

The Journal

Volume 17/4, December 2025

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

Contents

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

Contributed Research Articles

csurvey: Implementing Order Constraints in Survey Data Analysis	4
LHD: An All-encompassing R Package for Constructing Optimal Latin Hypercube Designs	20
longevity: An R Package for Modelling Excess Lifetimes	37
movieROC: Visualizing the Decision Rules Underlying Binary Classification	59
dtComb: A Comprehensive R Library and Web Tool for Combining Diagnostic Tests .	80
drclust: An R Package for Simultaneous Clustering and Dimensionality Reduction .	103
AcceptReject: An R Package for Acceptance-Rejection Method	133
qCBA: An R Package for Postoptimization of Rule Models Learnt on Quantized Data .	156
simlandr: Simulation-Based Landscape Construction for Dynamical Systems	173
rvif: a Decision Rule to Detect Troubling Statistical Multicollinearity Based on Redefined VIF	192
ASML: An R Package for Algorithm Selection with Machine Learning	216
elhmc: An R Package for Hamiltonian Monte Carlo Sampling in Bayesian Empirical Likelihood	237
GeRnika: An R Package for the Simulation, Visualization and Comparison of Tumor Phylogenies	255
MultiATSM: An R Package for Arbitrage-Free Macrofinance Multicountry Affine Term Structure Models.	275
memshare: Memory Sharing for Multicore Computation in R with an Application to Feature Selection by Mutual Information using PDE	304

News and Notes

Bioconductor Notes, December 2025	321
Changes on CRAN	325
News from the Forwards Taskforce	327
R Foundation News	329

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

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

Editor-in-Chief:

Rob J Hyndman, Monash University, Australia

Executive editors:

Mark van der Loo, Statistics Netherlands and Leiden University, Netherlands
Emi Tanaka, Australian National University, Australia
Emily Zabor, Cleveland Clinic, United States

Technical editors:

Mitchell O'Hara-Wild, Monash University, Australia

R Journal Homepage:

<http://journal.r-project.org/>

Email of editors and editorial board:

r-journal@R-project.org

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

Editorial

by Rob J Hyndman

Editorial changes

This is the last issue for 2025, and so it also marks the end of Mark van der Loo's term as an Executive Editor of the *R Journal*. We thank him for his work over the last four years, and especially for his service as Editor-in-Chief during 2024. He has consistently worked to improve the *R Journal*, and raise the standard of published articles and packages, and we are grateful for his contributions.

This is also the last issue for me as Editor-in-Chief of the *R Journal*. It has been an honour and a pleasure to serve in this role, and a privilege to work with a great group of editors and associate editors. I am happy to hand the reins to Dr Emi Tanaka as the incoming Editor-in-Chief, who has been an Executive Editor since 2024. Emi is a Senior Lecturer at the Australian National University, and is a very active member of the R community with several contributed packages on CRAN.

We also welcome Professor Vincent Arel-Bundock, from the University of Montreal, as a new Executive Editor for the period 2026–2029. He joins Emi Tanaka, Emily Zabor and me as the team of Executive Editors for 2026.

We also welcome two new Associate Editors: Selçuk Korkmaz and Maciej Beręsewicz. We are grateful to them, and the large team of Associate Editors, for their willingness to contribute to the *R Journal*.

Finally, thanks to Mitchell O'Hara-Wild, who is stepping down as Technical Editor of the Journal. He has provided wonderful support to the *R Journal* over many years, solving countless bewildering technical issues in order to make the Journal website function smoothly. In his place, we welcome Abhishek Ulayil, who will be the new Technical Editor from 2026. We are grateful to Abhishek for taking on this valuable role.

1 New guidelines

We recently introduced new guidelines for papers about R packages (which covers the vast majority of the papers we publish). The new guidelines are available on the [R Journal website](#). The intention is to make the expectations for authors clearer, and to improve the quality and consistency of articles published in the *R Journal*. Prospective authors should follow these guidelines when preparing their submissions.

In this issue

On behalf of the editorial board, I am pleased to present Volume 17 Issue 4 of the *R Journal*. This issue features 15 research articles. Each article relates to an R package available on CRAN, providing an overview of the package, its functionality, and examples of its use. Supplementary material for each article, with fully reproducible code, is available for download from the Journal website. We also include news from CRAN, Bioconductor, the Forwards Taskforce, and the R Foundation.

Rob J Hyndman
Monash University

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

csurvey: Implementing Order Constraints in Survey Data Analysis

by Xiyue Liao and Mary C. Meyer

Abstract Recent work in survey domain estimation has shown that incorporating *a priori* assumptions about orderings of population domain means reduces the variance of the estimators. The new R package `csurvey` allows users to implement order constraints using a design specified in the well-known `survey` package. The order constraints not only give estimates that satisfy *a priori* assumptions, but they also provide smaller confidence intervals with good coverage, and allow for design-based estimation in small-sample or empty domains. A test for constant versus increasing domain means is available in the package, with generalizations to other one-sided tests. A cone information criterion may be used to provide evidence that the order constraints are valid. Examples with well-known survey data sets show the utility of the methods. This package is now available from the Comprehensive R Archive Network at <http://CRAN.R-project.org/package=csurvey>.

1 Introduction

We assume that a finite population is partitioned into a number of domains, and the goal is to estimate the population domain means for the study variable and provide valid inference, such as confidence intervals and hypothesis tests.

Order constraints are a common type of *a priori* knowledge in survey data analysis. For example, we might know that salary increases with job rank within job type and location, or average cholesterol in a population increases with age category, or test scores decrease as poverty increases, or that the amount of pollution decreases with distance from the source. There might be a “block ordering” of salaries by location, where the locations are partitioned into blocks where salaries for all locations in one block (such as major metropolitan areas) are assumed to be higher, on average, than salaries in another block (such as rural areas), without imposing orderings within the blocks.

The order constraints may be imposed on domain mean estimates, or systematic component estimates if the response variable is not Gaussian but belongs to an exponential family of distributions. For example, if we are estimating the proportion of people with diabetes in a population, the study variable might be binary (whether or not the subject has diabetes), and perhaps we want to assume that diabetes prevalence increases with age, within ethnicity and socio-economic categories. The `csurvey` package extends the `survey` (Lumley, 2004) package in that it allows the user to impose linear inequality constraints on the domain means. If the inequality constraints are valid, this leads to improved inference, as well as the ability to estimate means for empty or small-sample domains without additional assumptions. The `csurvey` package provides constrained estimates of means or proportions, estimated variances of the estimates, and confidence intervals where the upper and lower bounds of the intervals also satisfy the order constraints.

The `csurvey` package implements one-sided tests. Suppose the study variable is salary, and interest is in whether the subject’s salary is affected by the level of education of the subject’s father, controlling for the subject’s education level and field. The null hypothesis is that there is no difference in salary by the level of father’s education. The one-sided test with alternative hypothesis that the salary increases with father’s education has a higher power than the two-sided test with the larger alternative “salary is not the same across the levels of father’s education.”

Finally, the `csurvey` package includes graphical functions for visualizing the order-constrained estimator and comparing it to the unconstrained estimator. Confidence bands can also be displayed to illustrate estimation uncertainty.

This package relies on `coneproj` (Liao and Meyer, 2014) and `survey` for its core computations and handling of complex sampling designs. The domain grid is constructed using the `data.table` (Barrett et al., 2025) package. Additional functionality, such as data filtering, variable transformation, and visualization of model results, leverages functions from `igraph` (Csárdi et al., 2025), `dplyr` (Wickham et al., 2023), and `ggplot2` (Wickham, 2016). When simulating the mixture covariance matrix and the sampling distribution of the one-sided test statistic, selected functions from `MASS` (Venables and Ripley, 2002) and `Matrix` (Bates et al., 2025) are employed.

2 Research incorporated in `csurvey`

The `csurvey` package provides estimation and inference on population domain variables with order constraints, using recently developed methodology. Wu et al. (2016) considered a complete ordering on a sequence of domain means. They applied the pooled adjacent violators algorithm Brunk (1958) for domain mean estimation, and derived asymptotic confidence intervals that have smaller width without sacrificing coverage, compared to the estimators that do not consider the ordering. Oliva-Avilés et al. (2020) developed methodology for partial orderings and more general constraints on domains. These include block orderings and orderings on domains arranged in grids by multiple variables of interest. Xu et al. (2021) refined these methods by proposing variance estimators based on a mixture of covariance matrices, and showed that the mixture covariance estimator improves coverage of confidence intervals while retaining smaller interval lengths. Liao et al. (2024) showed how to use the order constraints to provide conservative design-based inference in domains with small sample sizes, or even in empty domains, without additional model assumptions. Xu and Meyer (2023) developed a test for constant versus increasing domain means, and extended it to one-sided tests for more general orderings. Oliva-Avilés et al. (2019) proposed a cone information criterion (CIC) for survey data as a diagnostic method to measure possible departures from the assumed ordering. This criterion is similar to Akaike's information criterion (AIC) (Akaike, 1973) and the Bayesian information criterion (BIC) (Schwarz, 1978), and can be used for model selection. For example, we can use CIC to choose between an unconstrained estimator and an order constrained estimator. The one with a smaller CIC value will be chosen.

All of the above papers include extensive simulations showing that the methods substantially improve inference when the order restrictions are valid. These methods are easy for users to implement in `csurvey`, with commands to impose common orderings.

3 How to use `csurvey`

Statisticians and practitioners working with survey data are familiar with the R package `survey` (see Lumley (2004)) for analysis of complex survey data. Commands in the package allow users to specify the survey design with which their data were collected, then obtain estimation and inference for population variables of interest. The new `csurvey` package extends the utility of `survey` by allowing users to implement order constraints on domains.

The `csurvey` package relies on the functions in the `survey` package such as `svydesign` and `svrepdesign`, which allow users to specify the survey design. This function produces an object that contains information about the sampling design, allowing the user to specify strata, sampling weights, etc. This object is used in statistical functions in the `csurvey` package in the same manner as for the `survey` package. In addition, the mixture covariance matrix in Xu et al. (2021) is constructed from an initial estimate of covariance obtained from `survey`.

Consider a finite population with labels $U = \{1, \dots, N\}$, partitioned into domains U_d , $d = 1, \dots, D$, where U_d has N_d elements. For a study variable y , suppose interest is in

estimating the population domain means

$$\bar{y}_{U_d} = \frac{\sum_{k \in U_d} y_k}{N_d}$$

for each d , and providing inference such as confidence intervals for each \bar{y}_{U_d} . Given a survey design, a sample $s \subset U$ is chosen, and the unconstrained estimator of the population domain means is a weighted average of the sample observations in each domain. This estimator $\tilde{y}_s = (\tilde{y}_{s_1}, \dots, \tilde{y}_{s_D})$ in [Hájek \(1971\)](#) is provided by the [survey](#) package.

The desired orderings are imposed as linear inequality constraints on the domain means, in the form of an $m \times D$ constraint matrix \mathbf{A} . The [csurvey](#) package will find the constrained estimator $\tilde{\theta}$ by minimizing

$$\min_{\theta} (\tilde{y}_s - \theta)^\top \mathbf{W}_s (\tilde{y}_s - \theta) \quad \text{such that } \mathbf{A}\theta \geq \mathbf{0}$$

where the weights \mathbf{W}_s are provided by the survey design. (See [Oliva-Avilés et al. \(2020\)](#) for details.) For a simple example of a constraint matrix, consider five domains with a simple ordering, where we assume $\bar{y}_{U_1} \leq \bar{y}_{U_2} \leq \bar{y}_{U_3} \leq \bar{y}_{U_4} \leq \bar{y}_{U_5}$. Perhaps these are average salaries over five job levels. Then the constraint matrix is

$$\mathbf{A} = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}.$$

For simple orderings on D domains, the constraint matrix is $(D - 1) \times D$. For a block ordering example, suppose we again have five domains, and we know that each of the population means in the first two domains must be smaller than each of the means of the last three domains. The constraint matrix is

$$\mathbf{A} = \begin{pmatrix} -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 1 \end{pmatrix}.$$

The number of rows for a block ordering with two blocks is $D_1 \times D_2$, where D_1 is the number of domains in the first block and D_2 is the number in the second block. The package also allows users to specify grids of domains with various order constraints along the dimensions of the grid. The constraint matrices are automatically generated for some standard types of constraints.

The [csurvey](#) package allows users to specify a simple ordering with the symbolic `incr` function. For example, suppose `y` is the survey variable of interest (say, cholesterol level), and the variable `x` takes values 1 through D (age groups for example). Suppose we assume that average cholesterol level is increasing with age in this population, and that the design is specified by the object `ds`. Then

```
ans <- csvy(y ~ incr(x), design = ds)
```

creates an object containing the estimated population means and confidence intervals. The `decr` function is used similarly, to specify decreasing means.

Next, suppose `x1` takes values 1 through D_1 and `x2` takes values 1 through D_2 (say, age group and ethnicity), and we wish to estimate the population means over the $D_1 \times D_2$ grid of values of `x1` and `x2`. If we assume that the population domain means are ordered in `x1` but there is no ordering in `x2`, then the command

```
ans <- csvy(y ~ incr(x1) * x2, design = ds)
```

will provide domain mean estimates where the means are non-decreasing in age, within each ethnicity. Note that we don't allow “+” when defining a formula in `csvy()` because we consider all combinations $D_1 \times D_2$ given by `x1` and `x2`.

For an example of a block ordering with three blocks, the command

```
ans <- csvy(y ~ block.Ord(x, order = c(1,1,1,2,2,2,3,3,3)), design = ds)
```

specifies that the variable `x` takes values 1 through 9, and the domains with values 1, 2, and 3 each have population means not greater than each of the population means in the domains with `x` values 4, 5, and 6. The domains with `x` values 4, 5, and 6 each have population means not greater than each of the population means in the domains with `x` values 7, 8, and 9. More examples of implementation of constraints will be given below.

Implementing order constraints leads to “pooling” of domains where the order constraints are binding. This naturally leads to smaller confidence intervals as the averaging is over a larger number of observations. The mixture covariance estimator for θ that was derived in Xu et al. (2021) is provided by `csurvey`. This covariance estimator is constructed by recognizing that for different samples, different constraints are binding, so that different sets of domains are “pooled” to obtain the constrained estimator. The covariance estimator then is a mixture of pooled covariance matrix estimators, with the mixture distribution approximated via simulations. Using this mixture rather than the covariance matrix for the observed pooling provides confidence intervals with coverage that is closer to the target, while retaining the shorter lengths. The method introduced in Liao et al. (2024) further pools information across domains to provide upper and lower confidence interval bounds that also satisfy the constraints, effectively reducing the confidence interval length for domains with small sample sizes, and allowing for estimation and inference in empty domains.

The test $H_0 : A\bar{y}_U = 0$ versus the one-sided $H_1 : A\bar{y}_U \geq 0$ in `csurvey` has improved power over the F -test implemented using the `anova` command using the object from `svyglm` in the `survey` package. That F -test uses the two-sided alternative $H_2 : A\bar{y}_U \neq 0$. For example, suppose we have measures of amounts of pollutants in samples of water from small lakes in a certain region. We also have measurements of distances from sources, such as factories or waste dumps, that are suspected of contributing to the pollution. The test of the null hypothesis that the amount of pollution does not depend on the distance will have greater power if the alternative is one-sided, that is, that the pollution amount is, on average, larger for smaller distances.

In the following sections, we only show the main part of code used to generate the results. Supplementary or non-essential code is available in the accompanying R script submitted with this manuscript.

4 NHANES Example with monotonic domain means

The National Health and Nutrition Examination Survey (NHANES) combines in-person interviews and physical examinations to produce a comprehensive data set from a probability sample of residents of the U.S. The data are made available to the public at this [CDC NHANES website](#). The subset used in this example is derived from the 2009–2010 NHANES cycle and is provided in the `csurvey` package as the `nhdat2` data set. We consider the task of estimating the average total cholesterol level (mg/dL), originally recorded as `LBXTC` in the raw NHANES data, by age (in years), corresponding to the original variable `RIDAGEYR`. Focusing on young adults aged 21 to 45, we analyze a sample of $n = 1,933$ participants and specify the survey design using the associated weights and strata available in the data.

```
library(csurvey)
data(nhdat2, package = "csurvey")
dstrat <- svydesign(ids = ~id, strata = ~str, data = nhdat2, weight = ~wt)
```

Then, to get the proposed constrained domain mean estimate, we use the `csvy` function in the `csurvey` package. In this function, `incr` is a symbolic function used to define that the population domain means of `chol` are increasing with respect to the predictor `age`.

```
ans <- csvy(chol ~ incr(age), design = dstrat, n.mix = 100)
```

The `n.mix` parameter controls the number of simulations used to estimate the mixture covariance matrix, with a default of `n.mix = 100`. To speed up computation, users can set `n.mix` to be a smaller number, e.g., 10.

We can extract from `ans` the CIC value for the constrained estimator as

```
cat("CIC (constrained):", ans$CIC, "\n")
#> CIC (constrained): 32.99313
```

and the CIC value for the unconstrained estimator as

```
cat("CIC (unconstrained):", ans$CIC.un, "\n")
#> CIC (unconstrained): 51.65159
```

We see that for this example, the constrained estimator has a smaller CIC value and this implies it is a better fit.

If we want to construct a contrast of domain means and get its standard error, we can use the `svycontrast` function, which is inherited from the `survey` package. Note that in the `survey` package, it is impossible to get a contrast estimate when there is any empty domain. In the `csurvey` package, we inherit this feature. For example, suppose we want to compare the average cholesterol for the first thirteen age groups to the last twelve age groups, we can code it as:

```
cat(svycontrast(ans, list(avg = c(rep(-1, 13)/13, rep(1, 12)/12))), "\n")
#> 19.44752
```

The `csvy` function produces both the constrained fit and the corresponding unconstrained fit using methods from the `survey` package. A visual comparison between the two fits can be easily obtained by applying the `plot` method to the resulting `csvy` object by specifying the argument `type = "both"`. For illustration, Figure 1 displays the estimated domain means along with 95% confidence intervals, generated by the following code:

```
plot(ans, type = "both")
```

The `confint` function can be used to extract the confidence interval for domain mean. When the response is not Gaussian, then `type="link"` produces the confidence interval for the average systematic component over domains, while `type="response"` produces the confidence interval for the domain mean.

For this data set, the sample sizes for each of the twenty-five ages range from 54 to 99, so that none of the domains is “small.” To demonstrate `csvy` in the case of small domains, we next provide domain mean estimates and confidence intervals for 400 domains, arranged in a grid with 25 ages, four waist-size categories, and four income categories. We divide the waist measurement by height to make a relative girth, and split the observations into four groups with variable name `wcat`, which is a 4-level ordinal categorical variable representing waist-to-height ratio categories, computed from `BMXWAIST` (waist circumference in cm) and `BMXHT` (height in cm) in the body measures file `BMX_F.XPT` from NHANES. We have income information for the subjects, in terms of a multiple of the federal poverty level. Our first income category includes those with income that is 75% or less than the poverty line. The

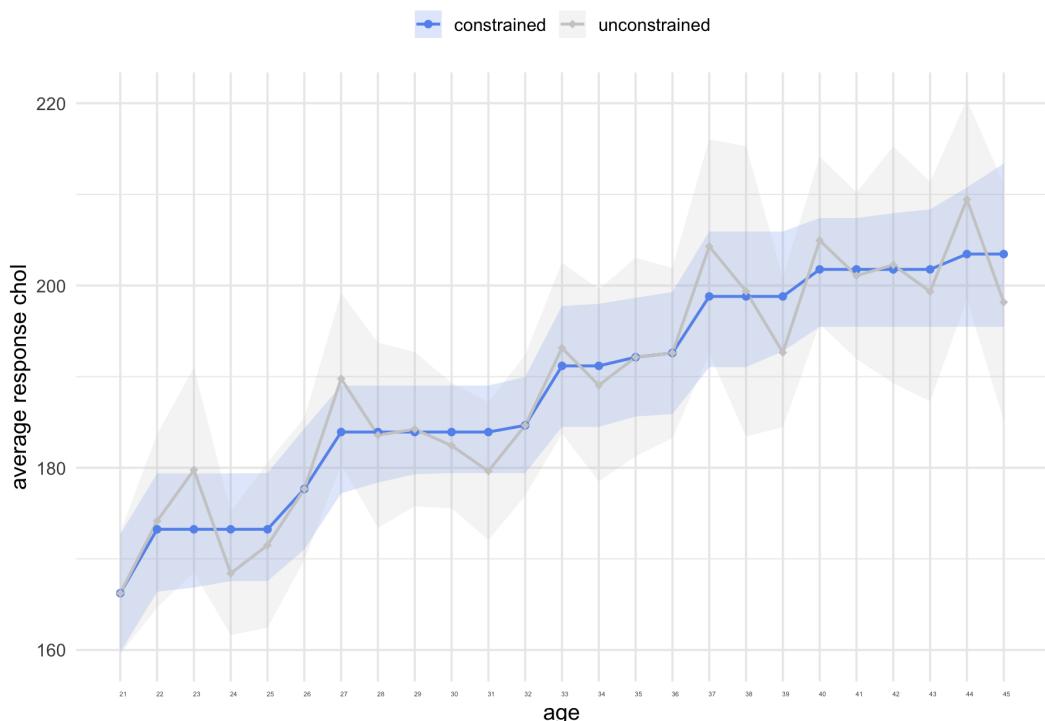


Figure 1: Estimates of average cholesterol level for 25 ages, with 95% confidence intervals, for a stratified sample in the R dataset ‘nhdat2’, $n = 1933$.

second category is .75 through 1.38 times the poverty level (1.38 determines Medicaid eligibility), the third category goes from above 1.38 to 3.5, and finally above 3.5 times the poverty level is the fourth category. These indicators are contained in the variable icat (categorized income). We create icat by the INDFMPIR variable from the file DEMO_F.XPT from NHANES.

The domain sample sizes average only 4.8 observations, and there are 16 empty domains. The sample size for each domain can be checked by ans\$nd. To estimate the population means we assume average cholesterol level is increasing in both age and waist size, but no ordering is imposed on income categories:

```
set.seed(1)
ans <- csvy(chol ~ incr(age)*incr(wcat)*icat, design = dstrat)
```

To extract estimates and confidence intervals for specific domains defined by the model, the user can use the predict function as follows:

```
domains <- data.frame(age = c(24, 35), wcat = c(2, 4), icat = c(2, 3))
pans <- predict(ans, newdata = domains, se.fit = TRUE)
cat("Predicted values, confidence intervals and standard errors for specified domains:\n")

##> Predicted values, confidence intervals and standard errors for specified domains:

print (pans)

##> $fit
##> [1] 162.9599 202.0061
##>
##> $lwr
##> [1] 148.6083 183.6379
##>
```

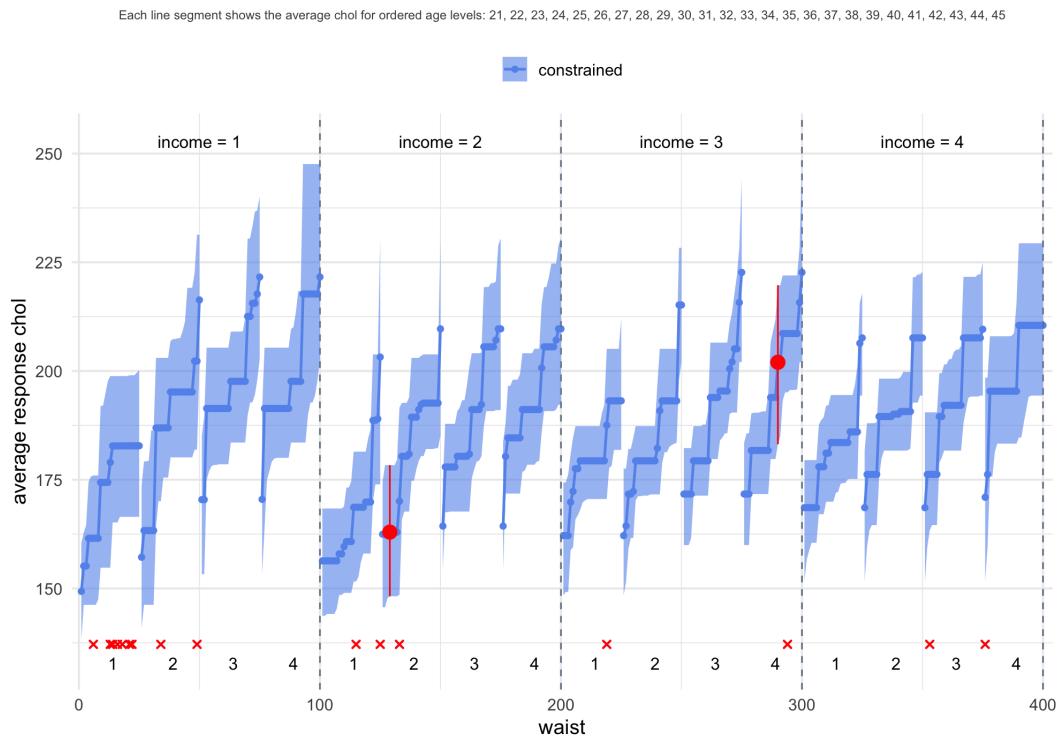


Figure 2: Constrained estimates of population domain means for 400 domains in a $25 \times 4 \times 4$ grid. The increasing population domain estimates for the 25 ages are shown within the waist size and income categories. The blue bands indicate 95% confidence intervals for the population domain means, with two specific domains, namely, $(\text{age}, \text{waist}, \text{income}) = (24, 2, 2)$ and $(35, 4, 3)$ marked in red. Empty domains are marked with a red 'x' sign.

```
#> $upp
#> [1] 177.8976 219.7764
#>
#> $se.fit
#> [1] 7.471908 9.219167
```

Figure 2 displays the domain mean estimates along with 95% confidence intervals, highlighting in red the two domains specified in the predict function. The code to create this plot is as below. The control argument is used to let the user adjust the aesthetics of the plot.

```
ctl <- list(x1lab = "waist", x2lab = "income", subtitle.size = 8)
plot(ans, x1 = "wcat", x2 = "icat", control = ctl, domains = domains)
```

The user can visualize the unconstrained fit by specifying `type = "unconstrained"` in the `plot` function. The corresponding output is presented in Figure 3.

```
plot(ans, x1 = "wcat", x2 = "icat", control = ctl, type="unconstrained")
```

Without the order constraints, the sample sizes are too small to provide valid estimates and confidence intervals, unless further model assumptions are used, as in some small area estimation methods. With order constraints, design-based estimation and inference are possible for substantially smaller domain sample sizes, compared to the unconstrained design-based estimation.

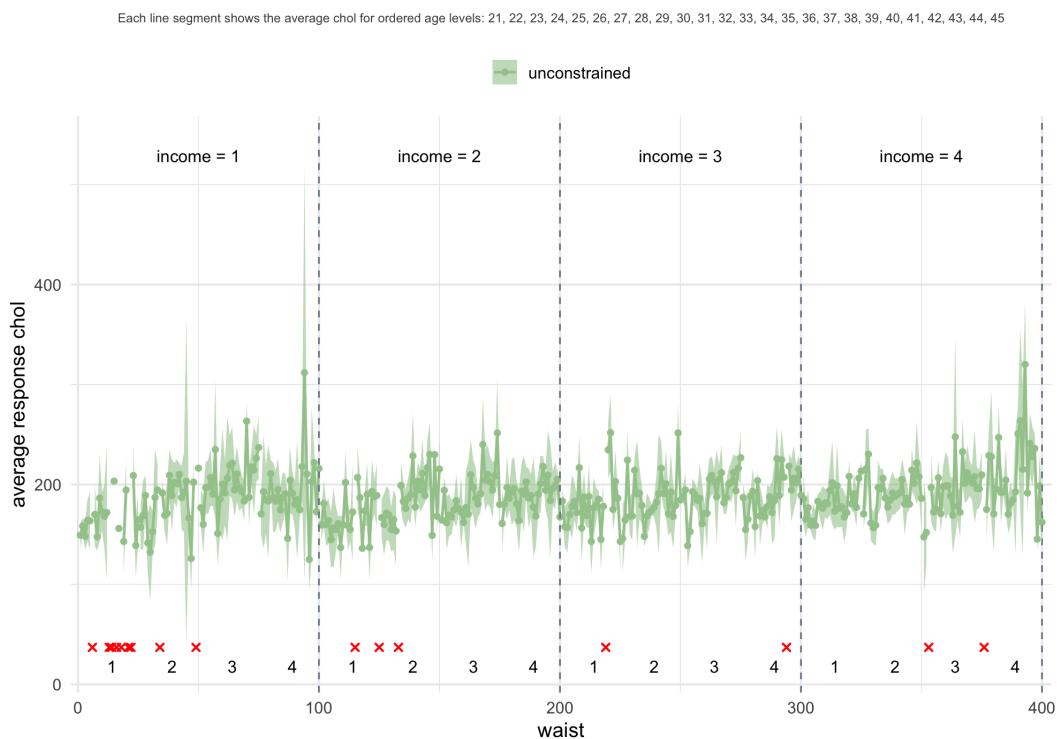


Figure 3: Unconstrained estimates of population domain means for 400 domains in a $25 \times 4 \times 4$ grid. The population domain estimates for the 25 ages are shown within the waist size and income categories. The green bands indicate 95% confidence intervals for the population domain means. Empty domains are marked with a red 'x' sign.

5 Constrained domain means with a block ordering

We consider the 2019 National Survey of College Graduates (NSCG), conducted by the U.S. Census Bureau and sponsored by the National Center for Science and Engineering Statistics (NCSES) within the National Science Foundation. The NSCG provides data on the characteristics of the nation's college graduates, with a focus on those in the science and engineering workforce. The datasets and documentation are available to the public on the [National Survey of College Graduates \(NSCF\) website](#). Replicate weights are available separately from NCSES upon request. Because the size of subsets of the NSCG survey used in this paper exceeds the size limit allowed for an R package stored on CRAN, the subsets are not included in the [csurvey](#) package. Instead, we provide the link to access the subsets `nscg19.rda` and `nscg19_2.rda` used in this section at this [website](#).

The study variable of interest is annual salary (denoted as `SALARY` in the dataset), which exhibits substantial right-skewness in its raw form. To reduce the influence of outliers and improve model stability, we restricted the analysis to observations with annual salaries between \$30,000 and \$400,000. A logarithmic transformation was applied to the salary variable to address skewness. Four predictors of salary were considered:

- Field of study (`field`, denoted by `NDGMEMG` in the raw dataset): This nominal variable defines the field of study for the highest degree. There are five levels: (1) Computer and mathematical sciences; (2) Biological, agricultural and environmental life sciences; (3) Physical and related sciences; (4) Social and related sciences; (5) Engineering. *Block ordering constraint*: given the other predictors, the average annual salary for each of the fields (2) and (4) is less than for the STEM fields (1), (3) and (5).
- Grouped year of award of highest degree (`hd_year_grouped`, denoted by `HDAY5`): This ordinal variable has five levels: (1) 1995 to 1999; (2) 2000 to 2004; (3) 2005 to 2009; (4) 2010 to 2014; (5) 2015 or later. *Isotonic constraint*: given the other predictors, the

average annual salary decreases with the year of award of highest degree; i.e., the more experience respondents have, on average, the higher the annual salary.

- Highest degree type (`hd_type`, denoted by DGRDG): The three levels are: (1) Bachelor's; (2) Master's; (3) Doctorate and Professional. *Isotonic constraint*: given the other predictors, the average annual salary increases with respect to the highest degree type.
- Region code for employer (`region`, denoted by EMRG): This nominal variable defines the regions in which the respondents worked within the U.S. Nine levels are: (1) New England; (2) Middle Atlantic; (3) East North Central; (4) West North Central; (5) South Atlantic; (6) East South Central; (7) West South Central; (8) Mountain; (9) Pacific and US Territories. There is no constraint for this predictor.

This data set contains $n = 30,368$ observations in a four-dimensional grid of 675 domains, where the sample size in the domains ranges from one to 491. Here, we specify the shape and order constraints in a similar fashion as we did in previous examples. The symbolic routine `block.Ord` is used to impose a block ordering on `field` and the order is specified in the `order` argument. The `svydesign` specifies a survey design with no clusters. The command `svrepdesign` creates a survey design with replicate weights, where the columns named as "RW0001", "RW0002", ..., "RW0320" are the 320 NSCG replicate weights and `weights(= ~w)` denotes the sampling weight. The variance is computed as the sum of squared deviation of the replicates from the mean. The general formula for computing a variance estimate using replicate weights follows:

$$v_{REP}(\hat{\theta}) = \sum_{r=1}^R c_r (\hat{\theta}_r - \hat{\theta})^2$$

where the estimate $\hat{\theta}$ is computed based on the final full sample survey weights and the calculation of each replicate estimate $\hat{\theta}_r$ is according to the r th set of replicate weights ($r = 1, \dots, R$). The replication adjustment factor c_r is a multiplier for the r th replicate of squared difference. The `scale(= 1)` is an overall multiplier and the `rscale(= 0.05)` denotes a vector of replicate-specific multipliers, which are the values of c_r in above formula.

```
load("./nscg19.rda")
rds <- svrepdesign(data = nscg, repweights = dplyr::select(nscg, "RW0001":"RW0320"),
  weights = ~w, combined.weights = TRUE, mse = TRUE, type = "other", scale = 1, rscale = 0.05)
```

Estimates of domain means for the 225 domains for which the highest degree is a Bachelor's are shown in Figure 4, as surfaces connecting the estimates over the grid of field indicators and year of award of highest degree. The block-ordering constraints can be seen in the surfaces, where the fields labeled 2 and 4 have lower means than those labeled 1, 3, and 5. The surfaces are also constrained to be decreasing in year of award of highest degree.

To improve computational efficiency, the simulations used to estimate the mixture covariance matrix and the sampling distribution of the test statistic can be parallelized. This can be enabled by setting the following R option

```
options(csurvey.multicore = TRUE)
```

The model is fitted using the following code

```
ans <- csvy(logSalary~decr(hd_year_grouped)*incr(hd_type)*block.Ord(field,
  order = c(2, 1, 2, 1, 2))*region, design = rds)
```

The `plotpersp` function is used to create Figure 4. It will create a three-dimensional estimated surface plot when there are at least two predictors. In this example, we use `plotpersp` to generate a 3D perspective plot of the estimated average log-transformed salary

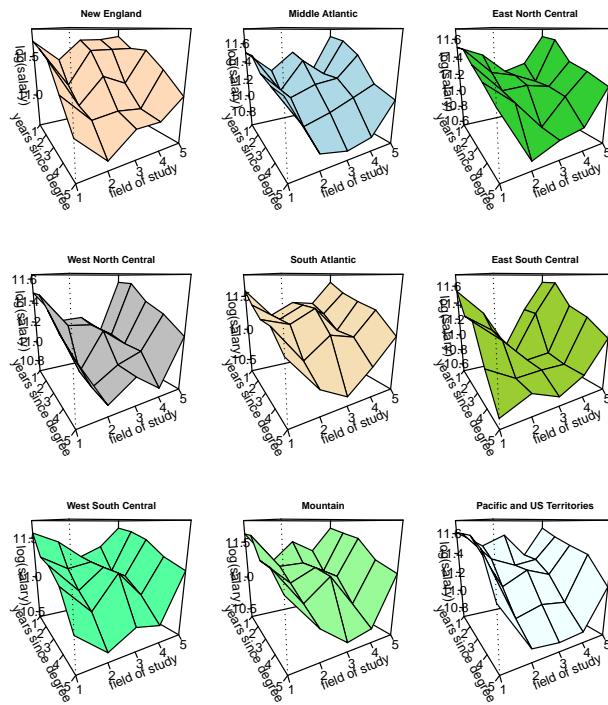


Figure 4: Estimates of average $\log(\text{salary})$ by field of study and year of degree, for observations where highest degree is a Bachelor's, for each of the nine regions.

with respect to field and year of award of highest degree. Among the three `hd_type` categories, the Bachelor's degree group has the highest number of observations. The `plotpersp` function visualizes the 225 domains corresponding to the most frequently observed level of this fourth predictor. Plot aesthetics are customized via the `control` argument:

```
ctl <- list(categ = "region", categnm = c("New England", "Middle Atlantic", "East North Central",
  "West North Central", "South Atlantic", "East South Central", "West South Central", "Mountain",
  "Pacific and US Territories"), NCOL = 3, th = 60, xlab = "years since degree",
  ylab = "field of study", zlab = "log(salary)")

plotpersp(ans, x1="hd_year_grouped", x2="field", control = ctl)
```

Estimates and confidence intervals for the 75 domain means associated with three of the regions are shown in Figure 5. The constrained estimates are shown as blue dots, and the unconstrained means are depicted as grey dots. The confidence intervals are indicated with blue bands for the constrained estimates and grey bands for the unconstrained estimates.

For the Northeast Region, the average length of the constrained confidence intervals is .353, while the average length for the unconstrained confidence intervals is .477. In the domain for those in the math and computer science field, with PhDs obtained in 2000-2004, the unconstrained log-salary estimate for this domain is much below the corresponding constrained estimate, because the latter is forced to be at least that of lower degrees, and that of newer PhDs. If the constraints hold in the population, then the unconstrained confidence interval is unlikely to capture the population value. As seen in the NHANES example in Section 2.4, the unconstrained estimators are unreliable when the sample domain size is small. The Pacific region has the largest sample sizes, ranging from 12 to 435 observations. With this larger sample size, most of the unconstrained estimates already satisfy the constraints, but the average length for the constrained estimator is .301, while the average length for the unconstrained estimator is .338, showing that the mixture covariance matrix leads to more precision in the confidence intervals. Also shown is the region with the smallest sample sizes: the East South Central region. Here the average length for the

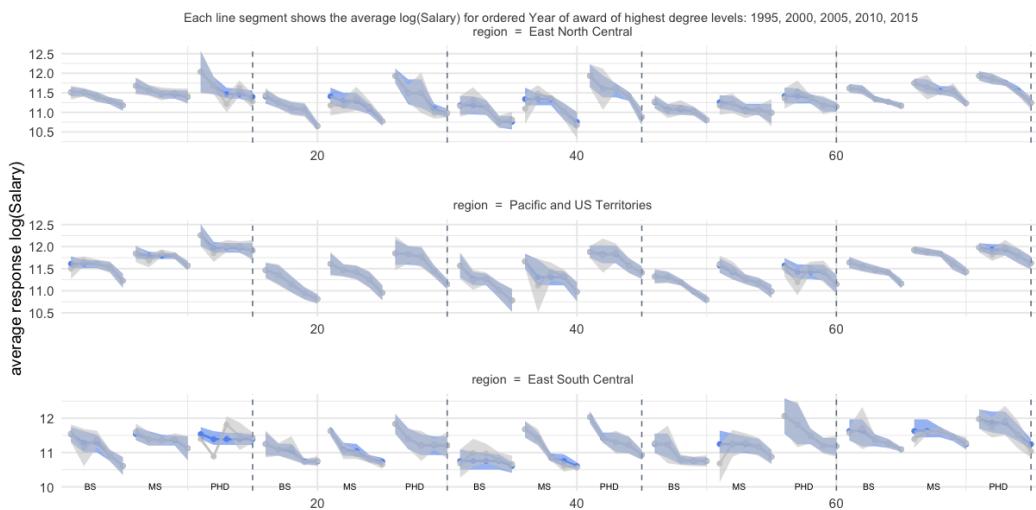


Figure 5: Estimates of average log(salary) for the 75 domains in each of three regions. The blue dots represent the constrained domain mean estimates, while the grey dots represent the unconstrained domain mean estimates. The blue band is the 95% confidence interval for the domains, using the constraints; the grey band is the 95% unconstrained domain mean confidence interval.

unconstrained confidence intervals is .488 while the average length for the constrained confidence intervals is .444.

6 One-sided testing

For this example we again use the NCGS data set. But for this example, we use some new variables and make some variable groupings. First, instead of using the grouped year of award of highest degree (`hd_year_grouped`), we use the actual calendar year when the highest degree was awarded (`hd_year`, denoted as `HDACYR` in the raw data set). We conduct the one-sided test for six pairs of consecutive calendar years. Besides, we choose father's education level (`daded`, denoted as `EDDAD`) as the main predictor, and the interest is in determining whether salaries are higher for people whose father has a higher education. In the raw data set, `EDDAD` has seven categories, we group them into five categories for this example: 1 = no high school degree, 2 = high school degree but no college degree, 3 = bachelor's degree, 4 = master's degree, and 5 = PhD or professional degree. We further group `region` (`EMRG`) as: 1 = Northeast, 2 = North Central, 3 = Southeast, 4 = West, 5 = Pacific and Territories, and group `field` (`NDGMEMG`) as: 1 = Computer and Mathematical Sciences, Physical and Related Sciences, Engineering, which can be considered as core STEM fields, 2 = Biological, Agricultural, and Environmental Life Sciences, Social and Related Sciences, which can be considered as life and social sciences, 3 = Science and Engineering-Related Fields, 4 = Non-Science and Engineering Fields, which is Non-STEM. Finally, people's highest degree type (denoted as `DGRDG` in the raw data) is used to limit the study sample to people whose highest degree is a Master's degree. The sample size is $n = 25,177$.

It seems reasonable that people whose parents are more educated are more educated themselves, but if we control for the person's level of education, as well as field, region, and year of degree, will the salary still be increasing with level of father's education? For this, the null hypothesis is that within the region, field, and year of degree, the salary is constant over levels of father's education. The one-sided alternative is that salaries increase with father's education level.

The constrained and unconstrained fits to the data under the alternative hypotheses are shown in Figure 6 for five regions and four fields, for degree years 2016-2017. The dots connected by solid blue lines represent the fit with the one-sided alternative, i.e., constrained

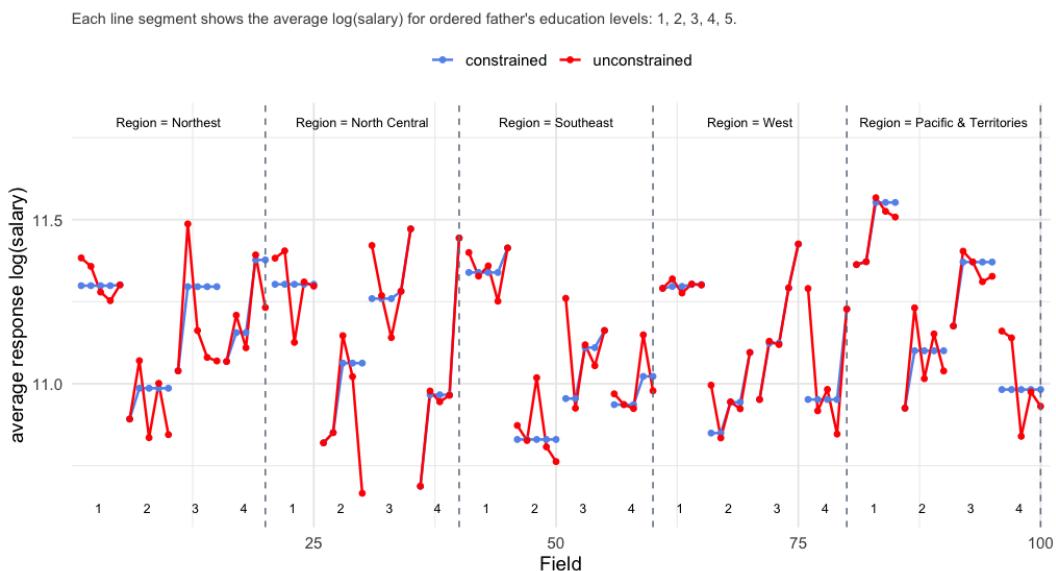


Figure 6: Estimates of average log(salary) by father's education level, for each of five regions and four fields, for subjects whose degree was attained in 2016-2017. The solid blue lines connect the estimates where the average salary is constrained to be increasing in father's education, and the solid red lines connect unconstrained estimates of average salary.

to be increasing in father's education. The dots connected by solid red lines represent the fit with the two-sided alternative, i.e., with unconstrained domain means.

The p -values for the test of the null hypotheses are given in Table 1, where n/a is shown for the two-sided p -value in the case of an empty domain. The test is conducted for six pairs of consecutive calendar years with the sample sizes to be 2,129, 2,795, 3,069, 2,895, 2,423, and 1,368 respectively. The one-sided test consistently has a small p -value, indicating that the father's education level is positively associated with salary earned, even after controlling for region, degree type, and field.

Table 1: One-sided and two-sided p -values for the test of the null hypothesis that salary is constant in father's education level. The two-sided test results in n/a when the grid has at least one empty domain.

2008-09		2010-11		2012-13		2014-15		2016-17		2018-19	
one	two	one	two	one	two	one	two	one	two	one	two
.008	n/a	<.001	.018	<.001	<.001	<.001	n/a	.003	.417	<.001	n/a

The p -value of the one-sided test is included in the object ans. For example, to check the p -value for the fit for the year 2008 to 2009, we can fit the model and print out its summary table as:

```
load("./nscg19_2.rda")
data <- nscg2 |>
  dplyr::filter(hd_year %in% c(2008, 2009))

rds <- svrepdesign(data = data, repweights = dplyr::select(data, "RW0001":"RW0320"), weights = ~w,
                    combined.weights = TRUE, mse = TRUE, type = "other",
                    scale = 1, rscale = 0.05)

set.seed(1)
ans <- csvy(logSalary ~ incr(daded) * field * region, design = rds, test = TRUE)

summary(ans)
```

```
#> Call:
#> csvy(formula = logSalary ~ incr(daded) * field * region, design = rds,
#>       test = TRUE)
#>
#> Null deviance: 460.6182 on 97 degrees of freedom
#> Residual deviance: 391.3722 on 51 degrees of freedom
#>
#> Approximate significance of constrained fit:
#>                               edf mixture.of.Beta p.value
#> incr(daded):field:region 47          0.5536 0.0068 **
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#> CIC (constrained estimator): 0.0275
#> CIC (unconstrained estimator): 0.0354
```

7 Binary outcome

Finally, we use another subset of the NHANES 2009–2010 data to demonstrate how our method applies when the outcome is binary. This subset is included in the [csurvey](#) package as `nhdat`. The construction of variables, sampling weights, and strata in this subset closely follows the approach described in Section 2.4. It contains $n = 1,680$ observations with complete records on total cholesterol, age, height, and waist circumference for adults aged 21–40. The binary outcome indicates whether an individual has high total cholesterol, coded as 1 if total cholesterol exceeds 200 mg/dL, and 0 otherwise. We estimate the population proportion with high cholesterol by age, waist, and gender (1 = male, 2 = female). The waist variable, denoted as `wcat`, is a 4-level categorized ordinal variable representing waist-to-height ratios.

It is reasonable to assume that, on average, the proportion of individuals with high cholesterol increases with both age and waist. The model is specified using the following code:

```
data(nhdat, package = "csurvey")
dstrat <- svydesign(ids = ~ id, strata = ~ str, data = nhdat, weight = ~ wt)
set.seed(1)
ans <- csvy(chol ~ incr(age) * incr(wcat) * gender, design = dstrat,
            family = binomial(link = "logit"), test = TRUE)
```

The CIC of the constrained estimator is smaller than that of the unconstrained estimator, and the one-sided hypothesis test has a p -value close to zero.

```
summary(ans)

#> Call:
#> csvy(formula = chol ~ incr(age) * incr(wcat) * gender, design = dstrat,
#>       family = binomial(link = "logit"), test = TRUE)
#>
#> Null deviance: 2054.947 on 159 degrees of freedom
#> Residual deviance: 1906.762 on 126 degrees of freedom
#>
#> Approximate significance of constrained fit:
#>                               edf mixture.of.Beta p.value
#> incr(age):incr(wcat):gender 34          0.5174 < 2.2e-16 ***
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#> CIC (constrained estimator): 0.3199
#> CIC (unconstrained estimator): 1.2778
```

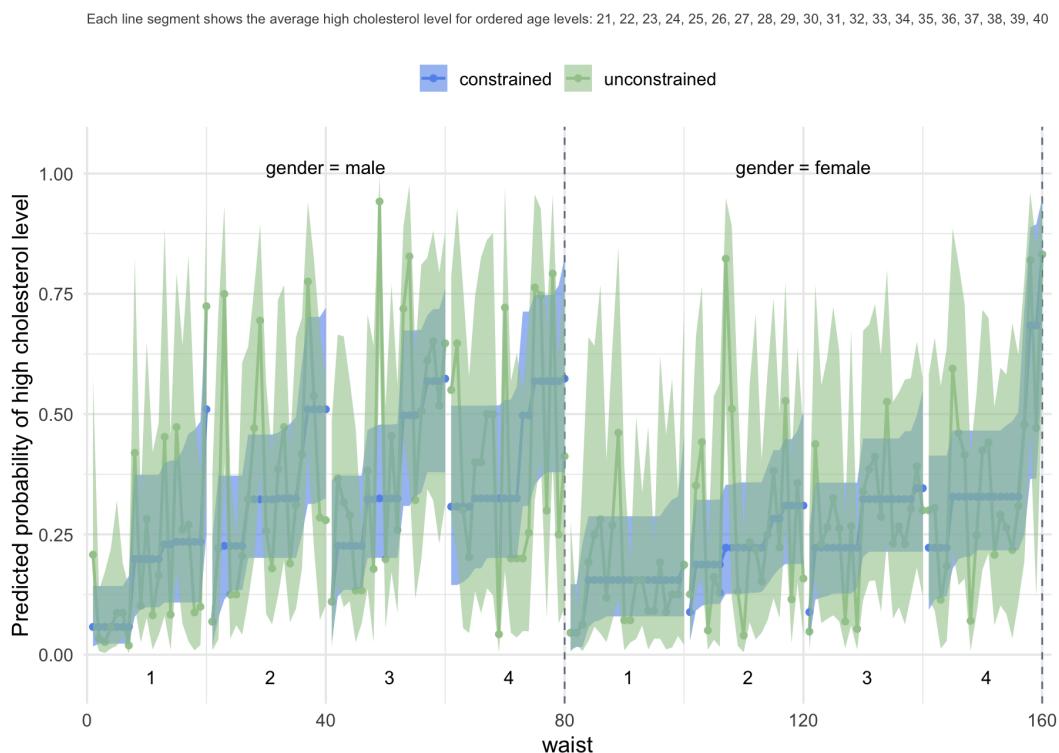


Figure 7: Estimates of probability of high cholesterol level for each combination of age, waist and gender. The blue dots represent the constrained domain mean estimates, while the green dots represent the unconstrained domain mean estimates. The blue band is the 95% confidence interval for the domains, using the constraints; the green band is the 95% unconstrained domain mean confidence interval.

The combination of age, waist, and gender gives 160 domains. This implies that the average sample size for each domain is only around 10. Due to the small sample sizes, the unconstrained estimator shows unlikely jumps as age increases within each waist category. On the other hand, the constrained estimator is more stable and tends to have smaller confidence intervals compared with the unconstrained Hájek estimator.

8 Discussion

While model-based small area estimators - such as those implemented in the `sae` package (Molina and Marhuenda, 2015) and the `emdi` package (Kreutzmann et al., 2019) - are powerful tools for borrowing strength across domains, they rely on parametric assumptions that may be violated in practice. Design-based methods remain essential for official statistical agencies, as they provide transparent and model-free inference that is directly tied to the survey design. Estimation and inference for population domain means with survey data can be substantially improved if constraints based on natural orderings are implemented. The `csurvey` package (version 1.15) (Liao, 2025) allows users to specify orderings on grids of domains, and obtain estimates of and confidence intervals for population domain means. This package also implements the design-based small area estimation method, which allows inference for population domain means for which the sample domain is empty, and further is used to improve estimates for domains with small sample size. The one-sided testing procedure available in `csurvey` has higher power than the standard two-sided test, and further can be applied in grids with some empty domains. Confidence intervals for domain means have better coverage rate and smaller interval width than what is produced by unconstrained estimation. Finally, the package provides functions to allow the user to easily visualize the data and the fits. The utility of the package has been demonstrated with well-known survey data sets.

Acknowledgment: This work was partially funded by NSF MMS-1533804.

References

- H. Akaike. Information theory as an extension of the maximum likelihood principle. In B. N. Petrov and F. Csaki, editors, *Second International Symposium on Information Theory*, pages 267–281. Akadémiai Kiadó, 1973. [p5]
- T. Barrett, M. Dowle, A. Srinivasan, J. Gorecki, M. Chirico, T. Hocking, B. Schwendinger, and I. Krylov. *data.table: Extension of ‘data.frame’*, 2025. URL <https://CRAN.R-project.org/package=data.table>. R package version 1.17.0. [p5]
- D. Bates, M. Maechler, and M. Jagan. *Matrix: Sparse and Dense Matrix Classes and Methods*, 2025. URL <https://CRAN.R-project.org/package=Matrix>. R package version 1.7-3. [p5]
- H. D. Brunk. Maximum likelihood estimates of monotone parameters. *The Annals of Mathematical Statistics*, 29(2):437–454, 1958. doi: 10.1214/aoms/1177728420. [p5]
- G. Csárdi, T. Nepusz, V. Traag, S. Horvát, F. Zanini, D. Noom, and K. Müller. *igraph: Network Analysis and Visualization in R*, 2025. URL <https://CRAN.R-project.org/package=igraph>. R package version 2.1.4. [p5]
- J. Hájek. Comment on a paper by D. Basu. In V. P. Godambe and D. A. Sprott, editors, *Foundations of Statistical Inference*, page 236. Holt, Rinehart and Winston, Toronto, 1971. [p6]
- A.-K. Kreutzmann, S. Pannier, N. Rojas-Perilla, T. Schmid, M. Templ, and N. Tzavidis. The R package emdi for estimating and mapping regionally disaggregated indicators. *Journal of Statistical Software*, 91(7):1–33, 2019. doi: 10.18637/jss.v091.i07. [p17]
- X. Liao. *csurvey: Constrained Regression for Survey Data*, 2025. URL <https://github.com/xliaosdsu/cssurvey>. R package version 1.15. [p17]
- X. Liao and M. C. Meyer. coneproj: An R package for the primal or dual cone projections with routines for constrained regression. *Journal of Statistical Software*, 61(12):1–22, 2014. doi: 10.18637/jss.v061.i12. [p5]
- X. Liao, M. C. Meyer, and X. Xu. Design-based estimation of small and empty domains in survey data analysis using order constraints. *Survey Methodology*, 50(2):303–321, 2024. URL <http://www.statcan.gc.ca/pub/12-001-x/2024002/article/00010-eng.pdf>. [p5, 7]
- T. Lumley. Analysis of complex survey samples. *Journal of Statistical Software*, 9(8):1–19, 2004. doi: 10.18637/jss.v009.i08. [p4, 5]
- I. Molina and Y. Marhuenda. sae: An R package for small area estimation. *The R Journal*, 7(1):81–98, jun 2015. URL <https://journal.r-project.org/archive/2015/RJ-2015-007/RJ-2015-007.pdf>. [p17]
- C. Oliva-Avilés, M. C. Meyer, and J. D. Opsomer. Checking validity of monotone domain mean estimators. *Canadian Journal of Statistics*, 47(2):315–331, 2019. doi: 10.1002/cjs.11496. [p5]
- C. Oliva-Avilés, M. C. Meyer, and J. D. Opsomer. Estimation and inference of domain means subject to qualitative constraints. *Survey Methodology*, 46(2):145–180, 2020. URL <http://www150.statcan.gc.ca/n1/pub/12-001-x/2020002/article/00002-eng.htm>. [p5, 6]
- G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464, 1978. doi: 10.1214/aos/1176344136. [p5]

- W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL <https://www.stats.ox.ac.uk/pub/MASS4/>. ISBN 0-387-95457-0. [p5]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p5]
- H. Wickham, R. François, L. Henry, K. Müller, and D. Vaughan. *dplyr: A Grammar of Data Manipulation*, 2023. URL <https://CRAN.R-project.org/package=dplyr>. R package version 1.1.4. [p5]
- J. Wu, M. C. Meyer, and J. D. Opsomer. Survey estimation of domain means that respect natural orderings. *Canadian Journal of Statistics*, 44(4):431–444, 2016. doi: 10.1002/cjs.11301. [p5]
- X. Xu and M. C. Meyer. One-sided testing of population domain means in surveys. *Survey Methodology*, 49(1):117–138, 2023. URL <https://www150.statcan.gc.ca/n1/en/pub/12-001-x/2023001/article/00001-eng.pdf>. [p5]
- X. Xu, M. C. Meyer, and J. D. Opsomer. Improved variance estimation for inequality-constrained domain mean estimators using survey data. *Journal of Statistical Planning and Inference*, 215:47–71, 2021. doi: 10.1016/j.jspi.2021.02.004. [p5, 7]

Xiyue Liao
San Diego State University
San Diego State University
Department of Mathematics and Statistics
San Diego, California 92182, United States of America
ORCID: 0000-0002-4508-9219
xliao@sdsu.edu

Mary C. Meyer
Colorado State University
Colorado State University
Department of Statistics
Fort Collins, Colorado 80523, United States of America
meyer@stat.colostate.edu

LHD: An All-encompassing R Package for Constructing Optimal Latin Hypercube Designs

by Hongzhi Wang, Qian Xiao and Abhyuday Mandal

Abstract Optimal Latin hypercube designs (LHDs), including maximin distance LHDs, maximum projection LHDs and orthogonal LHDs, are widely used in computer experiments. It is challenging to construct such designs with flexible sizes, especially for large ones, for two main reasons. One reason is that theoretical results, such as algebraic constructions ensuring the maximin distance property or orthogonality, are only available for certain design sizes. For design sizes where theoretical results are unavailable, search algorithms can generate designs. However, their numerical performance is not guaranteed to be optimal. Another reason is that when design sizes increase, the number of permutations grows exponentially. Constructing optimal LHDs is a discrete optimization process, and enumeration is nearly impossible for large or moderate design sizes. Various search algorithms and algebraic constructions have been proposed to identify optimal LHDs, each having its own pros and cons. We develop the R package LHD which implements various search algorithms and algebraic constructions. We embedded different optimality criteria into each of the search algorithms, and they are capable of constructing different types of optimal LHDs even though they were originally invented to construct maximin distance LHDs only. Another input argument that controls maximum CPU time is added to each of the search algorithms to let users flexibly allocate their computational resources. We demonstrate functionalities of the package by using various examples, and we provide guidance for experimenters on finding suitable optimal designs. The LHD package is easy to use for practitioners and possibly serves as a benchmark for future developments in LHD.

1 Introduction

Computer experiments are widely used in scientific research and industrial production, where complex computer codes, commonly high-fidelity simulators, generate data instead of real physical systems (Sacks et al., 1989; Fang et al., 2005). The outputs from computer experiments are deterministic (that is, free of random errors), and therefore replications are not needed (Butler, 2001; Joseph and Hung, 2008; Ba et al., 2015). Latin hypercube designs (LHDs, McKay et al. (1979)) may be the most popular type of experimental designs for computer experiments (Fang et al., 2005; Xiao and Xu, 2018), which avoid replications on every dimension and have uniform one-dimensional projections. According to practical needs, there are various types of optimal LHDs, including space-filling LHDs, maximum projection LHDs, and orthogonal LHDs. There is a rich literature on the construction of such designs, but it is still very challenging to find good ones for moderate to large design sizes (Ye, 1998; Fang et al., 2005; Joseph et al., 2015; Xiao and Xu, 2018). One key reason is that theoretical results, such as algebraic constructions which guarantee the maximin distance property or orthogonality, are only established for specific design sizes. These constructions provide theoretical guarantees on the design quality but are limited in their applicability. For design sizes where such theoretical guarantees do not exist, search algorithms can generate designs. However, the performance of search-based designs depends on the algorithm employed, the search space explored, and the computational resources allocated, meaning they cannot be guaranteed to be optimal. Constructing optimal LHDs is a discrete optimization process, where enumerating all possible solutions guarantees the optimal design for a given size. However, this approach becomes computationally infeasible as the number of permutations grows exponentially with increasing design sizes, making it another key reason that adds to the challenge.

An LHD with n runs and k factors is an $n \times k$ matrix with each column being a random permutation of numbers: $1, \dots, n$. Throughout this paper, n denotes the run size and k denotes the factor size. A space-filling LHD has its sampled region as scattered as possible, minimizing the unsampled region, thus accounting for the uniformity of all dimensions. Different criteria were proposed to measure designs' space-filling properties, including the maximin and minimax distance criteria (Johnson et al., 1990; Morris and Mitchell, 1995), the discrepancy criteria (Hickernell, 1998; Fang et al., 2002, 2005) and the entropy criterion (Shewry and Wynn, 1987). Since there are as many as $(n!)^k$ candidate LHDs for a given design size, it is nearly impossible to find the space-filling one by enumeration when n and k are moderate or large. In the current literature, both the search algorithms (Morris and Mitchell, 1995; Leary et al., 2003; Joseph and Hung, 2008; Ba et al., 2015; Ye et al., 2000; Jin et al., 2005; Liefvendahl and Stocki, 2006; Grosso et al., 2009; Chen et al., 2013) and algebraic constructions (Zhou and Xu, 2015; Xiao and Xu, 2017; Wang et al., 2018) are used to construct space-filling LHDs.

Space-filling designs often focus on the full-dimensional space. To further improve the space-filling properties of all possible sub-spaces, Joseph et al. (2015) proposed to use the maximum projection designs. Considering from two to $k - 1$ dimensional sub-spaces, maximum projection LHDs (MaxPro LHDs) are generally more space-filling compared to the classic maximin distance LHDs. The construction of MaxPro LHDs is also challenging, especially for large ones, and Joseph et al. (2015) proposed a simulated annealing (SA) based algorithm. In the LHD package, we incorporated the MaxPro criterion with other different algorithms such as the particle swarm optimization (PSO) and genetic algorithm (GA) framework, leading to many better MaxPro LHDs; see Section 3 for examples.

Unlike space-filling LHDs that minimize the similarities among rows, orthogonal LHDs (OLHDs) are another popular type of optimal design which consider similarities among columns. For example, OLHDs have zero column-wise correlations. Algebraic constructions are available for certain design sizes (Ye, 1998; Cioppa and Lucas, 2007; Steinberg and Lin, 2006; Sun et al., 2010, 2009; Yang and Liu, 2012; Georgiou and Efthimiou, 2014; Butler, 2001; Tang, 1993; Lin et al., 2009), but there are many design sizes where theoretical results are not available. In the LHD package, we implemented the average absolute correlation criterion and the maximum absolute correlation criterion (Georgiou, 2009) with SA, PSO, and GA to identify both OLHDs and nearly orthogonal LHDs (NOLHDs) for almost all design sizes.

This paper introduces the R package LHD available on the Comprehensive R Archive Network (<https://cran.r-project.org/web/packages/LHD/index.html>), which implements some currently popular search algorithms and algebraic constructions for constructing maximin distance LHDs, Maxpro LHDs, OLHDs and NOLHDs. We embedded different optimality criteria including the maximin distance criterion, the MaxPro criterion, the average absolute correlation criterion, and the maximum absolute correlation criterion in each of the search algorithms which were originally invented to construct maximin distance LHDs only (Morris and Mitchell, 1995; Leary et al., 2003; Joseph and Hung, 2008; Liefvendahl and Stocki, 2006; Chen et al., 2013), and each of them is capable of constructing different types of optimal LHDs through the package. To let users flexibly allocate their computational resources, we also embedded an input argument that limits the maximum CPU time for each of the algorithms, where users can easily define how and when they want the algorithms to stop. An algorithm can stop in one of two ways: either when the user-defined maximum number of iterations is reached or when the user-defined maximum CPU time is exceeded. For example, users can either allow the algorithm to run for a specified number of iterations without restricting the maximum CPU time or set a maximum CPU time limit to stop the algorithm regardless of the number of iterations completed. After an algorithm is completed or stopped, the number of iterations completed along with the average CPU time per iteration will be presented to users for their information. The R package LHD is an integrated tool for users with little or no background in design theory, and they can easily find optimal LHDs with desired sizes. Many new designs that are better than the existing ones are discovered; see Section 3.

The remainder of the paper is organized as follows. Section 2 illustrates different optimality criteria for LHDs. Section 3 demonstrates some popular search algorithms and their implementation details in the LHD package along with examples. Section 4 discusses some useful algebraic constructions as well as examples of how to implement them via the developed package. Section 5 concludes with a summary.

2 Optimality Criteria for LHDs

Various criteria are proposed to measure designs' space-filling properties (Johnson et al., 1990; Hickernell, 1998; Fang et al., 2002). In this paper, we focus on the currently popular maximin distance criterion (Johnson et al., 1990), which seeks to scatter design points over experimental domains so that the minimum distances between points are maximized. Let \mathbf{X} denote an LHD matrix throughout this paper. Define the L_q -distance between two runs x_i and x_j of \mathbf{X} as $d_q(x_i, x_j) = \left\{ \sum_{m=1}^k |x_{im} - x_{jm}|^q \right\}^{1/q}$ where q is an integer. Define the L_q distance of the design \mathbf{X} as $d_q(\mathbf{X}) = \min\{d_q(x_i, x_j), 1 \leq i < j \leq n\}$. In this paper, we consider $q = 1$ and $q = 2$, i.e. the Manhattan (L_1) and Euclidean (L_2) distances. A design \mathbf{X} is called a maximin L_q distance design if it has the unique largest $d_q(\mathbf{X})$ value among all designs of the same size. When more than one design has the same largest $d_q(\mathbf{X})$, the maximin distance design sequentially maximizes the next minimum inter-site distances. To evaluate the maximin distance criterion in a more convenient way, Morris and Mitchell (1995) and Jin et al. (2005) proposed to minimize a scalar value:

$$\phi_p = \left\{ \sum_{i=1}^{n-1} \sum_{j=i+1}^n d_q(x_i, x_j)^{-p} \right\}^{1/p}, \quad (1)$$

where p is a tuning parameter. This ϕ_p criterion in Equation (1) is asymptotically equivalent to the Maximin distance criterion as $p \rightarrow \infty$. In practice, $p = 15$ often suffices (Morris and Mitchell, 1995). In the LHD package, the function `phi_p()` implements this criterion.

Maximin distance LHDs focus on the space-filling properties in the full-dimensional space, but their space-filling properties in the subspaces are not guaranteed. Joseph et al. (2015) proposed the maximum projection criterion that considers designs' space-filling properties in all possible dimensional spaces. An LHD \mathbf{X} is called a maximum projection LHD (MaxPro LHD) if it minimizes the maximum projection criterion such that

$$\min_{\mathbf{X}} \psi(\mathbf{X}) = \left\{ \frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{1}{\prod_{l=1}^k (x_{il} - x_{jl})^2} \right\}^{1/k}. \quad (2)$$

From Equation (2), we can see that any two design points should be apart from each other in any projection to minimize the value of $\psi(\mathbf{X})$. Thus, the maximum projection LHDs consider the space-filling properties in all possible subspaces. Note that this criterion was originally defined using design points scaled to the unit hypercube $[0, 1]^k$ in Joseph et al. (2015), whereas our design points are represented as integer levels. A simple transformation can be applied to revert the scaling. For example, the transformation $\mathbf{X}_{Scaled} * n - 0.5$, can be applied, meaning that each element of every design point in scaled unit hypercube is multiplied by its run size n and then adding 0.5. The illustrative example at the end of Section 3 applies this transformation to ensure a fair comparison of performance. In the LHD package, the function `MaxProCriterion()` implements this criterion.

Orthogonal and nearly orthogonal designs that aim to minimize the correlations between factors are widely used in experiments (Georgiou, 2009; Steinberg and Lin, 2006; Sun and Tang, 2017). Two major correlation-based criteria to measure designs' orthogonality are the average absolute correlation criterion and the maximum absolute correlation criterion

(Georgiou, 2009), denoted as $\text{ave}(|q|)$ and $\max|q|$, respectively:

$$\text{ave}(|q|) = \frac{2 \sum_{i=1}^{k-1} \sum_{j=i+1}^k |q_{ij}|}{k(k-1)} \text{ and } \max|q| = \max_{i,j} |q_{ij}|, \quad (3)$$

where q_{ij} is the correlation between the i th and j th columns of the design matrix \mathbf{X} . Orthogonal designs have $\text{ave}(|q|) = 0$ and $\max|q| = 0$, which may not exist for all design sizes. Designs with a smaller $\text{ave}(|q|)$ or $\max|q|$ are generally preferred in practice. In the LHD package, functions `AvgAbsCor()` and `MaxAbsCor()` implement the criteria $\text{ave}(|q|)$ and $\max|q|$, respectively.

2.1 Illustrating Examples for the Introduced Optimality Criteria

This subsection demonstrates some examples of how to use the optimality criteria introduced above from the developed LHD package. To generate a random LHD matrix, the function `rLHD` can be used. For example,

```
> X = rLHD(n = 5, k = 3); X #This generates a 5 by 3 random LHD, denoted as X
      [,1] [,2] [,3]
[1,]    2     1     4
[2,]    4     3     3
[3,]    3     2     2
[4,]    1     4     5
[5,]    5     5     1
```

The input arguments for the function `rLHD` are the run-size n and the factor size k . Continuing with the above randomly generated LHD X , we evaluate it with respect to different optimality criteria. For example,

```
> phi_p(X)           #The maximin L1-distance criterion.
[1] 0.3336608
> phi_p(X, p = 10, q = 2)   #The maximin L2-distance criterion.
[1] 0.5797347
> MaxProCriterion(X)   #The maximum projection criterion.
[1] 0.5375482
> AvgAbsCor(X)         #The average absolute correlation criterion.
[1] 0.5333333
> MaxAbsCor(X)         #The maximum absolute correlation criterion.
[1] 0.9
```

The input arguments of the function `phi_p` are an LHD matrix X , p and q , where p and q come directly from the equation 1. Note that the default settings within function `phi_p` are $p = 15$ and $q = 1$ (the Manhattan distance) and user can change the settings. For functions `MaxProCriterion`, `AvgAbsCor`, and `MaxAbsCor`, there is only one input argument, which is an LHD matrix X .

3 Search Algorithms for Optimal LHDs with Flexible Sizes

3.1 Simulated Annealing Based Algorithms

Simulated annealing (SA, Kirkpatrick et al. (1983)) is a probabilistic optimization algorithm, whose name comes from the phenomenon of the annealing process in metallurgy. Morris and Mitchell (1995) proposed a modified SA that randomly exchanges the elements in LHD to seek potential improvements. If such an exchange leads to a better LHD under a given optimality criterion, the exchange is maintained. Otherwise, it is kept with a probability of $\exp[-(\Phi(\mathbf{X}_{\text{new}}) - \Phi(\mathbf{X}))/T]$, where Φ is a given optimality criterion, \mathbf{X} is the original LHD,

\mathbf{X}_{new} is the LHD after the exchange and T is the current temperature. In this article, we focus on minimizing the optimality criteria outlined in Section 2, meaning only minimization optimization problems are considered. Such probability guarantees that the exchange that leads to a slightly worse LHD has a higher chance of being kept than the exchange that leads to a significantly worse LHD, because an exchange which leads to a slightly worse LHD has a lower value of $\Phi(\mathbf{X}_{new}) - \Phi(\mathbf{X})$. Such an exchange procedure will be implemented iteratively to improve LHD. When there are no improvements after certain attempts, the current temperature T is annealed. Note that a large value of $\Phi(\mathbf{X}_{new}) - \Phi(\mathbf{X})$ (exchange leading to a significantly worse LHD) is more likely to remain during the early phase of the search process when T is relatively high, and it is less likely to stay later when T decreases (annealed). The best LHD is identified after the algorithm converges or the budget constraint is reached. In the LHD package, the function `SA()` implements this algorithm:

```
SA(n, k, N = 10, T0 = 10, rate = 0.1, Tmin = 1, Imax = 5, OC = "phi_p",
p = 15, q = 1, maxtime = 5)
```

Table 1 provides an overview of all the input arguments in `SA()`. n and k are the desired run size and factor size. T_0 is an initial temperature, $rate$ is the temperature decreasing rate, and T_{min} is the minimum temperature. If the current temperature is smaller than T_{min} , the current loop in the algorithm will stop and the current number of iterations will increase by one. There are two stopping criteria for the entire function: when the current number of iterations reaches the maximum (denoted as N in the function) or when the cumulative CPU time reaches the maximum (denoted as `maxtime` in the function), respectively. Either of those will trigger the stop of the function, whichever is earlier. For input argument OC (optimality criterion), "phi_p" returns maximin distance LHDs, "MaxProCriterion" returns MaxPro LHDs, and "AvgAbsCor" or "MaxAbsCor" returns orthogonal LHDs.

Table 1: Overview of Input Arguments of the SA Function

Argument	Description
n	A positive integer that defines the number of rows (or run size) of output LHD.
k	A positive integer that defines the number of columns (or factor size) of output LHD.
N	A positive integer that defines the maximum number of iterations in the algorithm. A large value of N will result in a high CPU time, and it is recommended to be no greater than 500. The default is set to be 10.
T_0	A positive number that defines the initial temperature. The default is set to be 10, which means the temperature anneals from 10 in the algorithm.
$rate$	A positive percentage that defines the temperature decrease rate, and it should be in $(0,1)$. For example, $rate=0.25$ means the temperature decreases by 25% each time. The default is set to be 10%.
T_{min}	A positive number that defines the minimum temperature allowed. When current temperature becomes smaller or equal to T_{min} , the stopping criterion for current loop is met. The default is set to be 1.
I_{max}	A positive integer that defines the maximum perturbations the algorithm will try without improvements before temperature is reduced. The default is set to be 5. For CPU time consideration, I_{max} is recommended to be no greater than 5.
OC	An optimality criterion. The default setting is "phi_p", and it could be one of the following: "phi_p", "AvgAbsCor", "MaxAbsCor", "MaxProCriterion".
p	A positive integer, which is one parameter in the ϕ_p formula, and p is preferred to be large. The default is set to be 15.
q	A positive integer, which is one parameter in the ϕ_p formula, and q could be either 1 or 2. If q is 1, the Manhattan (rectangular) distance will be calculated. If q is 2, the Euclidean distance will be calculated.
<code>maxtime</code>	A positive number, which indicates the expected maximum CPU time, and it is measured by minutes. For example, <code>maxtime=3.5</code> indicates the CPU time will be no greater than three and a half minutes. The default is set to be 5.

[Leary et al. \(2003\)](#) modified the SA algorithm in [Morris and Mitchell \(1995\)](#) to search for optimal orthogonal array-based LHDs (OALHDs). [Tang \(1993\)](#) showed that OALHDs tend to have better space-filling properties than random LHDs. The SA in [Leary et al. \(2003\)](#) starts with a random OALHD rather than a random LHD. The remaining steps are the same as the SA in [Morris and Mitchell \(1995\)](#). Note that the existence of OALHDs is determined by the existence of the corresponding initial OAs. In the LHD package, the function `OASA()` implements the modified SA algorithm.:

```
OASA(OA, N = 10, T0 = 10, rate = 0.1, Tmin = 1, Imax = 5, OC = "phi_p",
p = 15, q = 1, maxtime = 5),
```

where all the input arguments are the same as in SA except that OA must be an orthogonal array.

[Joseph and Hung \(2008\)](#) proposed another modified SA to identify the orthogonal-maximin LHDs, which considers both the orthogonality and the maximin distance criteria. The algorithm starts with generating a random LHD and then chooses the column that has the largest average pairwise correlations with all other columns. Next, the algorithm will select the row which has the largest total row-wise distance with all other rows. Then, the element at the selected row and column will be exchanged with a random element from the same column. The remaining steps are the same as the SA in [Morris and Mitchell \(1995\)](#). In the LHD package, the function `SA2008()` implements this algorithm:

```
SA2008(n, k, N = 10, T0 = 10, rate = 0.1, Tmin = 1, Imax = 5, OC = "phi_p",
p = 15, q = 1, maxtime = 5),
```

where all the input arguments are the same as in SA.

3.2 Particle Swarm Optimization Algorithms

Particle swarm optimization (PSO, [Kennedy and Eberhart \(1995\)](#)) is a metaheuristic optimization algorithm inspired by the social behaviors of animals. Recent research ([Chen et al., 2013](#)) adapted the classic PSO algorithm and proposed LaPSO to identify maximin distance LHDs. Since this is a discrete optimization task, LaPSO redefines the steps in which each particle updates its velocity and position in the general PSO framework. In the LHD package, the function `LaPSO()` implements this algorithm:

```
LaPSO(n, k, m = 10, N = 10, SameNumP = 0, SameNumG = n/4, p0 = 1/(k - 1),
OC = "phi_p", p = 15, q = 1, maxtime = 5)
```

Table 2 provides an overview of all the input arguments in `LaPSO()`, where n, k, N, OC, p, q, and maxtime are exactly the same as the input arguments in the function `SA()`. m is the number of particles, which represents candidate solutions in the PSO framework. SameNumP and SameNumG are two tuning parameters that denote how many exchanges would be performed to reduce the Hamming distance towards the personal best and the global best. p0 is the tuning parameter that denotes the probability of a random swap for two elements in the current column of the current particle to prevent the algorithm from being stuck at the local optimum. In [Chen et al. \(2013\)](#), they provided the following suggestions: SameNumP is approximately $n/2$ when SameNumG is 0, SameNumG is approximately $n/4$ when SameNumP is 0, and p0 should be between $1/(k - 1)$ and $2/(k - 1)$. The stopping criterion of the function is the same as that of the function SA.

3.3 Genetic Algorithms

The genetic algorithm (GA) is a nature-inspired metaheuristic optimization algorithm that mimics Charles Darwin's idea of natural selection ([Holland et al., 1992; Goldberg, 1989](#)). [Liefvendahl and Stocki \(2006\)](#) proposed a version of GA for identifying maximin distance LHDs. They implement the column exchange technique to solve the discrete optimization task. In the LHD package, the function `GA()` implements this algorithm:

Table 2: Overview of Input Arguments of the LaPSO Function

Argument	Description
n	A positive integer that defines the number of rows (or run size) of output LHD.
k	A positive integer that defines the number of columns (or factor size) of output LHD.
m	A positive integer that defines the number of particles, where each particle is a candidate solution. A large value of N will result in a high CPU time, and it is recommended to be no greater than 100. The default is set to be 10.
N	A positive integer that defines the maximum number of iterations in the algorithm. A large value of N will result in a high CPU time, and it is recommended to be no greater than 500. The default is set to be 10.
SameNumP	A non-negative integer that defines how many elements in current column of current particle should be the same as corresponding Personal Best. SameNumP can be 0, 1, 2, ..., n, where 0 means to skip the element exchange, which is the default setting.
SameNumG	A non-negative integer that defines how many elements in current column of current particle should be the same as corresponding Global Best. SameNumG can be 0, 1, 2, ..., n, where 0 means to skip the element exchange. The default setting is n/4. Note that SameNumP and SameNumG cannot be 0 at the same time.
p0	A probability of exchanging two randomly selected elements in current column of current particle LHD. The default is set to be 1/(k - 1).
OC	An optimality criterion. The default setting is "phi_p", and it could be one of the following: "phi_p", "AvgAbsCor", "MaxAbsCor", "MaxProCriterion".
p	A positive integer, which is one parameter in the ϕ_p formula, and p is preferred to be large. The default is set to be 15.
q	A positive integer, which is one parameter in the ϕ_p formula, and q could be either 1 or 2. If q is 1, the Manhattan (rectangular) distance will be calculated. If q is 2, the Euclidean distance will be calculated.
maxtime	A positive number, which indicates the expected maximum CPU time, and it is measured by minutes. For example, maxtime=3.5 indicates the CPU time will be no greater than three and a half minutes. The default is set to be 5.

```
GA(n, k, m = 10, N = 10, pmut = 1/(k - 1), OC = "phi_p", p = 15, q = 1,
maxtime = 5)
```

Table 3 provides an overview of all the input arguments in GA(), where n, k, N, OC, p, q, and maxtime are exactly the same as the input arguments in the function SA(). m is the population size, which represents how many candidate solutions in each iteration, and must be an even number. pmut is the tuning parameter that controls how likely the mutation would happen. When mutation occurs, two randomly selected elements will be exchanged in the current column of the current LHD. pmut serves the same purpose as p0 in LaPSO(), which prevents the algorithm from getting stuck at the local optimum, and it is recommended to be $1/(k - 1)$. The stopping criterion of the function is the same as that of the function SA.

3.4 Illustrating Examples for the Implemented Search Algorithms

This subsection demonstrates some examples on how to use the search algorithms in the developed LHD package. In Table 4, we summarize the R functions of the algorithms discussed in the previous subsections, which can be used to identify different types of optimal LHDs. Users who seek fast solutions can use the default settings of the input arguments after specifying the design sizes. See the following examples.

```
#Generate a 5 by 3 maximin distance LHD by the SA function.
> try.SA = SA(n = 5, k = 3); try.SA
```

Table 3: Overview of Input Arguments of the GA Function

Argument	Description
n	A positive integer that defines the number of rows (or run size) of output LHD.
k	A positive integer that defines the number of columns (or factor size) of output LHD.
m	A positive even integer, which stands for the population size and it must be an even number. The default is set to be 10. A large value of m will result in a high CPU time, and it is recommended to be no greater than 100.
N	A positive integer that defines the maximum number of iterations in the algorithm. A large value of N will result in a high CPU time, and it is recommended to be no greater than 500. The default is set to be 10.
pmut	A probability for mutation. When the mutation happens, two randomly selected elements in current column of current LHD will be exchanged. The default is set to be $1/(k - 1)$.
OC	An optimality criterion. The default setting is "phi_p", and it could be one of the following: "phi_p", "AvgAbsCor", "MaxAbsCor", "MaxProCriterion".
p	A positive integer, which is one parameter in the ϕ_p formula, and p is preferred to be large. The default is set to be 15.
q	A positive integer, which is one parameter in the ϕ_p formula, and q could be either 1 or 2. If q is 1, the Manhattan (rectangular) distance will be calculated. If q is 2, the Euclidean distance will be calculated.
maxtime	A positive number, which indicates the expected maximum CPU time, and it is measured by minutes. For example, maxtime=3.5 indicates the CPU time will be no greater than three and a half minutes. The default is set to be 5.

Table 4: Search algorithm functions in the LHD package

Function	Description
SA	Returns an LHD via the simulated annealing algorithm (Morris and Mitchell, 1995).
OASA	Returns an LHD via the orthogonal-array-based simulated annealing algorithm (Leary et al., 2003), where an OA of the required design size must exist.
SA2008	Returns an LHD via the simulated annealing algorithm with the multi-objective optimization approach (Joseph and Hung, 2008).
LaPSO	Returns an LHD via the particle swarm optimization (Chen et al., 2013).
GA	Returns an LHD via the genetic algorithm (Liefvendahl and Stocki, 2006).

```
[,1] [,2] [,3]
[1,] 2 2 1
[2,] 5 3 2
[3,] 4 5 5
[4,] 3 1 4
[5,] 1 4 3
> phi_p(try.SA) #\phi_p is smaller than that of a random LHD (0.3336608).
[1] 0.2169567

#Similarly, generations of 5 by 3 maximin distance LHD by the SA2008, LaPSO and GA functions.
> try.SA2008 = SA2008(n = 5, k = 3)
> try.LaPSO = LaPSO(n = 5, k = 3)
> try.GA = GA(n = 5, k = 3)

#Generate an OA(9,2,3,2), an orthogonal array with 9 runs, 2 factors, 3 levels, and 2 strength.
> OA = matrix(c(rep(1:3, each = 3), rep(1:3, times = 3)),
+               ncol = 2, nrow = 9, byrow = FALSE)
#Generates a maximin distance LHD with the same design size as the input OA
#by the orthogonal-array-based simulated annealing algorithm.
```

```
> try.OASA = OASA(OA)
> OA; try.OASA
 [,1] [,2]      [,1] [,2]
[1,]   1   1   [1,]   1   2
[2,]   1   2   [2,]   2   6
[3,]   1   3   [3,]   3   9
[4,]   2   1   [4,]   4   3
[5,]   2   2   [5,]   6   5
[6,]   2   3   [6,]   5   7
[7,]   3   1   [7,]   7   1
[8,]   3   2   [8,]   9   4
[9,]   3   3;  [9,]   8   8
```

Note that the default optimality criterion embedded in all search algorithms is "phi_p" (that is, the maximin distance criterion), leading to the maximin L_2 -distance LHDs. For other optimality criteria, users should change the setting of the input argument OC (with options "phi_p", "MaxProCriterion", "MaxAbsCor" and "MaxProCriterion"). The following examples illustrate some details of different argument settings.

```
#Below try.SA is a 5 by 3 maximin distance LHD generated by the SA with 30 iterations (N = 30).
#The temperature starts at 10 (T0 = 10) and decreases 10% (rate = 0.1) each time.
#The minimum temperature allowed is 1 (Tmin = 1) and the maximum perturbations that
#the algorithm will try without improvements is 5 (Imax = 5). The optimality criterion
#used is maximin distance criterion (OC = "phi_p") with p = 15 and q = 1, and the
#maximum CPU time is 5 minutes (maxtime = 5).
> try.SA = SA(n = 5, k = 3, N = 30, T0 = 10, rate = 0.1, Tmin = 1, Imax = 5, OC = "phi_p",
+               p = 15, q = 1, maxtime = 5); try.SA
 [,1] [,2] [,3]
[1,]   1   3   4
[2,]   2   5   2
[3,]   5   4   3
[4,]   4   1   5
[5,]   3   2   1
> phi_p(try.SA)
[1] 0.2169567

#Below try.SA2008 is a 5 by 3 maximin distance LHD generated by SA with
#the multi-objective optimization approach. The input arguments are interpreted
#the same as the design try.SA above.
> try.SA2008 = SA2008(n = 5, k = 3, N = 30, T0 = 10, rate = 0.1, Tmin = 1, Imax = 5,
+                       OC = "phi_p", p = 15, q = 1, maxtime = 5)

#Below try.OASA is a 9 by 2 maximin distance LHD generated by the
#orthogonal-array-based simulated annealing algorithm with the input
#OA (defined previously), and the rest input arguments are interpreted the
#same as the design try.SA above.
> try.OASA = OASA(OA, N = 30, T0 = 10, rate = 0.1, Tmin = 1, Imax = 5,
+                   OC = "phi_p", p = 15, q = 1, maxtime = 5)

#Below try.LaPSO is a 5 by 3 maximum projection LHD generated by the particle swarm
#optimization algorithm with 20 particles (m = 20) and 30 iterations (N = 30).
#Zero (or two) elements in any column of the current particle should be the same as
#the elements of corresponding column from personal best (or global best), because
#of SameNumP = 0 (or SameNumG = 2).
#The probability of exchanging two randomly selected elements is 0.5 (p0 = 0.5).
#The optimality criterion is maximum projection criterion (OC = "MaxProCriterion").
#The maximum CPU time is 5 minutes (maxtime = 5).
> try.LaPSO = LaPSO(n = 5, k = 3, m = 20, N = 30, SameNumP = 0, SameNumG = 2,
+                     p0 = 0.5, OC = "MaxProCriterion", maxtime = 5); try.LaPSO
 [,1] [,2] [,3]
```

```
[1,] 4 5 4
[2,] 3 1 3
[3,] 5 2 1
[4,] 2 3 5
[5,] 1 4 2
#Recall the value is 0.5375482 from the random LHD in Section 2.
> MaxProCriterion(try.LaPSO)
[1] 0.3561056

#Below try.GA is a 5 by 3 OLHD generated by the genetic algorithm with the
#population size 20 (m = 20), number of iterations 30 (N = 30), mutation
#probability 0.5 (pmut = 0.5), maximum absolute correlation criterion
#(OC = "MaxAbsCor"), and maximum CPU time 5 minutes (maxtime = 5).
> try.GA = GA(n = 5, k = 3, m = 20, N = 30, pmut = 0.5, OC = "MaxAbsCor",
+               maxtime = 5); try.GA
 [,1] [,2] [,3]
[1,] 2 1 2
[2,] 4 4 5
[3,] 3 5 1
[4,] 5 2 3
[5,] 1 3 4
#Recall the value is 0.9 from the random LHD in Section 2.
> MaxAbsCor(try.GA)
[1] 0.1      #The maximum absolute correlation between columns is 0.1
```

Next, we discuss some details of the implementation. In SA based algorithms (SA, SA2008, and OASA), the number of iterations N is recommended to be no greater than 500 for computing time considerations. The input rate determines the percentage of the decrease in current temperature (for example, 0.1 means a decrease of 10% each time). A high rate would make the temperature rapidly drop, which leads to a fast stop of the algorithm. It is recommended to set rate from 0.1 to 0.15. Imax indicates the maximum perturbations that the algorithm will attempt without improvements before the temperature reduces, and it is recommended to be no greater than 5 for computing time considerations. OC chooses the optimality criterion, and the "phi_p" criterion in (1) is set as default. OC has other options, including "MaxProCriterion", "AvgAbsCor" and "MaxAbsCor". Our algorithms support both the L_1 and L_2 distances.

For every algorithm, we incorporate a progress bar to visualize the computing time used. After an algorithm is completed, information on "average CPU time per iteration" and "numbers of iterations completed" will be presented. Users can set the limit for the CPU time used for each algorithm using the argument maxtime, according to their practical needs.

We also provide some illustrative code to demonstrate that the designs found in the LHD package are better than the existing ones, and the code in the following can be easily modified to construct other design sizes or other LHD types. Out of 100 trials, the code below shows the GA in the LHD package constructed better MaxPro LHDs 99 times compared to the algorithm in the MaxPro package, when 500 iterations are set for both algorithms. We did not compare the CPU time between these two packages since one is written in the R environment and the other one is written in the C++ environment, but with the same number of iterations, the GA in the LHD package almost always constructs better MaxPro LHDs.

```
#Make sure both packages are properly installed before load them
> library(LHD)
> library(MaxPro)

> count = 0 #Define a variable for counting purpose

> k = 5      #Factor size 5
> n = 10*k   #Run size = 10*factor size
```

```
#Setting 500 iterations for both algorithms, below loop counts
#how many times the GA from LHD package outperforms the algorithm
#from MaxPro package out of 100 times
> for (i in 1:100) {

  LHD = LHD::GA(n = n, k = k, m = 100, N = 500)
  MaxPro = MaxPro::MaxProLHD(n = n, p = k, total_iter = 500)$Design

  #MaxPro * n + 0.5 applied the transformation mentioned in Section 2
  #to revert the scaling.
  Result.LHD = LHD::MaxProCriterion(LHD)
  Result.MaxPro = LHD::MaxProCriterion(MaxPro * n + 0.5)

  if (Result.LHD < Result.MaxPro) {count = count + 1}

}

> count
[1] 99
```

4 Algebraic Constructions for Optimal LHDs with Certain Sizes

There are algebraic constructions available for certain design sizes, and theoretical results are developed to guarantee the efficiency of such designs. Algebraic constructions almost do not require any searching, which are especially attractive for large designs. In this section, we present algebraic constructions that are available in the LHD package for maximin distance LHDs and orthogonal LHDs.

4.1 Algebraic Constructions for Maximin Distance LHDs

[Wang et al. \(2018\)](#) proposed to generate maximin distance LHDs via good lattice point (GLP) sets ([Zhou and Xu, 2015](#)) and Williams transformation ([Williams, 1949](#)). In practice, their method can lead to space-filling designs with relatively flexible sizes, where the run size n is flexible but the factor size k must be no greater than the number of positive integers that are co-prime to n . They proved that the resulting designs of sizes $n \times (n - 1)$ (with n being any odd prime) and $n \times n$ (with $2n + 1$ or $n + 1$ being odd prime) are optimal under the maximin L_1 distance criterion. This construction method by [Wang et al. \(2018\)](#) is very attractive for constructing large maximin distance LHDs. In the LHD package, function `FastMmLHD()` implements this method:

```
FastMmLHD(n, k, method = "manhattan", t1 = 10),
```

where n and k are the desired run size and factor size. `method` is a distance measure method which can be one of the following: "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski". Any unambiguous substring can be given. `t1` is a tuning parameter, which determines how many repeats will be implemented to search for the optimal design. The default is set to be 10.

[Tang \(1993\)](#) proposed to construct orthogonal array-based LHDs (OALHDs) from existing orthogonal arrays (OAs), and [Tang \(1993\)](#) showed that the OALHDs can have better space-filling properties than the general ones. In the LHD package, function `OA2LHD()` implements this method:

```
OA2LHD(OA),
```

where OA is an orthogonal array matrix. Users only need to input an OA and the function will return an OALHD with the same design size as the input OA.

4.2 Algebraic Constructions for Orthogonal LHDs

Orthogonal LHDs (OLHDs) have zero pairwise correlation between any two columns, which are widely used by practitioners. There is a rich literature on the constructions of OLHDs with various design sizes, but they are often too hard for practitioners to replicate in practice. The LHD package implements some currently popular methods (Ye, 1998; Cioppa and Lucas, 2007; Sun et al., 2010; Tang, 1993; Lin et al., 2009; Butler, 2001) for practitioners and the functions are easy to use.

Ye (1998) proposed a construction for OLHDs with run sizes $n = 2^m + 1$ and factor sizes $k = 2m - 2$ where m is any integer bigger than 2. In the LHD package, function `OLHD.Y1998()` implements this algebraic construction:

```
OLHD.Y1998(m),
```

where input argument m is the m in the construction of Ye (1998). Cioppa and Lucas (2007) extended Ye (1998)'s method to construct OLHDs with run size $n = 2^m + 1$ and factor size $k = m + \binom{m-1}{2}$, where m is any integer bigger than 2. In the LHD package, function `OLHD.C2007()` implements this algebraic construction with input argument m remaining the same:

```
OLHD.C2007(m)
```

Sun et al. (2010) extended their earlier work (Sun et al., 2009) to construct OLHDs with $n = r2^{c+1} + 1$ or $n = r2^{c+1}$ and $k = 2^c$, where r and c are positive integers. In the LHD package, function `OLHD.S2010()` implements this algebraic construction:

```
OLHD.S2010(c, r, type = "odd"),
```

where input arguments C and r are c and r in the construction. When input argument `type` is "odd", the output design size would be $n = r2^{c+1} + 1$ by $k = 2^c$. When input argument `type` is "even", the output design size would be $n = r2^{c+1}$ by $k = 2^c$.

Lin et al. (2009) constructed OLHDs or NOLHDs with n^2 runs and $2fp$ factors by coupling `OLHD(n, p)` or `NOLHD(n, p)` with an $OA(n^2, 2f, n, 2)$. For example, an `OLHD(11, 7)`, coupled with an $OA(121, 12, 11, 2)$, would yield an `OLHD(121, 84)`. The design size of output OLHD or NOLHD highly depends on the existence of the OAs. In the LHD package, function `OLHD.L2009()` implements this algebraic construction:

```
OLHD.L2009(OLHD, OA),
```

where input arguments `OLHD` and `OA` are the OLHD and OA to be coupled, and their design sizes need to be aligned with the designated pattern of the construction.

Butler (2001) proposed a method to construct OLHDs with the run size n being odd primes and factor size k being less than or equal to $n - 1$ via the Williams transformation (Williams, 1949). In the LHD package, function `OLHD.B2001()` implements this algebraic construction with input arguments `n` and `k` exactly matching those in construction:

```
OLHD.B2001(n, k)
```

4.3 Illustrating Examples for the Implemented Algebraic Constructions

In Table 5, we summarize the algebraic constructions implemented by the developed LHD package, where `FastMmLHD` and `OA2LHD` are for maximin distance LHDs and `OLHD.Y1998`, `OLHD.C2007`, `OLHD.S2010`, `OLHD.L2009` and `OLHD.B2001` are for orthogonal LHDs. The following examples will illustrate how to use them.

Table 5: Algebraic constructions in the LHD package

Function	Description
FastMmLHD	Returns a maximin distance LHD matrix (Wang et al., 2018).
OA2LHD	Expands an orthogonal array to an LHD (Tang, 1993).
OLHD.Y1998	Returns a $2^m + 1$ by $2m - 2$ orthogonal LHD matrix (Ye, 1998) where m is an integer and $m \geq 2$.
OLHD.C2007	Returns a $2^m + 1$ by $m + \binom{m-1}{2}$ orthogonal LHD matrix (Cioppa and Lucas, 2007) where m is an integer and $m \geq 2$.
OLHD.S2010	Returns a $r2^{c+1} + 1$ or $r2^{c+1}$ by 2^c orthogonal LHD matrix (Sun et al., 2010) where r and c are positive integers.
OLHD.L2009	Couples an n by p orthogonal LHD with a n^2 by $2f$ strength 2 and level n orthogonal array to generate a n^2 by $2fp$ orthogonal LHD (Lin et al., 2009).
OLHD.B2001	Returns an orthogonal LHD (Butler, 2001) with the run size n being odd primes and factor size k being less than or equal to $n - 1$.

#FastMmLHD(8, 8) generates an optimal 8 by 8 maximin L_1 distance LHD.

```
>try.FastMm = FastMmLHD(n = 8, k = 8); try.FastMm
```

```
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,] 0 1 2 3 4 5 6 7
[2,] 1 3 5 7 6 4 2 0
[3,] 2 5 7 4 1 0 3 6
[4,] 3 7 4 0 2 6 5 1
[5,] 4 6 1 2 7 3 0 5
[6,] 5 4 0 6 3 1 7 2
[7,] 6 2 3 5 0 7 1 4
[8,] 7 0 6 1 5 2 4 3
```

#OA2LHD(OA) expands an input OA to an LHD of the same run size.

```
>try.OA2LHD = OA2LHD(OA)
```

```
>OA; try.OA2LHD
```

```
[,1] [,2] [,1] [,2]
[1,] 1 1 [1,] 1 2
[2,] 1 2 [2,] 2 4
[3,] 1 3 [3,] 3 9
[4,] 2 1 [4,] 4 3
[5,] 2 2 [5,] 5 5
[6,] 2 3 [6,] 6 7
[7,] 3 1 [7,] 9 1
[8,] 3 2 [8,] 8 6
[9,] 3 3; [9,] 7 8
```

#OLHD.Y1998(m = 3) generates a 9 by 4 orthogonal LHD.

#Note that $2^m+1 = 9$ and $2*m-2 = 4$.

```
> try.Y1998 = OLHD.Y1998(m = 3); try.Y1998
```

```
[,1] [,2] [,3] [,4]
```

```
[1,] 4 -3 -2 1
[2,] 3 4 -1 -2
[3,] 1 -2 3 -4
[4,] 2 1 4 3
[5,] 0 0 0 0
[6,] -4 3 2 -1
[7,] -3 -4 1 2
[8,] -1 2 -3 4
[9,] -2 -1 -4 -3
```

```
> MaxAbsCor(try.Y1998) #column-wise correlations are 0.
```

```
[1] 0
```

```
#OLHD.C2007(m = 4) generates a 17 by 7 orthogonal LHD.  

#Note that  $2^m+1 = 17$  and  $\sum_{i=1}^{m-1} \binom{m}{i} = 7$ .  

> try.C2007 = OLHD.C2007(m = 4); dim(try.C2007)  

[1] 17 7  

> MaxAbsCor(try.C2007)      #column-wise correlations are 0  

[1] 0  
  

#OLHD.S2010(C = 3, r = 3, type = "odd") generates a 49 by 8 orthogonal LHD.  

#Note that  $3 \times 2^4 + 1 = 49$  and  $2^3 = 8$ .  

> dim(OLHD.S2010(C = 3, r = 3, type = "odd"))  

[1] 49 8  

> MaxAbsCor(OLHD.S2010(C = 3, r = 3, type = "odd")) #column-wise correlations are 0  

[1] 0  
  

#OLHD.S2010(C = 3, r = 3, type = "even") generates a 48 by 8 orthogonal LHD.  

#Note that  $3 \times 2^4 = 48$  and  $2^3 = 8$ .  

> dim(OLHD.S2010(C = 3, r = 3, type = "even"))  

[1] 48 8  

> MaxAbsCor(OLHD.S2010(C = 3, r = 3, type = "even")) #column-wise correlations are 0  

[1] 0  
  

#Create a 5 by 2 OLHD.  

> OLHD = OLHD.C2007(m = 2)  
  

#Create an OA(25, 6, 5, 2).  

> OA = matrix(c(2,2,2,2,2,1,2,1,5,4,3,5,3,2,1,5,4,5,1,5,4,3,2,5,4,1,3,5,2,3,  

  1,2,3,4,5,2,1,3,5,2,4,3,1,1,1,1,1,4,3,2,1,5,5,5,5,5,5,1,4,4,4,4,4,1,  

  3,1,4,2,5,4,3,3,3,3,1,3,5,2,4,1,3,3,4,5,1,2,2,5,4,3,2,1,5,2,3,4,5,1,2,  

  2,5,3,1,4,4,1,4,2,5,3,4,4,2,5,3,1,4,2,4,1,3,5,3,5,3,1,4,2,4,5,2,4,1,3,3,  

  5,1,2,3,4,2,4,5,1,2,3,2), ncol = 6, nrow = 25, byrow = TRUE)  
  

#OLHD.L2009(OLHD, OA) generates a 25 by 12 orthogonal LHD.  

#Note that n = 5 so  $n^2 = 25$ . p = 2 and f = 3 so  $2fp = 12$ .  

> dim(OLHD.L2009(OLHD, OA))  

[1] 25 12  

> MaxAbsCor(OLHD.L2009(OLHD, OA))      #column-wise correlations are 0.  

[1] 0  
  

#OLHD.B2001(n = 11, k = 5) generates a 11 by 5 orthogonal LHD.  

> dim(OLHD.B2001(n = 11, k = 5))  

[1] 11 5
```

5 Other R Packages for Latin Hypercube and Comparative Discussion

Several R packages have been developed to facilitate Latin hypercube samples and design constructions for computer experiments. Among these, the `lhs` package (Carnell, 2024) is widely recognized for its utility. It provides functions for generating both random and optimized Latin hypercube samples (but not designs), and its methods are particularly useful for simulation studies where space-filling properties are desired but design optimality is not the primary focus. The `SLHD` package (Ba, 2015) was originally developed for generating sliced LHDs (Ba et al., 2015), while practitioners can set the number of slices to one to use the package for generating maximin LHDs. The `MaxPro` package (Ba and Joseph, 2018) focuses on constructing designs that maximize projection properties. One of its functions, `MaxProLHD`, generates MaxPro LHDs using a simulated annealing algorithm (Joseph et al., 2015).

While we acknowledge the contributions of other relevant R packages, we emphasize the distinguishing features of our developed package. The LHD package embeds multiple optimality criteria, enabling the construction of various types of optimal LHDs. In contrast, lhs and SLHD primarily focus on space-filling Latin hypercube samples and designs, while MaxPro primarily focuses on maximum projection LHDs. The LHD package implements various search algorithms and algebraic constructions, whereas the other three packages do not implement algebraic constructions, and both SLHD and MaxPro only implement one algorithm to construct LHDs. The primary application of LHD is in the design of computer experiments, whereas lhs is mainly used for sampling and simulation studies. Therefore, LHD emphasizes design optimality, while lhs emphasizes the space-filling properties of samples.

6 Conclusion and Recommendation

LHD package implements popular search algorithms, including the SA (Morris and Mitchell, 1995), OASA (Leary et al., 2003), SA2008 (Joseph and Hung, 2008), LaPSO (Chen et al., 2013) and GA (Liefvendahl and Stocki, 2006), along with some widely used algebraic constructions (Wang et al., 2018; Ye, 1998; Cioppa and Lucas, 2007; Sun et al., 2010; Tang, 1993; Lin et al., 2009; Butler, 2001), for constructing three types of commonly used optimal LHDs: the maximin distance LHDs, the maximum projection LHDs and the (nearly) orthogonal LHDs. We aim to provide guidance and an easy-to-use tool for practitioners to find appropriate experimental designs. Algebraic constructions are preferred when available, especially for large designs. Search algorithms are used to generate optimal LHDs with flexible sizes.

Among very few R libraries particularly for LHDs, LHD is comprehensive and self-contained as it not only has search algorithms and algebraic constructions, but also has other useful functions for LHD research and development such as calculating different optimality criteria, generating random LHDs, exchanging two random elements in a matrix, and calculating intersite distance between matrix rows. The help manual in the package documentation contains further details and illustrative examples for users who want to explore more of the functions in the package.

Acknowledgments

This research was partially supported by the National Science Foundation (NSF) grant DMS-2311186 and the National Key R&D Program of China 2024YFA1016200. The authors appreciate the reviewers' constructive comments and suggestions.

References

- S. Ba. *SLHD: Maximin-Distance (Sliced) Latin Hypercube Designs*, 2015. URL <https://CRAN.R-project.org/package=SLHD>. R package version 2.1-1. [p33]
- S. Ba and V. R. Joseph. *MaxPro: Maximum Projection Designs*, 2018. URL <https://CRAN.R-project.org/package=MaxPro>. R package version 4.1-2. [p33]
- S. Ba, W. R. Myers, and W. A. Brenneman. Optimal sliced Latin hypercube designs. *Technometrics*, 57(4):479–487, 2015. [p20, 21, 33]
- N. A. Butler. Optimal and orthogonal Latin hypercube designs for computer experiments. *Biometrika*, 88(3):847–857, 2001. [p20, 21, 31, 32, 34]
- R. Carnell. *lhs: Latin Hypercube Samples*, 2024. URL <https://CRAN.R-project.org/package=lhs>. R package version 1.2.0. [p33]
- R.-B. Chen, D.-N. Hsieh, Y. Hung, and W. Wang. Optimizing Latin hypercube designs by particle swarm. *Statistics and computing*, 23(5):663–676, 2013. [p21, 25, 27, 34]

- T. M. Cioppa and T. W. Lucas. Efficient nearly orthogonal and space-filling Latin hypercubes. *Technometrics*, 49(1):45–55, 2007. [p21, 31, 32, 34]
- K.-T. Fang, C.-X. Ma, and P. Winker. Centered L_2 -discrepancy of random sampling and Latin hypercube design, and construction of uniform designs. *Mathematics of Computation*, 71(237):275–296, 2002. [p21, 22]
- K.-T. Fang, R. Li, and A. Sudjianto. *Design and modeling for computer experiments*. CRC press, 2005. [p20, 21]
- S. D. Georgiou. Orthogonal Latin hypercube designs from generalized orthogonal designs. *Journal of Statistical Planning and Inference*, 139(4):1530–1540, 2009. [p21, 22, 23]
- S. D. Georgiou and I. Efthimiou. Some classes of orthogonal Latin hypercube designs. *Statistica Sinica*, 24(1):101–120, 2014. [p21]
- D. E. Goldberg. Genetic algorithms in search. *Optimization, and MachineLearning*, 1989. [p25]
- A. Grosso, A. Jamali, and M. Locatelli. Finding maximin Latin hypercube designs by iterated local search heuristics. *European Journal of Operational Research*, 197(2):541–547, 2009. [p21]
- F. Hickernell. A generalized discrepancy and quadrature error bound. *Mathematics of computation*, 67(221):299–322, 1998. [p21, 22]
- J. H. Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992. [p25]
- R. Jin, W. Chen, and A. Sudjianto. An efficient algorithm for constructing optimal design of computer experiments. *Journal of statistical planning and inference*, 134(1):268–287, 2005. [p21, 22]
- M. E. Johnson, L. M. Moore, and D. Ylvisaker. Minimax and maximin distance designs. *Journal of statistical planning and inference*, 26(2):131–148, 1990. [p21, 22]
- V. R. Joseph and Y. Hung. Orthogonal-maximin Latin hypercube designs. *Statistica Sinica*, pages 171–186, 2008. [p20, 21, 25, 27, 34]
- V. R. Joseph, E. Gul, and S. Ba. Maximum projection designs for computer experiments. *Biometrika*, 102(2):371–380, 2015. [p20, 21, 22, 33]
- J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995. [p25]
- S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983. [p23]
- S. Leary, A. Bhaskar, and A. Keane. Optimal orthogonal-array-based Latin hypercubes. *Journal of Applied Statistics*, 30(5):585–598, 2003. [p21, 24, 25, 27, 34]
- M. Liefvendahl and R. Stocki. A study on algorithms for optimization of Latin hypercubes. *Journal of statistical planning and inference*, 136(9):3231–3247, 2006. [p21, 25, 27, 34]
- C. D. Lin, R. Mukerjee, and B. Tang. Construction of orthogonal and nearly orthogonal Latin hypercubes. *Biometrika*, 96(1):243–247, 2009. [p21, 31, 32, 34]
- M. D. McKay, R. J. Beckman, and W. J. Conover. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979. [p20]
- M. D. Morris and T. J. Mitchell. Exploratory designs for computational experiments. *Journal of statistical planning and inference*, 43(3):381–402, 1995. [p21, 22, 23, 25, 27, 34]
- J. Sacks, S. B. Schiller, and W. J. Welch. Designs for computer experiments. *Technometrics*, 31(1):41–47, 1989. [p20]

- M. C. Shewry and H. P. Wynn. Maximum entropy sampling. *Journal of applied statistics*, 14(2):165–170, 1987. [p21]
- D. M. Steinberg and D. K. J. Lin. A construction method for orthogonal Latin hypercube designs. *Biometrika*, 93(2):279–288, 2006. [p21, 22]
- F. Sun and B. Tang. A general rotation method for orthogonal Latin hypercubes. *Biometrika*, 104(2):465–472, 2017. [p22]
- F. Sun, M.-Q. Liu, and D. K. J. Lin. Construction of orthogonal Latin hypercube designs. *Biometrika*, 96(4):971–974, 2009. [p21, 31]
- F. Sun, M.-Q. Liu, and D. K. J. Lin. Construction of orthogonal Latin hypercube designs with flexible run sizes. *Journal of Statistical Planning and Inference*, 140(11):3236–3242, 2010. [p21, 31, 32, 34]
- B. Tang. Orthogonal array-based Latin hypercubes. *Journal of the American statistical association*, 88(424):1392–1397, 1993. [p21, 25, 30, 31, 32, 34]
- L. Wang, Q. Xiao, and H. Xu. Optimal maximin L_1 -distance latin hypercube designs based on good lattice point designs. *The Annals of Statistics*, 46(6B):3741–3766, 2018. [p21, 30, 32, 34]
- E. Williams. Experimental designs balanced for the estimation of residual effects of treatments. *Australian Journal of Chemistry*, 2(2):149–168, 1949. [p30, 31]
- Q. Xiao and H. Xu. Construction of maximin distance Latin squares and related Latin hypercube designs. *Biometrika*, 104(2):455–464, 2017. [p21]
- Q. Xiao and H. Xu. Construction of maximin distance designs via level permutation and expansion. *Statistica Sinica*, 28(3):1395–1414, 2018. [p20]
- J. Yang and M.-Q. Liu. Construction of orthogonal and nearly orthogonal Latin hypercube designs from orthogonal designs. *Statistica Sinica*, pages 433–442, 2012. [p21]
- K. Q. Ye. Orthogonal column Latin hypercubes and their application in computer experiments. *Journal of the American Statistical Association*, 93(444):1430–1439, 1998. [p20, 21, 31, 32, 34]
- K. Q. Ye, W. Li, and A. Sudjianto. Algorithmic construction of optimal symmetric Latin hypercube designs. *Journal of statistical planning and inference*, 90(1):145–159, 2000. [p21]
- Y. Zhou and H. Xu. Space-filling properties of good lattice point sets. *Biometrika*, 102(4):959–966, 2015. [p21, 30]

Hongzhi Wang
wanghongzhi.ut@gmail.com

Qian Xiao
Department of Statistics, School of Mathematical Sciences, Shanghai Jiao Tong University
800 Dongchuan Road, Minhang, Shanghai, 200240
China
qian.xiao@sjtu.edu.cn

Abhyuday Mandal
Department of Statistics, University of Georgia
310 Herty Drive, Athens, GA 30602
USA
amandal@stat.uga.edu

longevity: An R Package for Modelling Excess Lifetimes

by Léo R. Belzile

Abstract The `longevity` R package provides a maximum likelihood estimation routine for modelling of survival data that are subject to non-informative censoring or truncation. It includes a selection of 12 parametric models of varying complexity, with a focus on tools for extreme value analysis and more specifically univariate peaks over threshold modelling. The package provides utilities for univariate threshold selection, parametric and nonparametric maximum likelihood estimation, goodness of fit diagnostics and model comparison tools. These different methods are illustrated using individual Dutch records and aggregated Japanese human lifetime data.

1 Introduction

Many data sets collected by demographers for the analysis of human longevity have unusual features and for which limited software implementations exist. The `longevity` package was initially built for dealing with human longevity records and data from the International Database on Longevity (IDL), which provides age at death of supercentenarians, i.e., people who died above age 110. Data for the statistical analysis of (human) longevity can take the form of aggregated counts per age at death, or most commonly life trajectory of individuals with both birth and death dates. Such lifetimes are often interval truncated (only age at death of individuals dying between two calendar dates are recorded) or left truncated and right censored (when data of individuals still alive at the end of the collection period are also included). Another frequent format is death counts, aggregated per age band. Censoring and truncation are typically of administrative nature and thus non-informative about death.

Supercentenarians are extremely rare and records are sparse. The most popular parametric models used by practitioners are justified by asymptotic arguments and have their roots in extreme value theory. Univariate extreme value distributions are well implemented in software and Belzile et al. (2023) provides a recent review of existing implementations. While there are many standard R packages for the analysis of univariate extremes using likelihood-based inference, such as `evd` (Stephenson, 2002), `mev` and `extRemes` (Gilleland and Katz, 2016), only the `evgam` package includes functionalities to fit threshold exceedance models with censoring, as showcased in Youngman (2022) with rounded rainfall measurements. Support for survival data for extreme value models is in general wholly lacking, which motivated the development of `longevity`.

The `longevity` package also includes several parametric models commonly used in demography. Many existing packages that focus on tools for modelling mortality rates, typically through life tables, are listed in the CRAN Task View *ActuarialScience* in the Life insurance section. They do not however allow for truncation or more general survival mechanisms, as the aggregated data used are typically complete except for potential right censoring for the oldest age group. The `survival` package (Therneau and Grambsch, 2000) includes utilities for accelerated failure time models for 10 parametric models. The `MortCast` package can be used to estimate age-specific mortality rates using the Kannisto and Lee–Carter approaches, among others (Ševčíková et al., 2016). The `demography` package provides forecasting methods for death rates based on constructed life tables using the Lee–Carter or ARIMA models. `MortalityLaws` includes utilities to download data from the Human Mortality Database (HMD), and fit a total of 27 parametric models for life table data (death counts and population at risk per age group) using Poisson, binomial or alternative loss functions. The `vitality` package fits the family of vitality models (Li and Anderson, 2009; Anderson, 2000) via maximum likelihood based on empirical survival data. The `fitdistrplus` (Delignette-Muller and Dutang, 2015) allows for generic parametric distributions to be fitted to interval censored data via maximum likelihood, with various S3 methods for

model assessment. The package also allows user-specified models, thereby permitting custom definitions for truncated distributions whose truncation bounds are passed as fixed vector parameters. Parameter uncertainty can be obtained via additional functions using nonparametric bootstrap. Many parametric distributions also appear in the **VGAM** package (Yee and Wild, 1996; Yee, 2015), which allows for vector generalized linear modelling. The **longevity** package is less general and offers support only for selected parametric distributions, but contrary to the aforementioned packages allows for truncation and general patterns. One strength of **longevity** is that it also includes model comparison tools that account for non-regular asymptotics and goodness of fit diagnostics.

Nonparametric methods are popular tools for the analysis of survival data with large samples, owing to their limited set of assumptions. They also serve for the validation of parametric models. Without explanatory variable, a closed-form estimator of the nonparametric maximum likelihood estimator of the survival function can be derived in particular instances, including the product limit estimator (Kaplan and Meier, 1958) for the case of random or non-informative right censoring and an extension allowing for left truncation (Tsai et al., 1987). In general, the nonparametric maximum likelihood estimator of the survival function needs to be computed using an expectation-maximization (EM) algorithm (Turnbull, 1976). Nonparametric estimators only assign probability mass on observed failure times and intervals and so cannot be used for extrapolation beyond the range of the data, limiting its utility in extreme value analysis.

The CRAN Task View on *Survival* Analysis lists various implementations of nonparametric maximum likelihood estimators of the survival or hazard functions: **survival** implements the Kaplan–Meier and Nelson–Aalen estimators. Many packages focus on the case of interval censoring (Groeneboom and Wellner, 1992, §3.2), including **prodlim**; Anderson-Bergman (2017a) reviews the performance of the implementations in **icenReg** (Anderson-Bergman, 2017b) and **Icens**. The latter uses the incidence matrix as input data. Routines for doubly censored data are provided by **dblCens**. The **interval** (Fay and Shaw, 2010) implements Turnbull’s EM algorithm for interval censored data. The case of left truncated and right censored data is handled by **survival**, and **tranSurv** provides transformation models with the potential to account for truncation dependent on survival times. For interval truncated data, dedicated algorithms that use gradient-based steps (Efron and Petrosian, 1999) or inverse probability weighting (Shen, 2010) exist and can be more efficient than the EM algorithm of Turnbull (1976). Many of these are implemented in the **DTDA** package. The **longevity** package includes a C⁺⁺ implementation of the corrected Turnbull’s algorithm (Turnbull, 1976), returning the nonparametric maximum likelihood estimator for arbitrary censoring and truncation patterns, as opposed to the specific existing aforementioned implementations already available which focus on specific subcases. With a small number of observations, it is also relatively straightforward to maximize the log likelihood for the concave program subject to linear constraints using constrained optimization algorithms; **longevity** relies for this on **Rsolnp**, which uses augmented Lagrangian methods and sequential quadratic programming (Ye, 1987).

1.1 Motivating examples

To showcase the functionality of the package and particularity for the modelling of threshold exceedances, we consider Dutch and Japanese life lengths. The **dutch** database contains the age at death (in days) of Dutch people who died above age 92 between 1986 and 2015; these data were obtained from Statistics Netherlands and analyzed in Einmahl et al. (2019) and Belzile et al. (2022). Records are interval truncated, as people are included in the database only if they died during the collection period. In addition, there are 226 interval censored and interval truncated records for which only the month and year of birth and death are known, as opposed to exact dates.

The second database we consider is drawn from Maier et al. (2021). The data frame **japanese2** consists of counts of Japanese above age 100 by age band and are stratified by both birth cohort and sex. To illustrate the format of the data, counts for female Japanese

Table 1: Death count by birth cohort and age band for female Japanese.

age	1874-1878	1879-1883	1884-1888	1889-1893	1894-1898	1899-1900
100	1648	2513	4413	8079	16036	9858
101	975	1596	2921	5376	11047	7091
102	597	987	1864	3446	7487	4869
103	345	597	1153	2230	5014	3293
104	191	351	662	1403	3242	2133
105	121	197	381	855	2084	1357
106	64	122	210	495	1284	836
107	34	74	120	274	774	521
108	16	41	66	152	433	297
109	12	30	39	83	252	167
110	6	17	21	49	130	92
111	4	10	15	26	69	47
112	3	3	11	15	29	22
113	2	2	8	5	15	9
114	1	2	4	3	7	2
115	0	1	1	0	3	2
116	0	1	1	0	1	1
117	0	0	0	0	1	1

are reproduced in Table 1. The data were constructed using the extinct cohort method and are interval censored between age and age + 1 and right truncated at the age reached by the oldest individuals of their birth cohort in 2020. The count variable lists the number of instances in the contingency table, and serves as a weight for likelihood contributions.

2 Package functionalities

The **longevity** package uses the S3 object oriented system and provides a series of functions with common arguments. The syntax used by the **longevity** package purposely mimics that of the **survival** package (Therneau and Grambsch, 2000), except that it does not specify models using a formula. Users must provide vectors for the time or age (or bounds in case of intervals) via arguments `time` and `time2`, as well as lower and upper truncation bounds (`ltrunc` and `rtrunc`) if applicable. The integer vector `event` is used to indicate the type of event, where following `survival` 0 indicates right-censoring, 1 observed events, 2 left-censoring and 3 interval censoring. Together, these five vectors characterize the data and the survival mechanisms at play. Depending on the sampling scheme, not all arguments are required or relevant and they need not be of the same length, but are common to most functions. Users can also specify a named list `args` to pass arguments: as illustrated below, this is convenient to avoid specifying repeatedly the common arguments in each function call. Default values are overridden by elements in `args`, with the exception of those that are passed by the user directly in the call. Relative to **survival**, functions have additional arguments `ltrunc` and `rtrunc` for left and right truncation limits, as these are also possibly matrices for the case of double interval truncation (Belzile et al., 2022), since both censoring and truncation can be present simultaneously.

We can manipulate the data set to build the time vectors and truncation bounds for the Dutch data. We re-scale observations to years for interpretability and keep only records above age 98 for simplicity. We split the data to handle the observed age at death first: these are treated as observed (uncensored) whenever `time` and `time2` coincide. When exact dates are not available, we compute the range of possible age at which individuals may have died, given their birth and death years and months. The truncation bounds for each individual can be obtained by subtracting from the endpoints of the sampling frame the birth dates,

Table 2: Sample of five Dutch records, formatted so that the inputs match the function arguments used by the package. Columns give the age in years at death (or plausible interval), lower and upper truncation bounds giving minimum and maximum age for inclusion, an integer indicating the type of censoring, gender and birth year.

time	time2	ltrunc	rtrunc	event	gender	byear
104.67	105.74	80.00	111.00	3	female	1905
103.50	104.58	78.00	109.00	3	female	1907
100.28	100.28	92.01	104.50	1	female	1911
100.46	100.46	92.01	102.00	1	male	1913
100.51	100.51	92.01	104.35	1	female	1911

with left and right truncation bounds

$$\text{ltrunc} = \min\{92 \text{ years}, 1986.01.01 - \text{bdate}\}, \quad \text{rtrunc} = 2015.12.31 - \text{bdate}.$$

Table 2 shows a sample of five individuals, two of whom are interval-censored, and the corresponding vectors of arguments along with two covariates, gender (gender) and birth year (byear).

We can proceed similarly for the Japanese data. Ages of centenarians are rounded down to the nearest year, so all observations are interval censored within one-year intervals. Assuming that the ages at death are independent and identically distributed with distribution function $F(\cdot; \theta)$, the log likelihood for exceedances $y_i = \text{age}_i - u$ above age u is

$$\ell(\theta) = \sum_{i:\text{age}_i > u} n_i [\log\{F(y_i + 1; \theta) - F(y_i; \theta)\} - \log F(r_i - u; \theta)]$$

where n_i is the count of the number of individuals in cell i and $r_i > \text{age}_i + 1$ is the right truncation limit for that cell, i.e., the maximum age that could have been achieved for that birth cohort by the end of the data collection period.

```
data(japanese2, package = "longevity")
# Keep only non-empty cells
japanese2 <- japanese2[japanese2$count > 0, ]
# Define arguments that are recycled
japanese2$rtrunc <- 2020 -
  as.integer(substr(japanese2$bcohort, 1, 4))
# The line above extracts the earliest year of the birth cohort
# Create a list with all arguments common to package functions
args_japan <- with(japanese2,
  list(
    time = age, # lower censoring bound
    time2 = age + 1L, # upper censoring bound
    event = 3, # define interval censoring
    type = "interval2",
    rtrunc = rtrunc, # right truncation limit
    weights = count)) # counts as weights
```

2.1 Parametric models and maximum likelihood estimation

Various models are implemented in `longevity`: their hazard functions are reported in Table 3. Two of those models, labelled perks and beard are logistic-type hazard functions proposed in Perks (1932) that have been used by Beard (1963), and popularized in work of Kannisto and Thatcher; we use the parametrization of Richards (2012), from which we also adopt the nomenclature. Users can compare the models with those available in `MortalityLaws`; see `?availableLaws` for the list of hazard functions and parametrizations.

Table 3: List of parametric models for excess lifetime supported by the package, with parametrization and hazard functions.

model	hazard function	constraints
exp	σ^{-1}	$\sigma > 0$
gomp	$\sigma^{-1} \exp(\beta t / \sigma)$	$\sigma > 0, \beta \geq 0$
gp	$(\sigma + \xi t)_+^{-1}$	$\sigma > 0, \xi \in \mathbb{R}$
weibull	$\sigma^{-\alpha} \alpha t^{\alpha-1}$	$\sigma > 0, \alpha > 0$
extgp	$\beta \sigma^{-1} \exp(\beta t / \sigma) [\beta + \xi \{\exp(\beta t / \sigma) - 1\}]^{-1}$	$\sigma > 0, \beta \geq 0, \xi \in \mathbb{R}$
extweibull	$\alpha \sigma^{-\alpha} t^{\alpha-1} \{1 + \xi(t/\sigma)^\alpha\}_+$	$\sigma > 0, \alpha > 0, \xi \in \mathbb{R}$
perks	$\alpha \exp(\nu x) / \{1 + \alpha \exp(\nu x)\}$	$\nu \geq 0, \alpha > 0$
beard	$\alpha \exp(\nu x) / \{1 + \alpha \beta \exp(\nu x)\}$	$\nu \geq 0, \alpha > 0, \beta \geq 0$
gompmake	$\lambda + \sigma^{-1} \exp(\beta t / \sigma)$	$\lambda \geq 0, \sigma > 0, \beta \geq 0$
perksmake	$\lambda + \alpha \exp(\nu x) / \{1 + \alpha \exp(\nu x)\}$	$\lambda \geq 0, \nu \geq 0, \alpha > 0$
beardmake	$\lambda + \alpha \exp(\nu x) / \{1 + \alpha \beta \exp(\nu x)\}$	$\lambda \geq 0, \nu \geq 0, \alpha > 0, \beta \geq 0$

Many of the models are nested and Figure 1 shows the logical relation between the various families. The function `fit_elife` allows users to fit all of the parametric models of Table 3: the `print` method returns a summary of the sampling mechanism, the number of observations, the maximum log likelihood and parameter estimates with standard errors. Depending on the data, some models may be overparametrized and parameters need not be numerically identifiable. To palliate such issues, the optimization routine, which uses `Rsolnp`, can try multiple starting values or fit various sub-models to ensure that the parameter values returned are indeed the maximum likelihood estimates. If one tries to compare nested models and the fit of the simpler model is better than the alternative, the `anova` function will return an error message.

The `fit_elife` function handles arbitrary censoring patterns over single intervals, along with single interval truncation and interval censoring. To accommodate the sampling scheme of the International Database on Longevity (IDL), an option also allows for double interval truncation (Belzile et al., 2022), whereby observations are included only if the person dies between time intervals, potentially overlapping, which defines the observation window over which dead individuals are recorded.

```
thresh <- 108
model0 <- fit_elife(arguments = args_japan,
                      thresh = thresh,
                      family = "exp")
(model1 <- fit_elife(arguments = args_japan,
                      thresh = thresh,
                      family = "gomp"))
#> Model: Gompertz distribution.
#> Sampling: interval censored, right truncated
#> Log-likelihood: -3599.037
#>
#> Threshold: 108
#> Number of exceedances: 2489
#>
#> Estimates
#> scale shape
#> 1.6855 0.0991
#>
#> Standard Errors
#> scale shape
#> 0.0523 0.0273
#>
```

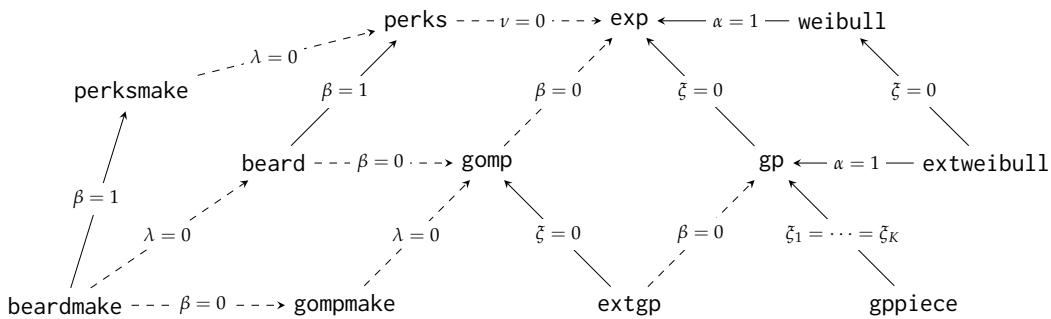


Figure 1: Relationship between parametric models showing nested relations. Dashed arrows represent restrictions that lead to nonregular asymptotic null distribution for comparison of nested models. Comparisons between models with Makeham components and exponential are not permitted by the software because of nonidentifiability issues.

```
#> Optimization Information
#>   Convergence: TRUE
```

2.2 Model comparisons

Goodness of fit of nested models can be compared using likelihood ratio tests via the anova method. Most of the interrelations between models yield non-regular model comparisons since, to recover the simpler model, one must often fix parameters to values that lie on the boundary of the parameter space. For example, if we compare a Gompertz model with the exponential, the limiting null distribution is a mixture of a point mass at zero and a χ^2_1 variable, both with probability half (Chernoff, 1954). Many authors (e.g., Camarda, 2022) fail to recognize this fact. The case becomes more complicated with more than one boundary constraint: for example, the deviance statistic comparing the Beard–Makeham and the Gompertz model, which constrains two parameters on the boundary of the parameter space, has a null distribution which is a mixture of $\chi^2_2/4 + \chi^2_1/2 + \chi^2_0/4$ (Self and Liang, 1987).

Nonidentifiability impacts testing: for example, if the rate parameter of the Perks–Makeham model (perksmake) $\nu \rightarrow 0$, the limiting hazard, $\lambda + \exp(\alpha)/\{1 + \exp(\alpha)\}$, is constant (exponential model), but neither α nor λ is identifiable. The usual asymptotics for the likelihood ratio test break down as the information matrix is singular (Rotnitzky et al., 2000). As such, all three families that include a Makeham component cannot be directly compared to the exponential in `longevity` and the call to `anova` returns an error message.

Users can also access information criteria, AIC and BIC. The correction factors implemented depend on the number of parameters of the distribution, but do not account for singular fit, non-identifiable parameters or singular models for which the usual corrections $2p$ and $\ln(n)p$ are inadequate (Watanabe, 2010).

To showcase how hypothesis testing is performed, we consider a simple example with two nested models. We test whether the exponential model is an adequate simplification of the Gompertz model for exceedances above 108 years — an irregular testing problem since $\beta = 0$ is a restriction on the boundary of the parameter space. The drop in log likelihood is quite large, indicating the exponential model is not an adequate simplification of the Gompertz fit. This is also what is suggested by the Bayesian information criterion, which is much lower for the Gompertz model than for the exponential.

```
# Model comparison
anova(model1, model0)
# Information criteria
c("exponential" = BIC(model0), "Gompertz" = BIC(model1))
```

	npar	Deviance	Df	Chisq	Pr(>Chisq)
gomp	2	7198.07			
exp	1	7213.25	1	15.17	0


```
#> exponential      Gompertz
#>    7221.065     7213.713
```

2.3 Simulation-based inference

Given the poor finite sample properties of the aforementioned tests, it may be preferable to rely on a parametric bootstrap rather than on the asymptotic distribution of the test statistic (Belzile et al., 2022) for model comparison. Simulation-based inference requires capabilities for drawing new data sets whose features match those of the original one. For example, the International Database on Longevity (IDL) (Jdanov et al., 2021) features data that are interval truncated above 110 years, but doubly interval truncated since the sampling period for semisupercentenarians (who died age 105 to 110) and supercentenarians (who died above 110) are not always the same (Belzile et al., 2022). The 2018 Istat semisupercentenarians database analyzed by Barbi et al. (2018) on Italians includes left truncated right censored records.

To mimic the postulated data generating mechanism while accounting for the sampling scheme, we could use the observed birth dates, or simulate new birth dates (possibly through a kernel estimator of the empirical distribution of birth dates) while keeping the sampling frame with the first and last date of data collection to define the truncation interval. In other settings, one could obtain the nonparametric maximum likelihood estimator of the distribution of the upper truncation bound (Shen, 2010) using an inverse probability weighted estimator, which for fixed data collection windows is equivalent to setting the birth date.

The `samp_elife` function includes multiple type2 arguments to handle these. For interval truncated data (`type2="ltrt"`), it uses the inversion method (Section 2 of Devroye (1986)): for F an absolutely continuous distribution function and F^{-1} the corresponding quantile function, a random variable distributed according to F truncated on $[a, b]$ is generated as $X \sim F^{-1}[F(a) + U\{F(b) - F(a)\}]$ where $U \sim U(0, 1)$ is standard uniform.

The function `samp_elife` also has an argument `upper` which serves for both right truncation, and right censoring. For the latter, any record simulated that exceeds `upper` is capped at that upper bound and declared partially observed. This is useful for simulating administrative censoring, whereby the birth date and the upper bound of the collection window fully determine whether an observation is right censored or not. An illustrative example is provided in the next section.

The `anova` method call uses the asymptotic null distribution for comparison of nested parametric distributions $\mathcal{F}_0 \subseteq \mathcal{F}_1$. We could use the bootstrap to see how good this approximation to the null distribution is. To mimic as closely as possible the data generating mechanism, which is custom in most scenarios, we condition on the sampling frame and the number of individuals in each birth cohort. The number dying at each age is random, but the right truncation limits will be the same for anyone in that cohort. We simulate excess lifetimes, then interval censor observations by keeping only the corresponding age bracket. Under the null hypothesis, the data are drawn from $\tilde{F}_0 \in \mathcal{F}_0$ and we generate observations from this right truncated distribution using the `samp_elife` utility, which also supports double interval truncation and left truncation right censoring. This must be done within a for loop since we have count attached to each upper bound, but the function is vectorized should we use a single vector containing all of the right truncation limits.

The bootstrap p -value for comparing models $M_0 \subset M_1$ would be obtained by repeating the following steps B times and calculating the rank of the observed test statistic among alternatives:

1. Simulate new birth dates d_i ($i = 1, \dots, n$) (e.g., drawing from a smoothed empirical distribution of birth dates); the latest possible birth date is one which ensures the person reached at least the threshold by the end of the period.
2. Subtract the endpoints of the sampling period, say c_1 and c_2 to get the minimum (maximum) age at death, $c_1 - d_i$ (respectively $c_2 - d_i$) days, which define the truncation bounds.
3. Use the function `samp_elife` to simulate new observations from a parametric interval truncated distribution from the null model M_0
4. Use the optimization procedure in `fit_elife` to fit the model with both M_1 and M_2 and calculate the deviance, and from them the likelihood ratio statistic.

The algorithm is implemented below for comparing the Gompertz and the exponential model. Since the procedure is computationally intensive, users must trade off between the precision of the bootstrap p -value estimate and the number of replications, B .

```
set.seed(2022)
# Count of unique right truncation limit
db_rtrunc <- aggregate(count ~ rtrunc,
                        FUN = "sum",
                        data = japanese2,
                        subset = age >= thresh)
B <- 1000L # Number of bootstrap replications
boot_anova <- numeric(length = B)
boot_gof <- numeric(length = B)
for(b in seq_len(B - 1L)){
  boot_samp <- # Generate bootstrap sample
  do.call(rbind, #merge data frames
          apply(db_rtrunc, 1, function(x){ # for each rtrunc and count
            count <- table( #tabulate count
                           floor( #round down
                                 samp_elife( # sample right truncated exponential
                                             n = x["count"],
                                             scale = model0$par,
                                             family = "exp", #null model
                                             upper = x["rtrunc"] - thresh,
                                             type2 = "lptrt")))
            data.frame( # return data frame
              count = as.integer(count),
              rtrunc = as.numeric(x["rtrunc"]) - thresh,
              eage = as.integer(names(count)))
          }))
  boot_mod0 <- # Fit null model to bootstrap sample
  with(boot_samp,
       fit_elife(time = eage,
                 time2 = eage + 1L,
                 rtrunc = rtrunc,
                 type = "interval",
                 event = 3,
                 family = "exp",
                 weights = count))
  boot_mod1 <- # Fit alternative model to bootstrap sample
  with(boot_samp,
       fit_elife(time = eage,
                 time2 = eage + 1L,
                 rtrunc = rtrunc,
                 type = "interval",
```

```

event = 3,
family = "gomp",
weights = count))
boot_anova[b] <- deviance(boot_mod0) -
  deviance(boot_mod1)
}
# Add original statistic
boot_anova[B] <- deviance(model1) - deviance(model0)
# Bootstrap p-value
(pval <- rank(boot_anova)[B] / B)
#> [1] 0.001

```

The asymptotic approximation is of similar magnitude as the bootstrap p -value. Both suggest that the more complex Gompertz model provides a significantly better fit.

2.4 Extreme value analysis

Extreme value theory suggests that, in many instances, the limiting conditional distribution of exceedances of a random variable Y with distribution function F is generalized Pareto, meaning

$$\lim_{u \rightarrow x^*} \Pr(Y - u > y \mid Y > u) = \begin{cases} (1 + \xi y/\sigma)_+^{-1/\xi}, & \xi \neq 0; \\ \exp(-y/\sigma), & \xi = 0; \end{cases} \quad (1)$$

with $x_+ = \max\{x, 0\}$ and $x^* = \sup\{x : F(x) < 1\}$. This justifies the use of Equation (1) for the survival function of threshold exceedances when dealing with rare events. The model has two parameters: a scale σ and a shape ξ which determines the behavior of the upper tail. Negative shape parameters correspond to bounded upper tails and a finite right endpoint for the support.

Study of population dynamics and mortality generally requires knowledge of the total population from which observations are drawn to derive rates. By contrast, the peaks over threshold method, by which one models the k largest observations of a sample, is a conditional analysis (e.g., given survival until a certain age), and is therefore free of denominator specification since we only model exceedances above a high threshold u . For modelling purposes, we need to pick a threshold u that is smaller than the upper endpoint x^* in order to have sufficient number of observations to estimate parameters. The threshold selection problem is a classical instance of bias-variance trade-off: the parameter estimators are possibly biased if the threshold is too low because the generalized Pareto approximation is not good enough, whereas choosing a larger threshold to ensure we are closer to the asymptotic regime leads to reduced sample size and increased parameter uncertainty.

To aid threshold selection, users commonly resort to parameter stability plots. These are common visual diagnostics consisting of a plot of estimates of the shape parameter $\hat{\xi}$ (with confidence or credible intervals) based on sample exceedances over a range of thresholds u_1, \dots, u_K . If the data were drawn from a generalized Pareto distribution, the conditional distribution above higher threshold $v > u$ is also generalized Pareto with the same shape: this threshold stability property is the basis for extrapolation beyond the range of observed records. Indeed, if the change in estimation of ξ is nearly constant, this provides reassurance that the approximation can be used for extrapolation. The only difference with survival data, relative to the classical setting, is that the likelihood must account for censoring and truncation. Note that, when we use threshold exceedances with a nonzero threshold (argument `thresh`), it however isn't possible to unambiguously determine whether left censored observations are still exceedances: such cases yield errors in the functions.

Theory on penultimate extremes suggests that, for finite levels and general distribution function F for which (1) holds, the shape parameter varies as a function of the threshold u , behaving like the derivative of the reciprocal hazard $r(x) = \{1 - F(x)\}/f(x)$. We can

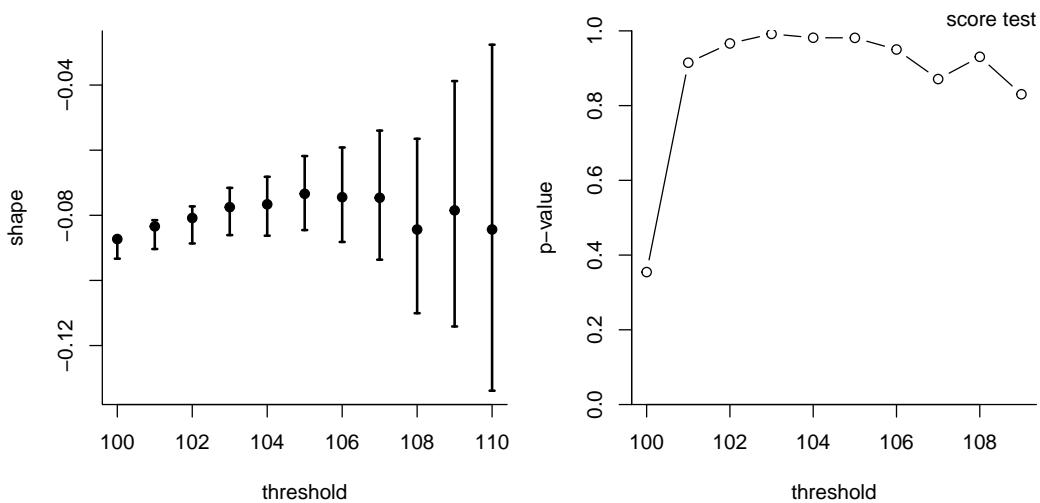


Figure 2: Threshold diagnostic tools: parameter stability plots for the generalized Pareto model (left) and Northrop–Coleman p -value path (right) for the Japanese centenarian dataset. Both suggest that a threshold as low as 100 may be suitable for peaks-over-threshold analysis.

thus model the shape as piece-wise constant by fitting a piece-wise generalized Pareto model due to [Northrop and Coleman \(2014\)](#) and adapted in [Belzile et al. \(2022\)](#) for survival data. The latter can be viewed as a mixture of generalized Pareto over K disjoint intervals with continuity constraints to ensure a smooth hazard, which reduces to the generalized Pareto if we force the $K + 1$ shape parameters to be equal. We can use a likelihood ratio test to compare the model, or a score test if the latter is too computationally intensive, and plot the p -values for each of the K thresholds, corresponding to the null hypotheses $H_k : \xi_k = \dots = \xi_K$ ($k = 1, \dots, K - 1$). As the model quickly becomes overparametrized, optimization is difficult and the score test may be a safer option as it only requires estimation of the null model of a single generalized Pareto over the whole range.

To illustrate these diagnostic tools, Figure 2 shows a threshold stability plot, which features a small increase in the shape parameters as the threshold increases, corresponding to a stabilization or even a slight decrease of the hazard at higher ages. We can envision a threshold of 108 years as being reasonable: the Northrop–Coleman diagnostic plot suggests lower thresholds are compatible with a constant shape above 100. Additional goodness-of-fit diagnostics are necessary to determine if the generalized Pareto model fits well.

```
par(mfrow = c(1, 2), mar = c(4, 4, 1, 1))
# Threshold sequence
u <- 100:110
# Threshold stability plot
tstab(arguments = args_japan,
      family = "gp",
      method = "profile",
      which.plot = "shape",
      thresh = u)
# Northrop-Coleman diagnostic based on score tests
nu <- length(u) - 1L
nc_score <- nc_test(arguments = c(args_japan, list(thresh = u)))
score_plot <- plot(nc_score)
graphics.off()
```

Each plot in the package can be produced using base R or using [ggplot2](#) ([Wickham, 2016](#)), which implements the grammar of graphics. To keep the list of package dependencies lean and adhere to the [tinyverse principle](#), the latter can be obtained by using the argument `plot.type` with the generic S3 method `plot`, or via `autoplot`, provided the [ggplot2](#) package is already installed.

2.5 Graphical goodness of fit diagnostics

Determining whether a parametric model fits well to survival data is no easy task due to the difficulty in specifying the null distribution of many goodness of fit statistics, such as the Cramér-von Mises statistic, which differ for survival data. As such, the `longevity` package relies mostly on visual diagnostic tools. [Waller and Turnbull \(1992\)](#) discusses how classical visual graphics can be adapted in the presence of censoring. Most notably, only observed failure times are displayed on the y -axis against their empirical plotting position on the x -axis. Contrary to the independent and identically distributed case, the uniform plotting positions $F_n(y_i)$ are based on the nonparametric maximum likelihood estimator discussed in Section 3.

The situation is more complicated with truncated data ([Belzile et al., 2022](#)), since the data are not identically distributed: indeed, the distribution function of observation Y_i truncated on the interval $[a_i, b_i]$ is $F_i(y_i) = \{F(y_i) - F(a_i)\} / \{F(b_i) - F(a_i)\}$, so the data arise from different distributions even if these share common parameters. One way out of this conundrum is using the probability integral transform and the quantile transform to map observations to the uniform scale and back onto the data scale. Taking $\tilde{F}(y_i) = F_n(y_i) = \text{rank}(y_i)/(n + 1)$ to denote the empirical distribution function estimator, a probability-probability plot would show $x_i = \tilde{F}_i(y_i)$ against $y_i = F_i(y_i)$, leading to approximately uniform samples if the parametric distribution F is suitable. Another option is to standardize the observation, taking the collection $\tilde{y}_i = F^{-1}\{F_i(y_i)\}$ of rescaled exceedances and comparing them to the usual plotting positions $x_{(i)} = \{i/(n + 1)\}$. The drawback of the latter approach is that the quantities displayed on the y -axis are not raw observations and the ranking of the empirical quantiles may change, a somewhat counter-intuitive feature. However, this means that the sample $\{F_i(y_i)\}$ should be uniform under the null hypothesis, and this allows one to use methods from [Säilynoja et al. \(2022\)](#) to obtain point-wise and simultaneous confidence intervals.

`longevity` offers users the choice between three types of quantile-quantile plots: regular (Q-Q, "qq"), Tukey's mean detrended Q-Q plots ("tmd") and exponential Q-Q plots ("exp"). Other options on uniform scale are probability-probability (P-P, "pp") plots and empirically rescaled plots (ERP, "erp") ([Waller and Turnbull, 1992](#)), designed to ease interpretation with censored observations by rescaling axes. We illustrate the graphical tools with the dutch data above age 105 in Figure 3. The fit is correct above 110, but there is a notable dip due to excess mortality around 109.

```
fit_dutch <- fit_elife(
  arguments = dutch_data,
  event = 3,
  type = "interval2",
  family = "gp",
  thresh = 105,
  export = TRUE)
par(mfrow = c(1, 2))
plot(fit_dutch,
  which.plot = c("pp", "qq"))
```

Censored observations are used to compute the plotting positions, but are not displayed. As such, we cannot use graphical goodness of fit diagnostics for the Japanese interval censored data. An alternative, given that data are tabulated in a contingency table, is to use a chi-squared test for independence, conditioning on the number of individuals per birth cohort. The expected number in each cell (birth cohort and age band) can be obtained by computing the conditional probability of falling in that age band. The asymptotic null distribution should be χ^2 with $(k - 1)(p - 1)$ degrees of freedom, where k is the number of age bands and p the number of birth cohorts. In finite samples, the expected count for large excess lifetimes are very low so one can expect the χ^2 approximation to be poor. To mitigate this, we can pool observations and resort to simulation to approximate the null

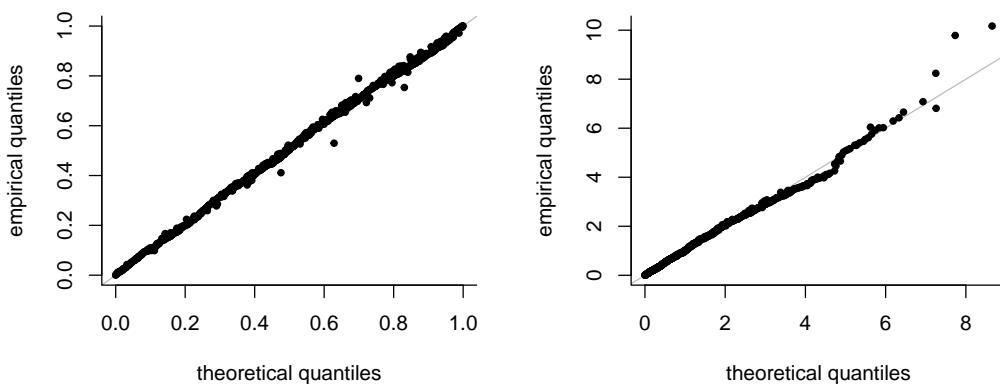


Figure 3: Probability-probability and quantile-quantile plots for generalized Pareto model fitted above age 105 years to Dutch data. The plots indicate broadly good agreement with the observation, except for some individuals who died age 109 for which too many have deaths close to their birthdates.

distribution of the test statistic. The bootstrap p -value for the exponential model above 108 years, pooling observations with excess lifetime of 5 years and above, is 0.872, indicating no evidence that the model is inadequate, but the test here may have low power.

2.6 Stratification

Demographers may suspect differences between individuals of different sex, from different countries or geographic areas, or by birth cohort. All of these are instances of categorical covariates. One possibility is to incorporate these covariates with suitable link function through parameters, but we consider instead stratification. We can split the data by levels of covariate (with factors) into sub-data and compare the goodness of fit of the K models relative to that which pools all observations. The `test_elife` function performs likelihood ratio tests for the comparisons. The test statistic is $-2\{\ell(\hat{\theta}_0) - \ell(\hat{\theta})\}$, where $\hat{\theta}_0$ is the maximum likelihood estimator under the null model with common parameters, and $\hat{\theta}$ is the unrestricted maximum likelihood estimator for the alternative model with the same distribution, but which allows for stratum-specific parameters. We illustrate this with a generalized Pareto model for the excess lifetime. The null hypothesis is $H_0 : \sigma_f = \sigma_m, \xi_f = \xi_m$ against the alternative that at least one equality doesn't hold and so the hazard and endpoints are different.

```

print(
  test_elife(
    arguments = args_japan,
    thresh = 110,
    family = "gp",
    covariate = japanese2$gender)
)
#> Model: generalized Pareto distribution.
#> Threshold: 110
#> Number of exceedances per covariate level:
#> female   male
#>     642     61
#>
#> Likelihood ratio statistic: 0.364
#> Null distribution: chi-square (2)
#> Asymptotic p-value:  0.833

```

In the present example, there is no evidence against any difference in lifetime distribution between male and female; this is unsurprising given the large imbalance between counts for each covariate level, with far fewer males than females.

2.7 Extrapolation

If the maximum likelihood estimator of the shape ξ for the generalized Pareto model is negative, then the distribution has a finite upper endpoint; otherwise, the latter is infinite. With $\xi < 0$, we can look at the profile log likelihood for the endpoint $\eta = -\sigma/\xi$, using the function `prof_gp_endpt`, to draw the curve and obtain confidence intervals. The argument `psi` is used to give a grid of values over which to compute the profile log likelihood. The bounds of the $(1 - \alpha)$ confidence intervals are obtained by fitting a cubic smoothing spline for $y = \eta$ as a function of the shifted profile curve $x = 2\{\ell_p(\eta) - \ell_p(\hat{\eta})\}$ on both sides of the maximum likelihood estimator and predicting the value of y when $x = -\chi_1^2(1 - \alpha)$. This technique works well unless the profile is nearly flat or the bounds lie beyond the range of values of `psi` provided; the user may wish to change them if they are too far. If $\hat{\xi} \approx 0$, then the upper bound of the confidence interval may be infinite and the profile log likelihood may never reach the cutoff value of the asymptotic χ_1^2 distribution.

The profile log likelihood curve for the endpoint, shifted vertically so that its value is zero at the maximum likelihood estimator, highlights the marked asymmetry of the distribution of η , shown in 4, with the horizontal dashed lines showing the limits for the 95% profile likelihood confidence intervals. These suggest that the endpoint, or a potential finite lifespan, could lie very much beyond observed records. The routine used to calculate the upper bound computes the cutoff value by fitting a smoothing spline with the role of the y and x axes reversed and by predicting the value of η at $y = 0$. In this example, the upper confidence limit is extrapolated from the model: more accurate measures can be obtained by specifying a longer and finer sequence of values of `psi` such that the profile log likelihood drops below the χ_1^2 quantile cutoff.

```
# Create grid of threshold values
thresholds <- 105:110
# Grid of values at which to evaluate profile
psi <- seq(120, 200, length.out = 101)
# Calculate the profile for the endpoint
# of the generalized Pareto at each threshold
endpt_tstab <- do.call(
  endpoint.tstab,
  args = c(
    args_japan,
    list(psi = psi,
         thresh = thresholds,
         plot = FALSE)))
# Compute corresponding confidence intervals
profile <- endpoint.profile(
  arguments = c(args_japan, list(thresh = 110, psi = psi)))
# Plot point estimates and confidence intervals
g1 <- autoplot(endpt_tstab, plot = FALSE, ylab = "lifespan (in years")
# Plot the profile curve with cutoffs for conf. int. for 110
g2 <- autoplot(profile, plot = FALSE)
patchwork::wrap_plots(g1, g2)
```

Depending on the model, the conclusions about the risk of mortality change drastically: the Gompertz model implies an ever increasing hazard, but no finite endpoint for the distribution of exceedances. The exponential model implies a constant hazard and no endpoint. By contrast, the generalized Pareto can accommodate both finite and infinite endpoints. The marked asymmetry of the distribution of lifespan defined by the generalized

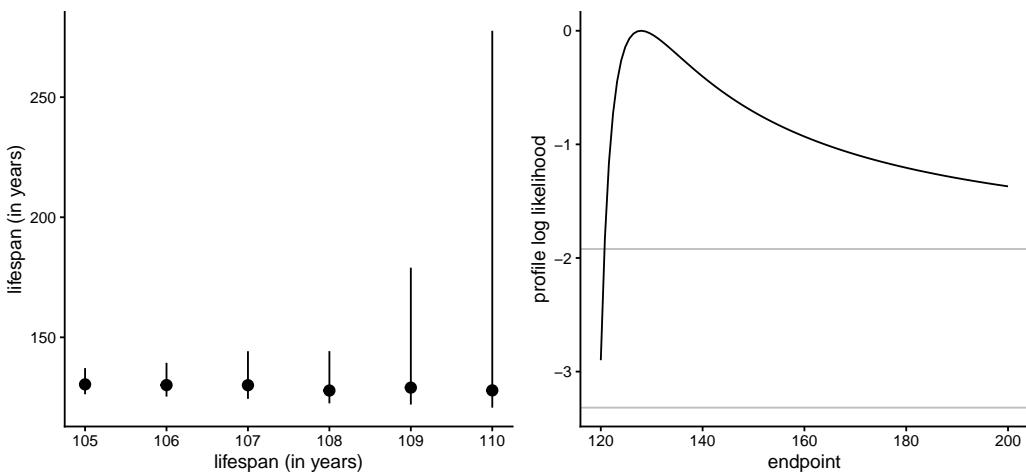


Figure 4: Maximum likelihood estimates with 95% confidence intervals as a function of threshold (left) and profile log likelihood for exceedances above 110 years (right) for Japanese centenarian data. As the threshold increases, the number of exceedances decreases and the intervals for the upper bound become wider. At 110, the right endpoint of the interval would go until infinity.

Pareto shows that inference obtained using symmetric confidence intervals (i.e., Wald-based) is likely very misleading: the drop in fit from having a zero or positive shape parameter ξ is seemingly smaller than the cutoffs for a 95% confidence interval, suggesting that while the best point estimate is around 128 years, the upper bound is so large (and extrapolated) that everything is possible. The model however also suggests a very high probability of dying in any given year, regardless of whether the hazard is constant, decreasing or increasing.

2.8 Hazard

The parameters of the models are seldom of interest in themselves: rather, we may be interested in a summary such as the hazard function. At present, `longevity` does not allow general linear modelling of model parameters or time-varying covariates, but other software implementations can tackle this task. For example, `casebase` (Bhatnagar et al., 2022) fits flexible hazard models using logistic or multinomial regression with potential inclusion of penalties for the parameters associated to covariates and splines effects. Another alternative is `flexsurv` (Jackson, 2016), which offers 10 parametric models and allows for user-specified models. The `bshazard` package (Rebora et al., 2014) provides nonparametric smoothing via B -splines, whereas `muhaz` handles kernel-based hazard for right censored data; both could be used for validation of the parametric models in the case of right censoring. The `rstpm2` package (Liu et al., 2018) handles generalized modelling for censoring with the Royston–Parmar model built from natural cubic splines (Royston and Parmar, 2002). Contrasting with all of the aforementioned approaches, we focus on parametric models: this is partly because there are few observations for the user case we consider and covariates, except perhaps for gender and birth year, are not available.

The hazard changes over time; the only notable exception is the exponential hazard, which is constant. `longevity` includes utilities for computing the hazard function from a fitted model object and computing point-wise confidence intervals using symmetric Wald intervals or the profile likelihood. Specifically, the `hazard_elife` function calculates the hazard $h(t; \theta)$ point-wise at times $t = x$; Wald-based confidence intervals are obtained using the delta-method, whereas profile likelihood intervals are obtained by reparametrizing the model in terms of $h(t)$ for each time t . More naturally perhaps, we can consider a Bayesian analysis of the Japanese excess lifetime above 108 years. Using the likelihood and encoded log posterior provided in `logpost_elife`, we obtained independent samples from the posterior of the generalized Pareto parameters (σ, ξ) with maximal data information prior using the `rust` package. Each parameter combination was then fed into `heliife` and

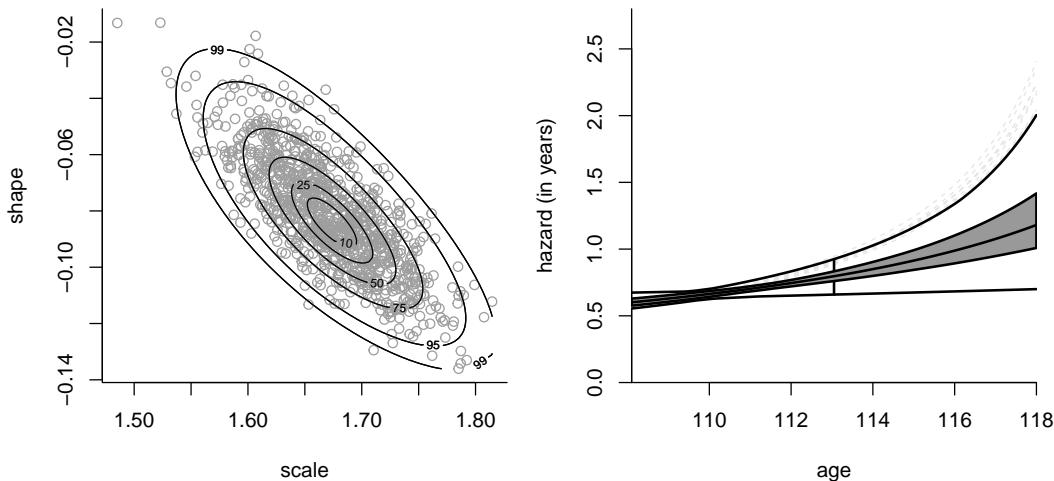


Figure 5: Left: scatterplot of 1000 independent posterior samples from generalized Pareto model with maximum data information prior; the contour curves give the percentiles of credible intervals, and show approximate normality of the posterior. Right: functional boxplots for the corresponding hazard curves, with increasing width at higher ages.

the hazard evaluated over a range of values. Figure 5 shows the posterior samples and functional boxplots (Sun and Genton, 2011) of the hazard curves, obtained using the **fda** package. The risk of dying increases with age, but comes with substantial uncertainty, as evidenced by the increasing width of the boxes and interquartile range.

2.9 Nonparametric maximum likelihood estimation

The nonparametric maximum likelihood estimator is unique only up to equivalence classes. The data for individual i consists of the tuple $\{L_i, R_i, V_i, U_i\}$, where the censoring interval is $[L_i, R_i]$ and the truncation interval is $[V_i, U_i]$, with $0 \leq V_i \leq L_i \leq R_i \leq U_i \leq \infty$. Turnbull (1976) shows how one can build disjoint intervals $C = \bigcup_{j=1}^m [a_j, b_j]$ where $a_j \in \mathcal{L} = \{L_1, \dots, L_n\}$ and $b_j \in \mathcal{R} = \{R_1, \dots, R_n\}$ satisfy $a_1 \leq b_1 < \dots < a_m \leq b_m$ and the intervals $[a_j, b_j]$ contain no other members of L or R except at the endpoints. This last condition notably ensures that the intervals created include all observed failure times as singleton sets in the absence of truncation. Other authors (Lindsey and Ryan, 1998) have taken interval censored data as semi-open intervals $(L_i, R_i]$, a convention we adopt here for numerical reasons. For interval censored and truncated data, Frydman (1994) shows that this construction must be amended by taking instead $a_j \in \mathcal{L} \cup \{U_1, \dots, U_n\}$ and $b_j \in \mathcal{R} \cup \{V_1, \dots, V_n\}$.

We assign probability $p_j = F(b_j^+) - F(a_j^-) \geq 0$ to each of the resulting m intervals under the constraint $\sum_{j=1}^m p_j = 1$ and $p_j \geq 0$ ($j = 1, \dots, m$). The nonparametric maximum likelihood estimator of the distribution function F is then

$$\hat{F}(t) = \begin{cases} 0, & t < a_1; \\ \hat{p}_1 + \dots + \hat{p}_j, & b_j < t < a_{j+1} \quad (1 \leq j \leq m-1); \\ 1, & t > b_m; \end{cases}$$

and is undefined for $t \in [a_j, b_j]$ ($j = 1, \dots, m$).

The procedure of Turnbull can be encoded using $m \times n$ matrices. For censoring, we build **A** whose (i, j) th entry $\alpha_{ij} = 1$ if $[a_j, b_j] \subseteq A_i$ and zero otherwise. Since the intervals forming the set C are disjoint and in increasing order, a more storage efficient manner of keeping track of the intervals is to find the smallest integer j such that $L_i \leq a_j$ and the largest $R_i \geq b_j$ ($j = 1, \dots, m$) for each observation. The same idea applies for the truncation sets $B_i = (V_i, U_i)$ and matrix **B** with (i, j) element β_{ij} .

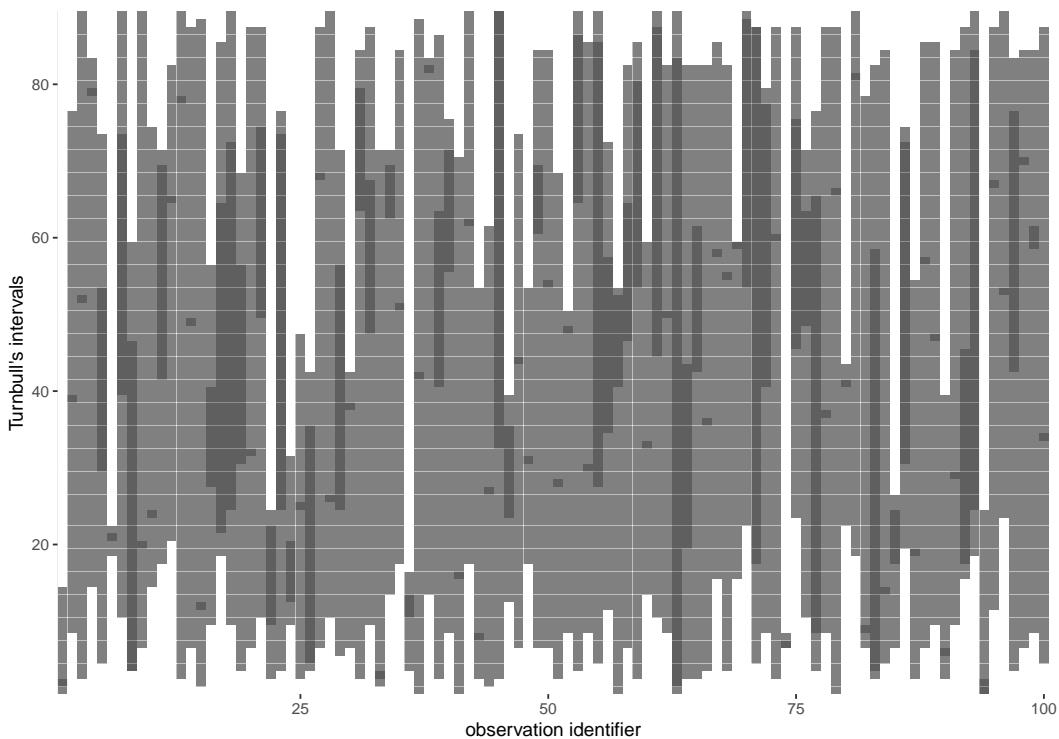


Figure 6: Illustration of the truncation (pale grey) and censoring intervals (dark grey) equivalence classes based on Turnbull’s algorithm. Observations must fall within equivalence classes defined by the former.

The log likelihood function is

$$\ell(\boldsymbol{p}) = \sum_{i=1}^n w_i \log \left(\sum_{j=1}^m \alpha_{ij} p_j \right) - w_i \log \left(\sum_{j=1}^m \beta_{ij} p_j \right)$$

The numerical implementation of the EM is in principle forward: first identify the equivalence classes C , next calculate the entries of A_i and B_i (or the vectors of ranges) and finally run the EM algorithm. In the second step, we need to account for potential ties in the presence of (interval) censoring and treat the intervals as open on the left for censored data. For concreteness, consider the toy example $T = (1, 1, 2)$ and $\delta = (0, 1, 1)$, where $\delta_i = 1$ if the observation is a failure time and $\delta_i = 0$ in case of right censoring. The left and right censoring bounds are $\mathcal{L} = \{1, 2\}$ and $\mathcal{R} = \{1, 2, \infty\}$ with $A_1 = (1, \infty)$, $A_2 = \{1\}$ and $A_3 = \{2\}$ and $C_1 = \{1\}$, $C_2 = \{2\}$. If we were to treat instead A_1 as a semi-closed interval $[1, \infty)$, direct maximization of the log likelihood in eq. 2.2 of [Turnbull \(1976\)](#) would give probability half to each observed failure time. By contrast, the Kaplan–Meier estimator, under the convention that right censored observations at time t were at risk up to and until t , assigns probability $1/3$ to the first failure. To retrieve this solution with Turnbull’s EM estimator, we need the convention that $C_1 \notin A_1$, but this requires comparing the bound with itself. The numerical tolerance in the implementation is taken to be the square root of the machine epsilon for doubles.

The maximum likelihood estimator (MLE) need not be unique, and the EM algorithm is only guaranteed a local maximum. For interval censored data, [Gentleman and Geyer \(1994\)](#) consider using the Karush–Kuhn–Tucker conditions to determine whether the probability in some intervals is exactly zero and whether the returned value is indeed the MLE.

Due to data scarcity, statistical inference for human lifespan is best conducted using parametric models supported by asymptotic theory, reserving nonparametric estimators to assess goodness of fit. The empirical cumulative hazard for Japanese is very close to linear

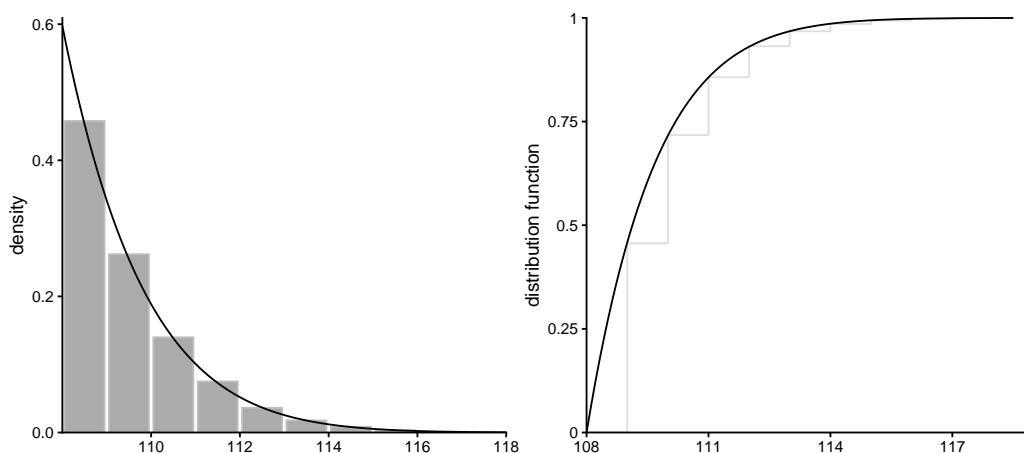


Figure 7: Nonparametric maximum likelihood estimate of the density (bar plot, left) and distribution function (staircase function, right), with superimposed generalized Pareto fit for excess lifetimes above 108 years. Except for the discreteness inherent to the nonparametric estimates, the two representations broadly agree at year marks.

from early ages, suggesting that the hazard may not be very far from exponential even if more complex models are likely to be favored given the large sample size.

The function `np_elife` returns a list with Turnbull's intervals $[a_j, b_j]$ and the probability weights assigned to each, provided the latter are positive. It also contains an object of class `stepfun` with a `weighting` argument that defines a cumulative distribution function.

We can use the nonparametric maximum likelihood estimator of the distribution function to assess a fitted parametric model by comparing the density with a binned distribution, or the cumulative distribution function. The distribution function is defined over equivalence classes, which may be isolated observations or intervals. The data interval over which there is non-zero probability of events is broken down into equispaced bins and the probability of failing in the latter estimated nonparametrically based on the distribution function. Alternatively, users can also provide a set of breaks, or the number of bins.

```
ecdf <- np_elife(arguments = args_japan, thresh = 108)
# Summary statistics, accounting for censoring
round(summary(ecdf), digits = 2)
#>   Min. 1st Qu. Median Mean 3rd Qu. Max.
#> 109.00 110.00 111.00 110.01 111.00 118.00
# Plots of fitted parametric model and nonparametric CDF
model_gp <- fit_elife(
  arguments = args_japan,
  thresh = 108,
  family = "gp",
  export = TRUE)
# ggplot2 plots, wrapped to display side by side
patchwork::wrap_plots(
  autoplot(
    model_gp, # fitted model
    plot = FALSE, # return list of ggplots
    which.plot = c("dens", "cdf"),
    breaks = seq(0L, 8L, by = 1L) # set bins for histogram
  )
)
```

3 Conclusion

This paper describes the salient features of **longevity**, explaining the theoretical underpinning of the methods and the design considerations followed when writing the package. While **longevity** was conceived for modelling lifetimes, the package could be used for applications outside of demography. Survival data in extreme value theory is infrequent yet hardly absent. For example, rainfall observations can be viewed as rounded due to the limited instrumental precision of rain gauges and treated as interval-censored. Some historical records, which are often lower bounds on the real magnitude of natural catastrophes, can be added as right-censored observations in a peaks-over-threshold analysis. In insurance, losses incurred by the company due to liability claims may be right-censored if they exceed the policy cap and are covered by a reinsurance company, or rounded (Belzile and Nešlehová, 2025). In climate science, attribution studies often focus on data just after a record-breaking event, and the stopping rule leads to truncation (Miralles and Davison, 2023), which biases results if ignored.

The package has features that are interesting in their own right, including adapted quantile-quantile and other visual goodness of fit diagnostics for model validation. The testing procedures correctly handle tests for restrictions lying on the boundary of the parameter space. Parametric bootstrap procedures for such tests are not straightforward to implement given the heavy reliance on the data generating mechanism and the diversity of possible scenarios: this paper however shows how the utilities of the package can be coupled to ease such estimation and the code in the supplementary material illustrates how this can be extended for goodness of fit testing.

Acknowledgements

The author thanks four anonymous reviewers for valuable feedback. This research was supported financially by the Natural Sciences and Engineering Research Council of Canada (NSERC) via Discovery Grant RGPIN-2022-05001.

References

- J. J. Anderson. A vitality-based model relating stressors and environmental properties to organism survival. *Ecological Monographs*, 70(3):445–470, 2000. doi: 10.1890/0012-9615(2000)070[0445:avbmrs]2.0.co;2. URL [https://doi.org/10.1890/0012-9615\(2000\)070\[0445:AVBMRS\]2.0.CO;2](https://doi.org/10.1890/0012-9615(2000)070[0445:AVBMRS]2.0.CO;2). [p37]
- C. Anderson-Bergman. An efficient implementation of the EMICM algorithm for the interval censored NPMLE. *Journal of Computational and Graphical Statistics*, 26(2):463–467, 2017a. doi: 10.1080/10618600.2016.1208616. URL <https://doi.org/10.1080/10618600.2016.1208616>. [p38]
- C. Anderson-Bergman. icenReg: Regression models for interval censored data in R. *Journal of Statistical Software*, 81(12):1–23, 2017b. doi: 10.18637/jss.v081.i12. URL <https://doi.org/10.18637/jss.v081.i12>. [p38]
- E. Barbi, F. Lagona, M. Marsili, J. W. Vaupel, and K. W. Wachter. The plateau of human mortality: Demography of longevity pioneers. *Science*, 360(6396):1459–1461, 2018. ISSN 0036-8075. doi: 10.1126/science.aat3119. URL <https://doi.org/10.1126/science.aat3119>. [p43]
- R. E. Beard. A theory of mortality based on actuarial, biological and medical considerations. In *Proceedings of the International Population Conference, New York*, volume 1, London, 1963. International Union for the Scientific Study of Population. [p40]

- L. R. Belzile and J. G. Nešlehová. Statistics of extremes for incomplete data, with application to lifetime and liability claim modeling. In M. de Carvalho, R. Huser, P. Naveau, and B. J. Reich, editors, *Handbook on Statistics of Extremes*, chapter 31, page in press. CRC Press, 2025. [p54]
- L. R. Belzile, A. C. Davison, J. Gampe, H. Rootzén, and D. Zholud. Is there a cap on longevity? A statistical review. *Annual Review of Statistics and its Application*, 9:22–45, 2022. doi: 10.1146/annurev-statistics-040120-025426. URL <https://doi.org/10.1146/annurev-statistics-040120-025426>. [p38, 39, 41, 43, 46, 47]
- L. R. Belzile, C. Dutang, P. J. Northrop, and T. Opitz. A modeler’s guide to extreme value software. *Extremes*, 26:595–638, 2023. doi: 10.1007/s10687-023-00475-9. URL <https://doi.org/10.1007/s10687-023-00475-9>. [p37]
- S. R. Bhatnagar, M. Turgeon, J. Islam, J. A. Hanley, and O. Saarela. casebase: An alternative framework for survival analysis and comparison of event rates. *The R Journal*, 14:59–79, 2022. ISSN 2073-4859. doi: 10.32614/rj-2022-052. URL <https://doi.org/10.32614/RJ-2022-052>. [p50]
- C. G. Camarda. The curse of the plateau. measuring confidence in human mortality estimates at extreme ages. *Theoretical Population Biology*, 144:24–36, 2022. doi: 10.1016/j.tpb.2022.01.002. URL <https://doi.org/10.1016/j.tpb.2022.01.002>. [p42]
- H. Chernoff. On the distribution of the likelihood ratio. *The Annals of Mathematical Statistics*, 25(3):573–578, 1954. doi: 10.1214/aoms/1177728725. URL <https://doi.org/10.1214/aoms/1177728725>. [p42]
- M. L. Delignette-Muller and C. Dutang. fitdistrplus: An R package for fitting distributions. *Journal of Statistical Software*, 64(4):1–34, 2015. doi: 10.18637/jss.v064.i04. [p37]
- L. Devroye. *Non-Uniform Random Variate Generation*. Springer, New York, 1986. URL <http://www.nrbook.com/devroye/>. [p43]
- B. Efron and V. Petrosian. Nonparametric methods for doubly truncated data. *Journal of the American Statistical Association*, 94(447):824–834, 1999. doi: 10.1080/01621459.1999.10474187. URL <https://doi.org/10.1080/01621459.1999.10474187>. [p38]
- J. J. Einmahl, J. H. J. Einmahl, and L. de Haan. Limits to human life span through extreme value theory. *Journal of the American Statistical Association*, 114(527):1075–1080, 2019. doi: 10.1080/01621459.2018.1537912. URL <https://doi.org/10.1080/01621459.2018.1537912>. [p38]
- M. P. Fay and P. A. Shaw. Exact and asymptotic weighted logrank tests for interval censored data: The interval R package. *Journal of Statistical Software*, 36(2):1–34, 2010. doi: 10.18637/jss.v036.i02. URL <https://doi.org/10.18637/jss.v036.i02>. [p38]
- H. Frydman. A note on nonparametric estimation of the distribution function from interval-censored and truncated observations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 56(1):71–74, 1994. ISSN 00359246. doi: 10.1111/j.2517-6161.1994.tb01960.x. URL <https://doi.org/10.1111/j.2517-6161.1994.tb01960.x>. [p51]
- R. Gentleman and C. J. Geyer. Maximum likelihood for interval censored data: Consistency and computation. *Biometrika*, 81(3):618–623, 1994. ISSN 0006-3444. doi: 10.1093/biomet/81.3.618. URL <https://doi.org/10.1093/biomet/81.3.618>. [p52]
- E. Gilletland and R. W. Katz. extRemes 2.0: An extreme value analysis package in R. *Journal of Statistical Software*, 72(8):1–39, 2016. doi: 10.18637/jss.v072.i08. [p37]
- P. Groeneboom and J. A. Wellner. *Information Bounds and Nonparametric Maximum Likelihood Estimation*. Basel, Switzerland, 1992. ISBN 978-3-7643-2794-1. doi: 10.1007/978-3-0348-8621-5. URL <https://doi.org/10.1007/978-3-0348-8621-5>. [p38]

- C. Jackson. flexsurv: A platform for parametric survival modeling in R. *Journal of Statistical Software*, 70(8):1–33, 2016. doi: 10.18637/jss.v070.i08. URL <https://www.jstatsoft.org/index.php/jss/article/view/v070i08>. [p50]
- D. A. Jdanov, V. M. Shkolnikov, and S. Gellers-Barkmann. The international database on longevity: Data resource profile. In H. Maier, B. Jeune, and J. W. Vaupel, editors, *Exceptional Lifespans*, Demographic Research Monographs, pages 22–24. Springer, Cham, Switzerland, 2021. doi: 10.1007/978-3-030-49970-9_2. URL https://doi.org/10.1007/978-3-030-49970-9_2. [p43]
- E. L. Kaplan and P. Meier. Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53:457–481, 1958. doi: 10.1080/01621459.1958.10501452. URL <https://doi.org/10.1080/01621459.1958.10501452>. [p38]
- T. Li and J. J. Anderson. The vitality model: A way to understand population survival and demographic heterogeneity. *Theoretical Population Biology*, 76(2):118–131, 2009. ISSN 0040-5809. doi: 10.1016/j.tpb.2009.05.004. URL <https://doi.org/10.1016/j.tpb.2009.05.004>. [p37]
- J. C. Lindsey and L. M. Ryan. Methods for interval-censored data. *Statistics in Medicine*, 17(2):219–238, 1998. doi: 10.1002/(sici)1097-0258(19980130)17:2<219::aid-sim735>3.0.co;2-o. URL [https://doi.org/10.1002/\(SICI\)1097-0258\(19980130\)17:2<219::AID-SIM735>3.0.CO;2-0](https://doi.org/10.1002/(SICI)1097-0258(19980130)17:2<219::AID-SIM735>3.0.CO;2-0). [p51]
- X.-R. Liu, Y. Pawitan, and M. Clements. Parametric and penalized generalized survival models. *Statistical Methods in Medical Research*, 27(5):1531–1546, 2018. doi: 10.1177/0962280216664760. URL <https://doi.org/10.1177/0962280216664760>. [p50]
- H. Maier, B. Jeune, and J. W. Vaupel, editors. *Exceptional Lifespans*. Demographic Research Monographs. Springer, 2021. doi: 10.1007/978-3-030-49970-9. URL <https://doi.org/10.1007/978-3-030-49970-9>. [p38]
- O. Miralles and A. C. Davison. Timing and spatial selection bias in rapid extreme event attribution. *Weather and Climate Extremes*, 41:100584, 2023. doi: 10.1016/j.wace.2023.100584. URL <https://doi.org/10.1016/j.wace.2023.100584>. [p54]
- P. J. Northrop and C. L. Coleman. Improved diagnostic plots for extreme value analyses. *Extremes*, 17:289–303, 2014. doi: 10.1007/s10687-014-0183-z. URL <https://doi.org/10.1007/s10687-014-0183-z>. [p46]
- W. Perks. On some experiments in the graduation of mortality statistics. *Journal of the Institute of Actuaries*, 63(1):12–57, 1932. doi: 10.1017/s0020268100046680. [p40]
- P. Rebora, A. Salim, and M. Reilly. bshazard: A flexible tool for nonparametric smoothing of the hazard function. *The R Journal*, 6(2):114–122, 2014. doi: 10.32614/rj-2014-028. URL <https://doi.org/10.32614/RJ-2014-028>. [p50]
- S. J. Richards. A handbook of parametric survival models for actuarial use. *Scandinavian Actuarial Journal*, 2012(4):233–257, 2012. doi: 10.1080/03461238.2010.506688. URL <https://doi.org/10.1080/03461238.2010.506688>. [p40]
- A. Rotnitzky, D. R. Cox, M. Bottai, and J. Robins. Likelihood-based inference with singular information matrix. *Bernoulli*, 6(2):243 – 284, 2000. doi: [bj/1081788028](https://doi.org/10.1081/BJ-1081788028). URL <https://doi.org/>. [p42]
- P. Royston and M. K. B. Parmar. Flexible parametric proportional-hazards and proportional-odds models for censored survival data, with application to prognostic modelling and estimation of treatment effects. *Statistics in Medicine*, 21(15):2175–2197, 2002. doi: 10.1002/sim.1203. URL <https://doi.org/10.1002/sim.1203>. [p50]

- T. Säilynoja, P.-C. Bürkner, and A. Vehtari. Graphical test for discrete uniformity and its applications in goodness of fit evaluation and multiple sample comparison. *Statistics and Computing*, 32(2):32, 2022. doi: 10.1007/s11222-022-10090-6. URL <https://doi.org/10.1007/s11222-022-10090-6>. [p47]
- S. G. Self and K.-Y. Liang. Asymptotic properties of maximum likelihood estimators and likelihood ratio tests under nonstandard conditions. *Journal of the American Statistical Association*, 82(398):605–610, 1987. doi: 10.1080/01621459.1987.10478472. URL <https://doi.org/10.1080/01621459.1987.10478472>. [p42]
- H. Ševčíková, N. Li, V. Kantorová, P. Gerland, and A. E. Raftery. *Age-Specific Mortality and Fertility Rates for Probabilistic Population Projections*, pages 285–310. Springer, Cham, Switzerland, 2016. ISBN 978-3-319-26603-9. doi: 10.1007/978-3-319-26603-9_15. URL https://doi.org/10.1007/978-3-319-26603-9_15. [p37]
- P.-S. Shen. Nonparametric analysis of doubly truncated data. *Annals of the Institute of Statistical Mathematics*, 62(5):835–853, 2010. ISSN 1572-9052. doi: 10.1007/s10463-008-0192-2. URL <https://doi.org/10.1007/s10463-008-0192-2>. [p38, 43]
- A. G. Stephenson. evd: Extreme value distributions. *R News*, 2(2), 2002. URL https://cran.r-project.org/doc/Rnews/Rnews_2002-2.pdf. [p37]
- Y. Sun and M. G. Genton. Functional boxplots. *Journal of Computational and Graphical Statistics*, 20(2):316–334, 2011. doi: 10.1198/jcgs.2011.09224. [p51]
- T. M. Therneau and P. M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer, New York, 2000. ISBN 0-387-98784-3. doi: 10.1007/978-1-4757-3294-8. URL <https://doi.org/10.1007/978-1-4757-3294-8>. [p37, 39]
- W.-Y. Tsai, N. P. Jewell, and M.-C. Wang. A note on the product-limit estimator under right censoring and left truncation. *Biometrika*, 74(4):883–886, 1987. doi: 10.1093/biomet/74.4.883. URL <https://doi.org/10.1093/biomet/74.4.883>. [p38]
- B. W. Turnbull. The empirical distribution function with arbitrarily grouped, censored and truncated data. *Journal of the Royal Statistical Society, Series B*, 38:290–295, 1976. doi: 10.1111/j.2517-6161.1976.tb01597.x. URL <https://doi.org/10.1111/j.2517-6161.1976.tb01597.x>. [p38, 51, 52]
- L. A. Waller and B. W. Turnbull. Probability plotting with censored data. *American Statistician*, 46(1):5–12, 1992. doi: 10.1080/00031305.1992.10475837. URL <https://doi.org/10.1080/00031305.1992.10475837>. [p47]
- S. Watanabe. Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory. *Journal of Machine Learning Research*, 11:3571–3594, 2010. ISSN 1532-4435. URL <https://dl.acm.org/doi/10.5555/1756006.1953045>. [p42]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p46]
- Y. Ye. *Interior Algorithms for Linear, Quadratic, and Linearly Constrained Non-Linear Programming*. PhD thesis, Department of ESS, Stanford University, 1987. [p38]
- T. W. Yee. *Vector Generalized Linear and Additive Models*. Springer, 2015. doi: 10.1007/978-1-4939-2818-7. URL <https://doi.org/10.1007/978-1-4939-2818-7>. [p38]
- T. W. Yee and C. J. Wild. Vector generalized additive models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(3):481–493, 1996. doi: 10.1111/j.2517-6161.1996.tb02095.x. URL <https://doi.org/10.1111/j.2517-6161.1996.tb02095.x>. [p38]
- B. D. Youngman. evgam: An R package for generalized additive extreme value models. *Journal of Statistical Software*, 103(3):1–26, 2022. doi: 10.18637/jss.v103.i03. URL <https://www.jstatsoft.org/index.php/jss/article/view/v103i03>. [p37]

Léo R. Belzile

*Department of Decision Sciences, HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec), Canada
H3T 2A7*
<https://lbelzile.bitbucket.io>
ORCID: 0000-0002-9135-014X
leo.belzile@hec.ca

movieROC: Visualizing the Decision Rules Underlying Binary Classification

by Sonia Pérez-Fernández, Pablo Martínez-Camblor and Norberto Corral-Blanco

Abstract The receiver operating characteristic (ROC) curve is a graphical tool commonly used to depict the binary classification accuracy of a continuous marker in terms of its sensitivity and specificity. The standard ROC curve assumes a monotone relationship between the marker and the response, inducing classification subsets of the form (c, ∞) with $c \in \mathbb{R}$. However, in non-standard cases, the involved classification regions are not so clear, highlighting the importance of tracking the decision rules. This paper introduces the R package movieROC, which provides visualization tools for understanding the ability of markers to identify a characteristic of interest, complementing the ROC curve representation. This tool accommodates multivariate scenarios and generalizations involving different decision rules. The main contribution of this package is the visualization of the underlying classification regions, with the associated gain in interpretability. Adding the time (videos) as a third dimension, this package facilitates the visualization of binary classification in multivariate problems. It constitutes a good tool to generate graphical material for presentations.

1 Introduction

The use of data to detect a characteristic of interest is a cornerstone of many disciplines such as medicine (to diagnose a pathology or to predict a patient outcome), finance (to detect fraud) or machine learning (to evaluate a classification algorithm), among others. Continuous markers are surrogate measures for the characteristic under study, or predictors of a potential subsequent event. They are measured in subjects, some of them with the characteristic (*positive*), and some without it (*negative*). In addition to reliability and feasibility, a good marker must have two relevant properties: interpretability and accuracy (Mayeux, 2004). High binary classification accuracy can be achieved if there exists a strong relationship between the marker and the response. The latter is assessed by a *gold standard* for the presence or absence of the characteristic of interest. *Interpretability* refers to the *decision rules* or *subsets* considered in the classification process. This piece of research seeks to elucidate both desirable properties for a marker by the implementation of a graphical tool in R language. We propose a novel approach involving the generation of videos as a solution to effectively capture the classification procedure for univariate and multivariate markers. Graphical analysis plays a pivotal role in data exploration, interpretation, and communication. Its burgeoning potential is underscored by the fast pace of technological advances, which empower the creation of insightful graphical representations.

A usual practice when the binary classification accuracy of a marker is of interest involves the representation of the *Receiver Operating Characteristic (ROC) curve*, summarized by the *Area Under the Curve (AUC)* (Hanley and McNeil, 1982). The resulting plot reflects the trade-off between the sensitivity and the complementary of the specificity. *Sensitivity* and *specificity* are probabilities of correctly classifying subjects, either positive or negative, respectively. Mathematically, let ξ and χ be the random variables modeling the marker values in the positive and the negative population, respectively, with $F_\xi(\cdot)$ and $F_\chi(\cdot)$ their associated cumulative distribution functions. Assuming that the expected value of the marker is larger in the positive than in the negative population, the standard ROC curve is based on *classification subsets* of the form $s = (c, \infty)$, where c is the so-called *cut-off value* or *threshold* in the support of the marker X , $\mathcal{S}(X)$. One subject is classified as a positive if its marker value is within this region, and as a negative otherwise. This type of subsets has two important advantages: first, their interpretability is clear; second, for each specificity $1 - t \in [0, 1]$, the corresponding $s_t = (c_t, \infty)$ is univocally defined by $c_t = F_\chi^{-1}(1 - t)$ for absolutely continuous markers.

When differences in marker distribution between the negative and the positive population are only in location but not in shape, then $F_\chi(\cdot) < F_\zeta(\cdot)$, and the classification is direct by using these decision rules. However, when this is not the case, the standard ROC curve may cross the main diagonal, resulting in an *improper* curve (Dorfman et al., 1997). This may be due to three different scenarios:

- i) the behavior of the marker in the two studied populations is different but it is not possible to determine the decision rules. Notice that the binary classification problem goes further than distinction between the two populations: the classification subsets should be highly likely in one population and highly unlikely in the other one (Martínez-Camblor, 2018);
- ii) there exists a relationship between the marker and the response with a potential classification use, but this is not monotone;
- iii) there is no relationship between the marker and the response at all (main diagonal ROC curve).

In the second case, we have to define classification subsets different from standard $s_t = (c_t, \infty)$. Therefore, the use of the marker becomes more complex. With the aim of accommodating scenarios where both higher and lower values of the marker are associated with a higher risk of having the characteristic, Martínez-Camblor et al. (2017) proposed the so-called *generalized ROC (gROC) curve*. This curve tracks the highest sensitivity for every specificity in the unit interval resulting from subsets of the form $s_t = (-\infty, x_t^L] \cup (x_t^U, \infty)$ with $x_t^L \leq x_t^U \in \mathcal{S}(X)$.

Despite final decisions are based on the underlying classification subsets, they are typically not depicted. This omission is not a shortcoming in standard cases, as for each specificity $1 - t \in [0, 1]$, there is only one rule of the form $s_t = (c_t, \infty)$ which such specificity. Particularly, s_t is univocally defined by $c_t = 1 - F_\chi^{-1}(1 - t)$; and the same applies if we fix a sensitivity. Nevertheless, if the gROC curve is taken, there are infinite subsets of the form $s_t = (-\infty, x_t^L] \cup (x_t^U, \infty)$ resulting in $\mathbb{P}(\chi \in s_t) = t$. This loss of univocality underlines the importance of reporting (numerically and/or graphically) the decision rules actually proposed for classifying. This gap is covered in the presented package.

An alternative approach to assess the classification performance of a marker involves considering a transformation of it. This transformation $h(\cdot)$ aims to capture differences in distribution between the two populations in the ROC sense. Once $h(\cdot)$ is identified, the standard ROC curve for $h(X)$ is represented, resulting in the *efficient ROC (eROC) curve* (Kauppi, 2016). Arguing as before, for a fixed specificity, the classification subsets $s_t = (c_t, \infty)$ in the transformed space are univocally defined, where a subject is classified as positive if $h(x) \in s_t$ and negative otherwise (with x representing its marker value). However, they may have any shape in the original space, depending on the monotonicity of the functional transformation $h(\cdot)$ (Martínez-Camblor et al., 2019). Emphasizing the importance of tracking the decision rules underlying the eROC curve, this monitoring process enables an assessment of whether the improved accuracy of the marker justifies the potential loss in interpretability.

The ROC curve is defined for classification accuracy evaluation of univariate markers. To deal with multivariate markers, the usual practice is to consider a transformation $h(\cdot)$ to reduce it to a univariate one, and then to construct the standard ROC curve. Same considerations as before apply when a functional transformation is taken. In the proposed R library, we consider methods from the literature to define and estimate $h(\cdot)$ in the multivariate case (Kang et al., 2016; Meisner et al., 2021).

Focusing on the classification subsets underlying the decision rules, the **movieROC** package incorporates methods to visualize the construction process of ROC curves by presenting the classification accuracy of these subsets. For univariate markers, the library includes both the classical (standard ROC curve) and the generalized (gROC curve) approach. Besides it enables the display of decision rules for various transformations of the marker,

seeking to maximize performance allowing for flexibility in the final shape of the subsets (eROC curve). For multidimensional markers, the proposed tool visualizes the evolution of decision subsets when different objective functions are employed for optimization, even imposing restrictions on the underlying regions. In this case, displaying the decision rules associated with every specificity in a single static image is no longer feasible. Therefore, *dynamic representations* (videos) are implemented, drawing on time as an extra dimension to capture the variation in specificity.

Much software available in R could be discussed here covering diverse topics related to ROC curves: the `pROC` package is a main reference including tools to visualize, estimate and compare ROC curves (Robin et al., 2011); `ROCnReg` explicitly considers covariate information to estimate the covariate-specific and the covariate-adjusted ROC curves (Rodríguez-Álvarez and Inácio, 2021); `smoothROCtime` implements smooth estimation of time-dependent ROC curves based on the bivariate kernel density estimator for $(X, \text{time-to-event})$ (Díaz-Coto, 2020); `OptimalCutpoints` includes point and interval estimation methods for optimal thresholds (López-Ratón et al., 2014); and `nsROC` performs non-standard analysis such as gROC estimation (Pérez-Fernández et al., 2018); among others.

This paper introduces and elucidates the diverse functionalities of the newly developed `movieROC` package, aimed at facilitating the visualization and comprehension of the decision rules underlying the binary classification process, encompassing various generalizations. Despite the availability of numerous R packages implementing related analyses, we have identified the main gaps covered in this library: tracking the decision rules underlying the ROC curve, including multivariate markers and non-standard scenarios (i.e. non-monotonous). The rest of the paper is structured as follows. In Section 2, we introduce the main R functions and objects implemented, and briefly explain the dataset employed throughout this manuscript to demonstrate the utility of the R library. Section 3 is devoted to reconsidering the definition of the standard ROC curve from the perspective of classification subsets, including an extension to multivariate scenarios. Sections 4 and 5 revisit the gROC curve and the eROC curve, respectively, covering various methods to capture the potential classification accuracy of the marker under study. Each of these sections begins with a state-of-the-art overview, followed by the main syntax of the corresponding R functions. In addition, examples of implementation using the dataset presented in Section 2.3 are provided. Finally, the paper concludes with a concise summary and computational details regarding the implemented tool.

2 Main functions of the `movieROC` package and illustrative dataset

Sections 2.1 and 2.2 provide a detailed description of the main objectives of the implemented R functions. To reflect the practical usage of the developed R package, we employ a real dataset throughout this manuscript, which is introduced in Section 2.3.

2.1 Functionality of the `movieROC` package

A graphical tool was developed to showcase static and dynamic graphics displaying the classification subsets derived from maximizing diagnostic accuracy under certain assumptions, ensuring the preservation of the interpretability. The R package facilitates the construction of the ROC curve across various specificities, providing visualizations of the resulting classification regions. The proposed tool comprises multiple R functions that generate objects with distinct class attributes (see function names where red arrows depart from and red nodes in Figure 1, respectively). Once the object of interest is created, different methods may be used, in order to plot the underlying regions (`plot_regions()`, `plot_funregions()`), to track the resulting ROC curve (`plot_buildROC()`, `plot()`), to predict decision rules for a particular specificity, and to print relevant information, among others. The main function of the package, `movieROC()`, produces videos to exhibit the classification procedure.

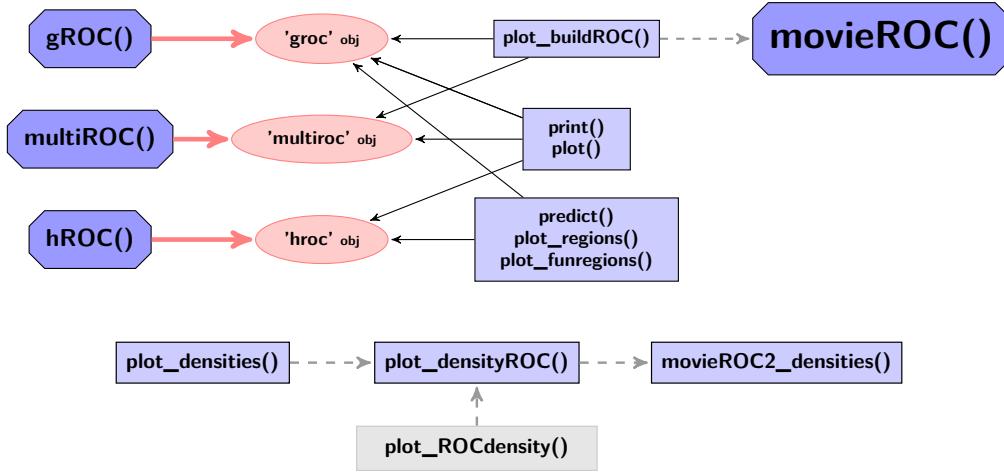


Figure 1: R functions of the **movieROC** package. The blue nodes include the names of the R functions and the red nodes indicate the different R objects that can be created and worked with. The red arrows depart from those R functions engaged in creating R objects and the black arrows indicate which R functions can be applied to which R objects. The grey dashed arrows show internal dependencies.

It includes algorithms to visualize the regions that underlie the binary classification problem, considering different approaches:

- make the classification subsets flexible in order to cover non-standard scenarios, by considering two cut-off values (`gROC()` function); explained in Section 4;
- transform the marker by a proper function $h(\cdot)$ (`hROC()` function); introduced in Section 5;
- when dealing with multivariate markers, consider a functional transformation with some fixed or dynamic parameters resulting from different methods available in the literature (`multiROC()` function); covered in Section 3.1.

2.2 Class methods for **movieROC** objects

By using the `gROC()`, the `multiROC()` or the `hROC()` function, the user obtains an R object of class ‘`groc`’, ‘`multiroc`’ or ‘`hroc`’, respectively. These will be called **movieROC** objects. Once the object of interest is created, the implemented package includes many functions (methods) to pass to it. Some of them are generic methods (`print()`, `plot()` and `predict()`), commonly used in R language over different objects according to their class attributes. The rest of the functions are specific for this library and therefore only applicable to **movieROC** objects. Table 1 summarizes all these functions and provides their target and main syntax (with default input parameters).

Generic functions	
<code>print()</code>	Print some relevant information.
<code>plot()</code>	Plot the ROC curve estimate.
<code>predict()</code>	Print the classification subsets corresponding to a particular false-positive rate (FPR) introduced by the user. For a 'groc' object, the user may specify a cut-off value C (for the standard ROC curve) or two cut-off values XL and XU (for the gROC curve).
Specific functions	
<code>plot_regions()</code>	<p>Applicable to a 'groc' or a 'hroc' object.</p> <p>Plot two graphics in the same figure: left, classification subsets for each false-positive rate (grey color by default); right, 90° rotated ROC curve.</p> <p>MAIN SYNTAX:</p> <pre>plot_regions(obj, plot.roc = TRUE, plot.auc = FALSE, FPR = 0.15, ...)</pre> <p>If the input parameter FPR is specified, the corresponding classification region reporting such false-positive rate and the point in the ROC curve are highlighted in blue color.</p>
<code>plot_funregions()</code>	<p>Applicable to a 'groc' or a 'hroc' object.</p> <p>Plot the transforming function and the classification subsets reporting the false-positive rate(s) indicated in the input parameter(s) FPR and FPR2.</p> <p>MAIN SYNTAX:</p> <pre>plot_funregions(obj, FPR = 0.15, FPR2 = NULL, plot.subsets = TRUE, ...)</pre>
<code>plot_buildROC()</code>	<p>Applicable to a 'groc' or a 'multiroc' object.</p> <ul style="list-style-type: none"> - For a 'groc' object: <p>Plot four (if input reduce is FALSE) or two (if reduce is TRUE, only those on the top) graphics in the same figure: top-left, density function estimates for the marker in both populations with the areas corresponding to FPR and TPR colored (blue and red, respectively) for the optional input parameter FPR, C or XL, XU; top-right, the empirical ROC curve estimate; bottom-left, boxplots in both groups; bottom-right, classification subsets for every FPR (grey color).</p> <p>MAIN SYNTAX:</p> <pre>plot_buildROC(obj, FPR = 0.15, C, XL, XU, h = c(1, 1), histogram = FALSE, breaks = 15, reduce = TRUE, build.process = FALSE, completeROC = FALSE, ...)</pre> <p>If build.process is FALSE, the whole ROC curve is displayed; otherwise, if completeROC is TRUE, the portion of the ROC curve until the fixed FPR is highlighted in black and the rest is shown in gray, while if completeROC is FALSE, only the first portion of the curve is displayed.</p> <ul style="list-style-type: none"> - For a 'multiroc' object: <p>Plot two graphics in the same figure: right, the ROC curve highlighting the point and the threshold for the resulting univariate marker; left, scatterplot with the marker values in both positive (red color) and negative (blue color) subjects</p> <ul style="list-style-type: none"> • for $p = 2$: over the original/feature bivariate space; • for $p > 2$: projected over two selected components of the marker (if <code>display.method = "OV"</code> with components selection in <code>displayOV, c(1, 2)</code> by default) or the first two principal components from PCA (if <code>display.method = "PCA"</code>, default); <p>and the classification subset (gold color) reporting the FPR selected by the user ($FPR \neq \text{NULL}$).</p> <p>MAIN SYNTAX:</p> <ul style="list-style-type: none"> • for $p = 2$: <pre>plot_buildROC(obj, FPR = 0.15, build.process = FALSE, completeROC = TRUE, ...)</pre> <ul style="list-style-type: none"> • for $p > 2$: <pre>plot_buildROC(obj, FPR = 0.15, display.method = c("PCA", "OV"), displayOV = c(1, 2), build.process = FALSE, completeROC = TRUE, ...)</pre> <p>If build.process is FALSE, the whole ROC curve is displayed; otherwise, if completeROC is TRUE, the portion of the ROC curve until the fixed FPR is highlighted in black and the rest is shown in gray, while if completeROC is FALSE, only the first portion of the curve is shown.</p>
<code>movieROC()</code>	<p>Applicable to a 'groc' or a 'multiroc' object.</p> <p>Save a video as a GIF illustrating the construction of the ROC curve.</p> <ul style="list-style-type: none"> - For a 'groc' object: <p>MAIN SYNTAX:</p> <pre>movieROC(obj, fpr=NULL, h=c(1,1), histogram = FALSE, breaks = 15, reduce = TRUE, completeROC = FALSE, videobar = TRUE, file = "animation1.gif", ...)</pre> <p>For each element in vector <code>fpr</code> (optional input parameter), the function executed is <code>plot_buildROC(obj, FPR = fpr[i], build.process = TRUE, ...)</code>. The vector of false-positive rates illustrated in the video is NULL by default: if length of output parameter <code>t</code> for <code>gROC()</code> function is lower than 150, such vector is taken as <code>fpr</code>; otherwise, an equally-space vector of length 100 covering the range of the marker values is considered.</p>

<p>- For a ‘multiroc’ object:</p> <p>MAIN SYNTAX:</p> <ul style="list-style-type: none"> • for $p = 2$: <pre>movieROC(obj,fpr = NULL,file = "animation1.gif",save = TRUE,border = TRUE, completeROC = FALSE,...);</pre> <ul style="list-style-type: none"> • for $p > 2$: <pre>movieROC(obj,fpr = NULL,display.method = c("PCA","OV"),displayOV = c(1,2), file = "animation1.gif",save = TRUE,border = TRUE, completeROC = FALSE,...).</pre> <p>The video is saved by default as a GIF with the name indicated in argument file (extension .gif should be added). A border for the classification subsets is drawn by default.</p> <p>For each element in vector fpr (optional input parameter), the function executed is</p> <ul style="list-style-type: none"> • for $p = 2$: <pre>plot_buildROC(obj,FPR = fpr[i],build.process = TRUE,completeROC,...);</pre> <ul style="list-style-type: none"> • for $p > 2$: <pre>plot_buildROC(obj,FPR = fpr[i],build.process = TRUE,display.method, displayOV,completeROC,...)</pre> <p>Same considerations about the input fpr as those for movieROC() over a ‘groc’ object.</p>
--

Table 1: Methods for a `movieROC` object: ‘groc’, ‘multiroc’ or ‘hroc’ object (output of the `gROC()`, `multiROC()` or `hROC()` function, respectively). The main input parameters are displayed.

2.3 Illustrative dataset

In order to illustrate the functionality of our R package, we consider the HCC data. This dataset is derived from gene expression arrays of tumor and adjacent non-tumor tissues of 62 Taiwanese cases of hepatocellular carcinoma (HCC). The goal of the original study (Shen et al., 2012) was to identify, with a genome-wide approach, additional genes hypermethylated in HCC that could be used for more accurate analysis of plasma DNA for early diagnosis, by using Illumina methylation arrays (Illumina, Inc., San Diego, CA) that screen 27,578 autosomal CpG sites. The complete dataset was deposited in NCBI’s Gene Expression Omnibus (GEO) and it is available through series accession number GSE37988 (www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE37988). It is included in the presented package (HCC dataset), selecting 948 genes with complete information.

The following code loads the R package and the HCC dataset (see the vignette for main structure).

```
R> library(movieROC)
R> data(HCC)
```

We selected the genes 20202438, 18384097, and 03515901. On the one hand, we chose the gene 03515901 as an example of a monotone relationship between the marker and the response, reporting a good ROC curve. On the other hand, relative gene expression intensities of the genes 20202438 and 18384097 tend to be more extreme in tissues with tumor than in those without it. These are non-standard cases, so if we limit ourselves to detect “appropriate” genes on the basis of the standard ROC curve, they would not be chosen. However, extending the decision rules by means of the gROC curve, those genes may be considered as potential biomarkers (locations) to differ between the two groups. The R code estimating and displaying the density probability function for gene expression intensities of the selected genes in each group (Figure 2) is included in the vignette.

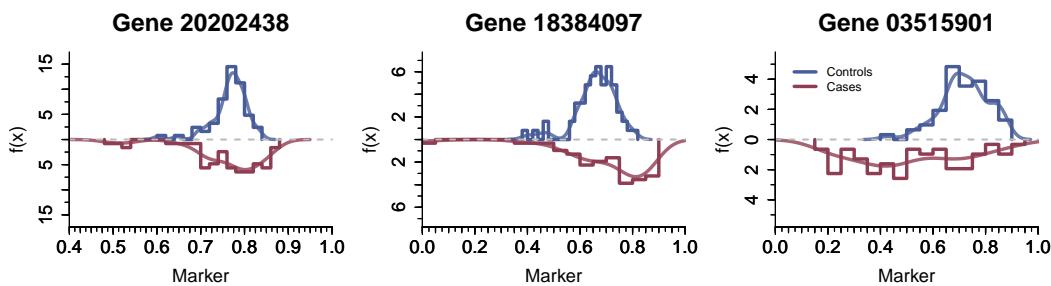


Figure 2: Density histograms and kernel density estimations (lighter) for gene expression intensities of the genes 20202438, 18384097 and 03515901 in negative (non-tumor) and positive (tumor) tissues.

3 Regular ROC curve

Assuming that there exists a monotone relationship between the marker and the response, the *regular, right-sided* or *standard ROC curve* associated with the marker X considers classification subsets of the form $s_t = (c_t, \infty)$. For each specificity $1 - t = \mathbb{P}(\chi \notin s_t) \in [0, 1]$, also called *true-negative rate*, there exists only one subset s_t reporting such specificity and thus a particular sensitivity, also called *true-positive rate*, $\mathbb{P}(\xi \in s_t)$. This results in a simple correspondence between each point of the ROC curve $\mathcal{R}_r(t) = 1 - F_\xi(F_\chi^{-1}(1 - t))$ and its associated classification region $s_t \in \mathcal{I}_r(t)$, where

$$\mathcal{I}_r(t) = \left\{ s_t = (c_t, \infty) : c_t \in \mathcal{S}(X), \mathbb{P}(\chi \in s_t) = t \right\}$$

is the *right-sided family of eligible classification subsets*. The definition of this family captures the shape of the decision rules and the target specificity.

If higher values of the marker are associated with a higher probability of not having the characteristic (see gene 03515901 in Figure 2), the ROC curve would be defined by the *left-sided family of eligible classification subsets* (Martínez-Camblor et al., 2017), $\mathcal{I}_l(t)$, similarly to $\mathcal{I}_r(t)$ but with the form $s_t = (\infty, c_t]$. It results in $\mathcal{R}_l(t) = F_\xi(F_\chi^{-1}(t))$, $t \in [0, 1]$, and the decision rules are also univocally defined in this case.

The ROC curve and related problems were widely studied in the literature; interested reader is referred to the monographs of Zhou et al. (2002), Pepe (2003), and Nakas et al. (2023), as well as the review by Inácio et al. (2021). By definition, the ROC curve is confined within the unit square, with optimal performance achieved when it approaches the left-upper corner (AUC closer to 1). Conversely, proximity to the main diagonal (AUC closer to 0.5) means diminished discriminatory ability, resembling a random classifier.

In practice, let $(\xi_1, \xi_2, \dots, \xi_n)$ and $(\chi_1, \chi_2, \dots, \chi_m)$ be two independent and identically distributed (i.i.d.) samples from the positive and the negative population, respectively. Different estimation procedures are implemented in the **movieROC** package, such as the empirical estimator (Hsieh and Turnbull, 1996) (by default in the `gROC()` function), accompanied by its summary indices: the AUC and the Youden index (Youden, 1950). Alternatively, semiparametric approaches based on kernel density estimation for the involved distributions may be considered (Zou et al., 1997). The `plot_densityROC()` function provides plots for both right- and left-sided ROC curved estimated by this method. On the other hand, assuming that the marker follows a gaussian distribution in both populations, that is, $\xi \sim \mathcal{N}(\mu_\xi, \sigma_\xi)$ and $\chi \sim \mathcal{N}(\mu_\chi, \sigma_\chi)$, parametric approaches propose plug-in estimators by estimating the unknown parameters while using the known distributions (Hanley, 1988). This parametric estimation is included in the `gROC_param()` function, which works similarly to `gROC()`.

MAIN SYNTAX:

```
gROC(X,D,side = "right",...)      gROC_param(X,D,side = "right",...)
```

Table 1 in the [vignette](#) provides the main input and output parameters of these R functions, which estimate the regular ROC curve (right-sided or left-sided with side = "right" or "left", respectively) and associated decision rules. Its output is an R object of class 'groc', to which the functions listed in Table 1 above can be applied. Most of them are visualization tools, but the user may also print() summary information and predict() classification regions for a particular specificity.

Figure 3 graphically represents the empirical estimation of the standard (gray line) and generalized (black line) ROC curves for each gene in Figure 2. To construct the standard ROC curve for the first two genes (20202438 and 18384097), the right-sided ROC curve is considered; and the left-sided curve for the third one (03515901). As expected following the discussion about Figure 2, the standard and gROC curves are similar for the third gene because there exists a monotone relationship between the marker and the response. However, these curves differ for the first two genes due to the lack of monotonicity in those scenarios. The empirical gROC curve estimator is explained in detail in Section 4.

Next chunk of code generates the figure, providing as example of the use of gROC() function, plot() and how to get access to the AUC.

```
R> for(gene in c("20202438", "18384097", "03515901")){
+   roc <- gROC(X = HCC[,paste0("cg",gene)], D = HCC$tumor,
+                 side = ifelse(gene == "03515901", "left", "right"))
+   plot(roc, col = "gray50", main = paste("Gene", gene), lwd = 3)
+   groc <- gROC(X = HCC[,paste0("cg",gene)], D = HCC$tumor, side = "both")
+   plot(groc, new = FALSE, lwd = 3)
+   legend("bottomright", paste(c("AUC =", "gAUC ="),
+                             format(c(roc$auc, groc$auc),
+                                   digits = 3)), col = c("gray50", "black"), lwd = 3, bty = "n",
+         inset = .01)}
```

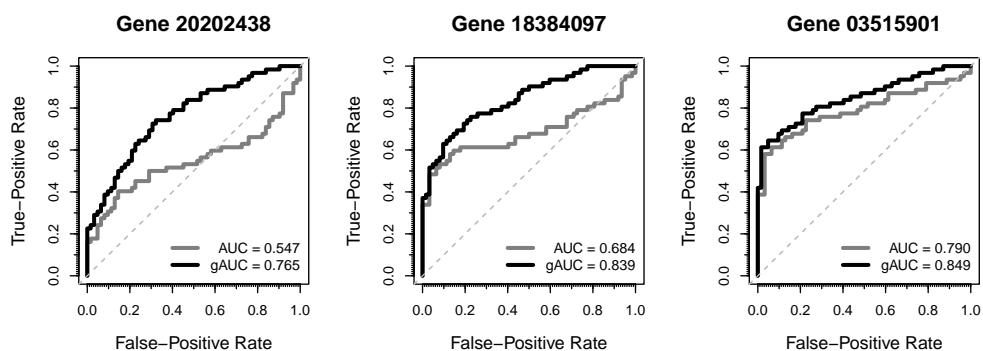


Figure 3: Standard ROC curve (in gray) and gROC curve (in black) empirical estimation for the capacity of genes 20202438, 18384097 and 03515901 to differ between tumor and non-tumor group.

The following code snippet estimates the standard ROC curve for gene 20202438, prints its basic information, and predicts the classification region and sensitivity resulting in a specificity of 0.9. It provides an illustrative example of utilizing the print() and predict() functions.

```
R> roc_selg1 <- gROC(X = HCC$cg20202438, D = HCC$tumor, side = "right")
R> roc_selg1
```

Data was encoded with nontumor (controls) and tumor (cases).
It is assumed that larger values of the marker indicate larger confidence that a given subject is a case.
There are 62 controls and 62 cases.
The specificity and sensitivity reported by the Youden index are 0.855 and 0.403, respectively, corresponding to the following classification subset: (0.799, Inf).
The area under the right-sided ROC curve (AUC) is 0.547.

```
R> predict(roc_selg1, FPR = .1)

$ClassSubsets           $Specificity          $Sensitivity
[1] 0.8063487    Inf      [1] 0.9032258   [1] 0.3064516
```

The following line of code displays the whole construction of the empirical standard ROC curve for gene 20202438. The video is saved by default as a GIF with the name provided.

```
R> movieROC(roc_selg1, reduce = FALSE, file = "StandardROC_gene20202438.gif")
```

3.1 Multivariate ROC curve

In practice, many cases may benefit from combining information from different markers to enhance classification accuracy. Rather than assessing univariate markers separately, taking the multivariate marker resulting from merging them can yield relevant gain. However, note that the ROC curve and related indices are defined only for univariate markers, as they require the existence of a total order. To address this limitation, a common approach involves transforming the p -dimensional multivariate marker X into a univariate one through a functional transformation $h : \mathbb{R}^p \rightarrow \mathbb{R}$. This transformation $h(\cdot)$ seeks to optimize an objective function related to the classification accuracy, usually the AUC (Su and Liu, 1993; McIntosh and Pepe, 2002; Martínez-Camblor et al., 2019).

We enumerate the methods included in the proposed R tool by the `multiROC()` function (with the input parameter `method`), listed according to the objective function to optimize. Recall that the output of `multiROC()` is an object of class ‘`multiroc`’, containing information about the estimation of the ROC curve and subsets for multivariate scenarios. Table 2 in the `vignette` includes the usage of this function.

MAIN SYNTAX:

```
multiROC(X,D,method = "lrm",
          formula = 'D ~ X.1 + I(X.1^2) + X.2 + I(X.2^2) + I(X.1*X.2)' ,...)
```

- 1.- AUC: Different procedures to estimate the $h(\cdot)$ maximizing the AUC in the multidimensional case have been studied in the literature. Among all families of functions, linear combinations ($\mathcal{L}_\beta(X) = \beta_1X_1 + \dots + \beta_pX_p$) are widely used due to their simplicity; an extensive review of the existing methods was conducted by Kang et al. (2016).

Computation: In the `multiROC()` function, fixing input parameters `method = "fixedLinear"` and `methodLinear` to one from “SuLiu” (Su and Liu, 1993), “PepeThompson” (Pepe and Thompson, 2000), or “minmax” (Liu et al., 2011). The R function also admits quadratic combinations when $p = 2$, i.e. $\mathcal{Q}_\beta(X) = \beta_1X_1 + \beta_2X_2 + \beta_3X_1X_2 + \beta_4X_1^2 + \beta_5X_2^2$, by fixing `method = "fixedQuadratic"` for a particular `coefQuadratic = beta = (beta1, ..., beta5)`.

- 2.- The risk score function $\text{logit}\{\mathbb{P}(D = 1 | X)\}$: Our package allows the user to fit a logistic regression model (`method = "lrm"`) considering any family of functions (linear, quadratic, whether considering interactions or not...) by means of the input parameter `formula.lrm`. A stepwise regression model is fitted if `stepModel = TRUE`. Details are explained in Section 5.

- 3.- The sensitivity for a particular specificity:

- a) Considering the theoretical discussion about the search of the optimal transformation $h(\cdot)$ pointed out in Section 5, Martínez-Camblor et al. (2021b) proposed to estimate it by multivariate kernel density estimation for positive and negative groups separately.

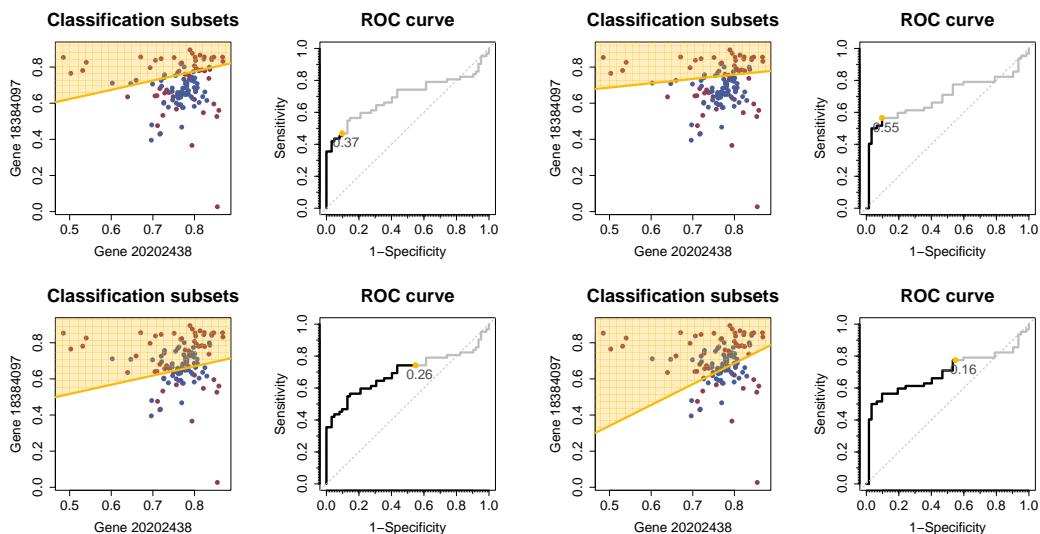
Computation: The `multiROC()` function integrates the estimation procedures for the bandwidth matrix developed by [Duong \(2007\)](#), by fixing `method = "kernelOptimal"` and choosing a proper method to estimate the bandwidth ("`kernelOptimal.H`").

- b) Mainly linear combinations have been explored to date in the scientific literature ([Meisner et al., 2021](#); [Pérez-Fernández et al., 2021](#)). For a fixed specificity $t \in [0, 1]$, we seek the linear combination $\mathcal{L}_{\beta(t)}(\mathbf{X}) = \beta_1(t)X_1 + \dots + \beta_p(t)X_p$ maximizing the true-positive rate by considering standard subsets for the transformed marker. The coefficients $\beta(t)$ are called 'dynamic parameters' because they may be different for each $t \in [0, 1]$.

Computation: Since our objective is to display the ROC curve, $\mathcal{L}_{\beta(t)}(\mathbf{X})$ is estimated for every t in a grid of the unit interval, resulting in one $\hat{\beta}(t)$ for each t . This approach is time-consuming, especially when it is based on the plug-in empirical estimators involved (`method = "dynamicEmpirical"`, only implemented for $p = 2$), and may result in overfitting. Instead, [Meisner et al. \(2021\)](#) method is recommended (`method = "dynamicMeisner"`).

Once the classification subsets for a multivariate marker are constructed by the `multiROC()` function, several R methods may be used for the output object (see Table 1). They include `print` relevant information or `plot` the resulting ROC curve. The main contribution of the package is to plot the construction of the ROC curve together with the classification subsets in static figure for a particular FPR (`plot_buildROC()` function), or in a video for tracking the whole process (`movieROC()` function).

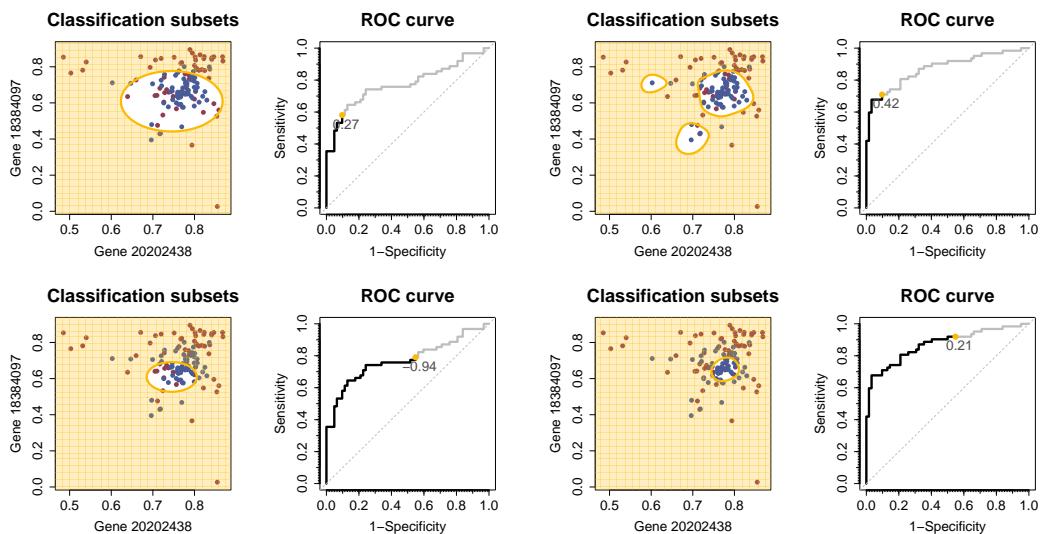
Figure 4 illustrates snapshots of videos resulting from `movieROC()` function for two FPR: 0.1 and 0.55. Particularly, classification accuracy of the bivariate marker (`cg20202438`, `cg18384097`) was studied by using four different approaches indicated on the captions, considering linear combinations (top) and nonlinear transformations (bottom). This figure was implemented by the code below, integrating `multiROC()` and `movieROC()` functions. Four videos are saved as GIF files with names "`PepeTh.gif`" (a), "`Meisner.gif`" (b), "`LRM.gif`" (c), and "`KernelDens.gif`" (d).



(a) Linear combinations with fixed parameters by [Pepe and Thompson \(2000\)](#).

(b) Linear combinations with dynamic parameters by [Meisner et al. \(2021\)](#).

```
R> X <- HCC[, c("cg20202438", "cg18384097")]; D <- HCC$tumor
R> biroc_12_PT <- multiROC(X, D, method = "fixedLinear", methodLinear = "PepeThompson")
R> biroc_12_Meis <- multiROC(X, D, method = "dynamicMeisner", verbose = TRUE)
R> biroc_12_lrm <- multiROC(X, D)
```



(c) Logistic regression model with quadratic formula by default (see `formula.lrm` in Table 2 of the vignette). **(d)** Optimal transformation by multivariate kernel density estimation with "Hbcv" method by default (Martinez-Camblor et al., 2021b).

Figure 4: Snapshots (from `movieROC()` videos) of the classification procedure and ROC curve for the bivariate marker (`cg20202438`, `cg18384097`) when false-positive rate equals 0.1 (top of each subfigure) and 0.55 (bottom of each subfigure). Four different methods for classification are displayed.

```
R> biroc_12_kernel <- multiROC(X, D, method = "kernelOptimal")
R> list_biroc <- list(PepeTh = biroc_12_PT, Meisner = biroc_12_Meis,
+                      LRM = biroc_12_lrm, KernelDens = biroc_12_kernel)
R> lapply(names(list_biroc), function(x) movieROC(list_biroc[[x]],
+                                               display.method = "OV", xlab = "Gene 20202438", ylab = "Gene 18384097",
+                                               cex = 1.2, alpha.points = 1, lwd.curve = 4, file = paste0(x, ".gif")))
```

When the marker has a dimension higher than two it is difficult to visualize the data and the classification regions. Therefore, the `movieROC()` function offers two options for showing the results, both on a bidimensional space. On the one hand, to choose two of the components of the multivariate marker and project the classification subsets on the plain defined by them (Figure 5, middle). On the other, to project the classification regions on the plain defined by the two first principal components (Figure 5, left). The R function `prcomp()` from `stats` is used to perform Principal Components Analysis (PCA) (Hotelling, 1933).

Figure 5 shows the difficulty in displaying the decision rules when $p > 2$ (the 3 genes used along this manuscript), even with the two options implemented in our package. It was generated using `multiROC()` and `plot_buildROC()`:

```
R> multiroc_PT <- multiROC(X = HCC[, c("cg20202438", "cg18384097", "cg03515901")],
+                           D = HCC$tumor, method = "fixedLinear", methodLinear = "PepeThompson")
R> multiroc_PT
```

Data was encoded with nontumor (controls) and tumor (cases).
There are 62 controls and 62 cases.
A total of 3 variables have been considered.
A linear combination with fixed parameters estimated by PepeThompson approach has been considered.
The specificity and sensitivity reported by the Youden index are 0.855 and 0.742, respectively, corresponding to the cut-off point -0.0755 for the transformation $h(X) = 0.81*cg20202438 - 0.1*cg18384097 - 1*cg03515901$.
The area under the ROC curve (AUC) is 0.811.

```
R> plot_buildROC(multiroc_PT, cex = 1.2, lwd.curve = 4)
R> plot_buildROC(multiroc_PT, display.method = "OV", displayOV = c(1,3), cex = 1.2,
+                   xlab = "Gene 20202438", ylab = "Gene 03515901", lwd.curve = 4)
```

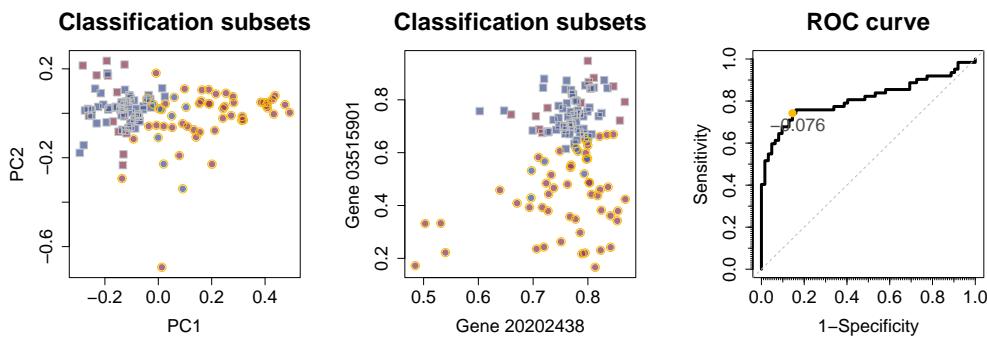


Figure 5: Multivariate ROC curve estimation for the simultaneous diagnostic accuracy of genes 20202438, 18384097 and 03515901. Pepe and Thompson (2000) approach was used to estimate the linear coefficients and classification rules (yellow and gray border for positive and negative class, respectively) for a FPR 0.15 are displayed. Left, projected over the 2 principal components from PCA; middle, over the 1st and the 3rd selected genes.

4 Generalized ROC curve

There are scenarios whose standard ROC curves are not concave (first two genes in Figure 3, gray solid line), reflecting that the standard initial assumption of existence of a monotone relationship between the marker and the response is misleading. In Figure 2, we may see that difference in gene 20202438 distribution between those tissues who have the characteristic and those who do not is mainly on dispersion. To accommodate this common type of scenarios, Martínez-Camblor et al. (2017) extended the ROC curve definition to the case where both extremes for marker values are associated with a higher risk of having the characteristic of interest, by considering the *both-sided family of eligible classification subsets*:

$$\mathcal{I}_g(t) = \left\{ s_t = (-\infty, x_t^L] \cup (x_t^U, \infty) : x_t^L \leq x_t^U \in \mathcal{S}(X), \mathbb{P}(\chi \in s_t) = t \right\}.$$

It becomes crucial to consider the supremum in the definition of the generalized ROC curve because the decision rule for each $t \in [0, 1]$ is not univocally defined: there exist infinite pairs $x_t^L \leq x_t^U$ reporting a specificity $1 - t$ (i.e. $\mathcal{I}_g(t)$ is uncountably infinite). Computationally, this optimization process incurs a time-consuming estimation, depending on the number of different marker values in the sample.

After the introduction of this extension, several studies followed up regarding estimation of the gROC curve (Martínez-Camblor et al., 2017; Martínez-Camblor and Pardo-Fernández, 2019a) and related measures such as its area (gAUC) (Martínez-Camblor et al., 2021a) and the Youden index (Martínez-Camblor and Pardo-Fernández, 2019b; Bantis et al., 2021). By considering this generalization, another property of the classification subsets may be lost: the regions may not be self-contained over the increase in false-positive rate. It may happen that a subject is classified as a positive for a particular FPR t_1 , but as a negative for a higher FPR t_2 . Therefore, it is natural to establish a restriction (C) on the classification subsets, ensuring that any subject classified as a positive for a fixed specificity (or sensitivity) will be also classified as a positive for any classification subset with lower specificity (higher sensitivity). Pérez-Fernández et al. (2021) proposed an algorithm to estimate the gROC curve under restriction (C), included in the gROC() function of the presented R package. See final Section for computational details about this algorithm implementation.

MAIN SYNTAX:

```
gROC(X,D,side = "both",...)      gROC_param(X,D,side = "both",...)
```

Table 1 in the [vignette](#) collects the input and output parameters of the `gROC()` function, which estimates the gROC curve, both in the mentioned direction (`side = "both"`) and in the opposite, i.e. when classification subsets of the form $s_t = (x_t^L, x_t^U]$ are considered (`side = "both2"`). In addition, all the particular methods for a 'groc' object collected in Table 1 above may be used in this general scenario.

Following the gene 20202438 expression intensity diagnostic accuracy is evaluated by the gROC curve without restrictions (`groc_selg1` object) and under the restriction (C) (`groc_selg1_C` object). The classification subsets and sensitivity for a specificity of 0.9 are displayed with the `predict()` function.

```
R> groc_selg1 <- gROC(X = HCC$cg20202438, D = HCC$tumor, side = "both")
R> predict(groc_selg1, FPR = .1)
```

\$ClassSubsets	\$Specificity	\$Sensitivity
[, 1] [, 2]	[1] 0.9032258	[1] 0.4032258
[1,] -Inf 0.7180623		
[2,] 0.8296072 Inf		

```
R> groc_selg1_C <- gROC(X = HCC$cg20202438, D = HCC$tumor, side = "both",
+                         restric = TRUE, optim = TRUE)
```

All the classification regions underlying the standard and the generalized ROC curves without and with restrictions are represented in Figure 6. The following code was used to generate the figure, illustrating the usage and output of the `plot_regions()` function. Besides displaying all the classification regions underlying every specificity (in gray), the one chosen by the user (`FPR = 0.15` by default) is highlighted in blue. Note that the ROC curves are rotated 90° to the right, in order to use the vertical axis for FPR in both plots.

```
R> plot_regions(roc_selg1, cex.legend = 1.5, plot.auc = TRUE,
+                 main = "Standard right-sided assumption [Classification subsets]")
R> plot_regions(groc_selg1, plot.auc = TRUE, legend = F, main.plotroc = "gROC curve",
+                 main = "General approach [Classification subsets]")
R> plot_regions(groc_selg1_C, plot.auc = TRUE, legend = F, main.plotroc = "gROC curve",
+                 main = "General approach with restriction (C) [Classific. subsets]",
+                 xlab = "Gene 20202438 expression intensity")
```

It is clear the gain achieved for considering the generalized scenario for this marker, which fits better its distribution in each group. Standard estimated AUC is 0.547, while the gAUC increases to 0.765. The gAUC is not especially affected by imposing the restriction (C), resulting in 0.762.

5 Efficient ROC curve: pursuing an optimal transformation

By keeping classification subsets of the form $s_t = (c_t, \infty)$, an alternative approach can be explored: transforming the univariate marker through a suitable function $h : \mathbb{R} \rightarrow \mathbb{R}$ to enhance its accuracy. Henceforth, the transformation $h^*(\cdot)$ reporting the dominant ROC curve compared to the one from any other function (i.e. $\mathcal{R}_{h^*}(\cdot) \geq \mathcal{R}_h(\cdot)$) will be referred to as *optimal transformation* (in the ROC sense), and the resulting ROC curve is called eROC ([Kauppi, 2016](#)). Following the well-known Neyman–Pearson lemma, [McIntosh and Pepe \(2002\)](#) proved that $h^*(\cdot)$ is the likelihood ratio.

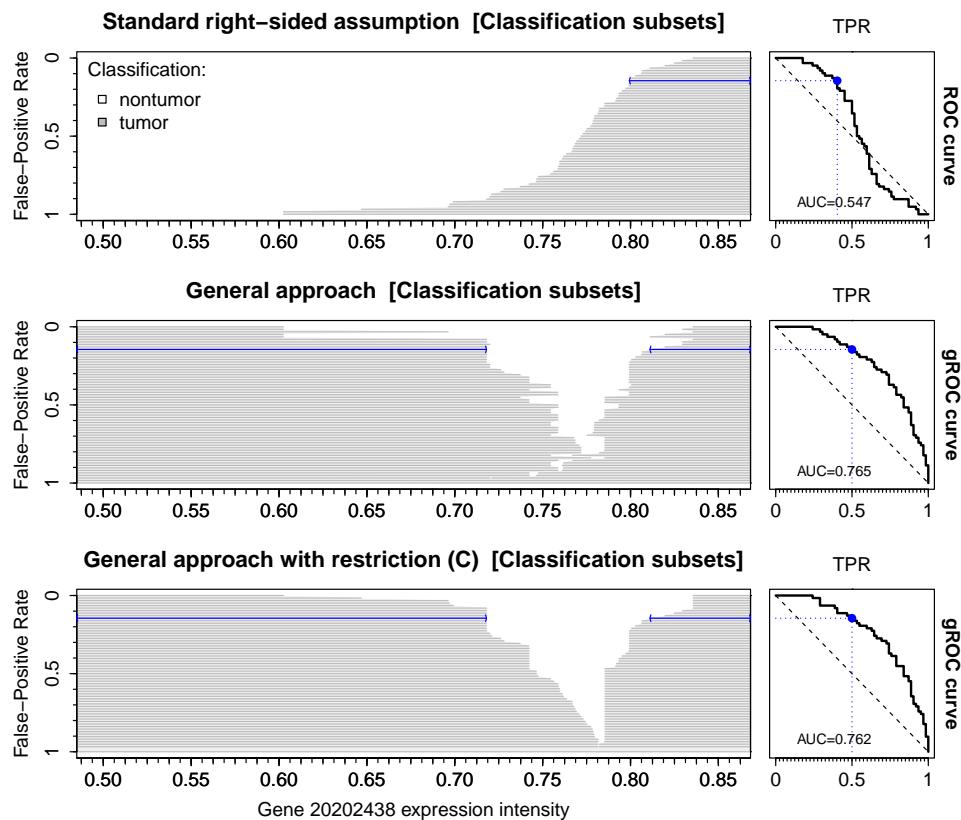


Figure 6: Classification regions and the ROC curve (90° rotated) for evaluation of gene 20202438 expression intensity assuming i) standard scenario (top), ii) generalized scenario without restrictions (middle), iii) generalized scenario under restriction (C) over the subsets (bottom).

We enumerate the methods included in the proposed R tool by the `hROC()` function (with the input parameter `type`), listed according to the procedure considered to estimate $h^*(\cdot)$. The output of this function is an object of class ‘`hroc`’. See Table 3 in the [vignette](#) for function usage and output details.

MAIN SYNTAX: `hROC(X, D, type = "lrm", formula.lrm = 'D ~ pol(X, 3)', ...)`

- 1.- [Martínez-Camblor et al. \(2019\)](#) exploited the result proved by [McIntosh and Pepe \(2002\)](#), suggesting to estimate the logit of the risk function by logistic regression, since it is a monotone increasing transformation of the likelihood ratio.

Computation: By the proposed R tool, the user can define any transformation $h(\cdot)$ for the right-hand side of the logistic regression model to be fitted, $\text{logit}\{\text{IP}(D = 1 | x)\} = h(x)$. Particularly, by fixing the input parameters: `type = "lrm"` and defining the function $h(\cdot)$ as `formula.lrm`.

- 2.- Arguing as in [Martínez-Camblor et al. \(2021b\)](#) for univariate markers instead of multivariate, the optimal transformation in the ROC sense is equivalent to $h^*(\cdot) = f_\xi(\cdot)/(f_\xi(\cdot) + f_\chi(\cdot))$, where $f(\cdot)$ denotes the density function. In order to estimate $h^*(\cdot)$, different estimation procedures for the density functions separately may be used, such as the kernel density estimator.

Computation: By the `hROC()` function, the user may fix `type = "kernel"` and choosing a proper bandwidth for the kernel estimation by `kernel.h` in order to compute this method.

- 3.- [Martínez-Camblor et al. \(2019\)](#) also included the estimation of the *overfitting function*, $h_{of}(\cdot)$, defined as the optimal one when no restrictions on the shape of $h^*(\cdot)$ are

imposed. It takes the value 1 for the positive marker values and 0 for the negative ones, reporting an estimated AUC of 1, but totally depending on the available sample (the resulting rules cannot be extended).

Computation: $h_{of}(\cdot)$ may be estimated by fixing the input parameter type = "overfitting".

Following code and figures study the capacity of improving the classification performance of the gene 18384097 expression intensity via the above functional transformations and its impact on the final decision rules. The first one considering an ordinary cubic polynomial formula (`hroc_cubic_selg2`), and a linear tail-restricted cubic splines (`hroc_rcs_selg2`) for the right-hand side of logistic regression model. The second one using two different bandwidths ($h = 1$ and $h = 3$) for density function estimation. For a comparative purpose, the last one estimates the gROC curve under restriction (C).

```
R> X <- HCC$cg18384097; D <- HCC$tumor

R> hroc_cubic_selg2 <- hROC(X, D); hroc_cubic_selg2

Data was encoded with nontumor (controls) and tumor (cases).
There are 62 controls and 62 cases.
A logistic regression model of the form D ~ pol(X,3) has been performed.
The estimated parameters of the model are the following:
  Intercept          X          X^2          X^3
  "1.551"    "32.054"   "-120.713"   "100.449"
The specificity and sensitivity reported by the Youden index are 0.935 and 0.532,
respectively, corresponding to the following classification subset:
(-Inf, 0.442) U (0.78, Inf).
The area under the ROC curve (AUC) is 0.759.

R> hroc_rcs_selg2 <- hROC(X, D, formula.lrm = "D ~ rcs(X,8)")
R> hroc_lkr1_selg2 <- hROC(X, D, type = "kernel")
R> hroc_lkr3_selg2 <- hROC(X, D, type = "kernel", kernel.h = 3)
R> hroc_overfit_selg2 <- hROC(X, D, type = "overfitting")

R> groc_selg2_C <- gROC(X, D, side = "both", restric = TRUE, optim = TRUE)
```

The following code snippet compares the AUC achieved from each approach considered above:

```
R> list_hroc <- list(Cubic = hroc_cubic_selg2, Splines = hroc_rcs_selg2,
+                      Overfit = hroc_overfit_selg2, LikRatioEst_h3 = hroc_lkr3_selg2,
+                      LikRatioEst_h1 = hroc_lkr1_selg2, gAUC_restC = groc_selg2_C)
```

```
R> AUCs <- sapply(list_hroc, function(x) x$auc)
R> round(AUCs, 3)

Cubic          Splines    Overfit   LikRatioEst_h3   LikRatioEst_h1       gAUC_restC
0.759          0.807      1.000      0.781          0.799          0.836
```

The shape of the classification regions over the original space $\mathcal{S}(X)$ depends on the monotonicity of $h^*(\cdot)$, which may be graphically studied by the `plot_funregions()` function (see Figure 7). These regions can be visualized by the R function `plot_regions()` (see Figure 8). Both are explained in Table 1 and illustrated below. The next chunk of code produced Figure 7, representing the different functional transformations estimated previously:

```
R> lapply(list_hroc, function(x) plot_funregions(x, FPR = .15, FPR2 = .5))
```

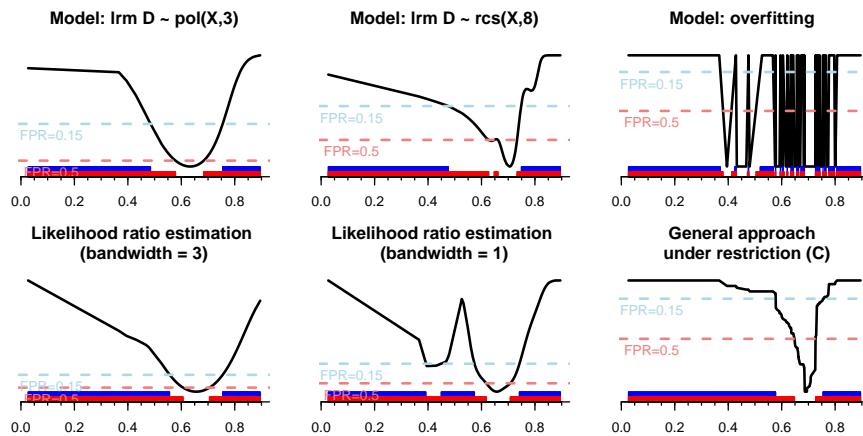


Figure 7: Different functional transformations and resulting classification subsets for gene 18384097. Rules for FPR 0.15 (blue) and 0.50 (red) are remarked. Top, from left to right: cubic polynomial function, restricted cubic splines (with 8 knots), and overfitted transformation. Bottom: likelihood ratio estimation with bandwidths 3 (left) and 1 (middle), and transformation resulting in the gROC curve under restriction (C).

Finally, using the `plot_regions()` function, Figure 8 shows the resulting classification subsets over the original space for the best two of the six methods above. First method (fitting a logistic regression model with restricted cubic splines with 8 knots) reports an AUC of 0.804 (compared to 0.684 by the standard ROC curve), but the shape of some classification rules is complex, such as $s_t = (-\infty, a_t] \cup (b_t, c_t] \cup (d_t, \infty)$. This area increases to 0.836 by considering subsets of the form $s_t = (-\infty, x_t^L] \cup (x_t^U, \infty)$, even imposing the restriction (C) to get a functional transformation $h(\cdot)$.

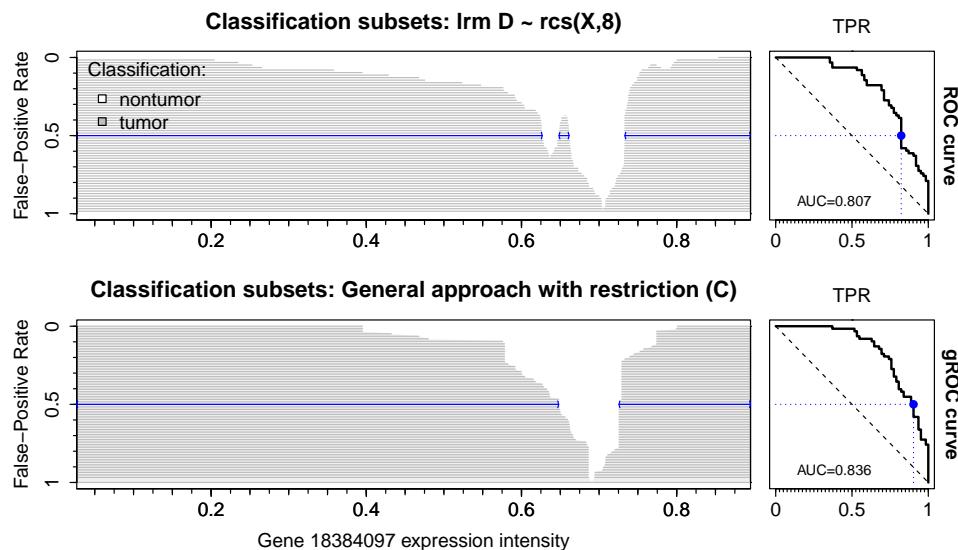


Figure 8: Classification regions and the resulting ROC curve (90° rotated) for the gene 18384097. Top, ROC curve for restricted cubic splines transformation with 8 knots; bottom, gROC curve under restriction (C) for the original marker.

6 Summary and conclusion

Conducting binary classification using continuous markers requires establishment of decision rules. In the standard case, each specificity $t \in [0, 1]$ entails a classification subset of the form $s_t = (c_t, \infty)$ univocally defined. However, in more complex situations – such as there is a non-monotone relationship between the marker and the response or in multivariate scenarios – these become not clear. Visualization of the decision rules becomes crucial in these cases. To address this, the [movieROC](#) package incorporates novel visualization tools complementing the ROC curve representation.

This R package offers a user-friendly and easily comprehensible software solution tailored for practical researchers. It implements statistical techniques to estimate and compare, and finally to graphically represent different classification procedures. While several R packages address ROC curve estimation, the proposed one emphasizes the classification process, tracking the decision rules underlying the studied binary classification problem. This tool incorporates different considerations and transformations which may be useful to capture the potential of the marker to classify in non-standard scenarios. Nevertheless, this library is also useful in standard cases, as well as when the marker itself comes from a classification or regression method (such as support vector machines), because it provides nice visuals and additional information not usually reported with the ROC curve.

The main function of the package, `movieROC()`, allows to monitor how the resulting classification subsets change along different specificities, thereby building the corresponding ROC curve. Notably, it introduces time as a third dimension to keep those specificities, generating informative videos. For interested readers or potential users of [movieROC](#), the [manual](#) available in CRAN provides complete information about the implemented functions and their parameters. In addition, a [vignette](#) is accessible, including mathematical formalism and details about the algorithms implemented.

Computational considerations

Dependencies Some functions of our package depend on other libraries available on CRAN:

- `gROC(X,D,side = "both", restric = TRUE, optim = TRUE, ...)` uses the `allShortestPaths()` function in the `e1071` package (Meyer et al., 2023).
- `hROC(X,D,type = "lrm", ...)` and `multiROC(X,D,method = "lrm", ...)` use the `lrm()` function in the `rms` package (Harrell Jr, 2023).
- `multiROC(X,D,method = "kernelOptimal", ...)` uses the `kde()` function in the `ks` package (Duong, 2023).
- `multiROC(X,D,method = "dynamicMeisner", ...)` uses the `maxTPR()` function in the `maxTPR` package (Meisner et al., 2021). This package was removed from the CRAN repository, so we integrated the code of the `maxTPR()` function into our package. This function uses `Rsolnp::solnp()` and `robustbase::BYlogreg()`.
- `multiROC(X,D,method = "fixedLinear", methodLinear, ...)` uses the R functions included in Kang et al. (2016) (Appendix). We integrated this code into our package.
- `movieROC(obj, save = TRUE, ...)` uses the `saveGIF()` function in the `animation` package (Xie et al., 2021).

Limitations Users should be aware of certain limitations while working with this package:

- Some methods are potentially time-consuming, especially with medium to large sample sizes:

The estimation of the gROC curve under restriction (C) can be computationally intensive, especially when considering different FPR to locally optimize the search using `gROC(X,D,side = "both", restric = TRUE, optim = TRUE, t0max = TRUE)`. Note that this method involves a quite exhaustive search of the self-contained classification subsets leading to the optimal gROC curve estimate. However, even selecting different false-positive rates t_0 to start from, it may not result in the optimal achievable estimate under restriction (C). Input parameters `restric`, `optim`, `t0` and `t0max` for `gROC()` function included in Table 1 of the `vignette` serve to control this search.

Similarly, it also occurs for multivariate markers when considering linear frontiers with dynamic parameters (by using `multiROC(X,D,method = "dynamicMeisner" | "dynamicEmpirical")`).

- Most implemented R functions consider empirical estimation for the resulting ROC curve, even if the procedure to estimate the decision rules is semi-parametric. An exception is the `gROC_param()` function, which accommodates the binormal scenario.
- When visualizing classification regions for multivariate markers with high dimension (`plot_buildROC()` and `movieROC()` functions for a ‘`multiroc`’ object), our package provides some alternatives, but additional improvements could provide further aid in interpretation.

Acknowledgements

The authors acknowledge support by the Grants PID2019-104486GB-I00 and PID2020-118101GB-I00 from Ministerio de Ciencia e Innovación (Spanish Government), and by a financial Grant for Excellence Mobility for lecturers and researchers subsidized by the University of Oviedo in collaboration with Banco Santander.

References

L. E. Bantis, J. V. Tsimikas, G. R. Chambers, M. Capello, S. Hanash, and Z. Feng. The length of the receiver operating characteristic curve and the two cutoff Youden index within a

- robust framework for discovery, evaluation, and cutoff estimation in biomarker studies involving improper receiver operating characteristic curves. *Statistics in Medicine*, 40(7):1767–1789, 2021. URL <https://doi.org/10.1002/sim.8869>. [p70]
- S. Díaz-Coto. smoothROCtime: an R package for time-dependent ROC curve estimation. *Computational Statistics*, 35:1231–1251, 2020. URL <https://doi.org/10.1007/s00180-020-00955-7>. [p61]
- D. Dorfman, K. Berbaum, C. Metz, R. Lenth, J. Hanley, and H. Dagga. Proper receiver operating characteristic analysis: The bigamma model. *Academic Radiology*, 4(2):138–149, 1997. URL [https://doi.org/10.1016/s1076-6332\(97\)80013-x](https://doi.org/10.1016/s1076-6332(97)80013-x). [p60]
- T. Duong. ks: Kernel density estimation and kernel discriminant analysis for multivariate data in R. *Journal of Statistical Software*, 21(7):1–16, 2007. URL <https://doi.org/10.18637/jss.v021.i07>. [p68]
- T. Duong. ks: Kernel Smoothing, 2023. URL <https://CRAN.R-project.org/package=ks>. R package version 1.14.1. [p76]
- J. A. Hanley. The robustness of the "binormal" assumptions used in fitting ROC curves. *Medical Decision Making*, 8(3):197–203, 1988. URL <https://doi.org/10.1177/0272989X8800800308>. PMID: 3398748. [p65]
- J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36, 1982. URL <https://doi.org/110.1148/radiology.143.1.7063747>. [p59]
- F. E. Harrell Jr. rms: Regression Modeling Strategies, 2023. URL <https://CRAN.R-project.org/package=rms>. R package version 6.7-1. [p76]
- H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417–441, 1933. URL <https://doi.org/10.1037/h0071325>. [p69]
- F. Hsieh and B. W. Turnbull. Nonparametric and semiparametric estimation of the receiver operating characteristic curve. *The Annals of Statistics*, 24(1):25–40, 1996. URL <https://doi.org/10.1214/aos/1033066197>. [p65]
- V. Inácio, M. X. Rodríguez-Álvarez, and P. Gayoso-Diz. Statistical evaluation of medical tests. *Annual Review of Statistics and Its Application*, 8(1):41–67, 2021. URL <https://doi.org/10.1146/annurev-statistics-040720-022432>. [p65]
- L. Kang, A. Liu, and L. Tian. Linear combination methods to improve diagnostic/prognostic accuracy on future observations. *Statistical Methods in Medical Research*, 25(4):1359–1380, 2016. URL <https://doi.org/10.1177/0962280213481053>. [p60, 67, 76]
- H. Kauppi. The generalized receiver operating characteristic curve. Discussion paper 114, Aboa Centre for Economics, 2016. URL <https://www.econstor.eu/bitstream/10419/233329/1/aboa-ce-dp114.pdf>. [p60, 71]
- C. Liu, A. Liu, and S. Halabi. A min–max combination of biomarkers to improve diagnostic accuracy. *Statistics in Medicine*, 30(16):2005–2014, 2011. URL <https://doi.org/10.1002/sim.4238>. [p67]
- M. López-Ratón, M. X. Rodríguez-Álvarez, C. C. Suárez, and F. G. Sampedro. OptimalCutpoints: An R package for selecting optimal cutpoints in diagnostic tests. *Journal of Statistical Software*, 61(8):1–36, 2014. URL <https://doi.org/10.18637/jss.v061.i08>. [p61]
- P. Martínez-Camblor. On the paper “Notes on the overlap measure as an alternative to the Youden index”. *Statistics in Medicine*, 37(7):1222–1224, 2018. URL <https://doi.org/10.1002/sim.7517>. [p60]

- P. Martínez-Camblor and J. C. Pardo-Fernández. Parametric estimates for the receiver operating characteristic curve generalization for non-monotone relationships. *Statistical Methods in Medical Research*, 28(7):2032–2048, 2019a. URL <https://doi.org/10.1177/0962280217747009>. [p70]
- P. Martínez-Camblor and J. C. Pardo-Fernández. The Youden index in the generalized receiver operating characteristic curve context. *The International Journal of Biostatistics*, 15(1), 2019b. URL <https://doi.org/10.1515/ijb-2018-0060>. [p70]
- P. Martínez-Camblor, N. Corral, C. Rey, J. Pascual, and E. Cernuda-Morollón. Receiver operating characteristic curve generalization for non-monotone relationships. *Statistical Methods in Medical Research*, 26(1):113–123, 2017. URL <https://doi.org/10.1177/0962280214541095>. [p60, 65, 70]
- P. Martínez-Camblor, S. Pérez-Fernández, and S. Díaz-Coto. Improving the biomarker diagnostic capacity via functional transformations. *Journal of Applied Statistics*, 46(9):1550–1566, 2019. URL <https://doi.org/10.1080/02664763.2018.1554628>. [p60, 67, 72]
- P. Martínez-Camblor, S. Pérez-Fernández, and S. Díaz-Coto. The area under the generalized receiver-operating characteristic curve. *The International Journal of Biostatistics*, 18(1):293–306, 2021a. URL <https://doi.org/10.1515/ijb-2020-0091>. [p70]
- P. Martínez-Camblor, S. Pérez-Fernández, and S. Díaz-Coto. Optimal classification scores based on multivariate marker transformations. *AStA Advances in Statistical Analysis*, 105(4):581–599, 2021b. URL <https://doi.org/10.1007/s10182-020-00388-z>. [p67, 69, 72]
- R. Mayeux. Biomarkers: Potential uses and limitations. *NeuroRx*, 1(2):182–188, 2004. URL <https://doi.org/10.1602/neurorx.1.2.182>. [p59]
- M. W. McIntosh and M. S. Pepe. Combining several screening tests: Optimality of the risk score. *Biometrics*, 58(3):657–664, 2002. URL <https://doi.org/10.1111/j.0006-341X.2002.00657.x>. [p67, 71, 72]
- A. Meisner, M. Carone, M. S. Pepe, and K. F. Kerr. Combining biomarkers by maximizing the true positive rate for a fixed false positive rate. *Biometrical Journal*, 63(6):1223–1240, 2021. URL <https://doi.org/10.1002/bimj.202000210>. [p60, 68, 76]
- 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, 2023. URL <https://CRAN.R-project.org/package=e1071>. R package version 1.7-13. [p76]
- C. Nakas, L. Bantis, and C. Gatsonis. *ROC Analysis for Classification and Prediction in Practice*. Chapman & Hall/CRC biostatistics series. CRC Press, 2023. ISBN 9781032480220. URL <https://www.routledge.com/ROC-Analysis-for-Classification-and-Prediction-in-Practice/Nakas-Bantis-Gatsonis/p/book/9781482233704>. [p65]
- M. S. Pepe. *The Statistical Evaluation of Medical Tests for Classification and Prediction*. Oxford Statistical Science Series. Oxford University Press, 2003. ISBN 9780198509844. URL <https://global.oup.com/academic/product/the-statistical-evaluation-of-medical-tests-for-classification-and-prediction-9780198565826?cc=es&lang=en&>. [p65]
- M. S. Pepe and M. L. Thompson. Combining diagnostic test results to increase accuracy. *Biostatistics*, 1(2):123–140, 2000. URL <https://doi.org/10.1093/biostatistics/1.2.123>. [p67, 68, 70]
- S. Pérez-Fernández, P. Martínez-Camblor, P. Filzmoser, and N. Corral. nsROC: An R package for non-standard ROC curve analysis. *The R Journal*, 10(2):55–74, 2018. URL <https://doi.org/10.32614/RJ-2018-043>. [p61]
- S. Pérez-Fernández, P. Martínez-Camblor, P. Filzmoser, and N. Corral. Visualizing the decision rules behind the ROC curves: understanding the classification process. *AStA Advances in Statistical Analysis*, 105(1):135–161, 2021. URL <https://doi.org/10.1007/s10182-020-00385-2>. [p68, 70]

- X. Robin, N. Turck, A. Hainard, N. Tiberti, F. Lisacek, J.-C. Sanchez, and M. Müller. pROC: an open-source package for R and S+ to analyze and compare ROC curves. *BMC Bioinformatics*, 12(77):1–8, 2011. URL <https://doi.org/10.1186/1471-2105-12-77>. [p61]
- M. X. Rodríguez-Álvarez and V. Inácio. ROCnReg: An R package for receiver operating characteristic curve inference with and without covariates. *The R Journal*, 13(1):525–555, 2021. URL <https://doi.org/10.32614/RJ-2021-066>. [p61]
- J. Shen, S. Wang, Y.-J. Zhang, M. Kappil, H.-C. Wu, M. G. Kibriya, Q. Wang, F. Jasmine, H. Ahsan, P.-H. Lee, et al. Genome-wide DNA methylation profiles in hepatocellular carcinoma. *Hepatology*, 55(6):1799–1808, 2012. URL <https://doi.org/10.1002/hep.25569>. [p64]
- J. Q. Su and J. S. Liu. Linear combinations of multiple diagnostic markers. *Journal of the American Statistical Association*, 88(424):1350–1355, 1993. URL <https://doi.org/10.1080/01621459.1993.10476417>. [p67]
- Y. Xie, C. Mueller, L. Yu, and W. Zhu. *animation: A Gallery of Animations in Statistics and Utilities to Create Animations*, 2021. URL <https://yihui.org/animation/>. R package version 2.7. [p76]
- W. Youden. Index for rating diagnostic tests. *Cancer*, 3(1):32–35, 1950. URL [https://doi.org/10.1002/1097-0142\(1950\)3:1<32::AID-CNCR2820030106>3.0.CO;2-3](https://doi.org/10.1002/1097-0142(1950)3:1<32::AID-CNCR2820030106>3.0.CO;2-3). [p65]
- X.-H. Zhou, D. K. McClish, and N. A. Obuchowski. *Statistical Methods in Diagnostic Medicine*, volume 414 of *Wiley Series in Probability and Statistics*. Wiley, 2002. URL <https://doi.org/10.1002/9780470317082>. [p65]
- K. H. Zou, W. J. Hall, and D. E. Shapiro. Smooth non-parametric receiver operating characteristic (ROC) curves for continuous diagnostic tests. *Statistics in Medicine*, 16(19):2143–2156, 1997. URL [https://doi.org/10.1002/\(SICI\)1097-0258\(19971015\)16:19<2143::AID-SIM655>3.0.CO;2-3](https://doi.org/10.1002/(SICI)1097-0258(19971015)16:19<2143::AID-SIM655>3.0.CO;2-3). [p65]

Sonia Pérez Fernández

Department of Statistics and Operations Research and Mathematics Didactics

University of Oviedo, Asturias, Spain

(ORCID: 0000-0002-2767-6399)

perezsonia@uniovi.es

Pablo Martínez Cambor

Department of Anesthesiology

Geisel School of Medicine at Dartmouth, New Hampshire, USA

and

Faculty of Health Sciences

Universidad Autónoma de Chile, Chile

(ORCID: 0000-0001-7845-3905)

Pablo.Martinez-Cambor@hitchcock.org

Norberto Corral Blanco

Department of Statistics and Operations Research and Mathematics Didactics

University of Oviedo, Asturias, Spain

(ORCID: 0000-0002-6962-6154)

norbert@uniovi.es

dtComb: A Comprehensive R Library and Web Tool for Combining Diagnostic Tests

S. İlayda Yerlitaş Taştan, Serra Bersan Gengeç, Necla Koçhan, Gözde Ertürk Zararsız, Selçuk Korkmaz and Gökmen Zararsız

Abstract

The combination of diagnostic tests has become a crucial area of research, aiming to improve the accuracy and robustness of medical diagnostics. While existing tools focus primarily on linear combination methods, there is a lack of comprehensive tools that integrate diverse methodologies. In this study, we present dtComb, a comprehensive R package and web tool designed to address the limitations of existing diagnostic test combination platforms. One of the unique contributions of dtComb is offering a range of 142 methods to combine two diagnostic tests, including linear, non-linear, machine learning algorithms, and mathematical operators. Another significant contribution of dtComb is its inclusion of advanced tools for ROC analysis, diagnostic performance metrics, and visual outputs such as sensitivity-specificity curves. Furthermore, dtComb offers classification functions for new observations, making it an easy-to-use tool for clinicians and researchers. The web-based version is also available at <https://biotools.erciyes.edu.tr/dtComb/> for non-R users, providing an intuitive interface for test combination and model training.

1 Introduction

A typical scenario often encountered in combining diagnostic tests is when the gold standard method combines two-category and two continuous diagnostic tests. In such cases, clinicians usually seek to compare these two diagnostic tests and improve the performance of these diagnostic test results by dividing the results into proportional results (Müller et al., 2019; Faria et al., 2016; Nyblom et al., 2006). However, this technique is straightforward and may not fully capture all potential interactions and relationships between the diagnostic tests. Linear combination methods have been developed to overcome such problems (Ertürk Zararsız, 2023).

Linear methods combine two diagnostic tests into a single score/index by assigning weights to each test, optimizing their performance in diagnosing the condition of interest (Neumann et al., 2023). Such methods improve accuracy by leveraging the strengths of both tests (Aznar-Gimeno et al., 2022; Bansal and Sullivan Pepe, 2013). For instance, Su and Liu (Su and Liu, 1993) found that Fisher's linear discriminant function generates a linear combination of markers with either proportional or disproportional covariance matrices, aiming to maximize sensitivity consistently across the entire selectivity spectrum under a multivariate normal distribution model. In contrast, another approach introduced by Pepe and Thomson (Pepe and Thompson, 2000) relies on ranking scores, eliminating the need for linear distributional assumptions when combining diagnostic tests. Despite the theoretical advances, when existing tools were examined, it was seen that they contained a limited number of methods. For instance, Kramar et al. developed a computer program called mROC that includes only the Su and Liu method (Kramar et al., 2001). Pérez-Fernández et al. presented a movieROC R package that includes methods such as Su and Liu, min-max, and logistic regression methods (Pérez-Fernández et al., 2021). An R package called maxmzpAUC that includes similar methods was developed by Yu and Park (Yu and Park, 2015).

On the other hand, non-linear approaches incorporating the non-linearity between the diagnostic tests have been developed and employed to integrate the diagnostic tests (Du et al., 2024; Ghosh and Chinnaiyan, 2005). These approaches incorporate the non-linear structure of tests into the model, which might improve the accuracy and reliability of the diagnosis. In contrast to some existing packages, which permit the use of non-linear

approaches such as splines¹, lasso² and ridge² regression, there is currently no package that employs these methods directly for combination and offers diagnostic performance. Machine-learning (ML) algorithms have recently been adopted to combine diagnostic tests (Ahsan et al., 2024; Sewak et al., 2024; Agarwal et al., 2023; Prinzi et al., 2023). Many publications/studies focus on implementing ML algorithms in diagnostic tests (Salvetat et al., 2022, 2024; Ganapathy et al., 2023; Alzyoud et al., 2024; Zararsiz et al., 2016). For instance, DeGroat et al. performed four different classification algorithms (Random Forest, Support Vector Machine, Extreme Gradient Boosting Decision Trees, and k-Nearest Neighbors) to combine markers for the diagnosis of cardiovascular disease (DeGroat et al., 2024). The results showed that patients with cardiovascular disease can be diagnosed with up to 96% accuracy using these ML techniques. There are numerous applications where ML methods can be implemented (`scikit-learn` (Pedregosa et al., 2011), `TensorFlow` (Abadi et al., 2015), `caret` (Kuhn, 2008)). The `caret` library is one of the most comprehensive tools developed in the R language (Kuhn, 2008). However, these are general tools developed only for ML algorithms and do not directly combine two diagnostic tests or provide diagnostic performance measures.

Apart from the aforementioned methods, several basic mathematical operations such as addition, multiplication, subtraction, and division can also be used to combine markers (Svart et al., 2024; Luo et al., 2024; Serban et al., 2024). For instance, addition can enhance diagnostic sensitivity by combining the effects of markers, whereas subtraction can more distinctly differentiate disease states by illustrating the variance across markers. On the other hand, there are several commercial (e.g. IBM SPSS, MedCalc, Stata, etc.) and open source (R) software packages (`ROCR` (Sing et al., 2005), (`pROC` (Robin et al., 2011), `PRROC` (Grau et al., 2015), `plotROC` (Sachs, 2017)) that researchers can use for Receiver operating characteristic (ROC) curve analysis. However, these tools are designed to perform a single marker ROC analysis. As a result, there is currently no software tool that covers almost all combination methods.

In this study, we developed `dtComb`, an R package encompassing nearly all existing combination approaches in the literature. `dtComb` has two key advantages, making it easy to apply and superior to the other packages: (1) it provides users with a comprehensive 142 methods, including linear and non-linear approaches, ML approaches and mathematical operators; (2) it produces turnkey solutions to users from the stage of uploading data to the stage of performing analyses, performance evaluation and reporting. Furthermore, it is the only package that illustrates linear approaches such as Minimax and Todor & Saplacan (Sameera et al., 2016; Todor et al., 2014). In addition, it allows for the classification of new, previously unseen observations using trained models. To our knowledge, no other tools were designed and developed to combine two diagnostic tests on a single platform with 142 different methods. In other words, `dtComb` has made more effective and robust combination methods ready for application instead of traditional approaches such as simple ratio-based methods. First, we review the theoretical basis of the related combination methods; then, we present an example implementation to demonstrate the applicability of the package. Finally, we present a user-friendly, up-to-date, and comprehensive web tool developed to facilitate `dtComb` for physicians and healthcare professionals who do not use the R programming language. The `dtComb` package is freely available on the CRAN network, the web application is freely available at <https://biotools.erciyes.edu.tr/dtComb/>, and all source code is available on GitHub³.

2 Material and methods

This section will provide an overview of the combination methods implemented in the literature. Before applying these methods, we will also discuss the standardization techniques

¹<https://cran.r-project.org/web/packages/splines/index.html>

²<https://cran.r-project.org/web/packages/glmnet/index.html>

³<https://github.com/gokmenzararsiz/dtComb>, https://github.com/gokmenzararsiz/dtComb_Shiny

available for the markers, the resampling methods during model training, and, ultimately, the metrics used to evaluate the model's performance.

2.1 Combination approaches

Linear combination methods

The `dtComb` package comprises eight distinct linear combination methods, which will be elaborated in this section. Before investigating these methods, we briefly introduce some notations which will be used throughout this section.

Notations:

Let $D_i, i = 1, 2, \dots, n_1$ be the marker values of the i th individual in the diseased group, where $D_i = (D_{i1}, D_{i2})$, and $H_j, j = 1, 2, \dots, n_2$ be the marker values of the j th individual in the healthy group, where $H_j = (H_{j1}, H_{j2})$. Let $x_{i1} = c(D_{i1}, H_{j1})$ be the values of the first marker, and $x_{i2} = c(D_{i2}, H_{j2})$ be values of the second marker for the i th individual $i = 1, 2, \dots, n$. Let $D_{i,min} = \min(D_{i1}, D_{i2})$, $D_{i,max} = \max(D_{i1}, D_{i2})$, $H_{j,min} = \min(H_{j1}, H_{j2})$, $H_{j,max} = \max(H_{j1}, H_{j2})$ and c_i be the resulting combination score of the i th individual.

- *Logistic regression:* Logistic regression is a statistical method used for binary classification. The logistic regression model estimates the probability of the binary outcome occurring based on the values of the independent variables. It is one of the most commonly applied methods in diagnostic tests, and it generates a linear combination of markers that can distinguish between control and diseased individuals. Logistic regression is generally less effective than normal-based discriminant analysis, like Su and Liu's multivariate normality-based method, when the normal assumption is met (Ruiz-Velasco, 1991; Efron, 1975). On the other hand, others have argued that logistic regression is more robust because it does not require any assumptions about the joint distribution of multiple markers (Cox and Snell, 1989). Therefore, it is essential to investigate the performance of linear combination methods derived from the logistic regression approach with non-normally distributed data.

The objective of the logistic regression model is to maximize the logistic likelihood function. In other words, the logistic likelihood function is maximized to estimate the logistic regression model coefficients.

$$c = \frac{\exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})}{1 + \exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})} \quad (1)$$

The logistic regression coefficients can provide the maximum likelihood estimation of the model, producing an easily interpretable value for distinguishing between the two groups.

- *Scoring based on logistic regression:* The method primarily uses a binary logistic regression model, with slight modifications to enhance the combination score. The regression coefficients, as predicted in Eq 1, are rounded to a user-specified number of decimal places and subsequently used to calculate the combination score (León et al., 2006).

$$c = \beta_1 x_{i1} + \beta_2 x_{i2} \quad (2)$$

- *Pepe & Thompson's method:* Pepe & Thompson have aimed to maximize the AUC or partial AUC to combine diagnostic tests, regardless of the distribution of markers (Pepe and Thompson, 2000). They developed an empirical solution of optimal linear combinations that maximize the Mann-Whitney U statistic, an empirical estimate of the ROC curve. Notably, this approach is distribution-free. Mathematically, they maximized the following objective function:

$$\text{maximize } U(a) = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} I [D_{i1} + \alpha D_{i2} \geq H_{j1} + \alpha H_{j2}] \quad (3)$$

$$c = x_{i1} + \alpha x_{i2} \quad (4)$$

where $\alpha \in [-1, 1]$ is interpreted as the relative weight of x_{i2} to x_{i1} in the combination, the weight of the second marker. This formula aims to find α to maximize $U(a)$. Readers are referred to see (Pepe and Thomson) (Pepe and Thompson, 2000).

- *Pepe, Cai & Langton's method:* Pepe et al. observed that when the disease status and the levels of markers conform to a generalized linear model, the regression coefficients represent the optimal linear combinations that maximize the area under the ROC curves (Pepe et al., 2006). The following objective function is maximized to achieve a higher AUC value:

$$\text{maximize } U(a) = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} I[D_{i1} + \alpha D_{i2} > H_{j1} + \alpha H_{j2}] + \frac{1}{2} I[D_{i1} + \alpha = H_{j1} + \alpha H_{j2}] \quad (5)$$

Before calculating the combination score using Eq 4, the marker values are normalized or scaled to be constrained within the scale of 0 to 1. In addition, it is noted that the estimate obtained by maximizing the empirical AUC can be considered as a particular case of the maximum rank correlation estimator from which the general asymptotic distribution theory was developed. Readers are referred to Pepe (2003, Chapters 4–6) for a review of the ROC curve approach and more details (Pepe, 2003).

- *Min-Max method:* The Pepe & Thomson method is straightforward if there are two markers. It is computationally challenging if we have more than two markers to be combined. To overcome the computational complexity issue of this method, Liu et al. (Liu et al., 2011) proposed a non-parametric approach that linearly combines the minimum and maximum values of the observed markers of each subject. This approach, which does not rely on the normality assumption of data distributions (i.e., distribution-free), is known as the Min-Max method and may provide higher sensitivity than any single marker. The objective function of the Min-Max method is as follows:

$$\text{maximize } U(a) = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} I[D_{i,max} + \alpha D_{i,min} > H_{j,max} + \alpha H_{j,min}] \quad (6)$$

$$c = x_{i,max} + \alpha x_{i,min} \quad (7)$$

where $x_{i,max} = \max(x_{i1}, x_{i2})$ and $x_{i,min} = \min(x_{i1}, x_{i2})$.

The Min-Max method aims to combine repeated measurements of a single marker over time or multiple markers that are measured with the same unit. While the Min-Max method is relatively simple to implement, it has some limitations. For example, markers may have different units of measurement, so standardization can be needed to ensure uniformity during the combination process. Furthermore, it is unclear whether all available information is fully utilized when combining markers, as this method incorporates only the markers' minimum and maximum values into the model (Kang et al., 2016).

- *Su & Liu's method:* Su and Liu examined the combination score separately under the assumption of two multivariate normal distributions when the covariance matrices were proportional or disproportionate (Su and Liu, 1993). Multivariate normal distributions with different covariances were first utilized in classification problems (Anderson and Bahadur, 1962). Then, Su and Liu also developed a linear combination method by extending the idea of using multivariate distributions to the AUC, showing that the best coefficients that maximize AUC are Fisher's discriminant coefficients. Assuming that $D \sim N(\mu_D, \Sigma_D)$ and $H \sim N(\mu_H, \Sigma_H)$ represent the multivariate normal distributions for the diseased and non-diseased groups, respectively. The Fisher's coefficients are as

follows:

$$(\alpha, \beta) = \left(\sum_D + \sum_H \right)^{-1} \mu \quad (8)$$

where $\mu = \mu_D - \mu_H$. The combination score in this case is:

$$c = \alpha x_{i1} + \beta x_{i2} \quad (9)$$

- *The Minimax method:* The Minimax method is an extension of Su & Liu's method (Sameera et al., 2016). Suppose that D follows a multivariate normal distribution $D \sim N(\mu_D, \Sigma_D)$, representing the diseased group, and H follows a multivariate normal distribution $H \sim N(\mu_H, \Sigma_H)$, representing the non-diseased group. Then Fisher's coefficients are as follows:

$$(\alpha, \beta) = \left[t \sum_D + (1-t) \sum_H \right]^{-1} (\mu_D - \mu_H) \quad (10)$$

Given these coefficients, the combination score is calculated using Eq 9. In this formula, t is a constant with values ranging from 0 to 1. This value can be hyper-tuned by maximizing the AUC.

- *Todor & Saplacan's method:* Todor and Saplacan's method uses the sine and cosine trigonometric functions to calculate the combination score (Todor et al., 2014). The combination score is calculated using $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, which maximizes the AUC within this interval. The formula for the combination score is given as follows:

$$c = \sin(\theta)x_{i1} + \cos(\theta)x_{i2} \quad (11)$$

Non-linear combination methods

In addition to linear combination methods, the `dtComb` package includes seven non-linear approaches, which will be discussed in this subsection. In this subsection, we will use the following notations: x_{ij} : the value of the j th marker for the i th individual, $i = 1, 2, \dots, n$ and $j = 1, 2, \dots, d$: degree of polynomial regressions and splines, $d = 1, 2, \dots, p$.

- *Logistic Regression with Polynomial Feature Space:* This approach extends the logistic regression model by adding extra predictors created by raising the original predictor variables to a certain power. This transformation enables the model to capture and model non-linear relationships in the data by including polynomial terms in the feature space (James et al., 2013). The combination score is calculated as follows:

$$c_i = \frac{\exp(\beta_0 + \beta_1 x_{ij} + \beta_2 x_{ij}^2 + \dots + \beta_p x_{ij}^p)}{1 + \exp(\beta_0 + \beta_1 x_{ij} + \beta_2 x_{ij}^2 + \dots + \beta_p x_{ij}^p)} \quad (12)$$

where c_i is the combination score for the i th individual and represents the posterior probabilities.

- *Ridge Regression with Polynomial Feature Space:* This method combines Ridge regression with expanded feature space created by adding polynomial terms to the original predictor variables. It is a widely used shrinkage method when we have multicollinearity between the variables, which may be an issue for least squares regression. This method aims to estimate the coefficients of these correlated variables by minimizing the residual sum of squares (RSS) while adding a term (referred to as a regularization term) to prevent overfitting. The objective function is based on the L2 norm of the coefficient vector, which prevents overfitting in the model (Eq 13). The Ridge estimate is defined

as follows:

$$\hat{\beta}^R = \operatorname{argmin}_{\beta} \text{RSS} + \lambda \sum_{j=1}^2 \sum_{d=1}^p \beta_j^d \quad (13)$$

where

$$\text{RSS} = \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^2 \sum_{d=1}^p \beta_j^d x_{ij}^d \right) \quad (14)$$

and $\hat{\beta}^R$ denotes the estimates of the coefficients of the Ridge regression, and the second term is called a penalty term where $\lambda \geq 0$ is a shrinkage parameter. The shrinkage parameter, λ , controls the amount of shrinkage applied to regression coefficients. A cross-validation is implemented to find the shrinkage parameter. We used the `glmnet` package (Friedman et al., 2010) to implement the Ridge regression in combining the diagnostic tests.

- *Lasso Regression with Polynomial Feature Space:* Similar to Ridge regression, Lasso regression is also a shrinkage method that adds a penalty term to the objective function of the least square regression. The objective function, in this case, is based on the L1 norm of the coefficient vector, which leads to the sparsity in the model. Some of the regression coefficients are precisely zero when the tuning parameter λ is sufficiently large. This property of the Lasso method allows the model to automatically identify and remove less relevant variables and reduce the algorithm's complexity. The Lasso estimates are defined as follows:

$$\hat{\beta}^L = \operatorname{argmin}_{\beta} \text{RSS} + \lambda \sum_{j=1}^2 \sum_{d=1}^p |\beta_j^d| \quad (15)$$

To implement the Lasso regression in combining the diagnostic tests, we used the `glmnet` package (Friedman et al., 2010).

- *Elastic-Net Regression with Polynomial Feature Space:* Elastic-Net Regression is a method that combines Lasso (L1 regularization) and Ridge (L2 regularization) penalties to address some of the limitations of each technique. The combination of the two penalties is controlled by two hyperparameters, $\alpha \in [0,1]$ and λ , which enable you to adjust the trade-off between the L1 and L2 regularization terms (James et al., 2013). For the implementation of the method, the `glmnet` package is used (Friedman et al., 2010).
- *Splines:* Another non-linear combination technique frequently applied in diagnostic tests is the splines. Splines are a versatile mathematical and computational technique that has a wide range of applications. These splines are piecewise functions that make interpolating or approximating data points possible. There are several types of splines, such as cubic splines. Smooth curves are created by approximating a set of control points using cubic polynomial functions. When implementing splines, two critical parameters come into play: degrees of freedom and the choice of polynomial degrees (i.e., degrees of the fitted polynomials). These user-adjustable parameters, which influence the flexibility and smoothness of the resulting curve, are critical for controlling the behavior of splines. We used the `splines` package in the R programming language to implement splines.
- *Generalized Additive Models with Smoothing Splines and Generalized Additive Models with Natural Cubic Splines:* Regression models are of great interest in many fields to understand the importance of different inputs. Even though regression is widely used, the traditional linear models often fail in real life as effects may not be linear. Another method called generalized additive models was introduced to identify and characterize non-linear regression (James et al., 2013). Smoothing Splines and Natural Cubic Splines are two standard methods used within GAMs to model non-linear relationships. To implement these two methods, we used the `gam` package in R (Hastie,

2015). The method of GAMs with Smoothing Splines is a more data-driven and adaptive approach where smoothing splines can automatically capture non-linear relationships without specifying the number of knots (specific points where two or more polynomial segments are joined together to create a piecewise-defined curve or surface) or the shape of the spline in advance. On the other hand, natural cubic splines are preferred when we have prior knowledge or assumptions about the shape of the non-linear relationship. Natural cubic splines are more interpretable and can be controlled by the number of knots (Elhakeem et al., 2022).

Mathematical Operators

This section will mention four arithmetic operators, eight distance measurements, and the exponential approach. Also, unlike other approaches, in this section, users can apply logarithmic, exponential, and trigonometric (sinus and cosine) transformations on the markers. Let x_{ij} represent the value of the j th variable for the i th observation, with $i = 1, 2, \dots, n$ and $j = 1, 2$. Let the resulting combination score for the i th individual be c_i .

- *Arithmetic Operators:* Arithmetic operators such as addition, multiplication, division, and subtraction can also be used in diagnostic tests to optimize the AUC, a measure of diagnostic test performance. These mathematical operations can potentially increase the AUC and improve the efficacy of diagnostic tests by combining markers in specific ways. For example, if high values in one test indicate risk, while low values in the other indicate risk, subtraction or division can effectively combine these markers.
- *Distance Measurements:* While combining markers with mathematical operators, a distance measure is used to evaluate the relationships or similarities between marker values. It's worth noting that, as far as we know, no studies have integrated various distinct distance measures with arithmetic operators in this context. Euclidean distance is the most commonly used distance measure, which may not accurately reflect the relationship between markers. Therefore, we incorporated a variety of distances into the package we developed. These distances are given as follows (Minaev et al., 2018; Pandit et al., 2011; Cha, 2007):

Euclidean:

$$c = \sqrt{(x_{i1} - 0)^2 + (x_{i2} - 0)^2} \quad (16)$$

Manhattan:

$$c = |x_{i1} - 0| + |x_{i2} - 0| \quad (17)$$

Chebyshev:

$$c = \max\{|x_{i1} - 0|, |x_{i2} - 0|\} \quad (18)$$

Kulczynskid:

$$c = \frac{|x_{i1} - 0| + |x_{i2} - 0|}{\min\{x_{i1}, x_{i2}\}} \quad (19)$$

Lorentzian:

$$c = \ln(1 + |x_{i1} - 0|) + \ln(1 + |x_{i2} - 0|) \quad (20)$$

Taneja:

$$c = z_1 \left(\log \left(\frac{z_1}{\sqrt{x_{i1}\epsilon}} \right) \right) + z_2 \left(\log \left(\frac{z_2}{\sqrt{x_{i2}\epsilon}} \right) \right) \quad (21)$$

where $z_1 = \frac{x_{i1}-0}{2}$, $z_2 = \frac{x_{i2}-0}{2}$

Kumar-Johnson:

$$c = \frac{(x_{i1}^2 - 0)^2}{2(x_{i1}\epsilon)^{\frac{3}{2}}} + \frac{(x_{i2}^2 - 0)^2}{2(x_{i2}\epsilon)^{\frac{3}{2}}}, \quad \epsilon = 0.0000 \quad (22)$$

Avg:

$$c = \frac{|x_{i1} - 0| + |x_{i2} - 0| + \max\{(x_{i1} - 0), (x_{i2} - 0)\}}{2} \quad (23)$$

- *Exponential approach:* The exponential approach is another technique to explore different relationships between the diagnostic measurements. The methods in which one of the two diagnostic tests is considered as the base and the other as an exponent can be represented as $x_{i1}^{(x_{i2})}$ and $x_{i2}^{(x_{i1})}$. The specific goals or hypothesis of the analysis, as well as the characteristics of the diagnostic tests, will determine which method to use.

Machine-Learning algorithms

Machine-learning algorithms have been increasingly implemented in various fields, including the medical field, to combine diagnostic tests. Integrating diagnostic tests through ML can lead to more accurate, timely, and personalized diagnoses, which are particularly valuable in complex medical cases where multiple factors must be considered. In this study, we aimed to incorporate almost all ML algorithms in the package we developed. We took advantage of the `caret` package in R (Kuhn, 2008) to achieve this goal. This package includes 190 classification algorithms that could be used to train models and make predictions. Our study focused on models that use numerical inputs and produce binary responses depending on the variables/features and the desired outcome. This selection process resulted in 113 models we further implemented in our study. We then classified these 113 models into five classes using the same idea given in (Zararsiz et al., 2016): (i) discriminant classifiers, (ii) decision tree models, (iii) kernel-based classifiers, (iv) ensemble classifiers, and (v) others. Like in the `caret` package, `m1Comb()` sets up a grid of tuning parameters for a number of classification routines, fits each model, and calculates a performance measure based on resampling. After the model fitting, it uses the `predict()` function to calculate the probability of the "event" occurring for each observation. Finally, it performs ROC analysis based on the probabilities obtained from the prediction step.

2.2 Standardization

Standardization is converting/transforming data into a standard scale to facilitate meaningful comparisons and statistical inference. Many statistical techniques frequently employ standardization to improve the interpretability and comparability of data. We implemented five different standardization methods that can be applied for each marker, the formulas of which are listed below:

- Z-score: $\frac{x - \text{mean}(x)}{\text{sd}(x)}$
- T-score: $\left(\frac{x - \text{mean}(x)}{\text{sd}(x)} \times 10 \right) + 50$
- min_max_scale: $\frac{x - \text{min}(x)}{\text{max}(x) - \text{min}(x)}$
- scale_mean_to_one: $\frac{x}{\text{mean}(x)}$
- scale_sd_to_one: $\frac{x}{\text{sd}(x)}$

2.3 Model building

After specifying a combination method from the `dtComb` package, users can build and optimize model parameters using functions like `mlComb()`, `linComb()`, `nonlinComb()`, and `mathComb()`, depending on the specific model selected. Parameter optimization is done using n-fold cross-validation, repeated n-fold cross-validation, and bootstrapping methods for linear and non-linear approaches (i.e., `linComb()`, `nonlinComb()`). Additionally, for machine-learning approaches (i.e., `mlComb()`), all of the resampling methods from the `caret` package are used to optimize the model parameters. The total number of parameters being optimized varies across models, and these parameters are fine-tuned to maximize the AUC. The returned object stores input data, preprocessed and transformed data, trained model, and resampling results.

2.4 Evaluation of model performances

A confusion matrix, as shown in Table 1, is a table used to evaluate the performance of a classification model and shows the number of correct and incorrect predictions. It compares predicted and actual

Table 1: Confusion Matrix

Predicted labels	Actual class labels		Total
	Positive	Negative	
Positive	TP	FP	TP+FP
Negative	FN	TN	FN+TN
Total	TP+FN	FP+TN	n

TP: True Positive, TN: True Negative, FP: False Positive, FN: False Negative, n: Sample size

class labels, with diagonal elements representing the correct predictions and off-diagonal elements representing the number of incorrect predictions. The `dtComb` package uses the `OptimalCutpoints` package (Yin and Tian, 2014) to generate the confusion matrix and then `epiR` (Stevenson et al., 2017), including different performance metrics, to evaluate the performances. Various performance metrics, accuracy rate (ACC), Kappa statistic (κ), sensitivity (SE), specificity (SP), apparent and true prevalence (AP, TP), positive and negative predictive values (PPV, NPV), positive and negative likelihood ratio (PLR, NLR), the proportion of true outcome negative subjects that test positive (False T+ proportion for true D-), the proportion of true outcome positive subjects that test negative (False T- proportion for true D+), the proportion of test-positive subjects that are outcome negative (False T+ proportion for T-), the proportion of test-negative subjects (False T- proportion for T-) that are outcome positive measures are available in the `dtComb` package. These metrics are summarized in Table 2.

2.5 Prediction of the test cases

The class labels of the observations in the test set are predicted with the model parameters derived from the training phase. It is critical to emphasize that the same analytical procedures employed during the training phase have also been applied to the test set, such as normalization, transformation, or standardization. More specifically, if the training set underwent Z-standardization, the test set would similarly be standardized using the mean and standard deviation derived from the training set. The class labels of the test set are then estimated based on the cut-off value established during the training phase and using the model's parameters that are trained using the training set.

Table 2: Performance Metrics and Formulas

Performance Metric	Formula
Accuracy	$ACC = \frac{TP+TN}{2}$
Kappa	$\kappa = \frac{ACC - P_e}{1 - P_e}$ $P_e = \frac{(TN+FN)(TP+FP)+(FP+TN)(FN+TN)}{n^2}$
Sensitivity (Recall)	$SE = \frac{TP}{TP+FN}$
Specificity	$SP = \frac{TN}{TN+FP}$
Apparent Prevalence	$AP = \frac{TP}{n} + \frac{FP}{n}$
True Prevalence	$TP = \frac{AP+SP-1}{SE+SP-1}$
Positive Predictive Value (Precision)	$PPV = \frac{TP}{TP+FP}$
Negative Predictive Value	$NPV = \frac{TN}{TN+FN}$
Positive Likelihood Ratio	$PLR = \frac{SE}{1-SP}$
Negative Likelihood Ratio	$NLR = \frac{1-SE}{SP}$
The Proportion of True Outcome Negative Subjects That Test Positive	$\frac{FP}{FP+TN}$
The Proportion of True Outcome Positive Subjects That Test Negative	$\frac{FN}{TP+FN}$
The Proportion of Test Positive Subjects That Are Outcome Negative	$\frac{FP}{TP+FN}$
The Proportion of Test Negative Subjects That Are Outcome Positive	$\frac{FN}{FN+TN}$

2.6 Technical details and the structure of dtComb

The `dtComb` package is implemented using the R programming language (<https://www.r-project.org/>) version 4.2.0. Package development was facilitated with `devtools` (Wickham et al., 2022) and documented with `roxygen2` (Wickham et al., 2024). Package testing was performed using 271 unit tests (Wickham, 2024). Double programming was performed using Python (<https://www.python.org/>) to validate the implemented functions (Shiralkar, 2010).

To combine diagnostic tests, the `dtComb` package allows the integration of eight linear combination methods, seven non-linear combination methods, arithmetic operators, and, in addition to these, eight distance metrics within the scope of mathematical operators and a total of 113 machine-learning algorithms from the `caret` package (Kuhn, 2008). These are summarized in Table 3.

Table 3: Features of dtComb

Modules (Tab Panels)	Features
Combination Methods	<ul style="list-style-type: none"> • Linear Combination Approach (8 Different methods) • Non-linear Combination Approach (7 Different Methods) • Mathematical Operators (14 Different methods) • Machine-Learning Algorithms (113 Different Methods) (Kuhn, 2008)
Preprocessing	<ul style="list-style-type: none"> • Five standardization methods applicable to linear, non-linear, mathematical methods • 16 preprocessing methods applicable to ML (Kuhn, 2008)
Resampling	<ul style="list-style-type: none"> • Three different methods for linear and non-linear combination methods <ul style="list-style-type: none"> – Bootstrapping – Cross-validation – Repeated cross-validation • 12 different resampling methods for ML (Kuhn, 2008)
Cutpoints	<ul style="list-style-type: none"> • 34 different methods for optimum cutpoints (Yin and Tian, 2014)

3 Results

Table 4 summarizes the existing packages and programs, including `dtComb`, along with the number of combination methods included in each package. While `mROC` offers only one linear combination method, `maxmzpAUC` and `movieROC` provide five linear combination techniques each, and `SLModels` includes four. However, these existing packages primarily focus on linear combination approaches. In contrast, `dtComb` goes beyond these limitations by integrating not only linear methods but also non-linear approaches, machine learning algorithms, and mathematical operators.

Table 4: Comparison of dtComb vs. existing packages and programs

Packages&Programs	Linear Comb.	Non-linear Comb.	Math. Operators	ML algorithms
<code>mROC</code> (Kramar et al., 2001)	1	-	-	-
<code>maxmzpAUC</code> (Yu and Park, 2015)	5	-	-	-
<code>movieROC</code> (Pérez-Fernández et al., 2021)	5	-	-	-
<code>SLModels</code> (Aznar-Gimeno et al., 2023)	4	-	-	-
<code>dtComb</code>	8	7	14	113

3.1 Dataset

To demonstrate the functionality of the `dtComb` package, we conduct a case study using four different combination methods. The data used in this study were obtained from patients who presented at Erciyes University Faculty of Medicine, Department of General Surgery, with complaints of abdominal pain (Zararsiz et al., 2016; Akyildiz et al., 2010). The dataset comprised D-dimer levels (*D_dimer*) and leukocyte counts (*log_leukocyte*) of 225 patients, divided into two groups (*Group*): the first group consisted of 110 patients who required an immediate laparotomy (*needed*). In comparison, the second group comprised 115 patients who did not (*not_needed*). After the evaluation of conventional treatment, the patients who underwent surgery due to their postoperative pathologies are placed in the first group. In contrast, those with a negative result from their laparotomy were assigned to the second group. All the analyses were performed by following a workflow given in Fig. 1. First of all, the `dtComb` package should be loaded in order to use related functions.

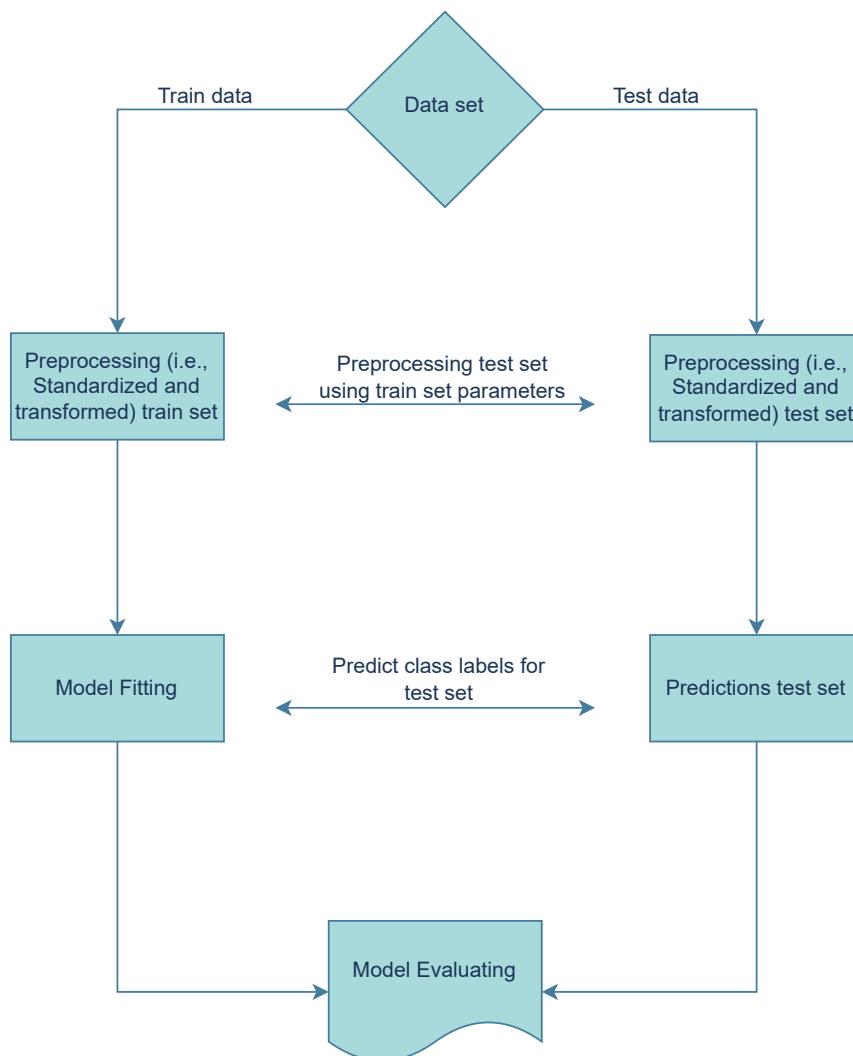


Figure 1: Combination steps of two diagnostic tests. The figure presents a schematic representation of the sequential steps involved in combining two diagnostic tests using a combination method.

```
# load dtComb package
library(dtComb)
```

Similarly, the laparotomy data can be loaded from the R database by using the following R code:

```
# load laparotomy data
data(laparotomy)
```

3.2 Implementation of the dtComb package

In order to demonstrate the applicability of the `dtComb` package, the implementation of an arbitrarily chosen method from each of the linear, non-linear, mathematical operator and machine learning approaches is demonstrated and their performance is compared. These methods are Pepe, Cai & Langton for linear combination, Splines for non-linear, Addition for mathematical operator and SVM for machine-learning. Before applying the methods, we split the data into two parts: a training set comprising 70% of the data and a test set comprising the remaining 30%.

```
# Splitting the data set into train and test (70%-30%)
set.seed(2128)
inTrain <- caret::createDataPartition(laparotomy$group, p = 0.7, list = FALSE)
trainData <- laparotomy[inTrain, ]
colnames(trainData) <- c("Group", "D_dimer", "log_leukocyte")
testData <- laparotomy[-inTrain, -1]

# define marker and status for combination function
markers <- trainData[, -1]
status <- factor(trainData$Group, levels = c("not_needed", "needed"))
```

The model is trained on `trainData` and the resampling parameters used in the training phase are chosen as ten repeat five fold repeated cross-validation. Direction = '`<`' is chosen, as higher values indicate higher risks. The Youden index was chosen among the cut-off methods. We note that markers are not standardised and results are presented at the confidence level (CI 95%). Four main combination functions are run with the selected methods as follows.

```
# PCL method
fit.lin.PCL <- linComb(markers = markers, status = status, event = "needed",
                         method = "PCL", resample = "repeatedcv", nfolds = 5,
                         nrepeats = 10, direction = "<", cutoff.method = "Youden")

# splines method (degree = 3 and degrees of freedom = 3)
fit.nonlin.splines <- nonlinComb(markers = markers, status = status, event = "needed",
                                    method = "splines", resample = "repeatedcv", nfolds = 5,
                                    nrepeats = 10, cutoff.method = "Youden", direction = "<",
                                    df1 = 3, df2 = 3)

#add operator
fit.add <- mathComb(markers = markers, status = status, event = "needed",
                      method = "add", direction = "<", cutoff.method = "Youden")

#SVM
fit.svm <- mlComb(markers = markers, status = status, event = "needed", method = "svmLinear",
                     resample = "repeatedcv", nfolds = 5, nrepeats = 10, direction = "<",
                     cutoff.method = "Youden")
```

Various measures were considered to compare model performances, including AUC, ACC, SEN, SPE, PPV, and NPV. AUC statistics, with 95% CI, have been calculated for each marker and method. The resulting statistics are as follows: 0.816 (0.751–0.880), 0.802 (0.728–0.877), 0.888 (0.825–0.930), 0.911 (0.868–0.954), 0.877 (0.824–0.929), and 0.875 (0.821–0.930) for D-dimer, Log(leukocyte), Pepe, Cai & Langton, Splines, Addition, and Support

Vector Machine (SVM). The results revealed that the predictive performances of markers and the combination of markers are significantly higher than random chance in determining the use of laparotomy ($p < 0.05$). The highest sensitivity and NPV were observed with the Addition method, while the highest specificity and PPV were observed with the Splines method. According to the overall AUC and accuracies, the combined approach fitted with the Splines method performed better than the other methods (Fig. 2). Therefore, the Splines method will be used in the subsequent analysis of the findings.

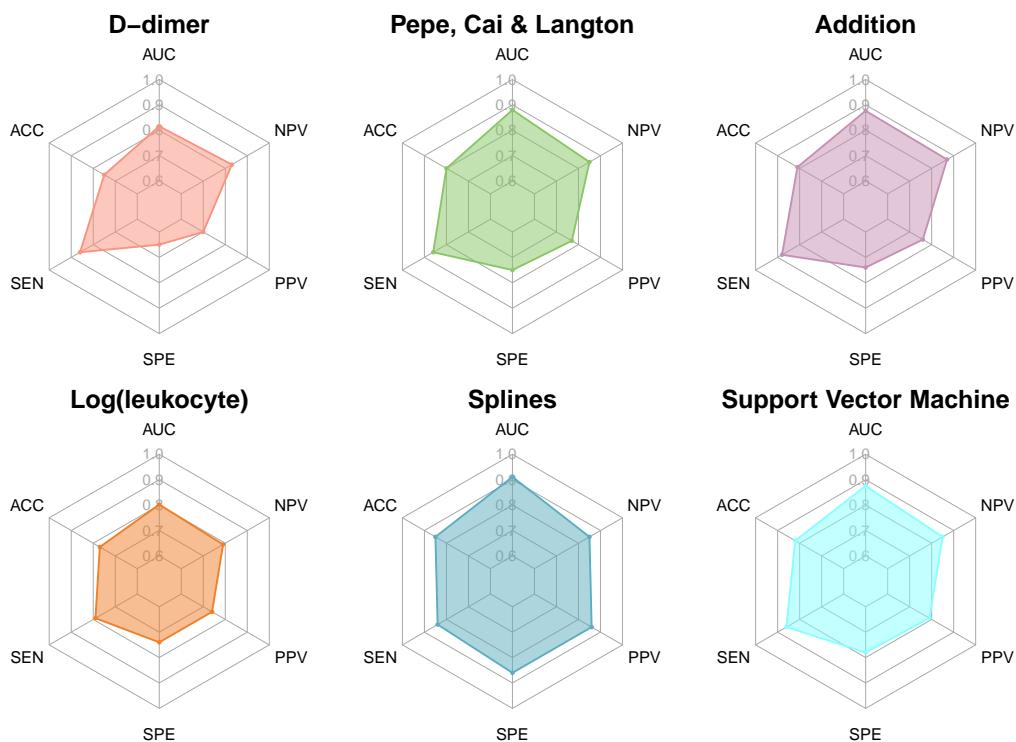


Figure 2: Radar plots of trained models and performance measures of two markers. Radar plots summarize the diagnostic performances of two markers and various combination methods in the training dataset. These plots illustrate the performance metrics such as AUC, ACC, SEN, SPE, PPV, and NPV measurements. In these plots, the width of the polygon formed by connecting each point indicates the model's performance in terms of AUC, ACC, SEN, SPE, PPV, and NPV metrics. It can be observed that the polygon associated with the Splines method occupies the most extensive area, which means that the Splines method performed better than the other methods.

For the AUC of markers and the spline model:

	AUC	SE.AUC	LowerLimit	UpperLimit	z	p.value
D_dimer	0.8156966	0.03303310	0.7509530	0.8804403	9.556979	1.212446e-21
log_leukocyte	0.8022286	0.03791768	0.7279113	0.8765459	7.970652	1.578391e-15
Combination	0.9111752	0.02189588	0.8682601	0.9540904	18.778659	1.128958e-78

Here:

SE: Standard Error.

The area under ROC curves for D-dimer levels and leukocyte counts on the logarithmic scale and combination score were 0.816, 0.802, and 0.911, respectively. The ROC curves generated with the combination score from the splines model, D-dimer levels, and leukocyte count markers are also given in Fig. 3, showing that the combination score has the highest AUC. It is observed that the splines method significantly improved between 9.5% and 10.9% in AUC statistics compared to D-dimer level and leukocyte counts, respectively.

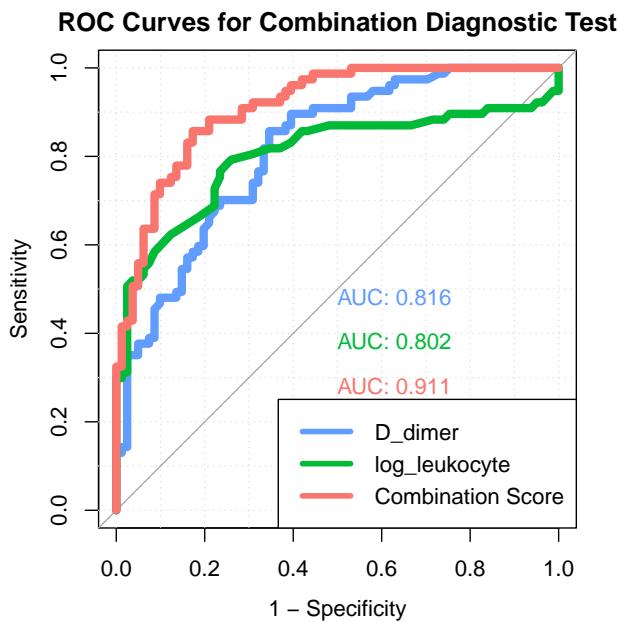


Figure 3: ROC curves. ROC curves for combined diagnostic tests, with sensitivity displayed on the y-axis and 1-specificity displayed on the x-axis. As can be observed, the combination score produced the highest AUC value, indicating that the combined strategy performs the best overall.

To see the results of the binary comparison between the combination score and markers:

```
fit.nonlin.splines$MultComp_table
```

Marker1 (A)	Marker2 (B)	AUC (A)	AUC (B)	A-B	SE(A-B)	z	p-value
1 Combination	D_dimer	0.9079686	0.8156966	0.09227193	0.02223904	4.1490971	3.337893e-05
2 Combination	log_leukocyte	0.9079686	0.8022286	0.10573994	0.03466544	3.0502981	2.286144e-03
3 D_dimer	log_leukocyte	0.8156966	0.8022286	0.01346801	0.04847560	0.2778308	7.811423e-01

Controlling Type I error using Bonferroni correction, comparison of combination score with markers yielded significant results ($p < 0.05$).

To demonstrate the diagnostic test results and performance measures for non-linear combination approach, the following code can be used:

```
fit.nonlin.splines$DiagStatCombined
    Outcome +    Outcome -    Total
Test +        66          13        79
Test -        11          68        79
Total         77          81       158
```

Point estimates and 95% CIs:

Apparent prevalence *	0.50 (0.42, 0.58)
True prevalence *	0.49 (0.41, 0.57)
Sensitivity *	0.86 (0.76, 0.93)
Specificity *	0.84 (0.74, 0.91)
Positive predictive value *	0.84 (0.74, 0.91)
Negative predictive value *	0.86 (0.76, 0.93)
Positive likelihood ratio	5.34 (3.22, 8.86)
Negative likelihood ratio	0.17 (0.10, 0.30)
False T+ proportion for true D- *	0.16 (0.09, 0.26)
False T- proportion for true D+ *	0.14 (0.07, 0.24)

False T+ proportion for T+ *	0.16 (0.09, 0.26)
False T- proportion for T- *	0.14 (0.07, 0.24)
Correctly classified proportion *	0.85 (0.78, 0.90)

* Exact CIs

Furthermore, if the diagnostic test results and performance measures of the combination score are compared with the results of the single markers, it can be observed that the TN value of the combination score is higher than that of the single markers, and the combination of markers has higher specificity and positive-negative predictive value than the log-transformed leukocyte counts and D-dimer level (Table 5). Conversely, D-dimer has a higher sensitivity than the others. Optimal cut-off values for both markers and the combined approach are also given in this table.

Table 5: Statistical diagnostic measures with 95% confidence intervals for each marker and the combination score

Diagnostic Measures (95% CI)	D-dimer level (> 1.6)	Log(leukocyte count) (> 4.16)	Combination score (> 0.448)
TP	66	61	65
TN	53	60	69
FP	28	21	12
FN	11	16	12
Apparent prevalence	0.59 (0.51-0.67)	0.52 (0.44-0.60)	0.49 (0.41-0.57)
True prevalence	0.49 (0.41-0.57)	0.49 (0.41-0.57)	0.49 (0.41-0.57)
Sensitivity	0.86 (0.76-0.93)	0.79 (0.68-0.88)	0.84 (0.74-0.92)
Specificity	0.65 (0.54-0.76)	0.74 (0.63-0.83)	0.85 (0.76-0.92)
Positive predictive value	0.70 (0.60-0.79)	0.74 (0.64-0.83)	0.84 (0.74-0.92)
Negative predictive value	0.83 (0.71-0.91)	0.79 (0.68-0.87)	0.85 (0.76-0.92)
Positive likelihood ratio	2.48 (1.81-3.39)	3.06 (2.08-4.49)	5.70 (3.35-9.69)
Negative likelihood ratio	0.22 (0.12-0.39)	0.28 (0.18-0.44)	0.18 (0.11-0.31)
False T+ proportion for true D-	0.35 (0.24-0.46)	0.26 (0.17-0.37)	0.15 (0.08-0.24)
False T- proportion for true D+	0.14 (0.07-0.24)	0.21 (0.12-0.32)	0.16 (0.08-0.26)
False T+ proportion for T+	0.30 (0.21-0.40)	0.26 (0.17-0.36)	0.16 (0.08-0.26)
False T- proportion for T-	0.17 (0.09-0.29)	0.21 (0.13-0.32)	0.15 (0.08-0.24)
Accuracy	0.75 (0.68-0.82)	0.77 (0.69-0.83)	0.85 (0.78-0.90)

For a comprehensive analysis, the `plotComb` function in `dtComb` can be used to generate plots of the kernel density and individual-value of combination scores of each group and the specificity and sensitivity corresponding to different cut-off point values Fig. 4. This function requires the result of the `nonlinComb` function, which is an object of the “`dtComb`” class and status which is of factor type.

```
# draw distribution, dispersion, and specificity and sensitivity plots
plotComb(fit.nonlin.splines, status)
```

If the model trained with Splines is to be tested, the generically written `predict` function is used. This function requires the test set and the result of the `nonlinComb` function, which is an object of the “`dtComb`” class. As a result of prediction, the output for each observation consisted of the combination score and the predicted label determined by the cut-off value derived from the model.

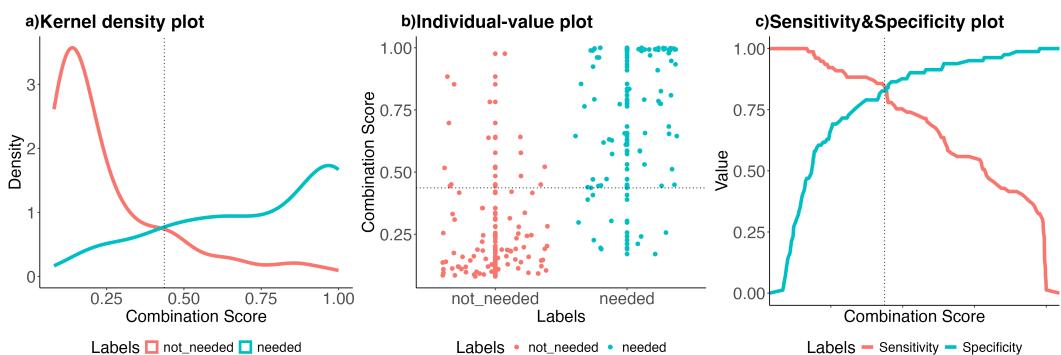


Figure 4: Kernel density, individual-value, and sens&spe plots of the combination score acquired with the training model. Kernel density of the combination score for two groups: needed and not needed (a). Individual-value graph with classes on the x-axis and combination score on the y-axis (b). Sensitivity and specificity graph of the combination score c. While colors show each class in Figures (a) and (b), in Figure (c), the colors represent the sensitivity and specificity of the combination score.

```
# To predict the test set
pred <- predict(fit.nonlin.splines, testData)
head(pred)
```

```
comb.score labels
1 0.6133884 needed
7 0.9946474 needed
10 0.9972347 needed
11 0.9925040 needed
13 0.9257699 needed
14 0.9847090 needed
```

Above, it can be seen that the estimated combination scores for the first six observations in the test set were labelled as **needed** because they were higher than the cut-off value of 0.448.

3.3 Web interface for the `dtComb` package

The primary goal of developing the `dtComb` package is to combine numerous distinct combination methods and make them easily accessible to researchers. Furthermore, the package includes diagnostic statistics and visualization tools for diagnostic tests and the combination score generated by the chosen method. Nevertheless, it is worth noting that using R code may pose challenges for physicians and those unfamiliar with R programming. We have also developed a user-friendly web application for `dtComb` using `Shiny` (Chang et al., 2024) to address this. This web-based tool is publicly accessible and provides an interactive interface with all the functionalities found in the `dtComb` package.

To initiate the analysis, users must upload their data by following the instructions outlined in the "Data upload" tab of the web tool. For convenience, we have provided three example datasets on this page to assist researchers in practicing the tool's functionality and to guide them in formatting their own data (as illustrated in Fig. 5a). We also note that ROC analysis for a single marker can be performed within the 'ROC Analysis for Single Marker(s)' tab in the data upload section of the web interface.

In the "Analysis" tab, one can find two crucial subpanels:

- Plots (Fig. 5b): This section offers various visual representations, such as ROC curves, kernel density plots, individual-value plots, and sensitivity and specificity plots. These visualizations help users assess single diagnostic tests and the combination score generated using user-defined combination methods.

- Results (Fig. 5c): In this subpanel, one can access a range of statistics. It provides insights into the combination score and single diagnostic tests, AUC statistics, and comparisons to evaluate how the combination score fares against individual diagnostic tests, and various diagnostic measures. One can also predict new data based on the model parameters set previously and stored in the "Predict" tab (Fig. 5d). If needed, one can download the model created during the analysis to keep the parameters of the fitted model. This lets users make new predictions by reloading the model from the "Predict" tab. Additionally, all the results can easily be downloaded using the dedicated download buttons in their respective tabs.

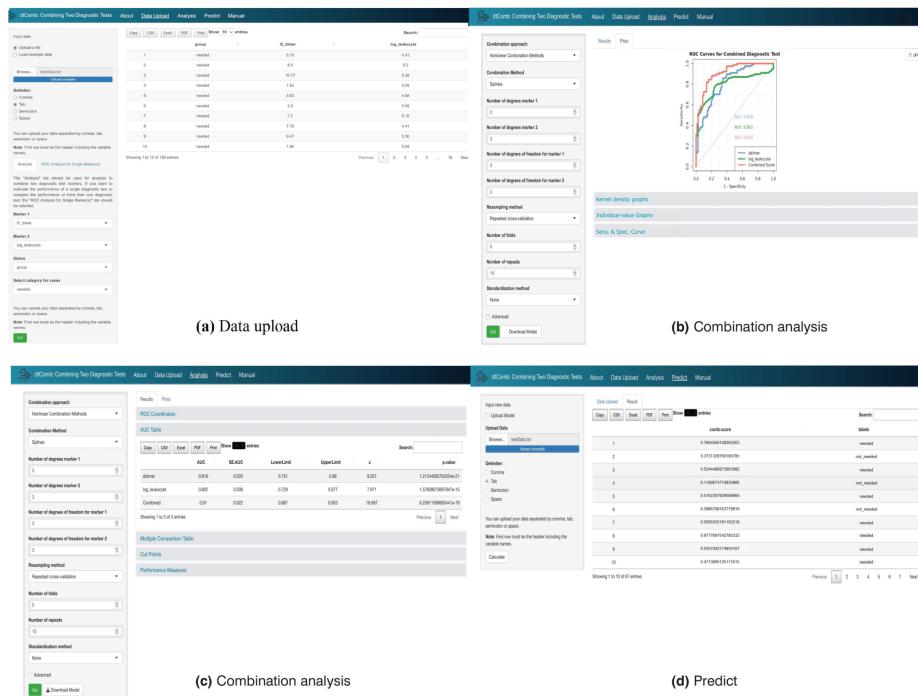


Figure 5: Web interface of the `dtComb` package. The figure illustrates the web interface of the `dtComb` package, which demonstrates the steps involved in combining two diagnostic tests. a) Data Upload: The user is able to upload the dataset and select relevant markers, a gold standard test, and an event factor for analysis. b) Combination Analysis: This panel allows the selection of the combination method, method-specific parameters, and resampling options to refine the analysis. c) Combination Analysis Output: Displays the results generated by the selected combination method, providing the user with key metrics and visualizations for interpretation. d) Predict: Displays the prediction results of the trained model when applied to the test set.

4 Summary and further research

In clinical practice, multiple diagnostic tests are possible for disease diagnosis (Yu and Park, 2015). Combining these tests to enhance diagnostic accuracy is a widely accepted approach (Su and Liu, 1993; Pepe and Thompson, 2000; Liu et al., 2011; Sameera et al., 2016; Pepe et al., 2006; Todor et al., 2014). As far as we know, the tools in Table 4 have been designed to combine diagnostic tests but only contain at most five different combination methods. As a result, despite the existence of numerous advanced combination methods, there is currently no extensive tool available for integrating diagnostic tests.

In this study, we presented `dtComb`, a comprehensive R package designed to combine diagnostic tests using various methods, including linear, non-linear, mathematical operators, and machine learning algorithms. The package integrates 142 different methods for combining two diagnostic markers to improve the accuracy of diagnosis. The package also provides ROC curve analysis, various graphical approaches, diagnostic performance scores, and

binary comparison results. In the given example, one can determine whether patients with abdominal pain require laparotomy by combining the D-dimer levels and white blood cell counts of those patients. Various methods, such as linear and non-linear combinations, were tested, and the results showed that the Splines method performed better than the others, particularly in terms of AUC and accuracy compared to single tests. This shows that diagnostic accuracy can be improved with combination methods.

Future work can focus on extending the capabilities of the `dtComb` package. While some studies focus on combining multiple markers, our study aimed to combine two markers using nearly all existing methods and develop a tool and package for clinical practice (Kang et al., 2016).

4.1 R Software

The R package `dtComb` is now available on the CRAN website <https://cran.r-project.org/web/packages/dtComb/index.html>.

4.2 Acknowledgment

We would like to thank the Proofreading & Editing Office of the Dean for Research at Erciyes University for the copyediting and proofreading service for this manuscript.

References

- M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <https://www.tensorflow.org/>. Software available from tensorflow.org. [p81]
- S. Agarwal, A. S. Yadav, V. Dinesh, K. S. S. Vatsav, K. S. S. Prakash, and S. Jaiswal. By artificial intelligence algorithms and machine learning models to diagnosis cancer. *Materials Today: Proceedings*, 80:2969–2975, 2023. [p81]
- M. Ahsan, A. Khan, K. R. Khan, B. B. Sinha, and A. Sharma. Advancements in medical diagnosis and treatment through machine learning: A review. *Expert Systems*, 41(3):e13499, 2024. [p81]
- H. Y. Akyildiz, E. Sozuer, A. Akcan, C. Kuçuk, T. Artis, İ. Biri, and N. Yılmaz. The value of D-dimer test in the diagnosis of patients with nontraumatic acute abdomen. *Turkish Journal of Trauma and Emergency Surgery*, 16(1):22–26, 2010. [p91]
- M. Alzyoud, R. Alzaidah, M. Aljaidi, G. Samara, M. Qasem, M. Khalid, and N. Al-Shanableh. Diagnosing diabetes mellitus using machine learning techniques. *International Journal of Data and Network Science*, 8(1):179–188, 2024. [p81]
- T. W. Anderson and R. R. Bahadur. Classification into two multivariate normal distributions with different covariance matrices. *The annals of mathematical statistics*, pages 420–431, 1962. [p83]
- R. Aznar-Gimeno, L. M. Esteban, R. del Hoyo-Alonso, Á. Borque-Fernando, and G. Sanz. A stepwise algorithm for linearly combining biomarkers under Youden index maximization. *Mathematics*, 10(8):1221, 2022. [p80]
- R. Aznar-Gimeno, L. Esteban, G. Sanz, and R. del Hoyo-Alonso. Comparing the min–max–median/IQR approach with the min–max approach, logistic regression and XGBoost,

- maximising the Youden index. *Symmetry (Basel)*, 15, 2023. doi: 10.3390/sym15030756. URL <https://doi.org/10.3390/sym15030756>. [p⁹⁰]
- A. Bansal and M. Sullivan Pepe. When does combining markers improve classification performance and what are implications for practice? *Statistics in medicine*, 32(11):1877–1892, 2013. [p⁸⁰]
- S.-H. Cha. Comprehensive survey on distance/similarity measures between probability density functions. *City*, 1(2):1, 2007. [p⁸⁶]
- W. Chang, J. Cheng, J. Allaire, C. Sievert, B. Schloerke, Y. Xie, J. Allen, J. McPherson, A. Dipert, and B. Borges. Shiny: Web application framework for R. 2024. URL <https://shiny.posit.co/>. R package version 1.9.1.9000, <https://github.com/rstudio/shiny>. [p⁹⁶]
- D. Cox and E. Snell. *Analysis of Binary Data*. Chapman and Hall/CRC, London, 2nd edition, 1989. [p⁸²]
- W. DeGroat, H. Abdelhalim, K. Patel, D. Mendhe, S. Zeeshan, and Z. Ahmed. Discovering biomarkers associated and predicting cardiovascular disease with high accuracy using a novel nexus of machine learning techniques for precision medicine. *Scientific reports*, 14(1):1, 2024. [p⁸¹]
- Z. Du, P. Du, and A. Liu. Likelihood ratio combination of multiple biomarkers via smoothing spline estimated densities. *Statistics in Medicine*, 43(7):1372–1383, 2024. [p⁸⁰]
- B. Efron. The efficiency of logistic regression compared to normal discriminant analysis. *Journal of the American Statistical Association*, 70(352):892–898, 1975. [p⁸²]
- A. Elhakeem, R. A. Hughes, K. Tilling, D. L. Cousminer, S. A. Jackowski, T. J. Cole, A. S. Kwong, Z. Li, S. F. Grant, A. D. Baxter-Jones, et al. Using linear and natural cubic splines, SITAR, and latent trajectory models to characterise nonlinear longitudinal growth trajectories in cohort studies. *BMC Medical Research Methodology*, 22(1):68, 2022. [p⁸⁶]
- G. Ertürk Zararsız. Linear combination of leukocyte count and D-Dimer levels in the diagnosis of patients with non-traumatic acute abdomen. *Med. Rec.*, 5:84–90, 2023. doi: 10.37990/medr.1166531. URL <https://doi.org/10.37990/medr.1166531>. [p⁸⁰]
- S. S. Faria, P. C. Fernandes Jr, M. J. B. Silva, V. C. Lima, W. Fontes, R. Freitas-Junior, A. K. Eterovic, and P. Forget. The neutrophil-to-lymphocyte ratio: a narrative review. *ecancermedicalscience*, 10, 2016. [p⁸⁰]
- J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010. [p⁸⁵]
- S. Ganapathy, H. KT, B. Jindal, P. S. Naik, and S. Nair N. Comparison of diagnostic accuracy of models combining the renal biomarkers in predicting renal scarring in pediatric population with vesicoureteral reflux (VUR). *Irish Journal of Medical Science (1971-)*, 192(5): 2521–2526, 2023. [p⁸¹]
- D. Ghosh and A. M. Chinnaian. Classification and selection of biomarkers in genomic data using LASSO. *BioMed Research International*, 2005(2):147–154, 2005. [p⁸⁰]
- J. Grau, I. Grosse, and J. Keilwagen. Prroc: computing and visualizing precision-recall and receiver operating characteristic curves in R. *Bioinformatics*, 31(15):2595–2597, 2015. [p⁸¹]
- T. Hastie. gam: Generalized additive models. 2015. URL <https://CRAN.R-project.org/package=gam>. R package version 1.22-5. [p⁸⁵]
- G. James, D. Witten, T. Hastie, R. Tibshirani, et al. *An introduction to statistical learning*, volume 112. Springer, 2013. [p^{84, 85}]

- L. Kang, A. Liu, and L. Tian. Linear combination methods to improve diagnostic/prognostic accuracy on future observations. *Statistical methods in medical research*, 25(4):1359–1380, 2016. [p83, 98]
- A. Kramar, D. Faraggi, A. Fortuné, and B. Reiser. mroc: a computer program for combining tumour markers in predicting disease states. *Computer methods and programs in biomedicine*, 66(2-3):199–207, 2001. [p80, 90]
- M. Kuhn. Building predictive models in R using the caret package. *Journal of statistical software*, 28:1–26, 2008. [p81, 87, 90]
- C. León, S. Ruiz-Santana, P. Saavedra, B. Almirante, J. Nolla-Salas, F. Álvarez-Lerma, J. Garnacho-Montero, M. Á. León, E. S. Group, et al. A bedside scoring system (“Candida score”) for early antifungal treatment in nonneutropenic critically ill patients with Candida colonization. *Critical care medicine*, 34(3):730–737, 2006. [p82]
- C. Liu, A. Liu, and S. Halabi. A min–max combination of biomarkers to improve diagnostic accuracy. *Statistics in medicine*, 30(16):2005–2014, 2011. [p83, 97]
- J. Luo, F. Yu, H. Zhou, X. Wu, Q. Zhou, Q. Liu, and S. Gan. AST/ALT ratio is an independent risk factor for diabetic retinopathy: A cross-sectional study. *Medicine*, 103(26):e38583, 2024. [p81]
- G. Minaev, R. Piché, and A. Visa. Distance measures for classification of numerical features. 2018. URL <https://trepo.tuni.fi/handle/10024/124353>. [p86]
- E. G. Müller, T. H. Edwin, C. Stokke, S. S. Nixelsaker, A. Babovic, N. Bogdanovic, A. B. Knapskog, and M. E. Revheim. Amyloid- β PET—correlation with cerebrospinal fluid biomarkers and prediction of Alzheimer’s disease diagnosis in a memory clinic. *PloS one*, 14(8):e0221365, 2019. [p80]
- M. Neumann, H. Kothare, and V. Ramanarayanan. Combining multiple multimodal speech features into an interpretable index score for capturing disease progression in Amyotrophic Lateral Sclerosis. *Interspeech*, 2353, 2023. [p80]
- H. Nyblom, E. Björnsson, M. Simrén, F. Aldenborg, S. Almer, and R. Olsson. The AST/ALT ratio as an indicator of cirrhosis in patients with PBC. *Liver International*, 26(7):840–845, 2006. [p80]
- S. Pandit, S. Gupta, et al. A comparative study on distance measuring approaches for clustering. *International journal of research in computer science*, 2(1):29–31, 2011. [p86]
- F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, 12:2825–2830, 2011. [p81]
- M. S. Pepe. *The statistical evaluation of medical tests for classification and prediction*. Oxford university press, 2003. [p83]
- M. S. Pepe and M. L. Thompson. Combining diagnostic test results to increase accuracy. *Biostatistics*, 1(2):123–140, 2000. [p80, 82, 83, 97]
- M. S. Pepe, T. Cai, and G. Longton. Combining predictors for classification using the area under the receiver operating characteristic curve. *Biometrics*, 62(1):221–229, 2006. [p83, 97]
- S. Pérez-Fernández, P. Martínez-Camblor, P. Filzmoser, and N. Corral. Visualizing the decision rules behind the ROC curves: understanding the classification process. *AStA Advances in Statistical Analysis*, 105(1):135–161, 2021. [p80, 90]
- F. Prinzi, C. Militello, N. Scichilone, S. Gaglio, and S. Vitabile. Explainable machine-learning models for COVID-19 prognosis prediction using clinical, laboratory and radiomic features. *IEEE Access*, 11:121492–121510, 2023. [p81]

- X. Robin, N. Turck, A. Hainard, N. Tiberti, F. Lisacek, J.-C. Sanchez, and M. Müller. proc: an open-source package for R and S+ to analyze and compare ROC curves. *BMC bioinformatics*, 12:1–8, 2011. [p81]
- S. Ruiz-Velasco. Asymptotic efficiency of logistic regression relative to linear discriminant analysis. *Biometrika*, 78(2):235–243, 1991. [p82]
- M. C. Sachs. plotroc: a tool for plotting ROC curves. *Journal of statistical software*, 79, 2017. [p81]
- N. Salvetat, F. J. Checa-Robles, V. Patel, C. Cayzac, B. Dubuc, F. Chimienti, J.-D. Abraham, P. Dupré, D. Vetter, S. Méreze, et al. A game changer for bipolar disorder diagnosis using rna editing-based biomarkers. *Translational Psychiatry*, 12(1):182, 2022. [p81]
- N. Salvetat, F. J. Checa-Robles, A. Delacrétaiz, C. Cayzac, B. Dubuc, D. Vetter, J. Dainat, J.-P. Lang, F. Gamma, and D. Weissmann. Ai algorithm combined with rna editing-based blood biomarkers to discriminate bipolar from major depressive disorders in an external validation multicentric cohort. *Journal of Affective Disorders*, 356:385–393, 2024. [p81]
- G. Sameera, R. V. Vardhan, and K. Sarma. Binary classification using multivariate receiver operating characteristic curve for continuous data. *Journal of biopharmaceutical statistics*, 26(3):421–431, 2016. [p81, 84, 97]
- D. Serban, N. Papanas, A. M. Dascalu, P. Kempler, I. Raz, A. A. Rizvi, M. Rizzo, C. Tudor, M. Silviu Tudosie, D. Tanasescu, et al. Significance of neutrophil to lymphocyte ratio (NLR) and platelet lymphocyte ratio (PLR) in diabetic foot ulcer and potential new therapeutic targets. *The International Journal of Lower Extremity Wounds*, 23(2):205–216, 2024. [p81]
- A. Sewak, S. Siegfried, and T. Hothorn. Construction and evaluation of optimal diagnostic tests with application to hepatocellular carcinoma diagnosis. *arXiv preprint arXiv:2402.03004*, 2024. [p81]
- P. Shiralkar. Programming validation: Perspectives and strategies. *PharmaSUG 2010—paper IB09*, 2010. [p89]
- T. Sing, O. Sander, N. Beerenwinkel, and T. Lengauer. Rocr: visualizing classifier performance in R. *Bioinformatics*, 21(20):3940–3941, 2005. [p81]
- M. Stevenson, T. Nunes, C. Heuer, J. Marshall, J. Sanchez, R. Thornton, J. Reiczigel, J. Robison-Cox, P. Sebastiani, P. Solymos, et al. epiR: tools for the analysis of epidemiological data. 2017. URL <https://cran.r-project.org/web/packages/epiR/index.html>. R package version 2.0.76. [p88]
- J. Q. Su and J. S. Liu. Linear combinations of multiple diagnostic markers. *Journal of the American Statistical Association*, 88(424):1350–1355, 1993. [p80, 83, 97]
- K. Svart, J. J. Korsbæk, R. H. Jensen, T. Parkner, C. S. Knudsen, S. G. Hasselbalch, S. M. Hagen, E. A. Wibroe, L. D. Molander, and D. Beier. Neurofilament light chain is elevated in patients with newly diagnosed idiopathic intracranial hypertension: A prospective study. *Cephalgia*, 44(5):03331024241248203, 2024. [p81]
- N. Todor, I. Todor, and G. Săplăcan. Tools to identify linear combination of prognostic factors which maximizes area under receiver operator curve. *Journal of clinical bioinformatics*, 4:1–7, 2014. [p81, 84, 97]
- H. Wickham. Testthat: Get started with testing. 2024. URL <https://cran.r-project.org/web/packages/testthat/index.html>. R package version 3.2.1.1. [p89]
- H. Wickham, J. Hester, W. Chang, and J. Bryan. Devtools: Tools to make developing R packages easier. 2022. URL <https://cran.r-project.org/web/packages/devtools/index.html>. R package version 2.4.5. [p89]

- H. Wickham, P. Danenberg, and M. Eugster. roxygen2: In-source documentation for R. 2024. URL <https://cran.r-project.org/web/packages/roxygen2/index.html>. R package version 7.3.2. [p89]
- J. Yin and L. Tian. Optimal linear combinations of multiple diagnostic biomarkers based on Youden index. *Statistics in medicine*, 33(8):1426–1440, 2014. [p88, 90]
- W. Yu and T. Park. Two simple algorithms on linear combination of multiple biomarkers to maximize partial area under the ROC curve. *Computational Statistics & Data Analysis*, 88:15–27, 2015. [p80, 90, 97]
- G. Zararsiz, H. Y. Akyildiz, D. GÖKSÜLÜK, S. Korkmaz, and A. ÖZTÜRK. Statistical learning approaches in diagnosing patients with nontraumatic acute abdomen. *Turkish Journal of Electrical Engineering and Computer Sciences*, 24(5):3685–3697, 2016. [p81, 87, 91]

S. İlayda Yerlitaş Taştan
Department of Biostatistics
Erciyes University
Türkiye
(ORCID: 0000-0003-2830-3006)
ilaydayerlitas340@gmail.com

Serra Bersan Gengeç
Department of Biostatistics
Erciyes University
Türkiye
serrabersan@gmail.com

Necla Koçhan
Department of Mathematics
Izmir University of Economics
Türkiye
(ORCID: 0000-0003-2355-4826)
necla.kayaalp@gmail.com

Gözde Ertürk Zararsız
Department of Biostatistics
Erciyes University
Türkiye
(ORCID if desired)
gozdeerturk9@gmail.com

Selçuk Korkmaz
Department of Biostatistics
Trakya University
Türkiye
(ORCID if desired)
selcukkorkmaz@gmail.com

Gökmen Zararsız
Department of Biostatistics
Erciyes University
Türkiye
(ORCID: 0000-0001-5801-1835)
gokmen.zararsiz@gmail.com

drclust: An R Package for Simultaneous Clustering and Dimensionality Reduction

by Ionel Prunila and Maurizio Vichi

Abstract The primary objective of simultaneous methodologies for clustering and variable reduction is to identify both the optimal partition of units and the optimal subspace of variables, all at once. The optimality is typically determined using least squares or maximum likelihood estimation methods. These simultaneous techniques are particularly useful when working with Big Data, where the reduction (synthesis) is essential for both units and variables. Furthermore, a secondary objective of reducing variables through a subspace is to enhance the interpretability of the latent variables identified by the subspace using specific methodologies. The drclust package implements double K-means (KM), reduced KM, and factorial KM to address the primary objective. KM with disjoint principal components addresses both the primary and secondary objectives, while disjoint principal component analysis and disjoint factor analysis address the latter, producing the sparsest loading matrix. The models are implemented in C++ for faster execution, processing large data matrices in a reasonable amount of time.

1 Introduction

Cluster analysis is the process of identifying homogeneous groups of units in the data so that those within clusters are perceived with a low degree of dissimilarity with each other. In contrast, units in different clusters are perceived as dissimilar, i.e., with a high degree of dissimilarity. When dealing with large or extremely large data matrices, often referred to as Big Data, the task of assessing these dissimilarities becomes computationally intensive due to the sheer volume of units and variables involved. To manage this vast amount of information, it is essential to employ statistical techniques that synthesize and highlight the most significant aspects of the data. Typically, this involves dimensionality reduction for both units and variables to efficiently summarize the data.

While cluster analysis synthesizes information across the rows of the data matrix, variable reduction operates on the columns, aiming to summarize the features and, ideally, facilitate their interpretation. This key process involves extracting a subspace from the full space spanned by the manifest variables, maintaining the principal informative content. The process allows for the synthesis of common information mainly among subsets of manifest variables, which represent concepts not directly observable. As a result, subspace-based variable reduction identifies a few uncorrelated latent variables that mainly capture common relationships within these subsets. When using techniques like Factor Analysis (FA) or Principal Component Analysis (PCA) for this purpose, interpreting the resulting factors or components can be challenging, particularly when variables significantly load onto multiple factors, a situation known as *cross-loading*. Therefore, a simpler structure in the loading matrix, focusing on the primary relationship between each variable and its related factor, becomes desirable for clarity and ease of interpretation. Furthermore, the latent variables derived from PCA or FA do not provide a unique solution. An equivalent model fit can be achieved by applying an orthogonal rotation to the component axes. This aspect of non-uniqueness is often exploited in practice through Varimax rotation, which is designed to improve the interpretability of latent variables, without affecting the fit of the analysis. The rotation promotes a simpler structure in the loading matrix, however, the rotations do not always ensure enhanced interpretability. An alternative approach has been proposed by Vichi and Saporta (2009) and Vichi (2017), with Disjoint Principal Component (DPCA) and Disjoint FA (DFA), suggesting to construct each component/factor from a distinct subset of manifest variables rather than using all available variables, still optimizing the same estimation as in PCA and FA, respectively.

It is important to note that data matrix reduction for both rows and columns is often performed without specialized methodologies by employing a "tandem analysis." This involves sequentially applying two methods, such as using PCA or FA for variable reduction, followed by Cluster Analysis using KM on the resulting factors. Alternatively, one could start with Cluster Analysis and then proceed to variable reduction. The outcomes of these two tandem analyses differ since each approach optimizes distinct objective functions, one before the other. For instance, when PCA is applied first, the components maximize the total variance of the manifest variables. However, if the manifest variables include high-variance variables that lack a clustering structure, these will be included in the components, even though they are not necessary for KM, which focuses on explaining only the variance between clusters. As a result, sequentially optimizing two different objectives may lead to sub-optimal solutions. In contrast, when combining KM with PCA or FA in a simultaneous approach, a single integrated objective function is utilized. This function aims to optimize both the clustering partition and the subspace simultaneously. The optimization is typically carried out using an Alternating Least Squares (ALS) algorithm, which updates the partition for the current subspace in one step and the subspace for the current partition in the next. This iterative process ensures convergence to a solution that represents at least a local minimum of the integrated objective function. In comparison, tandem analysis, which follows a sequential approach (e.g., PCA followed by KM), does not guarantee joint optimization. One potential limitation of this sequential method is that the initial optimization through PCA may obscure relevant information for the subsequent step of Cluster Analysis or emphasize irrelevant patterns, ultimately leading to sub-optimal solutions, as mentioned by DeSarbo et al. (1990). Indeed, the simultaneous strategy has been shown to be effective in various studies, like De Soete and Carroll (1994), Vichi and Kiers (2001), Vichi (2001), Vichi and Saporta (2009), Rocci and Vichi (2008), Timmerman et al. (2010), Yamamoto and Hwang (2014).

In order to spread access to these techniques and their use, software implementations are needed. Within the R Core Team (2015) environment, there are different libraries available to perform dimensionality reduction techniques. Indeed, the plain version of KM, PCA, and FA are available in the built-in package stats, namely: princomp, factanal, kmeans. Furthermore, some packages allow to go beyond the plain estimation and output of such algorithms. Indeed, one of the most rich libraries in R is psych (W. R. Revelle, 2017), which provides functions that allow to easily simulate data according to different schemes, testing routines, calculation of various estimates, as well as multiple estimation methods. ade4 (Dray and Dufour, 2007) allows for dimensionality reduction in the presence of different types of variables, along with many graphical instruments. The FactoMineR (Lê et al., 2008) package allows for unit-clustering and extraction of latent variables, also in the presence of mixed variables. FactoClass (Pardo and Del Campo, 2007) implements functions for PCA, Correspondence Analysis (CA) as well as clustering, including the tandem approach. factoextra (Kassambara, 2022) instead, provides visualization of the results, aiding their assessment in terms of choice of the number of latent variables, elegant dendograms, screeplots and more. More focused on the choice of the number of clusters is NbClust (Charrad et al., 2014), offering 30 indices for determining the number of clusters, proposing the best method by trying not only different numbers of groups but also different distance measures and clustering methods, going beyond the partitioning ones.

More closely related to the library here presented, to the knowledge of the authors, there are two packages that implement a subset of the techniques proposed within drclust. clustrd (Markos et al., 2019) implements simultaneous methods of clustering and dimensionality reduction. Besides offering functions for continuous data, they also allow for categorical (or mixed) variables. Even more, they formulate, at least for the continuous case, an implementation aligned with the objective function proposed by Yamamoto and Hwang (2014), based on which the reduced KM (RKM) and factorial KM (FKM) become special cases as results of a tuning parameter.

Finally, there is biplotbootGUI (Nieto Librero and Freitas, 2023), offering a GUI allowing to interact with graphical tools, aiding in the choice of the number of components and

clusters. Furthermore, it implements KM with disjoint PCA (DPCA), as described in (Vichi and Saporta, 2009). Even more, they propose an optimization algorithm for the choice of the initial starting point from which the estimation process for the parameters begins.

Like `clustrd`, the `drclust` package provides implementations of FKM and RKM. However, while `clustrd` also supports categorical and mixed-type variables, our implementation currently handles only continuous variables. That said, appropriate pre-processing of categorical variables, as suggested in Vichi et al. (2019), can make them compatible with the proposed methods. In extreme essence, one should dummy-encode all the qualitative variables. In terms of performance, `drclust` offers significantly faster execution. Moreover, regarding FKM, our proposal demonstrates superior results in both empirical applications and simulations, in terms of model fit and the Adjusted Rand Index (ARI). Another alternative, `biplotbootGUI`, implements KM with DPCA and includes built-in plotting functions and a SDP-based initialization of parameters. However, our implementation remains considerably faster and allows users to specify which variables should be grouped together within the same (or different) principal components. This capability enables a partially or fully confirmatory approach to variable reduction. Beyond speed and the confirmatory option, `drclust` offers three methods not currently available in other R packages: DPCA and DFA, both designed for pure dimensionality reduction, and double KM (DKM), which performs simultaneous clustering and variable reduction via KM. All methods are implemented in C++ for computational efficiency. Table 2 summarizes the similarities and differences between `drclust` and existing alternatives

The package presented within this work aims to facilitate the access to and usability of some techniques that fall in two main branches, which overlap. In order to do so, some statistical background is first recalled.

2 Notation and theoretical background

The main pillars of `drclust` fall in two main categories: dimensionality reduction and (partitioning) cluster analysis. The former may be carried out individually or blended with the latter. Because both rely on the language of linear algebra, Table 1 contains, for the convenience of the reader, the mathematical notation needed for this context. Then some theoretical background is reported.

2.1 Latent variables with simple-structure loading matrix

Classical methods of PCA (Pearson, 1901) or FA (Cattell, 1965; Lawley and Maxwell, 1962) build each latent factor from combination of *all* the manifest variables. As a consequence, the loading matrix, describing the relations between manifest and latent variables, is usually not immediately interpretable. Ideally, it is desirable to have variables that are associated to a single factor. This is typically called *simple structure*, which induces subsets of variables characterizing factors and frequently the partition of the variables. While factor rotation techniques go in this direction (especially Varimax), even if not exactly, they do not guarantee the result. Alternative solutions have been proposed. (Zou et al., 2006), by framing the PCA problem as a regression one, introducing an elastic-net penalty, aiming for a sparse solution of the loading matrix \mathbf{A} . For the present work, we consider two techniques for this purpose: DPCA and DFA, implemented in the proposed package.

Disjoint principal component analysis

Vichi and Saporta (2009) propose an alternative solution, DPCA, which leads to the simplest possible structure on \mathbf{A} , while still maximizing the explained variance. Such a result is obtained by building each latent factor from a subset of variables instead of allowing all the variables to contribute to all the components. This means that it provides J non-zero loadings instead of having JQ of them. To obtain this setting, variables are grouped in such a

Symbol	Description
n, J, K, Q	number of: units, manifest variables, unit-clusters, latent factors
\mathbf{X}	$n \times J$ data matrix, where the generic element x_{ij} is the real observation on the i -th unit within the j -th variable
\mathbf{x}_i	$J \times 1$ vector representing the generic row of \mathbf{X}
\mathbf{U}	$n \times K$ unit-cluster membership matrix, binary and row stochastic, with u_{ik} being the generic element
\mathbf{V}	$J \times Q$ variable-cluster membership matrix, binary and row stochastic, with v_{jq} as the generic element
\mathbf{B}	$J \times J$ variable-weighting diagonal matrix
\mathbf{Y}	$n \times Q$ component/factor score matrix defined on the reduced subspace
\mathbf{y}_i	$Q \times 1$ vector representing the generic row of \mathbf{Y}
\mathbf{A}	$J \times Q$ variables - factors, "plain", loading matrix
\mathbf{C}^+	Moore-Penrose pseudo-inverse of a matrix \mathbf{C} . $\mathbf{C}^+ = (\mathbf{C}'\mathbf{C})^{-1}\mathbf{C}'$
$\bar{\mathbf{X}}$	$K \times J$ centroid matrix in the original feature space, i.e., $\bar{\mathbf{X}} = \mathbf{U}^+\mathbf{X}$
$\bar{\mathbf{Y}}$	$K \times Q$ centroid matrix projected in the reduced subspace, i.e., $\bar{\mathbf{Y}} = \bar{\mathbf{X}}\mathbf{A}$
$\mathbf{H}_{\mathbf{C}}$	Projector operator $\mathbf{H}_{\mathbf{C}} = \mathbf{C}(\mathbf{C}'\mathbf{C})^{-1}\mathbf{C}'$ spanned by the columns of matrix \mathbf{C}
\mathbf{E}	$n \times J$ Error term matrix
$\ \cdot\ $	Frobenius norm

Table 1: Notation

way that they form a partition of the initial set. The model can be described as a constrained PCA, where the matrix \mathbf{A} is restricted to be reparametrized into the product $\mathbf{A} = \mathbf{BV}$. Thus, the model is described as:

$$\mathbf{X} = \mathbf{XAA}' + \mathbf{E} = \mathbf{XBVV}'\mathbf{B} + \mathbf{E}, \quad (1)$$

subject to

$$\mathbf{V} = [v_{jq} \in \{0,1\}] \quad (\text{binarity}), \quad (2)$$

$$\mathbf{V}\mathbf{1}_Q = \mathbf{1}_J \quad (\text{row-stochasticity}), \quad (3)$$

$$\mathbf{V}'\mathbf{BB}'\mathbf{V} = \mathbf{I}_Q \quad (\text{orthonormality}), \quad (4)$$

$$\mathbf{B} = \text{diag}(b_1, \dots, b_J) \quad (\text{diagonality}). \quad (5)$$

The estimation of the parameters \mathbf{B} and \mathbf{V} is carried out via least squares (LS) and, by solving the minimization problem,

$$RSS_{DPCA}(\mathbf{B}, \mathbf{V}) = \|\mathbf{X} - \mathbf{XBVV}'\mathbf{B}\|^2 \quad (6)$$

subject to the constraints (2, 3, 4, 5). An ALS algorithm is employed, guaranteeing at least a local optimum. In order to (at least partially) overcome this downside, multiple random starts are needed, and the best solution is retained.

Therefore, the DPCA method is subject to more structural constraints than standard PCA. Specifically, standard PCA does not enforce the reparameterization $\mathbf{A} = \mathbf{BV}$, meaning its loading matrix \mathbf{A} is free to vary among orthonormal matrices. In contrast, DPCA still requires an orthonormal matrix \mathbf{A} but also needs that each principal component is associated with a disjoint subset of variables that most reconstruct the data. This implies that each variable contributes to only one component, resulting in a sparse and block-diagonal loading matrix. In essence, DPCA fits Q separate PCAs on the Q disjoint subsets of variables, and from each, extracts the eigenvector associated with the largest eigenvalue. In general, the total variance explained by DPCA is slightly lower, and the residual of the objective function is larger compared to PCA. This trade-off is made in exchange for the added constraint that

clearly enhances interpretability. The extent of the reduction depends on the true underlying structure of the latent factors, specifically on whether they are truly uncorrelated. When the observed correlation matrix is block diagonal, with variables within blocks being highly correlated and variables between blocks being uncorrelated, DPCA can explain almost the same amount of variance of PCA, with the advantage to simplify interpretation.

It is important to note that, as DPCA is implemented, it allows for a blend of exploratory and confirmatory approaches. In the confirmatory framework, users can specify *a priori* which variables should collectively contribute to a factor using the `constr` argument, available for the last three functions in Table 2. The algorithm assigns the remaining manifest variables, for which no constraint has been specified, to the Q factors in a way that ensures the latent variables best reconstruct the manifest ones, capturing the maximum variance. This is accomplished by minimizing the loss function (6). Although each of the Q latent variables is derived from a different subset of variables, which involves the spectral decomposition of multiple covariance matrices, their smaller size, combined with the implementation in C++, enables very rapid execution of the routine.

A very positive side effect of the additional constraint in DPCA compared to standard PCA is the uniqueness of the solution, which eliminates the need for factor rotation in DPCA.

Disjoint factor analysis

Proposed by Vichi (2017), this technique is the model-based counterpart of the DPCA model. It pursues a similar goal in terms of building Q factors from J variables, imposing a simple structure on the loading matrix. However, the means by which the goal is pursued are different. Unlike DPCA, the estimation method adopted for DFA is Maximum Likelihood and the model requires additional statistical assumptions compared to DPCA. The model can be formulated in a matrix form as,

$$\mathbf{X} = \mathbf{Y}\mathbf{A}' + \mathbf{E}, \quad (7)$$

where \mathbf{X} is centered, meaning that the mean vector $\boldsymbol{\mu}$ has been subtracted from each multivariate unit \mathbf{x}_i . Therefore, for a multivariate, centered, unit, the previous model can be expressed as

$$\mathbf{x}_i = \mathbf{A}\mathbf{y}_i + \mathbf{e}_i, \quad i = 1, \dots, n. \quad (8)$$

where \mathbf{y}_i is the i -th row of \mathbf{Y} and \mathbf{x}_i , \mathbf{e}_i are, respectively, the i -th rows of \mathbf{X} and \mathbf{E} , with a multivariate normal distribution on the J -dimensional space,

$$\mathbf{x}_i \sim \mathcal{N}(\mathbf{0}, \Sigma_{\mathbf{X}}), \quad \mathbf{e}_i \sim \mathcal{N}(\mathbf{0}, \Psi) \quad (9)$$

The covariance structure of the FA model can be written,

$$\text{Cov}(\mathbf{x}_i) = \Sigma_{\mathbf{X}} = \mathbf{A}\mathbf{A}' + \Psi, \quad (10)$$

where additional, assumptions are needed,

$$\text{Cov}(\mathbf{y}_i) = \Sigma_{\mathbf{Y}} = \mathbf{I}_Q, \quad (11)$$

$$\text{Cov}(\mathbf{e}_i) = \Sigma_{\mathbf{E}} = \Psi, \quad \Psi = \text{diag}(\psi_1, \dots, \psi_Q : \psi_q > 0)', \quad j = 1, \dots, J \quad (12)$$

$$\text{Cov}(\mathbf{e}_i, \mathbf{y}_j) = \Sigma_{\mathbf{EY}} = 0 \quad (13)$$

$$\mathbf{A} = \mathbf{BV} \quad (14)$$

The objective function can be formulated as the maximization of the Likelihood function or as the minimization of the following discrepancy:

$$\begin{aligned} D_{DFA}(\mathbf{B}, \mathbf{V}, \boldsymbol{\Psi}) &= |\ln(\mathbf{BV}\mathbf{V}'\mathbf{B} + \boldsymbol{\Psi})| - \ln|\mathbf{S}| + \text{tr}((\mathbf{BV}\mathbf{V}'\mathbf{B} + \boldsymbol{\Psi})^{-1}\mathbf{S}) - J, \\ j &= 1, \dots, J, q = 1, \dots, Q, \\ \text{s.t. : } \mathbf{V} &= [v_{jq}], v_{jq} \in \{0, 1\}, \sum_q v_{jq} = 1, \end{aligned}$$

whose parameters are optimized by means of a coordinate descent algorithm.

Apart from the methodological distinctions between DPCA and DFA, the latter exhibits the scale equivariance property. The optimization of the Likelihood function implies a higher computational load, thus, a longer (compared to the DPCA) execution time.

As in the DPCA case, under the constraint $\mathbf{A} = \mathbf{BV}$, the solution provided by the model is, also in this case, unique.

2.2 Joint clustering and variable reduction

The four clustering methods discussed all follow the K -means framework, working to partition units. However, they differ primarily in how they handle variable reduction.

Double KM (DKM) employs a symmetric approach, clustering both the units (rows) and the variables (columns) of the data matrix at the same time. This leads to the simultaneous identification of mean profiles for both dimensions. DKM is particularly suitable for data matrices where both rows and columns represent units. Examples of such matrices include document-by-term matrices used in Text Analysis, product-by-customer matrices in Marketing, and gene-by-sample matrices in Biology.

In contrast, the other three clustering methods adopt an asymmetric approach. They treat rows and columns differently, focusing on means profiles and clustering for rows, while employing components or factors for the variables (columns). These methods are more appropriate for typical units-by-variable matrices, where it's beneficial to synthesize variables using components or factors. At the same time, they emphasize clustering and the mean profiles of the clusters specifically for the rows. The methodologies that fall into this category are RKM, FKM, and DPCAKM.

The estimation is carried out by the LS method, while the computation of the estimates is performed via ALS.

Double k-means (DKM)

Proposed by [Vichi \(2001\)](#), DKM is one of the first introduced bi-clustering methods that provides a simultaneous partition of the units and variables, resulting in a two-way extension of the plain KM ([McQueen, 1967](#)). The model is described by the following equation,

$$\mathbf{X} = \mathbf{U}\bar{\mathbf{Y}}\mathbf{V}' + \mathbf{E} \quad (15)$$

where $\bar{\mathbf{Y}}$ is the centroid matrix in the reduced space for the rows and columns, enabling a comprehensive summarization of units and variables. By optimizing a single objective function, the DKM method captures valuable information from both dimensions of the dataset simultaneously.

This bi-clustering approach can be applied in several impactful ways. One key application is in the realm of Big Data. DKM can effectively compress expansive datasets that includes a vast number of units and variables into a compressed more manageable and robust data matrix $\bar{\mathbf{Y}}$. This compressed matrix, formed by mean profiles both for rows and columns, can then be explored and analyzed using a variety of subsequent statistical techniques, thus facilitating efficient data handling and analysis of Big Data. The algorithm similarly to the well-known KM is very fast and converges quickly to a solution, which is at least a local minimum of the problem.

Another significant application of DKM is its capability to achieve optimal clustering for both rows and columns. This dual clustering ability is particularly advantageous in situations where it is essential to discern meaningful patterns and relationships within complex datasets, highlighting the utility of DKM in diverse fields and scenarios.

The Least Squares estimation of the parameters \mathbf{U} , \mathbf{V} and $\bar{\mathbf{Y}}$ leads to the minimization of the problem

$$RSS_{DKM}(\mathbf{U}, \mathbf{V}, \bar{\mathbf{Y}}) = \|\mathbf{X} - \mathbf{U}\bar{\mathbf{Y}}\mathbf{V}'\|^2, \quad (16)$$

$$\text{s.t. : } u_{ik} \in \{0, 1\}, \sum_k u_{ik} = 1, \quad i = 1, \dots, N, \quad k = 1, \dots, K, \quad (17)$$

$$v_{jq} \in \{0, 1\}, \sum_q v_{jq} = 1, \quad j = 1, \dots, J, \quad q = 1, \dots, Q. \quad (18)$$

Since $\bar{\mathbf{Y}} = \mathbf{U}^+ \mathbf{X} \mathbf{V}^{+'}$, then (16) can be framed in terms of projector operators, thus:

$$RSS_{DKM}(\mathbf{U}, \mathbf{V}) = \|\mathbf{X} - \mathbf{H}_{\mathbf{U}} \mathbf{X} \mathbf{H}_{\mathbf{V}}\|^2. \quad (19)$$

Minimizing in both cases the sum of squared-residuals (or, equivalently, the within deviances associated to the K unit-clusters and Q variable-clusters). In this way, one obtains a (hard) classification of both units and variables. The optimization of [19] is done via ALS, alternating, in essence, two assignment problems for rows and columns similar to KM steps.

Reduced k-means (RKM)

Proposed by De Soete and Carroll (1994), RKM performs the reduction of the variables by projecting the J -dimensional centroid matrix into a Q -dimensional subspace ($Q \leq J$), spanned by the columns of the loading matrix \mathbf{A} , such that it best reconstructs \mathbf{X} by using the orthogonal projector matrix $\mathbf{A}\mathbf{A}'$. Therefore, the model is described by the following equation,

$$\mathbf{X} = \mathbf{U}\bar{\mathbf{X}}\mathbf{A}\mathbf{A}' + \mathbf{E}. \quad (20)$$

The estimation of \mathbf{U} and \mathbf{A} can be done via LS, minimizing the following equation,

$$RSS_{RKM}(\mathbf{U}, \mathbf{A}) = \|\mathbf{X} - \mathbf{U}\bar{\mathbf{X}}\mathbf{A}\mathbf{A}'\|^2, \quad (21)$$

$$\text{s.t. : } u_{ik} \in \{0, 1\}, \sum_k u_{ik} = 1, \quad \mathbf{A}'\mathbf{A} = \mathbf{I}. \quad (22)$$

which can be optimized, once again, via ALS. In essence, the model alternates a KM step assigning each original unit x_i to the closest centroid in the reduced space and a PCA step based on the spectral decomposition of $\mathbf{X}'\mathbf{H}_{\mathbf{U}}\mathbf{X}$, conditioned on the results of the previous iteration. The iterations continue until when the difference between two subsequent objective functions is smaller than a small arbitrary chosen constant $\epsilon > 0$.

Factorial k-means (FKM)

Proposed by Vichi and Kiers (2001), FKM produces a dimension reduction both of the units and centroids differently from RKM. Its goal is to reconstruct the data in the reduced subspace, \mathbf{Y} , by means of the centroids in the reduced space. The FKM model can be obtained by considering the RKM model and post-multiplying the right- and left-hand side of it in equation (20), and rewriting the new error as \mathbf{E} ,

$$\mathbf{XA} = \mathbf{U}\bar{\mathbf{X}}\mathbf{A} + \mathbf{E}. \quad (23)$$

Its estimation via LS results in the optimization of the following equation,

$$RSS_{FKM}(\mathbf{U}, \mathbf{A}, \bar{\mathbf{X}}) = \|\mathbf{XA} - \mathbf{U}\bar{\mathbf{X}}\mathbf{A}\|^2, \quad (24)$$

$$\text{s.t. : } u_{ik} \in \{0, 1\}, \sum_k u_{ik} = 1, \mathbf{A}'\mathbf{A} = \mathbf{I}. \quad (25)$$

Although the connection with the RKM model appears straightforward, it can be shown that the loss function of the former is always equal or smaller compared to the latter. Practically, the KM step is applied on \mathbf{XA} , instead of just \mathbf{X} , as it happens in the DKM and RKM. In essence, FKM works better when the data and centroids are lying in the reduced subspace, and not just the centroids as in RKM.

In order to decide when RKM or FKM can be properly applied, it is important to recall that two types of residuals can be defined in dimensionality reduction: *subspace residuals*, lying on the subspace spanned by the columns of \mathbf{A} and *complement residuals*, lying on the complement of this subspace, i.e., those residual lying on the subspace spanned by the columns of \mathbf{A}^\perp , with \mathbf{A}^\perp a column-wise orthonormal matrix of order $J \times (J - Q)$ such that $\mathbf{A}^\perp \mathbf{A}^{\perp'} = \mathbf{O}_{J-Q}$, where \mathbf{O}_{J-Q} is the matrix of zeroes of order $Q \times (J - Q)$. FKM is more effective when there is significant residual variance in the subspace orthogonal to the clustering subspace. In other words, the complement residuals typically represent the error given by those observed variables that scarcely contribute to the clustering subspace to be identified. FKM tends to recover the subspace and clustering structure more accurately when the data contains variables with substantial variance that does not reflect the clustering structure and therefore mask it. FKM can better ignore these variables and focus on the relevant clustering subspace. On the other hand, RKM performs better when the data has significant residual variance within the clustering subspace itself. This means that when the variables within the subspace show considerable variance, RKM can more effectively capture the clustering structure.

In essence, when most of the variables in the dataset reflect the clustering structure, RKM is more likely to provide a good solution. If this is not the case, FKM may be preferred.

Disjoint principal component analysis k-means (DPCAkm)

Starting from the FKM model, the goal here, beside the partition of the units, is to have a parsimonious representation of the relationships between latent and manifest variables, provided by the loading matrix \mathbf{A} . [Vichi and Saporta \(2009\)](#) propose for FKM the parametrization of $\mathbf{A} = \mathbf{BV}$, that allows the simplest structure and thus simplifies the interpretation of the factors,

$$\mathbf{X} = \mathbf{U}\bar{\mathbf{X}}\mathbf{BV}'\mathbf{B} + \mathbf{E}. \quad (26)$$

By estimating \mathbf{U} , \mathbf{B} , \mathbf{V} and $\bar{\mathbf{X}}$ via LS, the loss function of the proposed method becomes:

$$RSS_{DPCAkm}(\mathbf{U}, \mathbf{B}, \mathbf{V}, \bar{\mathbf{X}}) = ||\mathbf{X} - \mathbf{U}\bar{\mathbf{X}}\mathbf{BV}'\mathbf{B}||^2, \quad (27)$$

$$\text{s.t. : } u_{ik} \in \{0, 1\}, \sum_k u_{ik} = 1, i = 1, \dots, N, k = 1, \dots, K, \quad (28)$$

$$v_{jq} \in \{0, 1\}, \sum_q v_{jq} = 1, j = 1, \dots, J, q = 1, \dots, Q, \quad (29)$$

$$\mathbf{V}'\mathbf{B}\mathbf{B}\mathbf{V} = \mathbf{I}, \mathbf{B} = \text{diag}(b_1, \dots, b_J). \quad (30)$$

In practice, this model has traits of the DPCA given the projection on the reduced subspace and the partitioning of the units, resulting in a sparse loading matrix, but also of the DKM, given the presence of both \mathbf{U} and \mathbf{V} . Thus, DPCAkm can be considered a bi-clustering methodology with an asymmetric treatment of the rows and columns of \mathbf{X} . By inheriting the constraint on \mathbf{A} , the overall fit of the model compared with the FKM for example, is generally worse although it offers an easier interpretation of the principal components. Nevertheless, it is potentially able to identify a better partition of the units. Like in the DPCA case, the difference is negligible when the true latent variables are really disjoint. As implemented, the assignment step is carried out by minimizing the unit-centroid squared-Euclidean distance in the reduced subspace.

3 The package

The library offers the implementation of all the models mentioned in the previous section. Each one of them corresponds to a specific function implemented using **Rcpp** (Eddelbuettel and Francois, 2011) and **RcppArmadillo** (Eddelbuettel and Sanderson, 2014).

Function	Model	Previous Implementations	Main differences in drclust
doublekm	DKM (Vichi, 2001)	None	Short runtime (C++);
redkm	RKM (De Soete and Carroll, 1994)	in clusterd; Mixed variables;	>50x faster (C++); Continuous variables;
factkm	FKM (Vichi and Kiers, 2001)	in clustrd; Mixed variables	>20x faster (C++); Continuous variables; Better fit and classification;
dpcakm	DPCAkm (Vichi and Saporta, 2009)	in biplotbootGUI; Continuous variables; SDP-based initialization of parameters;	>10x faster (C++); Constraint on variable allocation within principal components;
dispca	DPCA (Vichi and Saporta, 2009)	None	Short runtime (C++); Constraint on variable allocation within principal components;
disfa	DFA (Vichi, 2017)	None	Short runtime (C++); Constraint on variable allocation within factors;

Table 2: Statistical methods available in the drclust package

Some additional functions have been made available for the user. Most of them are intended to aid the user in evaluating the quality of the results, or in the choice of the hyper-parameters.

With regard to the auxiliary functions (Table 3), they have all been implemented in the R language, building on top of packages already available on CRAN, such as **cluster** by Maechler et al. (2023), **factoextra** by Kassambara (2022), **pheatmap** by Kolde (2019), which allowed for an easier implementation. One of the main goals of the proposed package, besides spreading the availability and usability of the statistical methods considered, is the speed of computation. By doing so (if the memory is sufficient), the results, also for large data matrices, can be obtained in a reasonable amount of time. A first mean adopted to pursue such a goal is the full implementation of the statistical methods in the C++ language. The libraries used are **Rcpp** (Eddelbuettel and Francois, 2011) and **RcppArmadillo** (Eddelbuettel and Sanderson, 2014), which significantly reduced the required runtime.

A practical issue that happens very often in crisp (hard) clustering, such as KM, is the presence of empty clusters after the assignment step. When this happens, a column of \mathbf{U} has all elements equal to zero, which can be proved to be a local minimum solution, and impedes obtaining a solution for $(\mathbf{U}'\mathbf{U})^{-1}$. This typically happens even more often when the number of clusters K specified by the user is larger than the true one or in the case of a sub-optimal solution. Among the possible solutions addressing this issue, the one implemented here consists in splitting the cluster with higher within-deviance. In practice, a KM with $K = 2$ is applied to it, assigning to the empty cluster one of the two clusters obtained by the procedure, which is iterated until all the empty clusters are filled. Such a strategy guarantees that the monotonicity of the ALS algorithm is preserved, although it is the most time-consuming one.

Function	Technique	Description	Goal
apseudoF	"relaxed" pseudoF	"Relaxed" version of Caliński and Harabasz (1974) . Selects the second largest pseudoF value if the difference with the first is less than a fraction.	Parameter tuning
dpseudoF	DKM-pseudoF	Adaptation of the pseudoF criterion proposed by Rocci and Vichi (2008) to bi-clustering.	Parameter tuning
kaiserCrit	Kaiser criterion	Kaiser rule for selecting the number of principal components (Kaiser, 1960).	Parameter tuning
centree	Dendrogram of the centroids	Graphical tool showing how close the centroids of a partition are.	Visualization
silhouette	Silhouette	Imported from cluster (Maechler et al., 2023) and factoextra (Kassambara, 2022).	Visualization, parameter tuning
heatm	Heatmap	Heatmap of distance-ordered units within distance-ordered clusters, adapted from pheatmap (Kolde, 2019).	Visualization
CronbachAlpha	Cronbach Alpha Index	Proposed by Cronbach (1951) . Assesses the unidimensionality of a dataset.	Assessment
mrand	ARI	Assesses clustering quality based on the confusion matrix (Rand, 1971).	Assessment
cluster	Membership vector	Returns a multinomial $1 \times n$ membership vector from a binary, row-stochastic $n \times K$ membership matrix; mimics kmeans\$cluster.	Encoding

Table 3: Auxiliary functions available in the library

Among all the six implementations of the statistical techniques, there are some arguments that are set to a default value. Table 4 describes all the arguments that have a default value. In particular, `print`, which displays a descriptive summary of the results, is set to zero (so the user should explicitly require to the function such output). `Rndstart` is set as default to 20, so that the algorithm is run 20 times until convergence. In order to have more confidence (not certainty) that the obtained solution is a global optimum, a higher value for this argument can be provided. With particular regard to `redkm` and `factkm`, the argument `rot`, which performs a Varimax rotation on the loading matrix, is set by default to 0. If the user would like to have this performed, it must be set equal to 1. Finally, the `constr` argument, which is available for `dpcakm` and `dispca`, is set by default to a vector (of length J) of zeros, so that each variable is selected to contribute to the most appropriate latent variable, according to the logic of the model.

By offering a fast execution time, all the implemented models allow to run multiple random starts of the algorithm in a reasonable amount of time. This feature comes particularly useful given the absence of guarantees of global optima for the ALS algorithm, which has an ad-hoc implementation for all the models. Table 5 shows that, compared to the two packages which implement 3 of the 6 models in `drclust`, our proposal is much faster than the corresponding versions implemented in R (Table 5), providing, nevertheless, compelling results.

The iris dataset has been used in order to measure the performance in terms of fit, runtime, and ARI ([Rand, 1971](#)). The z-transform has been applied on all the variables of the dataset. This implies that all the variables, post-transformation, have mean equal to 0 and variance equal to 1, by subtracting the mean to each variable and dividing the result by the standard deviation. The same result is typically obtained by the `scale(X)` R function.

Argument	Used In	Description	Default Value
Rndstart	doublekm, redkm, factkm, dpcakm, dispca, disfa	Number of times the model is run until convergence.	20
verbose	doublekm, redkm, factkm, dpcakm, dispca, disfa	Outputs basic summary statistics regarding each random start (1 = enabled; 0 = disabled).	0
maxiter	doublekm, redkm, factkm, dpcakm, dispca, disfa	Maximum number of iterations allowed for each random start (if convergence is not yet reached)	100
tol	doublekm, redkm, factkm, dpcakm, dispca, disfa	Tolerance threshold (maximum difference between the values of the objective function of two consecutive iterations such that convergence is assumed)	10^{-6}
tol	apseudoF	Approximation value. It is half of the length of the interval put for each pF value. $0 \leq tol < 1$	0.05
rot	redkm, factkm	performs varimax rotation of axes obtained via PCA (0 = False; 1 = True)	0
prep	doublekm, redkm, factkm, dpcakm, dispca, disfa	Pre-processing of the data. 1 performs the z-score transform; 2 performs the min-max transform; 0 leaves the data un-pre-processed	1
print	doublekm, redkm, factkm, dpcakm, dispca, disfa	Final summary statistics of the performed method (1 = enabled; 0 = disabled).	0
constr	dpcakm, dispca, disfa	Vector of length J (number of variables) specifying variable-to-cluster assignments. Each element can be an integer from 0 to Q (number of variable-clusters or components), indicating a fixed assignment, or 0 to leave the variable unconstrained (i.e., assigned by the algorithm).	rep(0, J)

Table 4: Arguments accepted by functions in the drclust package with default values

$$\mathbf{Z}_{\cdot j} = \frac{\mathbf{X}_{\cdot j} - \mu_j \mathbf{1}_n}{\sigma_j} \quad (31)$$

where μ_j is the mean of the j -th variable and σ_j its standard deviation. The subscript $\cdot j$ refers to the whole j -th column of the matrix. This operation avoids the measurement scale to have impact on the final result (and is used by default, unless otherwise specified by the user, within all the techniques implemented by drclust). In order to avoid the comparison between potentially different objective functions, the between deviance (intended as described by the authors in the articles where the methods have been proposed) has been used as a fit measure and computed based on the output provided by the functions, aiming at having homogeneity in the evaluation metric. $K=3$ and $Q=2$ have been used for the clustering algorithms, maintaining, for the two-dimensionality reduction techniques, just $Q = 2$.

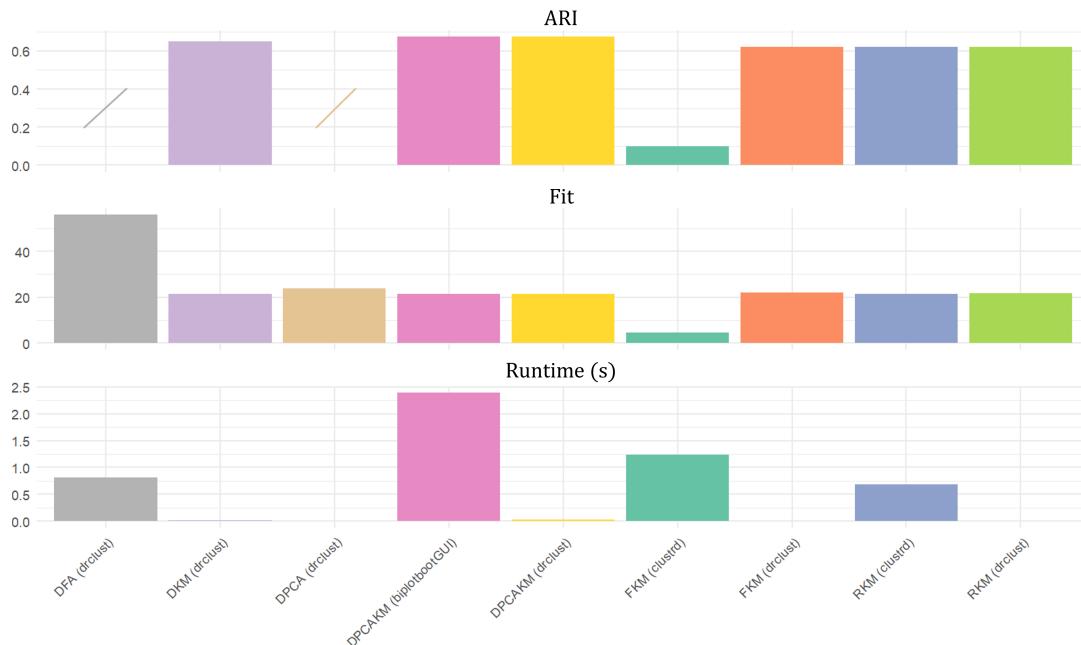
For each method, 100 runs have been performed and the best solution has been picked. For each run, the maximum allowed number of iterations = 100, with a tolerance error (i.e., precision) equal to 10^{-6} .

The results of table 5 are visually represented in figure 1.

Although the runtime heavily depends on the hardware characteristics, they have been reported within Table 5 for a relative comparison purpose only, having run all the techniques with the same one hardware. For all the computations within the present work, the specifics of the machine used are: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 2.00 GHz.

Besides the already mentioned difference between DPCA and DFA, it is worth mentioning that, in terms of implementation, they retrieve the latent variables differently. Indeed,

Library	Technique	Runtime	Fit	ARI	Fit Measure
clustrd	RKM	0.73	21.38	0.620	$\ \mathbf{U}\bar{\mathbf{Y}}'\ ^2$
drclust	RKM	0.01	21.78	0.620	$\ \mathbf{U}\bar{\mathbf{Y}}'\ ^2$
clustrd	FKM	1.89	4.48	0.098	$\ \mathbf{U}\bar{\mathbf{Y}}\ ^2$
drclust	FKM	0.03	21.89	0.620	$\ \mathbf{U}\bar{\mathbf{Y}}\ ^2$
biplotbootGUI	DPCA	2.83	21.32	0.676	$\ \mathbf{U}\bar{\mathbf{Y}}'\ ^2$
drclust	CDPCA	0.05	21.34	0.676	$\ \mathbf{U}\bar{\mathbf{Y}}'\ ^2$
drclust	DKM	0.03	21.29	0.652	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{H}_V\ ^2$
drclust	DPCA	<0.01	23.70	-	$\ \mathbf{Y}\mathbf{A}'\ ^2$
drclust	DFA	1.11	55.91	-	$\ \mathbf{Y}\mathbf{A}'\ ^2$

Table 5: Performance of the variable reduction and joint clustering-variable reduction models**Figure 1:** ARI, Fit, Runtime for the available implementations

while the DPCA relies on the eigendecomposition, the DFA uses an implementation of the power method (Hotelling, 1933).

In essence, the implementation of our proposal, while being very fast, exhibits a goodness of fit very close (sometimes better, compared) to the available alternatives.

4 Simulation study

To better understand the capabilities of the proposed methodologies and evaluate the performance of the drclust package, a simulation study was conducted. In this study, we assume that the number of clusters (K) and the number of factors (Q) are known, and we examine how results vary across the DKM, RKM, FKM, and DPCAKM methods.

4.1 Data generation process

The performance of these algorithms is tested on synthetic data generated through a specific procedure. Initially, centroids are created using eigendecomposition on a transformed distance matrix, resulting in three equidistant centroids in a reduced two-dimensional space.

To model the variances and covariances among the generated units within each cluster and to introduce heterogeneity among the units, a variance-covariance matrix (Σ_O) is derived from samples taken from a zero-mean Gaussian distribution, with a specified standard deviation (σ_u).

Membership for the 1,000 units is determined based on a ($K \times 1$) vector of prior probabilities, utilizing a multinomial distribution with (0.2, 0.3, 0.5) probabilities. For each unit, a sample is drawn from a multivariate Gaussian distribution centered around its corresponding centroid, using the previously generated covariance matrix (Σ_O). Additionally, four masking variables, which do not exhibit any clustering structure, are generated from a zero-mean multivariate Gaussian and scaled by a standard deviation of $\sigma=6$. These masking variables are added to the 2 variables that form the clustering structure of the dataset. Then, the final sample dataset is standardized.

It is important to note that the standard deviation σ_u controls the amount of variance in the reduced space, thus influencing the level of subspace residuals. Conversely, σ_m regulates the variance of the masking variables, impacting the complement residuals.

This study considers various scenarios where there are $J = 6$ variables, $n = 1,000$ units, $K = 3$ clusters and $Q = 2$ factors. We explore high, medium, and low variance σ_u of the heterogeneity within clusters with values of 0.8, 0.55, and 0.3. For each combination of these parameters, $s=100$ samples are generated. Since the design is fully crossed, a total of 300 datasets are produced. Examples of the generated samples are illustrated in Figure 2, which shows that as the level of within-cluster variance increases, the variables with a clustering structure tend to create overlapping clusters. It is worthy to inform that the two techniques dedicated solely to variable reduction, namely DPCA and DFA, were not included in the simulation study. This is because the study's primary focus is on clustering and dimension reduction and the comparison with competing implementations. However, it is worth noting that these methods are inherently quick, as can be observed from the speed of methodologies that combine clustering with DPCA or DFA dimension reduction methods.

4.2 Performance evaluation

The performance of the proposed methods was assessed through a simulation study. To evaluate the accuracy in recovering the true cluster membership of the units (\mathbf{U}), the ARI (Hubert and Arabie, 1985) was employed. The ARI quantifies the similarity between the hard partitions generated by the estimated classification matrices and those defined by the true partition. It considers both the reference partition and the one produced by the algorithm under evaluation. The ARI typically ranges from 0 to 1, where 0 indicates a level of agreement expected by random chance, and 1 denotes a perfect match. Negative values may also occur, indicating agreement worse than what would be expected by chance. In order to assess the models' ability to reconstruct the underlying data structure, the between deviance, denoted by f , was computed. This measure is defined in the original works proposing the evaluated methods and is reported in the second column (Fit Measure) of Table 6. For comparison, the true between deviance f^* , calculated from the known true, known, values of \mathbf{U} and \mathbf{A} , was also computed. The difference $f - f^*$ was considered, where negative values suggest potential overfitting. Furthermore, the squared Frobenius norm $\|\mathbf{A}^* - \mathbf{A}\|^2$ was computed to assess how accurately each model estimated the true loading matrix \mathbf{A}^* . This evaluation was not applicable to the DKM method, as it does not provide estimates of the loading matrix. For each performance metric presented in Table 6, the median value across $s = 100$ replicates, for each level of error (within deviance), is reported.

It is important to note that fit and ARI reflect distinct objectives. While fit measures the variance explained by the model, the ARI assesses clustering accuracy. As such, the two metrics may diverge. A model may achieve high fit by capturing subtle variation or even noise, which may not correspond to well-separated clusters, leading to a lower ARI. Conversely, a method focused on maximizing cluster separation may yield high ARI while explaining less overall variance. This trade-off is particularly relevant in unsupervised

settings, where there is no external supervision to guide the balance between reconstruction and partitioning. For this reason, we report both metrics to provide a more comprehensive assessment of model performance.

4.3 Algorithms performances and comparison with the competing implementations

For each sample, the algorithms DKM, RKM, FKM, and DPCAKM are applied using 100 random start solutions, selecting the best one. This significantly reduces the impact of local minima in the clustering and dimension reduction process. Figure 2 depicts the typical situation for each scenario (low, medium, high within-cluster variance). For the three

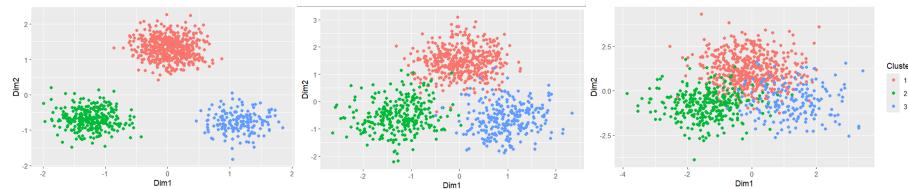


Figure 2: Within-cluster variance of the simulated data (in order: low, medium, high)

Technique	Fit Measure	Library	Runtime (s)	Fit	ARI	$f^* - f$	$\ \mathbf{A}^* - \mathbf{A}\ ^2$
Low							
RKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}'\ ^2$	clustrd	164.03	42.76	1.00	0.00	2.00
RKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}'\ ^2$	drclust	0.48	42.76	1.00	0.00	2.00
FKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}\ ^2$	clustrd	15.48	2.89	0.35	39.77	1.99
FKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}\ ^2$	drclust	0.52	42.76	1.00	0.00	2.00
DPCAKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}\ ^2$	biplotbootGUI	41.70	42.74	1.00	0.01	2.00
DPCAKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}\ ^2$	drclust	1.37	42.74	1.00	0.01	2.00
DKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{V}\ ^2$	drclust	0.78	61.55	0.46	-18.94	-
Medium							
RKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}'\ ^2$	clustrd	230.31	39.18	0.92	-0.27	2.00
RKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}'\ ^2$	drclust	0.70	39.18	0.92	-0.27	2.00
FKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}\ ^2$	clustrd	14.31	2.85	0.28	36.09	1.99
FKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}\ ^2$	drclust	0.76	39.18	0.92	-0.27	2
DPCAKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}\ ^2$	biplotbootGUI	47.76	39.15	0.92	-0.25	2.00
DPCAKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}\ ^2$	drclust	1.64	39.15	0.92	-0.25	2.00
DKM	$\ \mathbf{U}\bar{\mathbf{Y}}\mathbf{V}\ ^2$	drclust	0.81	5.93	0.39	-21.00	-
High							
RKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}'\ ^2$	clustrd	314.89	36.61	0.62	-2.11	2.00
RKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}'\ ^2$	drclust	0.94	36.61	0.61	-2.11	2.00
FKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}\ ^2$	clustrd	13.87	2.90	0.19	31.55	2.00
FKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}\ ^2$	drclust	1.02	36.61	0.61	-2.11	2.00
DPCAKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}\ ^2$	biplotbootGUI	55.49	36.53	0.64	-1.99	2.00
DPCAKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{A}\ ^2$	drclust	2.06	36.53	0.63	-2.01	2.00
DKM	$\ \mathbf{U}\bar{\mathbf{X}}\mathbf{V}\ ^2$	drclust	0.84	58.97	0.29	-24.37	-

Table 6: Comparison of joint clustering-variable reduction methods on simulated data

scenarios, the results are reported in 6.

Regarding the RKM, the `drclust` and `clustrd` performance is very close, both in terms of the ability to recover the data (fit) and in terms of identifying the true classification of the objects.

The FKM appears to be performing way better in the `drclust` case in terms of fit and ARI. Considering both ARI and fit for the CDPCA algorithm, the difference between the present proposal and the one of `biplotbootGUI` is almost absent. Referring to the CPU runtime, all

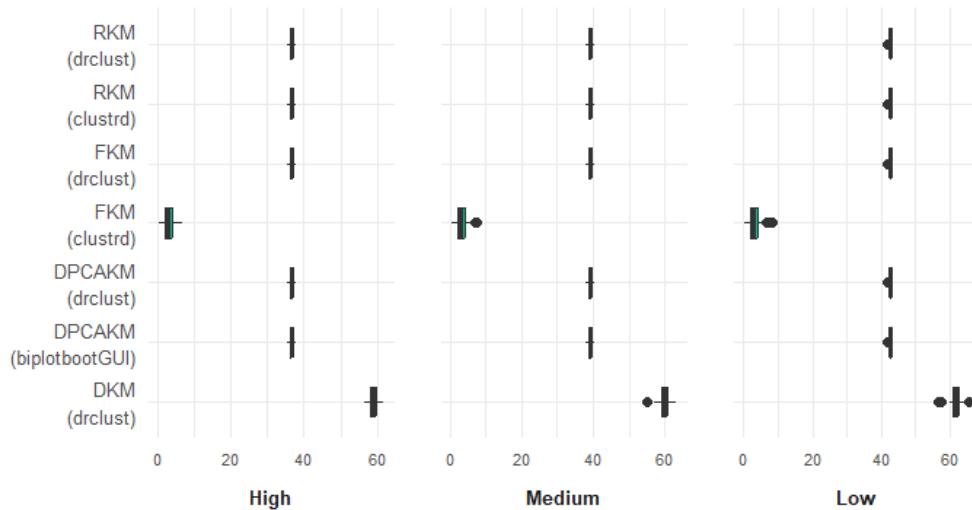


Figure 3: Boxplots of the Fit results in Table 6

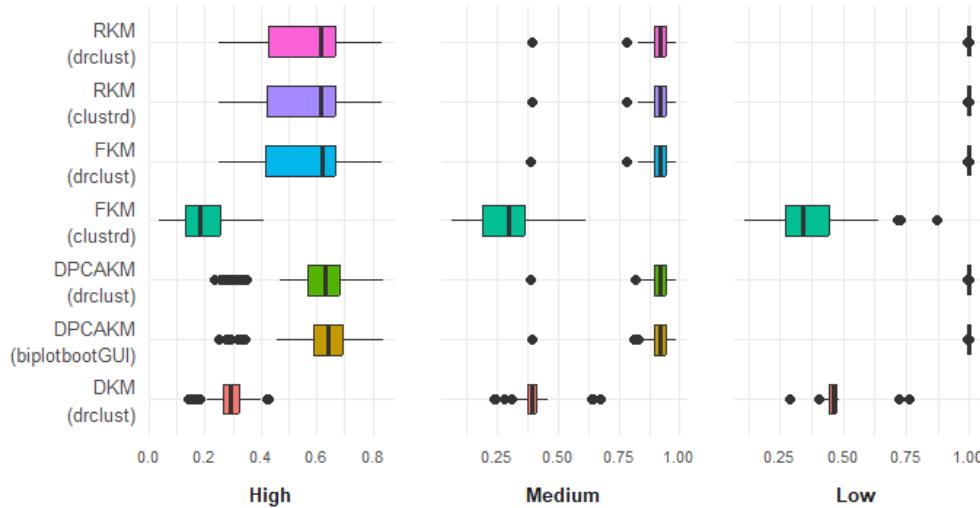


Figure 4: Boxplots of the ARI results in Table 6

of the models proposed are significantly faster compared to the previously available ones (RKM, FKM and KM with DPCA). For the architecture used for the experiments, the order of magnitude for such differences are specified in the last column of Table 2.

In general, the `drclust` shows a slight overfit, while there is no evident difference in the ability to recover the true A . There is no alternative implementation for the DKM, so no comparison can be made. However, except for the ARI which is lower than the other techniques, its fit is very close, showing a compelling ability to reconstruct the data. In general, except for the FKM, where our proposal outperforms the one in `clustrd`, our proposal is equivalent in terms of fit and ARI. However, our versions outperform every alternative in terms of runtime. Figures (3 - 8) visually depict the situation in 6, showing also the variability for each scenario, among 100 replicates. In general, with the exception of the FKM method, where our proposed approach outperforms the implementation available in `clustrd`, the methods are comparable in terms of both fit and ARI. Nevertheless, our implementations consistently outperform all alternatives in terms of runtime.

Figure (3 - 8) provide a visual summary of the results reported in Table 6, illustrating not only the central tendencies but also the variability across the 100 simulation replicates for each scenario.

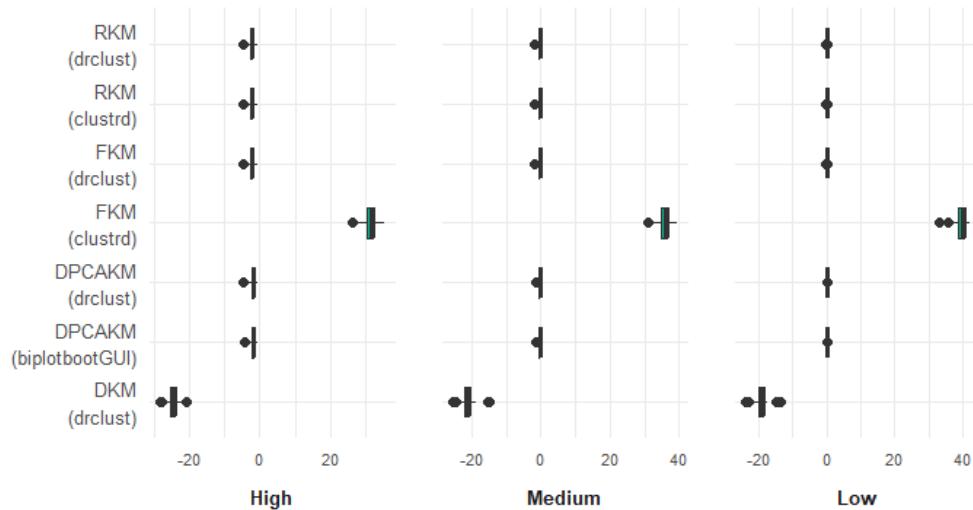


Figure 5: Boxplots of the $f^* - f$ results in Table 6

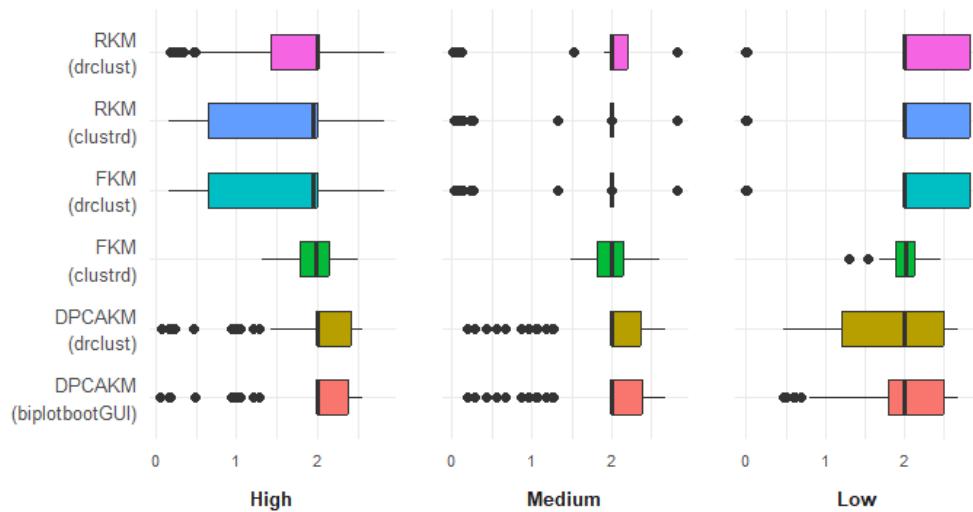


Figure 6: Boxplots of the $\|A - A^*\|^2$ metric results in Table 6

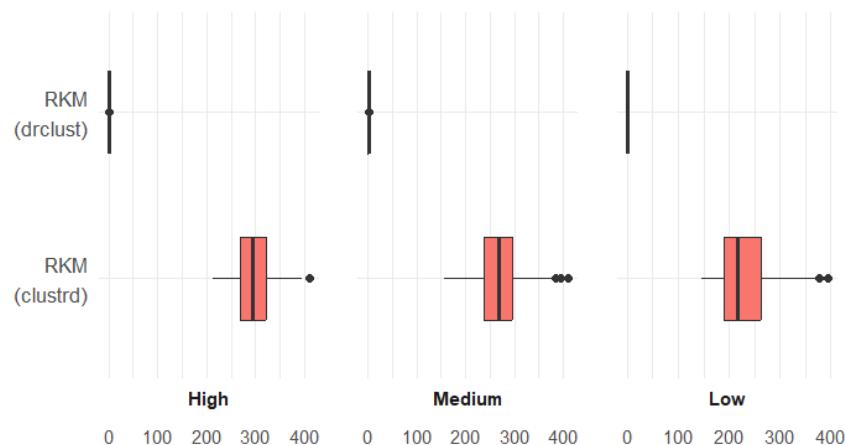


Figure 7: Boxplots of the runtime results in Table 6, for the RKM

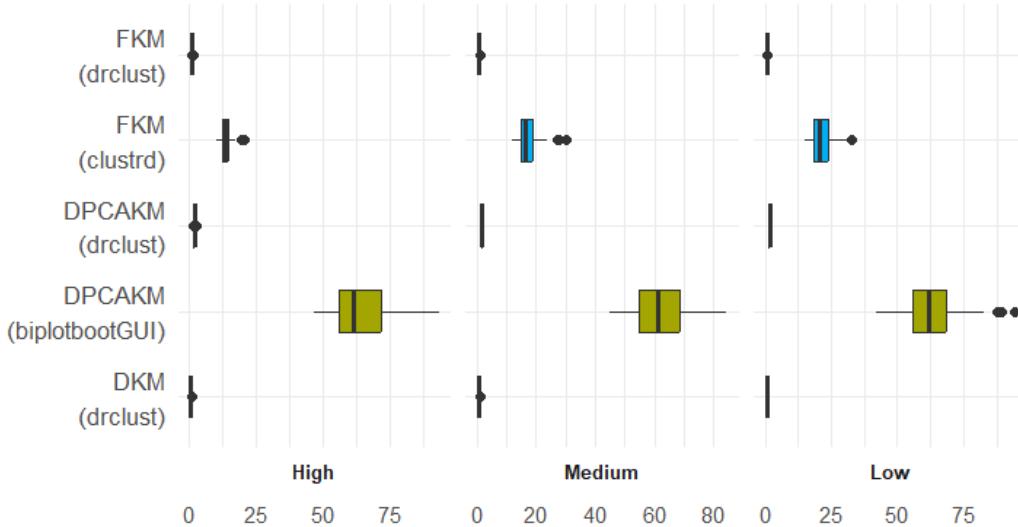


Figure 8: Boxplots of the runtime metric results in Table 6, for DKM, DPCAKM, FKM

5 Application on real data

The six statistical models implemented (Table 2) have a binary argument `print` which, if set to one, displays at the end of the execution the main statistics. In the following examples, such results are shown, using as dataset the same used by Vichi and Kiers (2001) and made available in `clustrd` (Markos et al., 2019) and named `macro`, which has been standardized by setting the argument `prep=1`, which is done by default by all the techniques. Moreover, the commands reported in each example do not specify all the arguments available for the function, for which the default values have been kept.

The first example refers to the DKM (Vichi, 2001). As shown, the output contains the fit expressed as the percentage of the total deviance (i.e., $\|\mathbf{X}\|^2$) captured by the between deviance of the model, implementing the fit measures in (Table 5). The second output is the centroid matrix $\bar{\mathbf{Y}}$, which describes the K centroids in the Q -dimensional space induced by the partition of the variables and its related variable-means. What follows are the sizes and within deviances of each unit cluster and each variable cluster. Finally, it shows the `pseudoF` (Caliński and Harabasz, 1974) index, which is always computed for the partition of the units. Please note that the data matrix provided to each function implemented in the package needs to be in matrix format.

```
# Macro dataset (Vichi & Kiers, 2001)
library(clustrd)
data(macro)
macro <- as.matrix(macro)
# DKM
> dkm <- doublekm(X = macro, K = 5, Q = 3, print = 1)

>> Variance Explained by the DKM (% BSS / TSS): 44.1039

>> Centroid Matrix (Unit-centroids x Variable-centroids):

          V-Clust 1   V-Clust 2   V-Clust 3
U-Clust 1  0.1282052 -0.31086968 -0.4224182
U-Clust 2  0.0406931 -0.08362029  0.9046692
U-Clust 3  1.4321347  0.51191282 -0.7813761
U-Clust 4 -0.9372541  0.22627768  0.1175189
U-Clust 5  1.2221058 -2.59078258 -0.1660691
```

```
>> Unit-clusters:
```

	U-Clust 1	U-Clust 2	U-Clust 3	U-Clust 4	U-Clust 5
Size	8	4	4	3	1
Deviance	23.934373	31.737865	5.878199	4.844466	0.680442

```
>> Variable-clusters:
```

	V-Clust 1	V-Clust 2	V-Clust 3
Size	3	2	1
Deviance	40.832173	23.024249	3.218923

```
>> pseudoF Statistic (Calinski-Harabasz): 2.23941
```

The second example shows as output the main quantities computed for the `redkm` (De Soete and Carroll, 1994). Differently from the DKM where the variable reduction is operated via averages, the RKM does this via PCA leading to a better overall fit altering also the final unit-partition, as observable from the sizes or deviances.

Additionally from the DKM example, the RKM also provides the loading matrix which projects the J -dimensional centroids in the Q -dimensional subspace. Another important difference is the summary of the latent factors: this table shows the information captured by the principal components with respect to the original data. In this sense, the output allows to distinguish between the loss due to the variable reduction (accounted in this table) and the overall loss of the algorithm (which accounts for the loss in the reduction of the units and the one due to the reduction of the variables, reported in the first line of the output).

```
# RKM
> rkm <- redkm(X = macro, K = 5, Q = 3, print = 1)
```

```
>> Variance Explained by the RKM (% BSS / TSS): 55.0935
```

```
>> Matrix of Centroids (Unit-centroids x Principal Components):
```

	PC 1	PC 2	PC 3
Clust 1	-1.3372534	-1.1457414	-0.6150841
Clust 2	1.8834878	-0.0853912	-0.8907303
Clust 3	0.5759906	0.4187003	0.3739608
Clust 4	-0.9538864	1.2392976	0.3454186
Clust 5	1.0417952	-2.2197178	3.0414445

```
>> Unit-clusters:
```

	Clust 1	Clust 2	Clust 3	Clust 4	Clust 5
Size	5	5	5	4	1
Deviance	26.204374	9.921313	11.231563	6.112386	0.418161

```
>> Loading Matrix (Manifest Variables x Latent Variables):
```

	PC 1	PC 2	PC 3
GDP	-0.5144915	-0.04436269	0.08985135
LI	-0.2346937	-0.01773811	-0.86115069
UR	-0.3529363	0.53044730	0.28002534
IR	-0.4065339	-0.42022401	-0.17016203
TB	0.1975072	0.69145440	-0.36710245

```
NNS  0.5927684 -0.24828525 -0.09062404
```

>> Summary of the latent factors:

	Explained Variance	Expl. Var. (%)	Cumulated Var.	Cum. Var (%)
PC 1	1.699343	28.322378	1.699343	28.322378
PC 2	1.39612	23.268663	3.095462	51.591041
PC 3	1.182372	19.706208	4.277835	71.297249

>> pseudoF Statistic (Calinski-Harabasz): 4.29923

The `factkm` ([Vichi and Kiers, 2001](#)) has the same output structure of the `redkm`. It exhibits, for the same data and hyperparameters, a similar fit (overall and variable-wise). However, the unit-partition, as well as the latent variables are different. This difference can be (at least) partially justified by the difference in the objective function, which is most evident in the assignment step.

```
# factorial KM
> fkm <- factkm(X = macro, K = 5, Q = 3, print = 1, rot = 1)
```

>> Variance Explained by the FKM (% BSS / TSS): 55.7048

>> Matrix of Centroids (Unit-centroids x Principal Components):

	PC 1	PC 2	PC 3
Clust 1	-0.7614810	2.16045496	-1.21025666
Clust 2	1.1707159	-0.08840133	-0.29876729
Clust 3	-0.9602731	-1.33141866	0.02370092
Clust 4	1.0782934	1.17952330	3.59632116
Clust 5	-1.7634699	0.65075735	0.46486440

>> Unit-clusters:

	Clust 1	Clust 2	Clust 3	Clust 4	Clust 5
Size	9	5	3	2	1
Deviance	6.390576	2.827047	5.018935	3.215995	0

>> Loading Matrix (Manifest Variables x Latent Variables):

	PC 1	PC 2	PC 3
GDP	-0.6515084	-0.1780021	0.37482509
LI	-0.3164139	0.1809559	-0.68284917
UR	-0.2944864	-0.5235492	0.01561022
IR	-0.3316254	0.5884434	-0.22101070
TB	0.1848264	-0.5367239	-0.57166730
NNS	0.4945307	0.1647067	0.13164438

>> Summary of the latent factors:

	Explained Variance	Expl. Var. (%)	Cumulated Var.	Cum. Var (%)
PC 1	1.68496	28.082675	1.68496	28.082675
PC 2	1.450395	24.173243	3.135355	52.255917
PC 3	1.079558	17.992635	4.214913	70.248552

>> pseudoF Statistic (Calinski-Harabasz): 4.26936

`dpcakm` ([Vichi and Saporta, 2009](#)) shows the same output as RKM and FKM. The partition of the variables, described by the V term in (29) - (30), is readable within the loading matrix,

considering a 1 for each non-zero value. For the `iris` dataset, the additional constraint $\mathbf{A} = \mathbf{B}\mathbf{V}$ does not cause a significant decrease in the objective function. The clusters, however, differ from the previous cases as well.

```
# K-means DPCA
> cdPCA <- dpcakm(X = macro, K = 5, Q = 3, print = 1)

>> Variance Explained by the DPCAkm (% BSS / TSS): 54.468

>> Matrix of Centroids (Unit-centroids x Principal Components):

          PC 1      PC 2      PC 3
Clust 1  0.6717536  0.01042978 -2.7309458
Clust 2  3.7343724 -1.18771685  0.6320673
Clust 3 -0.6729575 -1.80822745  0.7239541
Clust 4 -0.2496002  1.54537904  0.5263009
Clust 5 -0.1269212 -0.12464388 -0.1748282

>> Unit-clusters:
      Clust 1  Clust 2  Clust 3  Clust 4  Clust 5
Size      7       6       4       2       1
Deviance 3.816917 2.369948 1.14249  4.90759  0

>> Loading Matrix (Manifest Variables x Latent Variables):

          PC 1      PC 2      PC 3
GDP    0.5567605  0.0000000   0
LI     0.0000000  0.7071068   0
UR    0.5711396  0.0000000   0
IR     0.0000000  0.0000000   1
TB     0.0000000  0.7071068   0
NNS   -0.6031727  0.0000000   0

>> Summary of the latent factors:
      Explained Variance Expl. Var. (%) Cumulated Var. Cum. Var. (%)
PC 1 1                  16.666667    1        16.666667
PC 2 1.703964           28.399406    2.703964  45.066073
PC 3 1.175965           19.599421    3.87993   64.665494

>> pseudoF Statistic (Calinski-Harabasz): 3.26423
```

For the `dispca` ([Vichi and Saporta, 2009](#)), the output is mostly similar (except for the part of unit-clustering) to the ones already shown. Nevertheless, because the focus here is exclusively on the variable reduction process, some additional information is reported in the summary of the latent factors. Indeed, because a single principal component summarises a subset of manifest variables, the variance of the second component related to each of the subsets, along with the [Cronbach \(1951\)](#) Alpha index is computed, in order for the user to know when the evidence supports such strategy of dimensionality reduction. As mentioned, this function, like in the DPCAkm case, as well as the DFA case, it allows to constrain a subset of the J variables to belong to the same cluster. In the example that follows, the first two manifest variables are constrained to contribute to the same principal component (which is confirmed by the output `A`). Note that the manifest variables that have indices (column-position in the data matrix) in correspondence of the zeros in `constr` remain unconstrained.

```
# DPCA
# Impose GDP and LI to be in the same cluster
```

```

> out <- dispca(X = macro, Q = 3, print = 1, constr = c(1,1,0,0,0,0))

>> Variance explained by the DPCA (% BSS / TSS)= 63.9645

>> Loading Matrix (Manifest Variables x Latent variables)

      PC 1       PC 2       PC 3
GDP  0.0000000  0.0000000  0.7071068
LI   0.0000000  0.0000000  0.7071068
UR   -0.7071068 0.0000000  0.0000000
IR   0.0000000 -0.7071068 0.0000000
TB   0.0000000  0.7071068 0.0000000
NNS  0.7071068 0.0000000 0.0000000

>> Summary of the latent factors:
      Explained Variance Expl. Var. (%) Cumulated Var.
PC 1           1.388294    23.13824    1.388294
PC 2           1.364232    22.73721    2.752527
PC 3           1.085341    18.08902    3.837868
      Cum. Var (%) Var. 2nd component Cronbach's Alpha
PC 1           23.13824    0.6117058    -1.269545
PC 2           45.87544    0.6357675    -1.145804
PC 3           63.96447    0.9146585     0.157262

```

The disfa ([Vichi, 2017](#)), by assuming a probabilistic underlying model, allows additional evaluation metrics and statistics as well. The overall objective function is not directly comparable with the other ones, and is expressed in absolute (not relative, like in the previous cases) terms. The χ^2 (X^2), along with BIC, AIC and RMSEA allow a robust evaluation of the results in terms of fit/parsimony. Additionally to the DPCA case, for each variable, the function displays the commonality with the factors, providing a standard error, as well as an associated p -value for the estimate.

It is possible to assess by comparing the loading matrix in the DPCA case with the DFA one, the similarity in terms of latent variables. Part of the difference can be justified (besides the well-known distinctions between PCA and FA) with the method used to compute each factor. While in all the previous cases, the eigendecomposition has been employed for this purpose, the DFA makes use of the power iteration method for the computation of the loading matrix ([Hotelling, 1933](#)).

```

# disjoint FA
> out <- disfa(X = macro, Q = 3, print = 1)
>> Discrepancy of DFA: 0.296499

>> Summary statistics:

      Unknown Parameters Chi-square Degrees of Freedom BIC
      9                   4.447531    12                  174.048102
      AIC                 RMSEA
      165.086511  0.157189

>> Loading Matrix (Manifest Variables x Latent Variables)

      Factor 1 Factor 2 Factor 3
GDP  0.5318618      0  0.0000000
LI   0.0000000      1  0.0000000
UR   0.5668542      0  0.0000000
IR   0.0000000      0  0.6035160
TB   0.0000000      0 -0.6035152

```

```
NNS -0.6849942      0  0.0000000
```

>> Summary of the latent factors:

	Explained Variance	Expl. Var. (%)	Cum. Var	Cum. Var (%)
Factor 1	1.0734177	17.89029	1.073418	17.89029
Factor 2	1.0000000	16.66667	2.073418	34.55696
Factor 3	0.7284622	12.14104	2.801880	46.69800
	Var. 2nd component	Cronbach's Alpha		
Factor 1	0.7001954	-0.6451803		
Factor 2	0.0000000	1.0000000		
Factor 3	0.6357675	-1.1458039		

>> Detailed Manifest-variable - Latent-factor relationships

	Associated Factor	Corr.	Coeff.	Std. Error	Pr(p> Z)
GDP	1	0.5318618	0.1893572	0.0157923335	
LI	2	1.0000000	0.0000000	0.0000000000	
UR	1	0.5668542	0.1842113	0.0091557523	
IR	3	0.6035160	0.1782931	0.0048411219	
TB	3	-0.6035152	0.1782932	0.0048411997	
NNS	1	-0.6849942	0.1629084	0.0008606488	
	Var. Error	Communality			
GDP	0.7171230	0.2828770			
LI	0.0000000	1.0000000			
UR	0.6786764	0.3213236			
IR	0.6357684	0.3642316			
TB	0.6357695	0.3642305			
NNS	0.5307830	0.4692170			

In practice, usually the K and Q hyper-parameters are not known a priori. In such case, a possible tool that allows to investigate plausible values for Q is the Kaiser criterion ([Kaiser, 1960](#)), in R, `kaiserCrit`), takes as a single argument the dataset and outputs a message, as well as a scalar output indicating the number of the optimal components based on this rule.

```
# Kaiser criterion for the choice of Q, the number of latent components
> kaiserCrit(X = macro)
```

The number of components suggested by the Kaiser criterion is: 3

For selecting the number of clusters, K, one of the most commonly used indices is the *pseudoF* statistic, which, however, tends to underestimate the optimal number of clusters. To address this limitation, a "relaxed" version, referred to as *apseudoF*, has been implemented. The *apseudoF* procedure computes the standard *pseudoF* index over a range of possible values up to *maxK*. If a higher value of K yields a *pseudoF* that is less than *tol* · *pseudoF* (compared to the maximum value suggested by the plain *pseudoF*), then *apseudoF* selects this alternative K as the optimal number of clusters. Additionally, it generates a plot of the *pseudoF* values computed across the specified K range. Given the hybrid nature of the proposed methods, the function also requires specifying the clustering model to be used: 1 = *doublekm*, 2 = *redkm*, 3 = *factkm*, 4 = *dpcakm*. Furthermore, the number of components, Q, must be provided, as it also influences the final quality of the resulting partition.

```
> apseudoF(X = macro, maxK=10, tol = 0.05, model = 2, Q = 3)
The optimal number of clusters based on the pseudoF criterion is: 5
```

While this index has been thought for one-mode clustering methods, ([Rocci and Vichi, 2008](#)) extended it for two-mode clustering methods, allowing to apply it for methods like

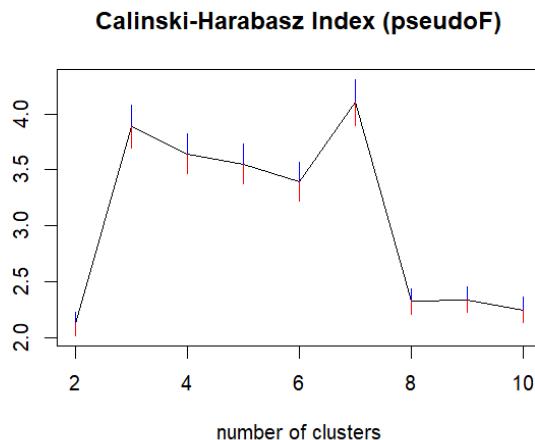


Figure 9: Interval-pseudoF polygonal chain

the `doublekm`. The `dpseudoF` function implements it and, besides the dataset, one provides the maximum K and Q values.

```
> dpseudoF(X = macro, maxK = 10, maxQ = 5)
      Q = 2      Q = 3      Q = 4      Q = 5
K = 2  38.666667 22.800000 16.000000 12.222222
K = 3  22.800000 13.875000 9.818182 7.500000
K = 4  16.000000 9.818182 6.933333 5.263158
K = 5  12.222222 7.500000 5.263158 3.958333
K = 6  9.818182 6.000000 4.173913 3.103448
K = 7  8.153846 4.950000 3.407407 2.500000
K = 8  6.933333 4.173913 2.838710 2.051282
K = 9  6.000000 3.576923 2.400000 1.704545
K = 10 5.263158 3.103448 2.051282 1.428571
```

Here, the indices of the maximum value within the matrix are chosen as the best Q and K values.

Just by providing the centroid matrix, one can check how those are related. Such information is usually not provided by partitive clustering methods, but rather for the hierarchical ones. Nevertheless, it is always possible to construct a distance matrix based on the centroids and represent it via a dendrogram, using an arbitrary distance. The `centree` function does exactly this, using the the [Ward \(1963\)](#) distance, which corresponds to the squared Euclidean one. In practice, one provides as an argument the output of one of the 4 methods performing clustering.

```
> out <- factkm(X = macro, K = 10, Q = 3)
> centree(drclust_out = out)
```

If, instead, one wants to assess visually the quality of the obtained partition, there are another instrument typically used for this purpose. The silhouette ([Rousseeuw, 1987](#)), besides summarizing this numerically, allows to also graphically represent it. By employing `cluster` for the computational part and `factoextra` for the graphical part, `silhouette` takes as argument the output of one of the four `drclust` clustering methods and the dataset, returning the results of the two functions with just one command.

```
# Note: The same data must be provided to dpcakm and silhouette
> out <- dpcakm(X = macro, K = 5, Q = 3)
> silhouette(X = macro, drclust_out = out)
```

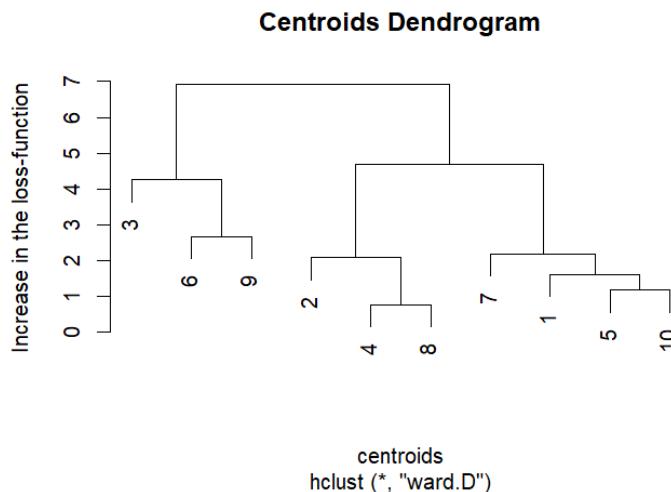


Figure 10: Dendrogram of a 10-centroids

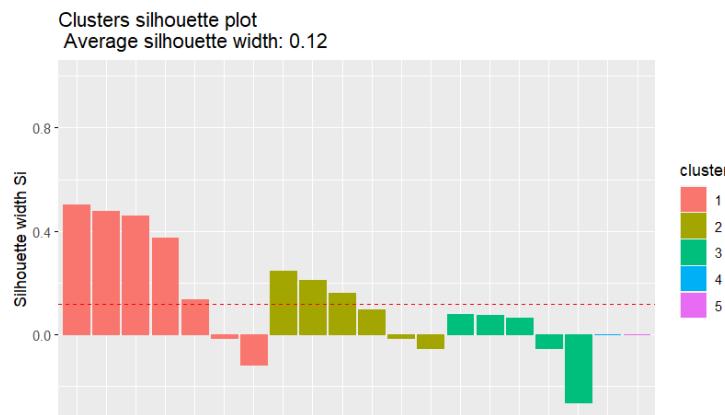


Figure 11: Silhouette of a DPCA KM solution

As can be seen in Figure 11, the average silhouette width is also displayed as a scalar above the plot.

A purely graphical tool used to assess the dis/homogeneity of the groups is the heatmap. By employing the [pheatmap](#) library (Kolde, 2019) and the result of `doublekm`, `redkm`, `factkm` or `dpcakm`, the function orders each cluster of observations in ascending order with regard to the distance between observation and cluster to which it has been assigned. After doing so for each group, groups are sorted based on the distance between their centroid and the grand mean (i.e., the mean of all observations). The `heatm` function allows to obtain such result. Figure 11 represents its graphical output.

```
# Note: The same data must be provided to dpcakm and silhouette
> out <- doublekm(X = macro, K = 5, Q = 3)
> heatm(X = macro, drclust_out = out)
```

Biplots and parallel coordinates plots can be obtained based on the output of the techniques in the proposed package by means of few instructions, using libraries available on CRAN, such as: [ggplot2](#) Wickham et al. (2024), `grid` (which now became a base package, [dplyr](#) Wickham et al. (2023) and [GGally](#) by Schloerke et al. (2024)). Therefore, the user can easily visualize the subspaces provided by the statistical techniques. In future versions of the package, the two functions will be available as built-in. Currently, for the biplot, we have:

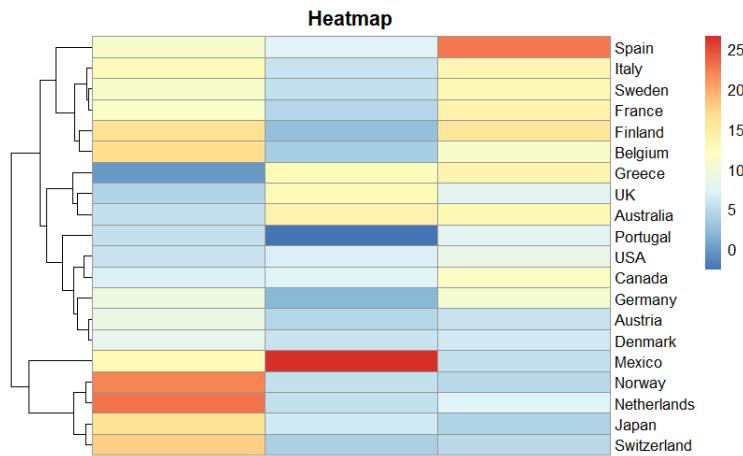


Figure 12: heatmap of a double-KM solution

```

library(ggplot2)
library(grid)
library(dplyr)

out <- factkm(macro, K = 2, Q = 2, Rndstart = 100)

# Prepare data
Y <- as.data.frame(macro%*%out$A); colnames(Y) <- c("Dim1", "Dim2")
Y$cluster <- as.factor(cluster(out$U))

arrow_scale <- 5
A <- as.data.frame(out$A)[, 1:2] * arrow_scale
colnames(A) <- c("PC1", "PC2")
A$var <- colnames(macro)

# Axis limits
lims <- range(c(Y$Dim1, Y$Dim2, A$PC1, A$PC2)) * 1.2

# Circle
circle <- data.frame(x = cos(seq(0, 2*pi, length.out = 200)) * arrow_scale,
                      y = sin(seq(0, 2*pi, length.out = 200)) * arrow_scale)

ggplot(Y, aes(x = Dim1, y = Dim2, color = cluster)) +
  geom_point(size = 2) +
  geom_segment(
    data = A, aes(x = 0, y = 0, xend = PC1, yend = PC2),
    arrow = arrow(length = unit(0.2, "cm")), inherit.aes = FALSE, color = "gray40"
  ) +
  geom_text(
    data = A, aes(x = PC1, y = PC2, label = colnames(macro)), inherit.aes = FALSE,
    hjust = 1.1, vjust = 1.1, size = 3
  ) +
  geom_path(data = circle, aes(x = x, y = y), inherit.aes = FALSE,
            linetype = "dashed", color = "gray70") +
  coord_fixed(xlim = lims, ylim = lims) +
  labs(x = "Component 1", y = "Component 2", title = "Biplot") +
  theme_minimal()

```

which leads to the result shown in Figure 13.

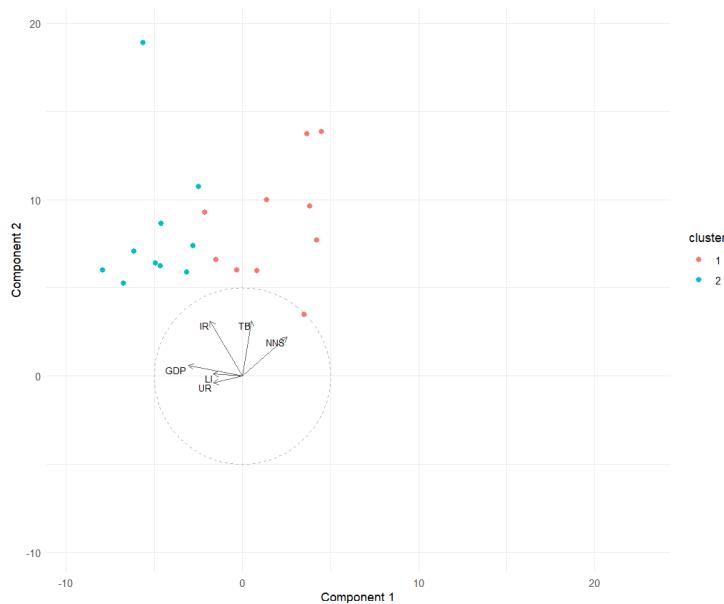


Figure 13: Biplot of a FKM solution

By using essential information in the output provided by `factkm`, we are able to see the cluster of each observation, represented in the estimated subspace induced by \mathbf{A} , as well as the relationships between observed and latent variables via the arrows.

In order to obtain the parallel coordinates plot, a single instruction is sufficient, based on the same output as a starting point.

```
library(GGally)
out <- factkm(macro, K = 3, Q = 2, Rndstart = 100)
ggparcoord(
  data = Y, columns = 1:(ncol(Y)-1),
  groupColumn = "cluster", scale = "uniminmax",
  showPoints = FALSE, alphaLines = 0.5
) +
  theme_minimal() +
  labs(title = "Parallel Coordinate Plot",
       x = "Variables", y = "Normalized Value")
```

For FKM applied on `macro` dataset, the output is reported in figure 14.

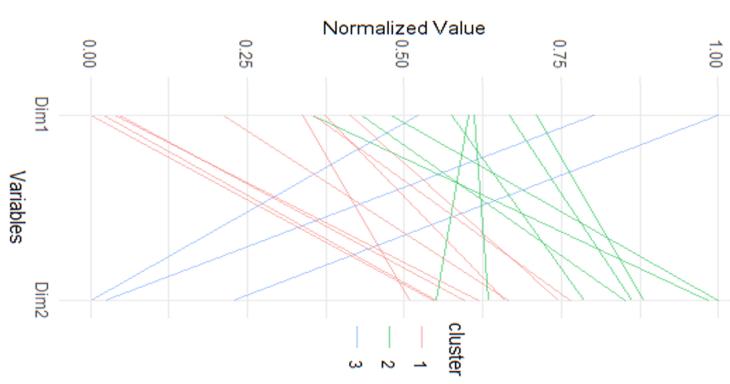


Figure 14: Parallel coordinates plot of a FKM solution

6 Conclusions

This work presents an R library that implements techniques of joint dimensionality reduction and clustering. Some of them are already implemented by other packages. In general, the performance between the proposed implementations and the earlier ones is very close, except for the FKM, where the new one is always better for the metrics considered here. As an element of novelty, the empty cluster(s) issue that may occur in the estimation process has been addressed by applying 2-means on the cluster with the highest deviance, preserving the monotonicity of the algorithm and providing slightly better results, at a higher computational costs.

The implementation of the two dimensionality reduction methods, `dispca` and `disfa`, as well as `doublekkm` offered by our library are novel in the sense that they do not find previous implementation in R. Besides the methodological difference between these last two, the latent variables are computed differently: the former uses the well-known eigen-decomposition, while the latter adopts the power method. In general, by implementing all the models in C/C++, the speed advantage has been shown to be remarkable compared to all the existing comparisons. These improvements allow the application of the techniques on datasets that are relatively large, to obtain results in reasonable amounts of time. Some additional functions have been implemented for the purpose of helping in the choice process for the values of the hyperparameters. Additionally, they can also be used as an assessment tool in order to evaluate the quality of the results provided by the implementations.

References

- T. Caliński and J. Harabasz. A dendrite method for cluster analysis. *Communications in Statistics*, 3(1):1–27, 1974. URL <https://doi.org/10.1080/03610927408827101>. [p112, 119]
- R. B. Cattell. Factor analysis: An introduction to essentials i. the purpose and underlying models. *Biometrics*, 21(1):190–215, 1965. URL <https://doi.org/10.2307/2528364>. [p105]
- M. Charrad, N. Ghazzali, V. Boiteau, and A. Niknafs. Nbclust: An R package for determining the relevant number of clusters in a data set. *Journal of Statistical Software*, 61(6):1–36, 2014. URL <https://doi.org/10.18637/jss.v061.i06>. [p104]
- L. J. Cronbach. Coefficient alpha and the internal structure of tests. *Psychometrika*, 16(3): 297–334, 1951. URL <https://doi.org/10.1007/BF02310555>. [p112, 122]
- G. De Soete and J. D. Carroll. K-means clustering in a low-dimensional Euclidean space. In E. Diday, Y. Lechevallier, M. Schader, P. Bertrand, and B. Burtschy, editors, *New Approaches in Classification and Data Analysis*, chapter 24. Springer, Berlin, Heidelberg, 1994. URL https://doi.org/10.1007/978-3-642-51175-2_24. [p104, 109, 111, 120]
- W. S. DeSarbo, K. Jedidi, K. Cool, and D. Schendel. Simultaneous multidimensional unfolding and cluster analysis: An investigation of strategic groups. *Marketing Letters*, 2:129–146, 1990. URL <https://doi.org/10.1007/BF00436033>. [p104]
- S. Dray and A.-B. Dufour. The ade4 package: Implementing the duality diagram for ecologists. *Journal of Statistical Software*, 22(4):1–20, 2007. URL <https://doi.org/10.18637/jss.v022.i04>. [p104]
- 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>. [p111]
- 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>. [p111]

- H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441, and 498–520, 1933. URL <https://doi.org/10.1037/h0071325>. [p114, 123]
- L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985. URL <https://doi.org/10.1007/BF01908075>. [p115]
- H. F. Kaiser. The application of electronic computers to factor analysis. *Educational and Psychological Measurement*, 20(1):141–151, April 1960. URL <https://doi.org/10.1177/001316446002000116>. [p112, 124]
- A. Kassambara. *factoextra: Extract and Visualize the Results of Multivariate Data Analyses*. R package version 1.0.7, 2022. URL <https://cran.r-project.org/package=factoextra>. [p104, 111, 112]
- R. Kolde. *pheatmap: Pretty Heatmaps*. R package 1.0.12, 2019. URL <https://cran.r-project.org/package=pheatmap>. [p111, 112, 126]
- D. N. Lawley and A. E. Maxwell. Factor analysis as a statistical method. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 12(3):209–229, 1962. URL <https://doi.org/10.2307/2986915>. [p105]
- S. Lê, J. Josse, and F. Husson. Factominer: An R package for multivariate analysis. *Journal of Statistical Software*, 25(1):1–18, 2008. URL <https://doi.org/10.18637/jss.v025.i01>. [p104]
- M. Maechler, P. Rousseeuw, A. Struyf, M. Hubert, and K. Hornik. *cluster: Cluster Analysis Basics and Extensions*. R package version 2.1.6, 2023. URL <https://CRAN.R-project.org/package=cluster>. [p111, 112]
- A. Markos, A. I. D’Enza, and M. van de Velden. Beyond tandem analysis: Joint dimension reduction and clustering in R. *Journal of Statistical Software*, 91(10):1–24, 2019. URL <https://doi.org/10.18637/jss.v091.i10>. [p104, 119]
- J. McQueen. Some methods for classification and analysis of multivariate observations. *Computer and Chemistry*, 4:257–272, 1967. URL <https://www.cs.cmu.edu/~bhiksha/courses/mlsp.fall2010/class14/macqueen.pdf>. [p108]
- A. B. Nieto Librero and A. Freitas. biplotbootgui: Bootstrap on classical biplots and clustering disjoint biplot, 2023. URL <https://cran.r-project.org/web/packages/biplotbootGUI/index.html>. [p104]
- C. E. Pardo and P. C. Del Campo. Combination of factorial methods and cluster analysis in R: The package factoclass. *Revista Colombiana de Estadística*, 30(2):231–245, 2007. URL <https://revistas.unal.edu.co/index.php/estad/article/view/29478>. [p104]
- K. Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. URL <https://doi.org/10.1080/14786440109462720>. [p105]
- R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. URL <http://www.R-project.org/>. [p104]
- W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971. URL <https://doi.org/10.2307/2284239>. [p112]
- R. Rocci and M. Vichi. Two-mode multi-partitioning. *Computational Statistics & Data Analysis*, 52(4):1984–2003, 2008. URL <https://doi.org/10.1016/j.csda.2007.06.025>. [p104, 112, 124]

- P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. ISSN 0377-0427. URL [https://doi.org/10.1016/0377-0427\(87\)90125-7](https://doi.org/10.1016/0377-0427(87)90125-7). [p125]
- B. Schloerke, D. Cook, H. Hofmann, H. Wickham, S. Garnier, K. Morrissey, and A. Elberg. *GGally: Extension to 'ggplot2'*. R package version 2.1.2, 2024. URL <https://CRAN.R-project.org/package=GGally>. [p126]
- M. E. Timmerman, E. Ceulemans, H. A. Kiers, and M. Vichi. Factorial and reduced k-means reconsidered. *Computational Statistics & Data Analysis*, 54(7):1858–1871, 2010. ISSN 0167-9473. URL <https://doi.org/10.1016/j.csda.2010.02.009>. [p104]
- M. Vichi. Double k-means clustering for simultaneous classification of objects and variables. In S. Borra, R. Rocci, M. Vichi, and M. Schader, editors, *Advances in Classification and Data Analysis*, chapter 6. Springer, Berlin, Heidelberg, 2001. URL https://doi.org/10.1007/978-3-642-59471-7_6. [p104, 108, 111, 119]
- M. Vichi. Disjoint factor analysis with cross-loadings. *Advances in Data Analysis and Classification*, 11(4):563–591, 2017. URL <https://doi.org/10.1007/s11634-016-0263-9>. [p103, 107, 111, 123]
- M. Vichi and H. A. L. Kiers. Factorial k-means analysis for two-way data. *Computational Statistics & Data Analysis*, 37(1):49–64, 2001. ISSN 0167-9473. URL [https://doi.org/10.1016/S0167-9473\(00\)00064-5](https://doi.org/10.1016/S0167-9473(00)00064-5). [p104, 109, 111, 119, 121]
- M. Vichi and G. Saporta. Clustering and disjoint principal component analysis. *Computational Statistics & Data Analysis*, 53(8):3194–3208, 2009. ISSN 0167-9473. URL <https://doi.org/10.1016/j.csda.2008.05.028>. [p103, 104, 105, 110, 111, 121, 122]
- M. Vichi, D. Vicari, and H. A. L. Kiers. Clustering and dimension reduction for mixed variables. *Behaviormetrika*, pages 243–269, 2019. URL <https://doi.org/10.1007/s41237-018-0068-6>. [p105]
- W. R. Revelle. psych: Procedures for personality and psychological research, 2017. URL <https://cran.r-project.org/web/packages/psych/index.html>. [p104]
- J. H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963. URL <https://doi.org/10.1080/01621459.1963.10500845>. [p125]
- H. Wickham, R. François, L. Henry, and K. Müller. *dplyr: A Grammar of Data Manipulation*. R package version 1.1.4, 2023. URL <https://CRAN.R-project.org/package=dplyr>. [p126]
- H. Wickham, W. Chang, L. Henry, T. L. Pedersen, K. Takahashi, C. G. Wilke, H. Yutani, and D. Dunnington. *ggplot2: Elegant Graphics for Data Analysis*. R package version 3.4.4, 2024. URL <https://CRAN.R-project.org/package=ggplot2>. [p126]
- M. Yamamoto and H. Hwang. A general formulation of cluster analysis with dimension reduction and subspace separation. *Behaviormetrika*, 41:115–129, 2014. URL <https://doi.org/10.2333/bhmk.41.115>. [p104]
- H. Zou, T. Hastie, and R. Tibshirani. Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15(2):265–286, 2006. doi: <https://doi.org/10.1198/106186006X113430>. [p105]

Ionel Prunila

Department of Statistical Sciences, Sapienza University of Rome

P.le Aldo Moro 5, 00185 Rome

Italy

ORCID: 0009-0009-3773-0481
ionel.prunila@uniroma1.it

Maurizio Vichi
Department of Statistical Sciences, Sapienza University of Rome
P.le Aldo Moro 5, 00185 Rome
Italy
ORCID: 0000-0002-3876-444X
maurizio.vichi@uniroma1.it

AcceptReject: An R Package for Acceptance-Rejection Method

by Pedro Rafael Diniz Marinho, Vera L. D. Tomazella, and Pedro Luiz Ramos

Abstract The AcceptReject package, available for the R programming language on the Comprehensive R Archive Network (CRAN), versioned and maintained on GitHub, offers a simple and efficient solution for generating pseudo-random observations of discrete or continuous random variables using the acceptance-rejection method. This method provides a viable alternative for generating pseudo-random observations in univariate distributions when the inverse of the cumulative distribution function is not in closed form or when suitable transformations involving random variables that we know how to generate are unknown, thereby facilitating the generation of observations for the variable of interest. The package is designed to be simple, intuitive, and efficient, allowing for the rapid generation of observations and supporting multicore parallelism on Unix-based operating systems. Some components are written using C++, and the package maximizes the acceptance probability of the generated observations, resulting in even more efficient execution. The package also allows users to explore the generated pseudo-random observations by comparing them with the theoretical probability mass function or probability density function and to inspect the underlying probability density functions that can be used in the method for generating observations of continuous random variables. This article explores the package in detail, discussing its functionalities, benefits, and practical applications, and provides various benchmarks in several scenarios.

1 Introduction

The class of Monte Carlo methods and algorithms is a powerful and versatile computational technique that has been widely used in a variety of fields, from physics, statistics, and economics to biology and areas such as social sciences. Through the modeling of complex systems and the performance of stochastic experiments, Monte Carlo simulation allows researchers to explore scenarios that would be impractical or impossible to investigate through traditional experimental methods.

A critical component of any Monte Carlo simulation is the ability to generate pseudo-random observations of a sequence of random variables that follow a given probability distribution. In many cases, these variables can be discrete or continuous, and the ability to generate observations from a sequence of random variables efficiently and accurately is fundamental in simulation studies.

There are several techniques to generate observations from a sequence of random variables, many of which depend on the availability of a closed-form quantile function of the distribution of interest or knowledge of some transformation involving random variables that we know how to generate observations from. Depending on the distribution one wishes to generate observations from, the inverse of the cumulative distribution function (quantile function) does not have a closed form, and in many cases, we do not know transformations involving random variables that we know how to generate observations from, and thus, we can generate observations from the random variable of interest. In this sense, computational methods, such as the Acceptance-Rejection Method (ARM), proposed by John von Neumann in 1951, see [von Neumann \(1951\)](#), are a viable alternative for generating observations in the univariate context. For example, in the current scenario, where various distributions and probability distribution generators are being proposed, with these generators being functions that, from the knowledge of a probability distribution, can generate a new probability distribution, the ARM is widely used because it proves to be useful for generating pseudo-random observations of random variables of a discrete or continuous nature.

In the case of univariate distributions, the ARM is an effective technique for generating pseudo-random observations of random variables and has several advantages over other methods such as the Metropolis-Hastings (MH) algorithm, for example. The ARM, often also called the Rejection Method, is an algorithm that can be easily parallelized. Moreover, this method is not sensitive to initial parameters, as in the case of the MH algorithm, and leads to observations that are not dependent. Additionally, in the ARM, it is not necessary to perform a “burn-in,” and it is not required to wait for the algorithm to reach a stationary level, as in the case of MH. The ARM, when well implemented, can be a great alternative for generating observations of random variables in the univariate case and can be considered in many situations before opting for computationally more expensive methods such as MH or Gibbs sampling.

The **AcceptReject** package (Marinho and Tomazella, 2024) for the R programming language, also available and maintained at <https://github.com/prdm0/AcceptReject/>, was specifically developed to handle these challenges. It offers a simple and efficient solution for generating pseudo-random observations of discrete or continuous random variables in univariate distributions, using the ARM to generate pseudo-random observations of a sequence of random variables whose probability mass function (discrete case) or probability density function (continuous case) are complex or poorly explored. The library has detailed documentation and vignettes to assist user understanding. On the package’s website, you can find usage examples and the complete documentation of the package <https://prdm0.github.io/AcceptReject/>, in addition to the vignettes.

The design of the **AcceptReject** package is simple and intuitive, allowing users to generate observations of random variables quickly and efficiently, and in a parallelized manner, using multicore parallelism on Unix-based operating systems. The package also performs excellently on Windows, as much of the library’s performance comes from optimizing the probability of accepting observations of an auxiliary random variable as observations of the random variable of interest. Additionally, some points have been further optimized using the **Rcpp** (Eddelbuettel et al., 2024) and **RcppArmadillo** (Eddelbuettel and Sanderson, 2014) libraries.

The library, through the `accept_reject()` function https://prdm0.github.io/AcceptReject/reference/accept_reject.html, only requires the user to provide the probability mass function (for discrete variables) or the probability density function (for continuous variables) they wish to generate observations from, in addition to the list of arguments of the distribution of interest. Other arguments can be passed and are optional, such as the normalization constant value, which by default is obtained automatically, arguments involving the optimization method, a probability mass function (discrete case), or probability density function (continuous case), if the user wishes to specify a base distribution, which can be useful in cases of more complex distributions.

With the use of simple functions such as `inspect()` and `plot()` the user can inspect the base probability density function and the quality of generating pseudo-random observations quickly, without spending much time on plotting, making the analysis process faster, which is as important as computational efficiency.

In this article, we will explore the **AcceptReject** package in detail. We will discuss how it can be used, the functionalities it offers, and the benefits it brings to users. Through examples and discussions, we hope to demonstrate the value of the **AcceptReject** package as a useful addition to the toolbox of any researcher or professional working with Monte Carlo simulations.

Initially, in Section 2, the article will discuss the ARM and how it can be used to generate pseudo-random observations of random variables. In Section 3, the dependencies used in the current version of the package will be listed and referenced, and some computational details considered in the examples and simulations will be presented. In Section 4, it shows how to install and load the package from CRAN and GitHub. In Section 5, each of the functions exported by the **AcceptReject** package is discussed, and some examples are presented to help expose the details. In Section 6, more examples of the package’s use are presented, showing the package for generating pseudo-random observations of

discrete and continuous random variables, and Section 7 is dedicated to presenting a more complex problem, in a more realistic scenario, where a probability distribution generator, proposed by Nadarajah et al. (2014), is used to generate the Modified Beta Weibull (MBW) probability density function, where we do not know its quantile function, and thus, the ARM applied using the **AcceptReject** package can solve the problem of generating observations of a random variable with MBW distribution. In Section 8, benchmarks are presented in different scenarios to demonstrate the computational efficiency of the package. In Section 9, similar works are compared with the **AcceptReject** package, and layout and performance advantages are presented. Finally, in Section 10, a conclusion is presented with suggestions for future improvements to enhance the package.

2 Acceptance-Rejection Method - ARM

The acceptance-rejection method - ARM, proposed by (von Neumann, 1951) and often simply called the rejection method, is a useful method for generating pseudo-random observations of discrete or continuous random variables, mainly in the context of univariate distributions, to which the **AcceptReject** package is intended. The method is based on the idea that, if we can find a probability distribution G_Y of a random variable Y that envelops (bounds) the probability distribution F_X of the random variable X of interest, and G and F share the same support, then we can generate observations of X . In addition to sharing the same support and having G_Y bounding f_X , it is necessary that we can generate observations of Y , that is, in practice, we will have a function that generates observations of this random variable. Through the generator of Y , we can then generate observations of the random variable of interest X , accepting or rejecting the observations generated by the observation generator of Y , based on the probability density functions or probability mass functions of X and Y , in the continuous or discrete case, respectively.

Consider a hypothetical example, with $x, y \in [0, 5]$, where x and y are observations of the random variables X (variable of interest) and Y (variable from which we can generate observations), respectively, and assuming we do not know how to generate observations of X , for example, from the Weibull distribution (in practice, this is not true), where $X \sim \text{Weibull}(\alpha = 2, \beta = 1)$, with $c \times Y \sim \mathcal{U}(a = 0, b = 5)$, where $c \geq 1$. In this example, given a density function used as a basis (density function of Y), the initial idea of ARM is to make the base density, denoted by g_Y , envelop (bound) the probability density function of interest, that is, envelop f_X . The Figure 1 (a) and Figure 1 (b) illustrate the procedure for choosing the constant c for ARM, with $c = 1$ and $c = 4.3$, respectively.

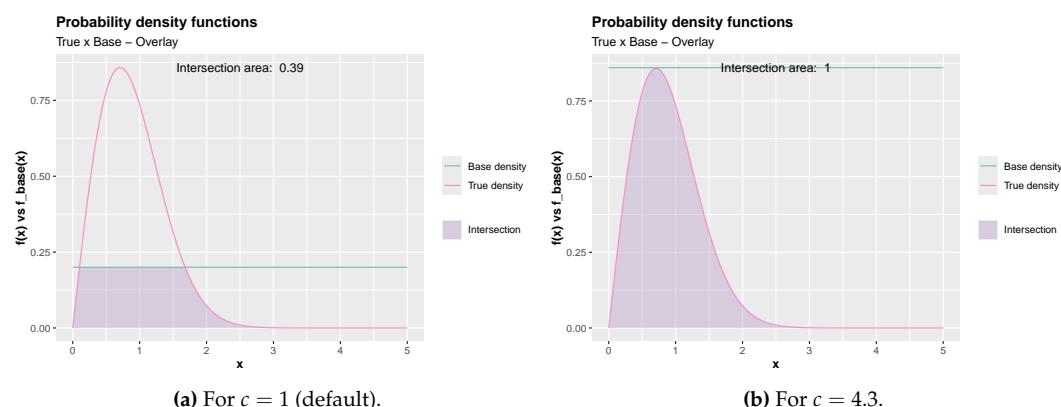


Figure 1: Inspection of the probability density function of the random variable of interest with the base probability density function, with $c = 1$ (default) (a) and $c = 4.3$ (b).

Notice that the support of the base probability density function g_Y must be the same as f_X , since the observations generated from Y are accepted or rejected as observations of X

(the random variable of interest). The interval $x, y \in [0, 5]$ was chosen because the density near the upper limit of this interval quickly drops to values close to zero, eliminating the need to consider a broader interval, although it could be considered. It is possible to see, with the help of Figure 1 (a), that $c = 1$ is not a good choice, as much of the density f_X would be left out and not captured by g_Y , i.e., it would not be bounded by g_Y . Therefore, increasing the value of c in this hypothetical example is necessary for good generation of observations of X . Note in Figure 1 (a) that the intersection area is approximately 0.39, and in the optimal case, we should have $c \geq 1$, such that the intersection area between f_X and g_Y is 1, and c is the smallest possible value. A small value of c will always imply a higher probability of accepting observations of Y as observations of X . Thus, the choice of c will be a trade-off between computational efficiency (higher acceptance probability) and ensuring that the base density bounds the density of interest.

Note now that for $c = 4.3$, in Figure 1 (b), the intersection area is equal to 1 and g_Y does not excessively bound f_X , thus becoming a convenient value. In this way, $c = 4.3$ could be an appropriate value for the given example. However, a larger value of c could be used, which would decrease the computational efficiency of ARM, for reasons that will be presented later.

What makes the method interesting is that the iterations of ARM are not mathematically dependent, making it easily parallelizable. Although it can be extended to the bivariate or multivariate case, the method is most commonly used to generate univariate observations. Moreover, if the distribution of the random variable from which pseudo-random observations are to be generated is indexed by many parameters, ARM can be applied without major impacts, as the number of parameters usually does not affect the efficiency of the method.

Considering the discrete case, suppose X and Y are random variables with probability density function (pdf) or probability function (pf) f and g , respectively. Furthermore, suppose there exists a constant c such that

$$\frac{f(x)}{g(y)} \leq c,$$

for every value of x , with $f(x) > 0$. To use the acceptance-rejection method to generate observations of the random variable X , using the algorithm below, first find a random variable Y with pdf or pf g , such that it satisfies the above condition.

It is important that the chosen random variable Y is such that you can easily generate its observations. This is because the acceptance-rejection method is computationally more intensive than more direct methods such as the transformation method or the inversion method, which only requires the generation of pseudo-random numbers with a uniform distribution.

Algorithm of the Acceptance-Rejection Method:

- 1 - Generate an observation y from a random variable Y with pdf/pf g ;
- 2 - Generate an observation u from a random variable $U \sim \mathcal{U}(0, 1)$;
- 3 - If $u < \frac{f(y)}{cg(y)}$ accept $x = y$; otherwise, reject y as an observation of the random variable X and go back to step 1.

Proof: Consider the discrete case, that is, X and Y are random variables with pfs f and g , respectively. By step 3 of the algorithm above, we have $\{\text{accept}\} = \{x = y\} = u < \frac{f(y)}{cg(y)}$. That is,

$$P(\text{accept} | Y = y) = \frac{P(\text{accept} \cap \{Y = y\})}{g(y)} = \frac{P(U \leq f(y)/cg(y)) \times g(y)}{g(y)} = \frac{f(y)}{cg(y)}.$$

Hence, by the **Law of Total Probability**, we have:

$$P(\text{accept}) = \sum_y P(\text{accept}|Y=y) \times P(Y=y) = \sum_y \frac{f(y)}{cg(y)} \times g(y) = \frac{1}{c}.$$

Therefore, by the acceptance-rejection method, we accept the occurrence of Y as an occurrence of X with probability $1/c$. Moreover, by Bayes' Theorem, we have

$$P(Y=y|\text{accept}) = \frac{P(\text{accept}|Y=y) \times g(y)}{P(\text{accept})} = \frac{[f(y)/cg(y)] \times g(y)}{1/c} = f(y).$$

The result above shows that accepting $x = y$ by the algorithm's procedure is equivalent to accepting a value from X that has pf f . For the continuous case, the proof is similar.

Notice that to reduce the computational cost of the method, we should choose c in such a way that we can maximize $P(\text{accept})$. Therefore, choosing an excessively large value of the constant c will reduce the probability of accepting an observation from Y as an observation of the random variable X .

Computationally, it is convenient to consider Y as a random variable with a uniform distribution on the support of f , since generating observations from a uniform distribution is straightforward on any computer. For the discrete case, considering Y with a discrete uniform distribution might be a good alternative.

Choosing an excessively large value for c will increase the chances of generating good observations of X , as an excessive value for c , in many situations, will allow the base distribution g to bound f . The problem, however, will be the computational cost. Thus, the problem of choosing an appropriate value for c is an optimization problem, where selecting an appropriate value for c is a task that can be automated. Additionally, it is important to note that choosing an excessively large value for the constant c , although it will lead to excessively slow code, is not as significant a problem as choosing an excessively small value for c , which will lead to the generation of poor observations of X . Since it is an optimization problem, it is possible to choose a convenient value for c that is neither too large nor too small, generating good observations for X with a very acceptable computational cost.

Therefore, the **AcceptReject** package aims to automate some tasks, including the optimization of the value of c . Thus, it becomes clear that, given a probability density function (continuous case) or probability mass function (discrete case) g , obtaining the smallest value of c , such that $c \geq 1$, combined with the possibility of code parallelism, are excellent ways to reduce the computational cost of ARM. This is, among other things, what the **AcceptReject** package does through its `accept_reject()` function and other available functions. In the following sections, we will discuss each of them in detail.

The first article describing the acceptance-rejection method is by von Neumann, titled "Various techniques used in connection with random digits" from 1951 (von Neumann, 1951). Several common books in the field of stochastic simulations and Monte Carlo methods also detail the method, such as (Kroese et al., 2013), (Asmussen and Glynn, 2007), (Kemp, 2003), and (Gentle, 2003).

3 Dependencies and Some Computational Details

The **AcceptReject** package (Marinho and Tomazella, 2024) has several dependencies, which, in its most current version, can be observed in the **DESCRIPTION** file. In the current version on GitHub, version 0.1.2, the dependencies are **assertthat** (Wickham, 2019), **cli** (Csárdi, 2023), **ggplot2** (Wickham, 2016), **glue** (Hester and Bryan, 2024), **numDeriv** (Gilbert and Varadhan, 2019), **purrr** (Wickham and Henry, 2023), **Rcpp** (Eddelbuettel et al., 2024), **RcppArmadillo** (Eddelbuettel and Sanderson, 2014), **rlang** (Henry and Wickham, 2024), **scales** (Wickham et al., 2023), and **scattermore** (Kulichova and Kratochvil, 2023). Suggested libraries include **knitr** (Xie, 2024), **rmarkdown** (Allaire et al., 2024), **cowplot** (Wilke, 2024), and **testthat** (Wickham, 2011).

In some examples that use the pipe operator `|>`, internal to the R language, it is necessary to have version 4.1.0 or higher of the language. However, this operator is not essential in the examples and can be easily omitted. The `%>%` operator from the `magrittr` package (Bache and Wickham, 2022) can also be used, as the `purrr` package exports it. In fact, the `|>` operator has not been used internally in any function of the `AcceptReject` package, in the current version on GitHub, so that it can pass the automatic checks of GitHub Actions, whose tests also consider older versions of the R language.

Additionally, some examples load the `parallel` library, included in the R language since version 2.14 in 2011. In these examples, where ARM executions are performed in parallel, to ensure the reproducibility of the results, it is necessary to call the instructions `RNGkind("L'Ecuyer-CMRG")` and `mc.reset.stream()`. The `RNGkind("L'Ecuyer-CMRG")` instruction sets the L'Ecuyer-CMRG pseudo-random number generator (L'ecuyer, 1999) and (L'ecuyer et al., 2002), which is a safe type of generator to use in parallel calculations, as it is a generator of multiple independent streams. The `mc.reset.stream()` function resets the stream of random numbers, ensuring that in a subsequent execution, the same sequence is generated again.

To take advantage of parallelism in ARM implemented in the `AcceptReject` package, which works on Unix-based operating systems, it is not necessary to load the `parallel` library, as the `accept_reject()` function of the `AcceptReject` package already makes use of parallelism. Loading the `parallel` library is only interesting if reproducibility is desired when execution is done in parallel using multiple processes, meaning when there is interest in executing the instructions `RNGkind("L'Ecuyer-CMRG")` and `mc.reset.stream()`.

Simulations for benchmarking in parallel and non-parallel scenarios were performed on a computer with the Arch Linux operating system, an Intel Core(TM) i7-1260P processor with 16 threads, a maximum processor frequency of 4.70 GHz, and 16 GB of RAM. The version of the R language was 4.4.0, and the computational times considered are in logarithmic scale, base 10.

The `AcceptReject` package makes use of the S3 object-oriented system in R. Simple functions like `plot()`, `qqplot()`, and `print()` were exported to dispatch for the `accept_reject` class of the `AcceptReject` package, making it easier to use. Other object-oriented systems, such as the R6 system (Chang, 2021), were not considered, as this would make the use of the package non-idiomatic, and some users might feel discouraged from using the library. Historical details about the programming paradigms of the R language can be found in (Chambers, 2014).

4 Installation and loading the package

The `AcceptReject` package is available on CRAN and GitHub and can be installed using the following command:

```
# Install CRAN version
install.packages("remotes")

# Installing the development version from GitHub
# or install.packages("remotes")
remotes::install_github("prdm0/AcceptReject", force = TRUE)

# Load the package
library(AcceptReject)
```



Figure 2: Logo of the package.

To access the latest updates of the **AcceptReject** package versions, check the [changelog](#). For suggestions, questions, or to report issues and bugs, please open an [issue](#). For a more general and quick overview of the package, read the [README.md](#) file or visit the package's website at <https://prdm0.github.io/AcceptReject/> to explore usage examples.

5 Function details and package usage

The **AcceptReject** package provides functions not only to generate observations using ARM in an optimized manner, but it also exports auxiliary functions that allow inspecting the base density function g_Y , as well as the generation of pseudo-random observations after the creation, making the generation and inspection work efficient, easy, and intuitive. Efficient from a computational point of view by automatically optimizing the constant c and, therefore, maximizing the acceptance probability $1/c$ of accepting observations of Y as observations of X , in addition to allowing multicore parallelism in Unix-based operating systems. The computational efficiency combined with the ease of inspecting g_Y and the generated observations makes the package pleasant to use. The **AcceptReject** library provides the following functions:

- `inspect()`: a useful function for inspecting the probability density function f_X with the base probability density function g_Y , highlighting the intersection between the two functions and the value of the areas, allowing experimentation with different values of c . This function does not perform any optimization; it simply facilitates the inspection of the proposed g_Y as a base probability density function, returning an object of the secondary class `gg` and the primary class `ggplot` for graphs created with the `ggplot2` library, as shown in Figure 1 (a) and Figure 1 (b);
- `accept_reject()`: implements ARM, optimizes the constant c , and performs parallelism if specified by the user. The user can also specify details in the optimization process or define a value for c and thus assume this value of c , omitting the optimization process. Additionally, it is possible to specify or not a base probability mass function or probability density function g_Y , depending on whether X is a discrete or continuous random variable, respectively. If omitted, $Y \sim \mathcal{U}$ discrete or continuous will be considered, depending on the nature of X . Moreover, the user can specify, in the case of not specifying the value of the constant c , an initial guess used in the optimization process of the constant c . A good guess can be given by graphically inspecting the relationship between f_X and g_Y . In most cases, the `accept_reject()` function will provide good acceptances without specifying g_Y different from the default discrete or continuous uniform distribution and will make a good estimation of c ;
- `print.accept_reject()`: a function responsible for printing useful information on the screen about objects of the `accept_reject` class returned by the `accept_reject()` function, such as the number of generated observations, the estimated constant c , and the estimated acceptance probability of accepting observations of Y as observations of X ;

- `plot.accept_reject()`: operates on objects of the `accept_reject` class returned by the `accept_reject()` function. The `plot.accept_reject()` function returns an object of the secondary class `gg` and the primary class `ggplot` from the `ggplot2` package, allowing easy graphical comparison of the probability density function or probability mass function f_X with the observed probability density function or mass function from the generated data;
- `qqplot()`: constructs a Quantile-Quantile plot (QQ-plot) to compare the distribution of data generated by ARM (observed distribution) with the distribution of the random variable X (theoretical distribution, denoted by f_X). In a very simple way, the QQ-plot is produced by passing an object of the `accept_reject` class, returned by the `accept_reject()` function, to the `qqplot()` function.

In the following subsections, more details about the functions exported by the `AcceptReject` package will be presented. Examples are provided to facilitate understanding. Additional details about the functions can be found in the function documentation, which can be accessed using `help(package = "AcceptReject")` or on the package's website, hosted in the GitHub repository that versions the development of the library at <https://prdm0.github.io/AcceptReject/>.

5.1 Inspecting Density Functions

The `inspect()` function of the `AcceptReject` package is useful when we want to generate pseudo-random observations of a continuous random variable using ARM. It is possible to skip this inspection since the `accept_reject()` function already automatically considers the continuous uniform distribution as the base density and optimizes, based on this base distribution, the best value for the constant c . Additionally, other base densities g_Y can be specified to the `accept_reject()` function, where the search for the constant c will be done automatically, optimizing the value of c and its relationship with f_X and g_Y , given g_Y .

To specify other base density functions g_Y , it is prudent to perform a graphical inspection of the relationship between f_X and g_Y to get an idea of the reasonableness of the candidate base probability density function g_Y . Thus, the `inspect()` function will automatically plot a graph with some useful information, as well as the functions f_X and g_Y . The `inspect()` function will return an object with the secondary class `gg` and the primary class `ggplot` (a graph made with the `ggplot2` library) highlighting the intersection area between f_X and g_Y and the value of this area for a given c , specified in the `c` argument of the `inspect()` function.

Theoretically, you can use any function g_Y that has support equivalent to that of the function f_X , finding the appropriate value of c that will make g_Y envelop f_X , that is, so that the value of the intersection area integrates to 1. The `inspect()` function has the following form:

```
inspect(
  f,
  args_f,
  f_base,
  args_f_base,
  xlim,
  c = 1,
  alpha = 0.4,
  color_intersection = "#BB9FC9",
  color_f = "#FE4F0E",
  color_f_base = "#7BBDB3"
)
```

where:

1. f is the probability density function f_X of the random variable X of interest;
2. args_f : is a list of arguments that will be passed to the probability density function f_X and that specify the parameters of the density of X ;
3. f_{base} : is the probability density function that will supposedly be used as the base density g_Y ;
4. $\text{args}_{f_{\text{base}}}$: is a list of arguments that will be passed to the base probability density function g_Y and that specify the parameters of the density of Y ;
5. xlim : is a vector of size two that specifies the support of the functions f_Y and g_Y (they must be equivalent);
6. c : is the constant c that will be used to multiply the base probability density function g_Y , so it envelops (bounds) f_X . The default is $c = 1$;
7. alpha : is the transparency of the intersection area (default is $\text{alpha} = 0.4$);
8. $\text{color_intersection}$: is the color of the intersection area between f_X and g_Y , where the default is $\text{color_intersection} = \text{\#BB9FC9}$;
9. color_f : is the color of the curve of the probability density function f_X (default is $\text{color}_f = \text{\#FE4F0E}$);
10. $\text{color}_{f_{\text{base}}}$: is the color of the curve of the base probability density function g_Y , where the default is $\text{color}_{f_{\text{base}}} = \text{\#7BBDB3}$.

The Figure 1 (a) and Figure 1 (b), presented at the beginning of this paper, were automatically created with the `inspect()` function. As an example, in addition to those available in the package documentation and vignettes, here is the code that generated the graph shown in Figure 1 (a):

```
library(AcceptReject)

# Considering c = 1 (default)
inspect(
  f = dweibull,
  f_base = dunif,
  xlim = c(0, 5),
  args_f = list(shape = 2, scale = 1),
  args_f_base = list(min = 0, max = 5),
  c = 1
)
```

5.2 Generating Observations with ARM

The most important function of the `AcceptReject` package is the `accept_reject()` function, as it is the function that implements ARM and all the optimizations to obtain a good generation of X . The `accept_reject()` function has the following signature:

```
accept_reject(
  n = 1L,
  continuous = TRUE,
  f = NULL,
  args_f = NULL,
  f_base = NULL,
  random_base = NULL,
  args_f_base = NULL,
  xlim = NULL,
  c = NULL,
  parallel = FALSE,
  cores = NULL,
  warning = TRUE,
  ...
)
```

where:

1. *n*: is the number of observations to be generated (default *n* = 1L);
2. *continuous*: is a logical value that indicates whether the probability density function f_X is continuous (default *continuous* = TRUE) or discrete, if *continuous* = FALSE;
3. *f*: is the probability density function f_X of the random variable X of interest;
4. *args_f*: is a list of arguments that will be passed to the probability density function f_X and that specify the parameters of the density of X . No matter how many parameters there are in *f*, they should be passed as a list to *args_f*;
5. *f_base*: is the probability density function that will supposedly be used as the base density g_Y . It is important to note that this argument is only useful in this package when *continuous* = TRUE, as for the discrete case the package already has quite satisfactory computational performance. Additionally, visualizing a probability mass function that bounds f_X is more complicated than for the continuous case. In the discrete case, when *continuous* = FALSE, *f_base* = NULL will be considered the discrete uniform distribution;
6. *random_base*: is a function that generates observations from the base distribution Y (default *random_base* = NULL). If *random_base* = NULL, the base distribution Y will be considered uniform over the interval specified in the *xlim* argument;
7. *args_f_base*: is a list of arguments that will be passed to the base probability density function g_Y and that specify the parameters of the density of Y . No matter how many parameters there are in *f_base*, they should be passed as a list to *args_f_base*;
8. *xlim*: is a vector of size two that specifies the support of the functions f_Y and g_Y . It is important to remember that the support of f_Y and g_Y must be equivalent and will be informed by a single vector of size two passed as an argument to *xlim*;
9. *c*: is the constant *c* that will be used to multiply the base probability density function g_Y , so it envelops (bounds) f_X . The default is *c* = 1. Unless there is a very strong reason, considering *c* = NULL as the default will be a good choice because the *accept_reject()* function will attempt to find an optimal value for the constant *c*;
10. *parallel*: is a logical value that indicates whether the generation of observations will be done in parallel (default *parallel* = FALSE). If *parallel* = TRUE, the generation of observations will be done in parallel on Unix-based systems, using the total number of cores available on the system. If *parallel* = TRUE and the operating system is Windows, the code will run serially, and no error will be returned even if *parallel* = TRUE;
11. *cores*: is the number of cores that will be used for parallel observation generation. If *parallel* = TRUE and *cores* = NULL, the number of cores used will be the total number of cores available on the system. If the user wishes to use a smaller number of cores, they can define it in the *cores* argument;
12. *warning*: is a logical value that indicates whether warning messages will be printed during the execution of the function (default *warning* = TRUE). If the user specifies a very small domain in *xlim*, the *accept_reject()* function will issue a warning informing that the specified domain is too small and that the generation of observations may be compromised;
13. ...: additional arguments that the user might want to pass to the *optimize* function used to optimize the value of *c*.

In a simpler use case, where the user does not wish to specify the base density function g_Y passed as an argument to *f_base*, useful for generating observations of a sequence of continuous random variables (x_1, \dots, x_n) , the use of the *accept_reject()* function is quite straightforward. For example, to generate 100 observations of a random variable X with a probability density function $f_X(x) = 2x$, $0 \leq x \leq 1$, the user could do:

```
set.seed(0)

# Generate 100 observations from a random variable X with
```

```
# f_X(x) = 2x, 0 <= x <= 1.
x <- accept_reject(
  n = 100L,
  f = function(x) 2 * x,
  args_f = list(),
  xlim = c(0, 1),
  warning = FALSE
)
print(x[1L:8L])

#> [1] 0.8966972 0.3721239 0.5728534 0.8983897 0.9446753 0.6607978 0.6870228
#> [8] 0.7698414
```

Note that in this case, if `warning = TRUE` (default), the `accept_reject()` function cannot know that `xlim = c(0, 1)` specifies the entire support of the probability density function $f_X(x) = 2x$, $0 \leq x \leq 1$, and that is why it issues a warning. Typically, one chooses a support that is passed to the `xlim` argument in which below the lower limit (first element of `xlim`) and above the upper limit (second element of the vector `xlim`), the probability mass (in the discrete case) or density (in the continuous case) is close to zero or not defined. In this case, we have deliberately set `warning = FALSE`.

5.3 Printing the Object with Generated Observations

The `accept_reject()` function returns an object of class `accept_reject`. An object of the `accept_reject` class is essentially an atomic vector with observations generated by ARM, carrying attributes and marked with a specific class from the **AcceptReject** package, i.e., marked with the `accept_reject` class. These attributes are useful for the package and carry information used by the `plot.accept_reject()` function.

Using the S3 object-oriented system in R, the `print()` function will dispatch for objects of the `accept_reject()` class, invoking the `print.accept_reject()` method from the **AcceptReject** package. For an object of the `accept_reject` class returned by the `accept_reject()` function, it is possible to pass methods that operate on atomic vectors, such as `summary()`, `mean()`, `var()`, among others. The `print()` function will print useful information about the `accept_reject()` class object, such as the number of observations generated, the value of `c` used, the acceptance probability, the first generated observations, and the considered `xlim` interval. The `summary()` function will work by returning some descriptive statistics about the generated observations, such as the mean, median, variance, standard deviation, minimum, maximum, first quartile, and third quartile.

```
# setting a seed for reproducibility
set.seed(0)
x <- accept_reject(
  n = 2000L,
  f = dbinom,
  continuous = FALSE,
  args_f = list(size = 5, prob = 0.5),
  xlim = c(0, 10)
)

# Printing the first 10 (default) observations
print(x)

# Printing the first 20 observations
print(x, n_min = 20L)

# Summary
summary(x)
```

```
#>      V1
#> Min. :0.000
#> 1st Qu.:2.000
#> Median :3.000
#> Mean   :2.538
#> 3rd Qu.:3.000
#> Max.   :5.000
```

5.4 Plotting Data Generated by ARM

Often, when generating observations using ARM, we are interested in visualizing the generated observations and graphically comparing them with the probability mass function (discrete case) or probability density function (continuous case) to get an idea of the quality of the data generation. Constructing a graph with your favorite library for each generation can be time-consuming. The idea of the `plot.accept_reject()` method is to facilitate this task, allowing you to easily and quickly generate this type of graph. In fact, since the S3 object-oriented system is used, you only need to use the `plot()` function on an `accept_reject` class object.

The `plot()` function applied to an `accept_reject` class object will return an object with the secondary class `gg` and the primary class `ggplot` (returns a graph). The returned object is a graph constructed with the `ggplot2` library and can be modified to your preferred `ggplot2` standards, such as its theme. However, the `plot` function has some specific arguments that allow you to modify some elements. The general usage form is:

```
## S3 method for class 'accept_reject'
plot(
  x,
  color_observed_density = "#BB9FC9",
  color_true_density = "#FE4F0E",
  color_bar = "#BB9FC9",
  color_observable_point = "#7BBDB3",
  color_real_point = "#FE4F0E",
  alpha = 0.3,
  hist = TRUE,
  ...
)
```

1. `x`: An object of the `accept_reject` class;
2. `color_observed_density`: observed density color (continuous case);
3. `color_true_density`: theoretical density color (continuous case);
4. `color_bar`: bar chart fill color (discrete case);
5. `color_observable_point`: color of generated points (discrete case);
6. `color_real_point`: color of the real points (discrete case);
7. `alpha`: transparency of the bars (discrete case);
8. `hist`: if `TRUE`, a histogram will be plotted in the continuous case, comparing the theoretical density with the observed one. If `FALSE`, `ggplot2::geom_density()` will be used instead of the histogram;
9. `...: additional arguments.`

The following code demonstrates the use of the `plot()` function. In Figure 3 (a), an example of a graph is presented with the histogram replaced by the observed density, if this type of representation is more useful to the user. To do this, simply pass the argument `hist = FALSE`. To illustrate the use of the arguments, the parameters `color_true_density`, `color_observed_density`, and `alpha` were also changed. In Figure 3 (b), an example of generating a graph for the discrete case is presented. The simple use of the `plot()` function without passing arguments should be sufficient to meet the needs of most users.

```

library(AcceptReject)

# Generating and plotting the theoretical density with the
# observed density.

# setting a seed for reproducibility
set.seed(0)

# Continuous case
accept_reject(
  n = 2000L,
  continuous = TRUE,
  f = dweibull,
  args_f = list(shape = 2.1, scale = 2.2),
  xlim = c(0, 10)
) |>
  plot(
    hist = FALSE,
    color_true_density = "#2B8b99",
    color_observed_density = "#F4DDB3",
    alpha = 0.6
  ) # Changing some arguments in plot()

# Discrete case
accept_reject(
  n = 1000L,
  f = dbinom,
  continuous = FALSE,
  args_f = list(size = 5, prob = 0.5),
  xlim = c(0, 10)
) |> plot()

```

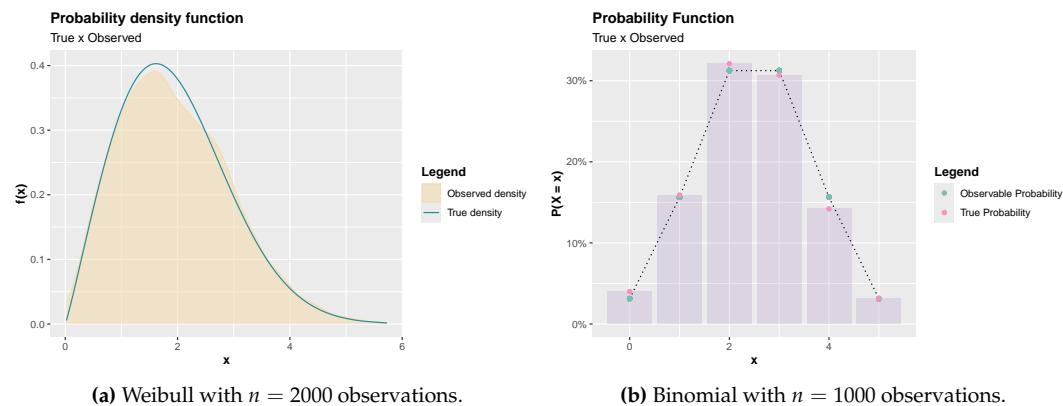


Figure 3: Plotting the theoretical density function (a) and the probability mass function (b), with details of the respective parameters in the code.

6 Examples

Below are some examples of using the `accept_reject()` function to generate pseudo-random observations of discrete and continuous random variables. It should be noted

that in the case of X being a discrete random variable, it is necessary to provide the argument `continuous = FALSE`, whereas in the case of X being continuous (the default), you must consider `continuous = TRUE`.

6.1 Generating discrete observations

As an example, let $X \sim Poisson(\lambda = 0.7)$. We will generate $n = 1000$ observations of X using the acceptance-rejection method, using the `accept_reject()` function. Note that it is necessary to provide the `xlim` argument. Try to set an upper limit value for which the probability of X assuming that value is zero or very close to zero. In this case, we choose `xlim = c(0, 20)`, where `dpois(x = 20, lambda = 0.7)` is very close to zero ($1.6286586 \times 10^{-22}$). Note Figure 4 (a) and Figure 4 (b), respectively, and note that in Figure 4 (b), there is a proximity of the observed probabilities to the theoretical probabilities, indicating that ARM is generating observations that approximate the true probability mass function as the sample size increases.

```
library(AcceptReject)
library(parallel)
library(cowplot) # install.packages("cowplot")

# Ensuring reproducibility in parallel computing
RNGkind("L'Ecuyer-CMRG")
set.seed(0)
mc.reset.stream()

# Simulation
simulation <- function(n, lambda = 0.7)
  accept_reject(
    n = n,
    f = dpois,
    continuous = FALSE, # discrete case
    args_f = list(lambda = lambda),
    xlim = c(0, 20),
    parallel = TRUE # Parallelizing the code in Unix-based systems
  )

# Generating observations
# n = 25 observations
system.time({x <- simulation(25L)})

#>     user   system elapsed
#>   0.006   0.038   0.047

plot(x)

# n = 2500 observations
system.time({y <- simulation(2500L)})

#>     user   system elapsed
#>   0.006   0.033   0.043

plot(y)
```

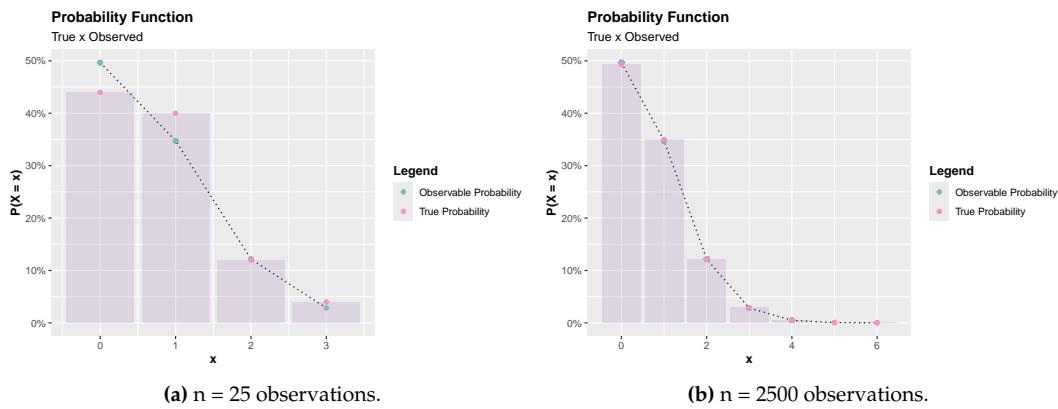


Figure 4: Generating observations from a Poisson distribution using the acceptance-rejection method, with $n = 25$ (a) and $n = 2500$ (b), respectively.

Note that it is necessary to specify the nature of the random variable from which observations are desired to be generated. In the case of discrete variables, the argument `continuous = FALSE` must be passed. In Section 6, examples of how to generate continuous observations will be presented.

6.2 Generating continuous observations

Considering the default base function, which is the uniform distribution, where it is not necessary to specify it, the code below exemplifies the continuous case, where $X \sim \mathcal{N}(\mu = 0, \sigma^2 = 1)$. Not specifying a base probability density function implies `f_base = NULL`, `random_base = NULL`, and `args_base = NULL`, which are the defaults for the `accept_reject()` function. If at least one of them is specified as `NULL` and, by mistake, another is specified, no error will occur. In this situation, the `accept_reject()` function will assume the base probability density function is the density function of a uniform distribution over `xlim`. Note also the use of the `plot()` function, which generates a graph with the theoretical density function and the histogram of the generated observations, allowing a quick visual check of the quality of the observations generated by ARM.

```
library(AcceptReject)
library(parallel)

# Ensuring reproducibility in parallel computing
RNGkind("L'Ecuyer-CMRG")
set.seed(0)
mc.reset.stream()

# Generating observations
accept_reject(
  n = 50L,
  f = dnorm,
  continuous = TRUE,
  args_f = list(mean = 0, sd = 1),
  xlim = c(-4, 4),
  parallel = TRUE
) |> plot()

accept_reject(
  n = 500L,
  f = dnorm,
```

```
continuous = TRUE,
args_f = list(mean = 0, sd = 1),
xlim = c(-4, 4),
parallel = TRUE
) |> plot()
```

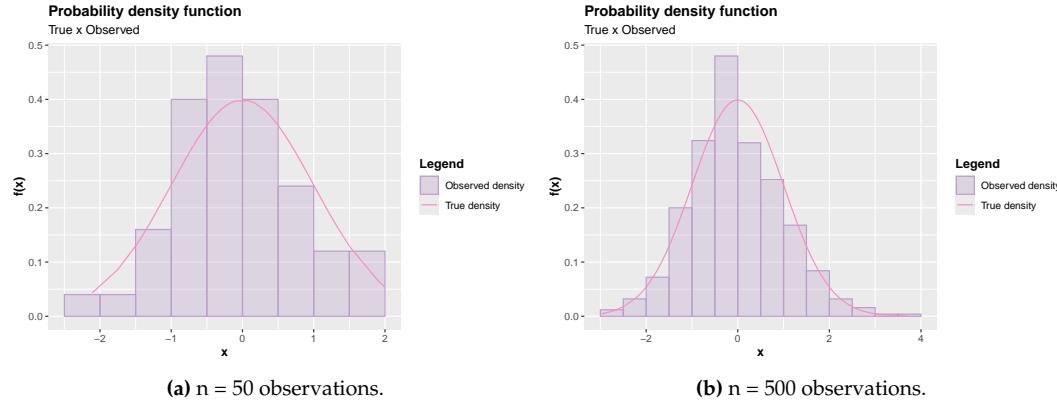


Figure 5: Generating observations from a continuous random variable with a Standard Normal distribution, with $n = 50$ and $n = 500$ observations, respectively.

7 A practical scenario for the use of the package

So far, the use of the package to generate distributions that are relatively simple and involve few parameters has been demonstrated. In this section, the idea is to demonstrate the use of the **AcceptReject** package in a practical situation where the use of ARM becomes necessary. Let's consider the generator of Modified Beta Distributions, proposed by (Nadarajah et al., 2014). It is a family of probability distributions since it is possible to generate various probability density functions through the proposed density generator, whose general density function is defined by:

$$f_X(x) = \frac{\beta^a}{B(a, b)} \times \frac{g(x)G(x)^{a-1}(1-G(x))^{b-1}}{[1 - (1-\beta)G(x)]^{a+b}},$$

with $x \geq 0$ and $\beta, a, b > 0$, where $g(x)$ is a probability density function, $G(x)$ is the cumulative distribution function of $g(x)$, and $B(a, b)$ is the beta function.

Notice that $f_X(x)$ has a certain complexity since it depends on another probability density function $g(x)$ and its cumulative distribution function $G(x)$. Thus, $f_X(x)$ has three parameters, c, a, b , plus additional parameters inherited from $g(x)$. Here, we will consider $g(x)$ as the Weibull probability density function. Therefore, $f_X(x)$ is the modified Weibull beta density function with five parameters. The implementation of the Modified Beta Distributions generator is presented below:

```
#|
library(numDeriv)

pdf <- function(x, G, ...){
  numDeriv::grad(
    func = \((x)\) G(x, ...),
    x = x
  )
}
```

```

# Modified Beta Distributions
# Link: https://link.springer.com/article/10.1007/s13571-013-0077-0
generator <- function(x, G, a, b, beta, ...){
  g <- pdf(x = x, G = G, ...)
  numerator <- beta^a * g * G(x, ...)^(a - 1) * (1 - G(x, ...))^(b - 1)
  denominator <- beta(a, b) * (1 - (1 - beta) * G(x, ...))^(a + b)
  numerator/denominator
}

# Probability density function - Modified Beta Weibull
pdf_mbw <- function(x, a, b, beta, shape, scale)
  generator(
    x = x,
    G = pweibull,
    a = a,
    b = b,
    beta = beta,
    shape = shape,
    scale = scale
  )

# Checking the value of the integral
integrate(
  f = \(x) pdf_mbw(x, 1, 1, 1, 1, 1),
  lower = 0,
  upper = Inf
)

#> 1 with absolute error < 5.7e-05

```

Notice that the value `pdf_mbw()` integrates to 1, being a probability density function. Thus, the `generator()` function generates probability density functions based on another distribution $G_X(x)$. In the case of the code above, the cumulative distribution function of the Weibull distribution is passed to the `generator()` function, but it could be any other.

In the following code, we will adopt the strategy of investigating (inspecting) a coherent proposal for a base density function to be passed as an argument to `f_base` in the `accept_reject()` function. The investigation could be skipped, in which case the `accept_reject()` function would assume the uniform distribution as the base.

We will consider the Weibull distribution since it is a particular case of the Modified Beta Weibull distribution. As we know how to generate observations from the Weibull distribution using the `rweibull()` function, the Weibull distribution is a viable candidate for the base density $g_Y(y)$. Consider the true parameters $a = 10.5$, $b = 4.2$, $\beta = 5.9$, $shape = 1.5$, and $scale = 1.7$. Thus, using the `inspect()` function, we can quickly inspect by doing:

```

library(AcceptReject)

# True parameters
a <- 10.5
b <- 4.2
beta <- 5.9
shape <- 1.5
scale <- 1.7

# c = 1 (default)
inspect(

```

```

f = pdf_mbw,
f_base = dweibull,
xlim = c(0, 4),
args_f = list(
  a = a,
  b = b,
  beta = beta,
  shape = shape,
  scale = scale
),
args_f_base = list(shape = 2, scale = 1.2),
c = 1
)

# c = 2.2
inspect(
  f = pdf_mbw,
  f_base = dweibull,
  xlim = c(0, 4),
  args_f = list(
    a = a,
    b = b,
    beta = beta,
    shape = shape,
    scale = scale
),
  args_f_base = list(shape = 2, scale = 1.2),
  c = 2.2
)

```

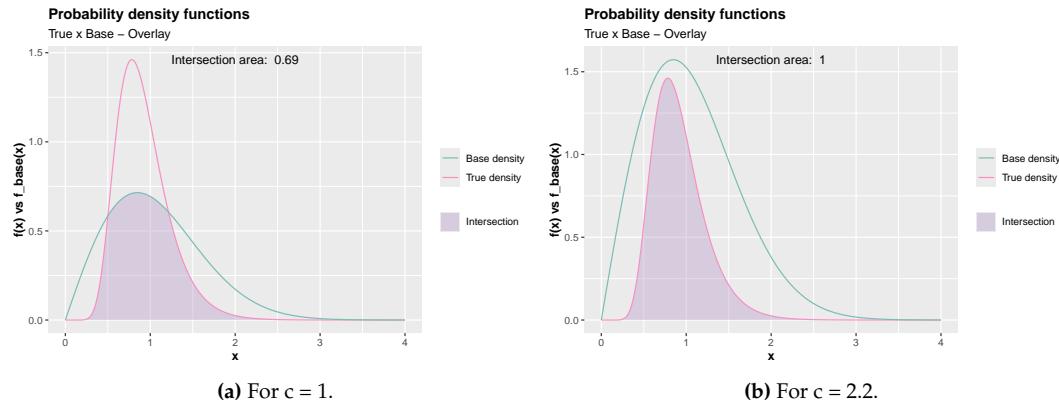


Figure 6: Inspecting the Weibull distribution with $\text{shape} = 2$, $\text{scale} = 1.2$, with the support $\text{xlim} = \text{c}(0, 4)$ and $c = 1$ (default) (a) and $c = 2.2$ (b), respectively.

Notice that in Figure 6 (b), when $c = 2.2$, the density g_Y bounds the density f_X , which is the Modified Beta Weibull density of X from which we want to generate observations. Thus, the density g_Y used as a base is a viable candidate to be passed to the `f_base` argument of the `accept_reject()` function in the **AcceptReject** package. Also, note that the area between f_X and g_Y is smaller than it would be if g_Y were considered the uniform probability density function in the `xlim` support. In the following section, we will discuss the computational cost for different sample sizes, considering the base density g_Y as the Weibull density or the default (uniform density).

8 Benchmarking

The benchmarks considered in this section were performed on a computer with Arch Linux operating system, Intel Core(TM) i7-1260P processor with 16 threads, maximum frequency of 4.70 GHz, with computational times in logarithmic scale, base ten. More specifications were presented in Section 3.

Considering the case of the Modified Beta Weibull probability density function (a probability density function with five parameters) implemented in the `pdf_mbw()` function presented in Section 7, several benchmarks were conducted to evaluate the computational cost of the `accept_reject()` function for various sample sizes, considering both the parallelized and non-parallelized scenarios. Additionally, the benchmarks took into account the specification of g_Y considering the continuous uniform distribution in the `xlim` interval (default of the `accept_reject()` function) and the Weibull distribution with parameters `shape = 2` and `scale = 1.2`, which bounds the probability density function of a random variable with Modified Beta Weibull with true parameters as in Figure 6. The sample sizes considered were $n = 50, 250, 500, 1000, 5000, 10000, 15000, 25000, 50000, 100000, 150000, 250000, 500000$, and 1000000 .

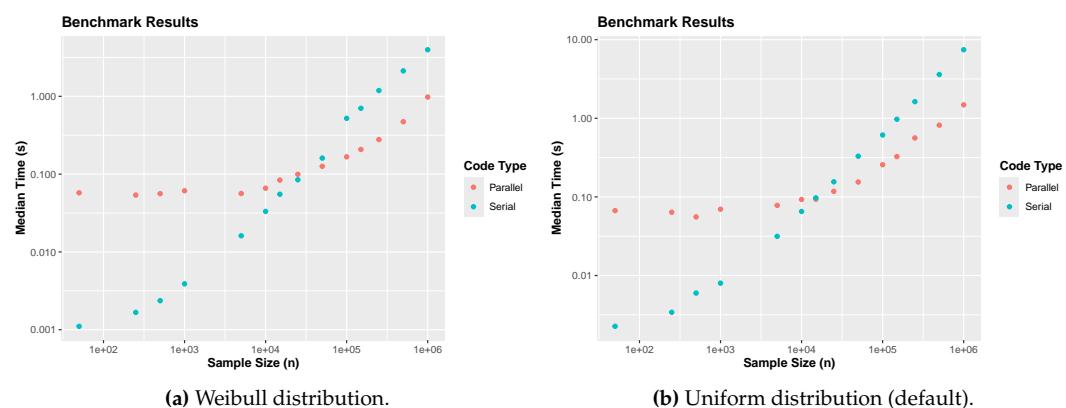


Figure 7: Benchmarking for different sample sizes, considering the Weibull distribution and the uniform distribution as the base density, with Weibull distribution and Uniform distribution (default), respectively.

Observing the Figures in 7, it is possible to see that the serial code, which is the default for the `accept_reject()` function, already performs excellently even with large samples, both when considering the specification of a base function g_Y or using the uniform distribution as the base. Only with very large samples does the parallelized code perform better, as the parallel code may impose an overhead in thread creation that might not be justified with very small samples. For sample sizes above 10,000, the parallelized code shows better performance. However, depending on the complexity of the probability distributions involved, there may be situations where parallelizing with moderate samples could be a good alternative. The package user should, in practice, conduct tests and decide whether to consider `parallel = TRUE` or `parallel = FALSE`.

It can be observed Figure 7 that the choice of the base distribution, for the simulated case, did not significantly influence the computational performance of the `accept_reject()` function. Often, depending on the complexity of f_X , the user might not need to worry about choosing a g_Y to pass to the `f_base` argument of the `accept_reject()` function. Additionally, it is not very common in Monte-Carlo simulation studies to consider sample sizes much larger than those considered here. Therefore, users of non-Unix-based systems will not experience significant issues regarding the computational cost of the `accept_reject()` function.

9 Related works

An implementation of ARM is provided by the **AR** library (Parchami, 2018). This library exports the **AR.Sim()** function, which has an educational utility to demonstrate the functioning of ARM. Additionally, the design of the **AcceptReject** package allows, in the continuous case, the use of base probability density functions that do not necessarily need to be implemented in the R language or in specific packages like in the case of the **AR** library, which is a significant advantage. The specifications of the base densities in the **AR** package are made considering only the densities implemented in the **DISTRIB** package (Parchami, 2016).

Another library that implements ARM is **SimDesign** (Chalmers and Adkins, 2020), through the **rejectionSampling()** function. The **rejectionSampling()** function is more efficient than the **AR.Sim()** function from the **AR** library, but its efficiency still does not surpass that of the **accept_reject()** function from the **AcceptReject** package. The **rejectionSampling()** function from the **SimDesign** library also does not support parallelism, which is a disadvantage compared to the **accept_reject()** function from the **AcceptReject** package. Furthermore, the design of the **AcceptReject** package uses the S3 object-oriented system, with exports of simple functions that allow the inspection and analysis of the generated data.

The **AcceptReject** library presents several advantages over the mentioned libraries, particularly in the way the package is designed, facilitating the use of functions. The library, for example, provides functions that allow the inspection of f_X with g_Y when X and Y are continuous random variables, making it easy to create highly informative graphs for the user, visually informing about the quality of the generated observations. It is a common interest for those using ARM to observe the generated data and check if they conform to the desired probability distribution.

Figure 8 (a) presents a simulation study comparing the **accept_reject()** function from the **AcceptReject** package with the **rejectionSampling()** function from the **SimDesign** package, considering small to moderate sample sizes. In this scenario, the **accept_reject()** function was executed serially, just like the **rejectionSampling()** function, which does not support parallelism. Equivalent performance was observed between the functions. In Figure 8 (b), it is observed that the performance of the **accept_reject()** function on large samples, considering the parallel execution of the **accept_reject()** function, surpasses the performance of the **rejectionSampling()** function from the **SimDesign** package.

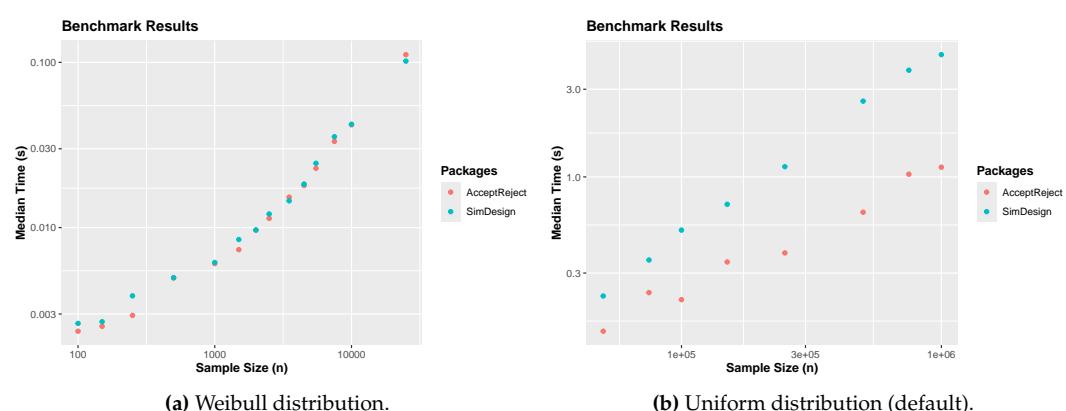


Figure 8: Comparison between the **AcceptReject** and **SimDesign** package for different sample sizes, considering the generation of observations from a random variable with a Modified Beta Weibull distribution, serial processing with **AcceptReject** (a) and parallel processing with **AcceptReject** package (b), respectively.

10 Conclusion and future developments

The `AcceptReject` package is an efficient tool for generating pseudo-random numbers in univariate distributions of discrete and continuous random variables using the acceptance-rejection method (ARM). The library is built with the aim of delivering ease of use and computational efficiency. The `AcceptReject` library can generate pseudo-random numbers both serially and in parallel, with computational efficiency in both cases, in addition to allowing easy inspection and analysis of the generated data.

For future developments, the `AcceptReject` library can be extended with functions that allow the visualization of ARM in an iterative application using Shiny for educational purposes. The most important development step for future versions will be to seek even more performance. Additionally, the project is also open to contributions from the community.

References

- J. Allaire, Y. Xie, C. Dervieux, J. McPherson, J. Luraschi, K. Ushey, A. Atkins, H. Wickham, J. Cheng, W. Chang, and R. Iannone. *rmarkdown: Dynamic Documents for R*, 2024. URL <https://github.com/rstudio/rmarkdown>. R package version 2.26. [p137]
- S. Asmussen and P. W. Glynn. *Stochastic simulation: algorithms and analysis*, volume 57. Springer, 2007. [p137]
- S. M. Bache and H. Wickham. *magrittr: A Forward-Pipe Operator for R*, 2022. URL <https://CRAN.R-project.org/package=magrittr>. R package version 2.0.3. [p138]
- R. P. Chalmers and M. C. Adkins. Writing effective and reliable Monte Carlo simulations with the SimDesign package. *The Quantitative Methods for Psychology*, 16(4):248–280, 2020. doi: 10.20982/tqmp.16.4.p248. [p152]
- J. M. Chambers. Object-oriented programming, functional programming and R. 2014. [p138]
- W. Chang. *R6: Encapsulated Classes with Reference Semantics*, 2021. URL <https://CRAN.R-project.org/package=R6>. R package version 2.5.1. [p138]
- G. Csárdi. *cli: Helpers for Developing Command Line Interfaces*, 2023. URL <https://CRAN.R-project.org/package=cli>. R package version 3.6.2. [p137]
- D. Eddelbuettel and C. Sanderson. RcppArmadillo: Accelerating R with high-performance C++ linear algebra. *Computational statistics & data analysis*, 71:1054–1063, 2014. [p134, 137]
- D. Eddelbuettel, R. Francois, J. Allaire, K. Ushey, Q. Kou, N. Russell, D. Bates, J. Chambers, and M. D. Eddelbuettel. Package ‘Rcpp’. 2024. [p134, 137]
- J. E. Gentle. *Random number generation and Monte Carlo methods*, volume 381. Springer, 2003. [p137]
- P. Gilbert and R. Varadhan. *numDeriv: Accurate Numerical Derivatives*, 2019. URL <https://CRAN.R-project.org/package=numDeriv>. R package version 2016.8-1.1. [p137]
- L. Henry and H. Wickham. *rlang: Functions for Base Types and Core R and ‘Tidyverse’ Features*, 2024. URL <https://CRAN.R-project.org/package=rlang>. R package version 1.1.3. [p137]
- J. Hester and J. Bryan. *glue: Interpreted String Literals*, 2024. URL <https://CRAN.R-project.org/package=glue>. R package version 1.7.0. [p137]
- D. Kemp. Discrete-event simulation: modeling, programming, and analysis, 2003. [p137]
- D. P. Kroese, T. Taimre, and Z. I. Botev. *Handbook of monte carlo methods*. John Wiley & Sons, 2013. [p137]

- T. Kulichova and M. Kratochvil. *scattermore: Scatterplots with More Points*, 2023. URL <https://CRAN.R-project.org/package=scattermore>. R package version 1.2. [p137]
- P. L'ecuyer. Good parameters and implementations for combined multiple recursive random number generators. *Operations Research*, 47(1):159–164, 1999. [p138]
- P. L'ecuyer, R. Simard, E. J. Chen, and W. D. Kelton. An object-oriented random-number package with many long streams and substreams. *Operations research*, 50(6):1073–1075, 2002. [p138]
- P. R. D. Marinho and V. L. D. Tomazella. *AcceptReject: Acceptance-Rejection Method for Generating Pseudo-Random Observations*, 2024. URL <https://prdm0.github.io/AcceptReject/>. R package version 0.1.1. [p134, 137]
- S. Nadarajah, M. Teimouri, and S. H. Shih. Modified beta distributions. *Sankhya B*, 76:19–48, 2014. [p135, 148]
- A. Parchami. *DISTRIB: Four Essential Functions for Statistical Distributions Analysis: A New Functional Approach*, 2016. URL <https://CRAN.R-project.org/package=DISTRIB>. R package version 1.0. [p152]
- A. Parchami. *AR: Another Look at the Acceptance-Rejection Method*, 2018. URL <https://CRAN.R-project.org/package=AR>. R package version 1.1. [p152]
- J. von Neumann. Various techniques used in connection with random digits. *Notes by GE Forsythe*, pages 36–38, 1951. [p133, 135, 137]
- H. Wickham. testthat: Get started with testing. *The R Journal*, 3:5–10, 2011. URL https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf. [p137]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p137]
- H. Wickham. assertthat: Easy Pre and Post Assertions, 2019. URL <https://CRAN.R-project.org/package=assertthat>. R package version 0.2.1. [p137]
- H. Wickham and L. Henry. *purrr: Functional Programming Tools*, 2023. URL <https://CRAN.R-project.org/package=purrr>. R package version 1.0.2. [p137]
- H. Wickham, T. L. Pedersen, and D. Seidel. *scales: Scale Functions for Visualization*, 2023. URL <https://CRAN.R-project.org/package=scales>. R package version 1.3.0. [p137]
- C. O. Wilke. *cowplot: Streamlined Plot Theme and Plot Annotations for 'ggplot2'*, 2024. URL <https://CRAN.R-project.org/package=cowplot>. R package version 1.1.3. [p137]
- Y. Xie. *knitr: A General-Purpose Package for Dynamic Report Generation in R*, 2024. URL <https://yihui.org/knitr/>. R package version 1.46. [p137]

Pedro Rafael Diniz Marinho
Federal University of Paraíba
Department of Statistics
Cidade Universitária, s/n, Departamento de Estatística - UFPB
<https://github.com/prdm0>
ORCID: 0000-0003-1591-8300
pedro.rafael.mariho@gmail.com

Vera L. D. Tomazella
Federal University of São Carlos
Department of Statistics
Rodovia Washington Luiz, Km 235, Monjolinho

<https://www.servidores.ufscar.br/vera/>
ORCID: 0000-0002-6780-2089
vera@ufscar.br

Pedro Luiz Ramos
Pontificia Universidad Católica de Chile
Faculty of Mathematics
Edificio Rolando Chuaqui, Campus San Joaquín. Avda. Vicuña Mackenna 4860, Macul.
<https://www.mat.uc.cl/>
pedro.ramos@mat.uc.cl

qCBA: An R Package for Postoptimization of Rule Models Learnt on Quantized Data

Tomas Kliegr

Abstract A popular approach to building rule models is association rule classification. However, these often produce larger models than most other rule learners, impeding the comprehensibility of the created classifiers. Also, these algorithms decouple discretization from model learning, often leading to a loss of predictive performance. This package presents an implementation of Quantitative Classification based on Associations (QCBA), which is a collection of postprocessing algorithms for rule models built over discretized data. The QCBA method improves the fit of the bins originally produced by discretization and performs additional pruning, resulting in models that are typically smaller and often more accurate. The qCBA package supports models created with multiple packages for rule-based classification available in CRAN, including `arc`, `arulesCBA`, `rCBA` and `sbrl`.

1 Introduction

There is a resurgence of interest in interpretable machine learning models, with rule learning providing an appealing combination of intrinsic comprehensibility, as humans are naturally used to working with rules, with well-documented predictive performance and scalability. Association rule classification (ARC) is a subclass of rule-learning algorithms that can quickly generate a large number of candidate rules, a subset of which is subsequently chosen for the final classifier. The first, and with at least three packages (Hahsler et al., 2019), the most popular such algorithm was Classification Based on Associations (CBA) (Liu et al., 1998). There are also multiple newer approaches, such as SBRL (Yang et al., 2017) or RCAR (Azmi and Berrado, 2020), both available in R (Yang et al., 2024; Michael Hahsler, 2024).

A major limitation of ARC approaches is that they typically trade off the ability to process numerical data for the speed of generation. On the input, these approaches require categorical data. If there are numerical attributes, these need to be converted to categories, typically through some discretization (quantization) approach, such as MDLP (Fayyad and Irani, 1993). Association rule learning then operates on prediscretized datasets, which results in the loss of predictive performance and larger rule sets.

The *Quantitative Classification Based on Associations* method (QCBA) (Kliegr and Izquierdo, 2023) is a collection of several algorithms that postoptimize rule-based classifiers learnt on prediscretized data with respect to the original raw dataset with numerical attributes. As was experimentally shown in Kliegr and Izquierdo (2023), this makes the models often more accurate and consistently smaller and thus more interpretable.

This paper presents the `qCBA` R package, which implements QCBA and is available on CRAN. The `qCBA` R package was initially developed to postprocess results of CBA implementations, as these were the most common rule learning systems in R, but it can now also handle results of other rule learning approaches such as SBRL. The three CBA implementations in CRAN – `rCBA` (Kuchař and Kliegr, 2019), `arc` (Kliegr, 2016) and `arulesCBA` (Michael Hahsler, 2024) introduced in Hahsler et al. (2019) rely on the fast and proven `arules` package (Hahsler et al., 2011) to mine association rules, which is also the main dependency of the `qCBA` R package.

2 Primer on building rule-based classifiers for QCBA in R

This primer will show how to use QCBA with CBA as the base rule learner. Out of the rule learners supported by QCBA, CBA is the most widely supported in CRAN and also scientifically most cited (as of writing). This primer is standalone and intends to show the main concepts through code-based examples. However, it uses the same dataset and setting

as the going example in the open-access publication (Kliegr and Izquierdo, 2023), which contains graphical illustrations as well as formal definitions of the algorithms (as opposed to R-code examples here).

2.1 Brief introduction to association rule classification

Before we present the details of the QCBA algorithm, we start by covering the foundational methods of association rule learning and classification.

Input data Association rules are historically mined on *transactions*, which is also the format used by the most commonly used R package `arules`. A standard data table (data frame) can be converted to a transaction matrix. This can be viewed as a binary incidence matrix, with one dummy variable for each attribute-value pair (called an *item*). In the case of numerical variables, discretization (quantization) is a standard preprocessing step. It is required to ensure that the search for rules is fast and the conditions in the discovered rules are sufficiently broad.

Association rules Algorithms such as Apriori (Agrawal and Srikant, 1994) are used for association rule mining. An association rule has the form $r : \text{antecedent} \rightarrow \text{consequent}$, where both *antecedent* and *consequent* are sets of items. The `arules` package, which is the most popular Apriori implementation in CRAN, calls the antecedent the left-hand side of the rule (*lhs*) and the consequent the right-hand side (*rhs*). Each set is interpreted as a conjunction of conditions corresponding to individual items. When all these conditions are met, then the rule predicts that its consequent is true for a given input data point. Formally, we say that a rule *covers* a transaction if all items in the rule's antecedent are contained in the transaction. A rule *correctly classifies* the transaction if the transaction is covered and, at the same time, the items in the rule consequent are contained in the transaction.

Rule interest measures Each rule is associated with some quality metrics (also called rule interest measures). The two most important ones are the confidence and support of rule r . Confidence is calculated as $\text{conf}(r) = a/(a + b)$, where a is the number of transactions matching both the antecedent and consequent and b is the number of transactions matching the antecedent but not the consequent. Support is calculated either as a (absolute support) or a divided by the total number of transactions (relative support). In the association rule generation step, confidence and support are used as constraints in association rule learning. Another important constraint to prevent a combinatorial explosion is a restriction on the length of the rule (`minLen` and `maxLen` parameters in `arules`), defined as the threshold on the minimum and maximum count of items in the rule antecedent and consequent.

Association Rule Classification models ARC algorithms process candidate association rules into a classifier. An input to the ARC algorithm is a set of pre-mined *class association rules* (CARs). A class association rule is an association rule that contains exactly one item corresponding to one of the target classes in the consequent. The classifier-building process typically amounts to a selection of a subset of CARs (Vanhoof and Depaire, 2010). Some ARC algorithms, such as CBA, use rule interest measures for this: rules are first ordered and then some of them are removed using *data coverage pruning* and *default rule pruning* (step 2 and step 3 of the CBA-CB algorithm as described in Liu et al. (1998)). Some ARC algorithms also include a rule with an empty antecedent (called *default rule*) at the end of the rule list to ensure that the classifier can always provide a prediction even if there is no specific rule matching that particular transaction.

Types of ARC models CBA produces *rule lists*: the rule with the highest priority in the pruned rule list is used for classification. Other recent algorithms producing rule lists include SBRL (Yang et al., 2024). The other common ARC approach is based on *rule sets*, where rules are not ordered, and multiple rules can be used for classification, e.g., through voting. This second group includes algorithms such as CMAR (Li et al., 2001) or CPAR (Yin and Han, 2003).

Postprocessing with Quantitative CBA QCBA is a postprocessing algorithm for ARC models built over quantized data. On the input, it can take both rule lists or rule sets.

However, it always outputs a rule list. While association rule learning often faces issues with a combinatorial explosion (Kliegr and Kuchar, 2019), the postprocessing by QCBA is performed on a relatively small number of rules that passed the pruning steps within the specific ARC algorithm. QCBA is a modular approach with six steps that can be performed independently of each other. The only exception is the initial refit tuning step, which processes all items in a given rule that are the result of quantization. QCBA adjusts the item boundaries so that they correspond to actual values appearing in the original training data before discretization. Benchmarks in Kliegr and Izquierdo (2023) have shown that, on average, the postprocessing with QCBA took a similar time as building the input CBA model. The most expensive step can be extension, with its complexity depending on the number of unique numerical values.

Comparison with decision tree induction A common question is the relationship between association rule classifiers and decision trees. Trees can be decomposed into rules that are similar to association rules, as each path from the root to the leaf corresponds to one rule. However, individual trees in algorithms such as C4.5 (Quinlan, 1993) are built in such a way that the resulting rules are non-overlapping, while association rule learning outputs overlapping rules. Algorithms such as Apriori will output all rules that are valid in the data, given the user-specified thresholds. In contrast, decision tree induction algorithms use a heuristic, such as information gain, which results in a large number of otherwise valid patterns being skipped as rules prioritize splits that maximize the chosen heuristic.

2.2 Example

Dataset

Let's look at the `humtemp` synthetic data set, which we will be using throughout this tutorial and is bundled with the `arcPackage`. There are two quantitative explanatory attributes (Temperature and Humidity). The target attribute is preference (subjective comfort level).

The first six rows of `humtemp` obtained with `head(humtemp)`:

```
##   Temperature Humidity Class
## 1         45       33     2
## 2         27       29     3
## 3         40       48     2
## 4         40       65     1
## 5         38       82     1
## 6         37       30     3
```

Quantization

The essence of QCBA is the optimization of literals (conditions) created over numerical attributes in rules. QCBA thus needs to translate back the bins created during the discretization to the continuous space.

For clarity, we will perform the quantization manually using equidistant binning and user-defined cut points. An alternative approach using automatic supervised discretization is shown in Section 2.4.

```
library(qCBA)

temp_breaks <- seq(from = 15, to = 45, by = 5)
# Another possibility with user-defined cutpoints
hum_breaks <- c(0, 40, 60, 80, 100)

data_discr <- arc::applyCuts(
  df = humtemp,
  cutp = list(temp_breaks, hum_breaks, NULL),
  infinite_bounds = TRUE,
  labels = TRUE
```

```
)
head(data_discr)
```

The result of quantization is:

```
##   Temperature Humidity Class
## 1     (40;45]    (0;40]     2
## 2     (25;30]    (0;40]     3
## 3     (35;40]    (40;60]     2
## 4     (35;40]    (60;80]     1
## 5     (35;40]    (80;100]    1
## 6     (35;40]    (0;40]     3
```

The purpose of the `applyCuts()` function is to ensure that within intervals, a semicolon is used as a separator instead of the more common comma. A semicolon is used as the standard interval separator in some countries, such as Czechia. However, the main reason is that a comma is already used within rules for another purpose – to separate conditions. The use of a different separator fosters unambiguity, for example, in situations when a rule set aimed to be optimized is read from a file.

Discovery of candidate class association rules

ARC algorithms typically first generate a large number of association rules or frequent itemsets. Typically, this step is handled internally by the ARC library (as shown in Section [Demonstration of individual QCBA optimization steps](#)).

For better clarity, in the example below, the list of CARs is generated by manually invoking the `apriori` algorithm.

```
txns <- as(data_discr, "transactions")
appearance <- list(rhs = c("Class=1", "Class=2", "Class=3", "Class=4"))
rules <- arules::apriori(
  data = txns,
  parameter = list(
    confidence = 0.5,
    support = 3 / nrow(data_discr),
    minlen = 1,
    maxlen = 3
  ),
  appearance = appearance
)
```

The first line converts the input data into *items*, attribute-value pairs such as `Temperature=(40;45]` or `Class=3`. The `appearance` defines which items can appear in the consequent of the rules (right-hand side, *rhs*). This data format is required for association rule learning. On the second line, there are several standard additional parameters typically used for the extraction of association rules. The confidence threshold of 0.5 and support of 1% is recommended ([Liu et al., 1998](#)). However, since the `humtemp` dataset has fewer than 100 rows, the support threshold would correspond to the support of just 1 transaction, which would miss the purpose of this threshold to address overfitting by eliminating rules that are backed by only a small number of instances. We, therefore, set the minimum support threshold to 3 transactions (expressed as a percentage in the code snippet). The `minlen` and `maxlen` express that rules must contain at most two items in the condition part (antecedent).

The discovered rules, shown with `inspect(rules)`, are shown below:

lhs	rhs	support	confidence	coverage	lift	#
{Humidity=(80;100]}	=> {Class=1}	0.1111111	0.8000000	0.1388889	3.600000	4
{Temperature=(15;20]}	=> {Class=2}	0.1111111	0.5714286	0.1944444	2.057143	4
{Temperature=(30;35]}	=> {Class=4}	0.1388889	0.6250000	0.2222222	2.045455	5
{Temperature=(25;30]}	=> {Class=4}	0.1388889	0.5000000	0.2777778	1.636364	5
{Temperature=(25;30], Humidity=(40;60]}	=> {Class=4}	0.0833333	0.6000000	0.1388889	1.963636	3

In this listing, coverage and lift, as defined in the Hahsler et al. (2007) documentation, are additional rule quality measures not used by the **qCBA** package. Count (abbreviated with # in the listing) corresponds to the support of the rule represented as an integer value rather than a proportion.

Learn classifier from candidate CARs

Out of the five discovered rules, we create a CBA classifier. The following lists the conceptual steps performed by CBA to transform the input rule list into a CBA model:

1. Rule *precedence* is established: rules are sorted according to confidence, support and length.
2. Rules are subject to *pruning*
 - *data coverage pruning*: the algorithm iterates through the rules in the order of precedence removing any rule which does not correctly classify at least one instance. If the rule correctly classifies at least one instance, it is retained and the instances removed (only for the purpose of subsequent steps of data coverage pruning).
 - *default rule pruning*: the algorithm iterates through the rules in the sort order, and cuts off the list once keeping the current rule would result in worse accuracy of the model than if a default rule was inserted at the place of the current rule and the rules below it were removed.
3. *Default rule* is inserted at the bottom of the list.

To supply our own list of candidate rules instead of using one generated with `cba()`, we will call:

```
classAtt <- "Class"
rmCBA <- cba_manual(
  datadf_raw = humtemp,
  rules = rules,
  txns = txns,
  rhs = appearance$rhs,
  classAtt = classAtt,
  cutp = list()
)
inspect(rmCBA@rules)
```

The reason why we invoke `cba_manual()` from the `arc` package is that `cba_manual()` instead of `cba()` allows us to supply a custom list of rules from which the CBA model will be built. This function would also do the quantization, but since we already did this as part of the preprocessing, we use `cutp = list()` to express that no cutpoints are specified.

In this toy example, the CBA model, which could be displayed through the function call `inspect(rmCBA@rules)`, is almost identical to the candidate list of rules shown above. The main difference is the reordering of rules by confidence and support (higher is better) and the addition of the *default rule* – a rule with an empty antecedent to the end. Note that for brevity, the conditions in the rules in the printout below were replaced by {...} as they are the same as in the printout on the previous page (although mind the different order of rules). The values of support, confidence, coverage and lift are also the same and were omitted.

	lhs	rhs	{...}	count	lhs_length	orderedConf	orderedSupp	cumulativeConf
[1]	{...} => {Class=1}	{...}	4	1	0.8000000	4	0.8000000	
[2]	{...} => {Class=4}	{...}	5	1	0.7142857	5	0.7500000	
[3]	{...} => {Class=4}	{...}	3	2	0.6000000	3	0.7058824	
[4]	{...} => {Class=2}	{...}	4	1	0.5000000	3	0.6521739	
[5]	{...} => {Class=4}	{...}	5	1	0.5000000	2	0.6296296	
[6]	{}	=> {Class=2}	{...}	0	0.5555556	5	0.6111111	

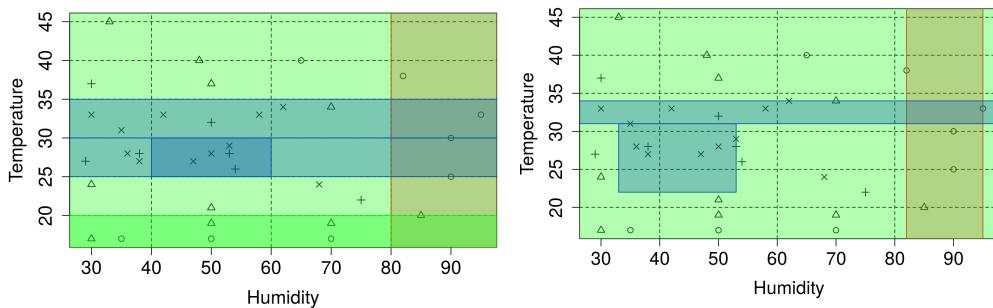


Figure 1: Illustration of postpruning algorithm (HumTemp dataset). Left: CBA model; right: QCBA model.

The default rule ensures that the rule list covers every possible instance. The rule list is visualized in Figure 1, where the green background corresponds to Class 2 predicted by the default rule. The CBA output contains several additional statistics for each rule. The *ordered* versions of confidence and support are computed only from those training instances reaching the given rule. For the first rule, the ordered confidence is identical to standard confidence. The ordered support is also semantically the same but is expressed as an absolute count rather than a proportion. To compute these values for the subsequent rules, instances covered by rules higher in the list have been removed. The cumulative confidence is an experimental measure described in [arc](#) documentation.

Applying the classifier

The toy example does not contain enough data for a meaningfully large train/test split. Therefore, we will evaluate the training data. The accuracy of the CBA model on training data:

```
prediction_cba <- predict(rmCBA, data_discr)
acc_cba <- CBARuleModelAccuracy(
  prediction = prediction_cba,
  groundtruth = data_discr[[classAtt]]
)
```

The accuracy is 0.61, and the contents of `prediction_cba` are the predicted values of comfort:

```
[1] 2 4 2 2 1 2 2 4 4 4 4 4 1 4 1 4 4 4 4 4 4 1 2 2 2 2 1 2 2 2 2 2 2  
Levels: 1 2 4
```

Explaining the prediction

The `predict()` function from the [arc](#) library allows for additional output to enhance the explainability of the result. By setting `outputFiringRuleIDs=TRUE` we can obtain the ID of a particular rule that was used to classify each instance in the passed dataset.

```
ruleIDs <- predict(rmCBA, data_discr, outputFiringRuleIDs = TRUE)
```

For example, we may now explain the classification of the first row of `data_discr`, which is, for convenience, reproduced below:

```
##   Temperature Humidity Class
## 1     (40;45]    (0;40]     2
```

To do so, we invoke:

```
inspect(rmCBA@rules[ruleIDs[1]])
```

This returns the default rule (number 6). The reason is that the values in this instance are out of bounds of the conditions in all other rules. Now, we have a rule list ready for postoptimization with QCBA.

Postprocessing with QCBA

By working with the original continuous data, the QCBA algorithm can improve the fit of the rules and consequently reduce their count.

We will use the rmCBA model built previously:

```
rmqCBA <- qcba(cbaRuleModel = rmCBA, datadf = humtemp)
```

This ran QCBA with the default set of optimization steps enabled, which correspond to the best-performing configuration #5 from (Kliegr and Izquierdo, 2023).

The resulting rule list is

rules	support	confidence	#
1 {Humidity=[82;95]} => {Class=1}	0.1111111	0.8000000	1
2 {Temperature=[22;31],Humidity=[33;53]} => {Class=4}	0.1666667	0.7500000	2
3 {Temperature=[31;34]} => {Class=4}	0.1388889	0.6250000	1
4 {} => {Class=2}	0.2777778	0.2777778	0

Note that the ‘condition_count’ column was abbreviated as # in the listing and the ordered-Conf and orderedSupp columns were omitted for brevity.

Figure 1 (right) shows the QCBA model. Compared to the CBA model in Figure 1 (left), QCBA removed two rules and refined the boundaries of the remaining rules.

Predictive performance is computed in the same way as for CBA:

```
prediction <- predict(rmqCBA, humtemp)
acc <- CBARuleModelAccuracy(prediction, humtemp[[rmqCBA@classAtt]])
```

The accuracy is unchanged at 0.61, but we got a smaller model. Similarly, as with CBA, we could use the argument *outputFiringRuleIDs*.

Note that the QCBA algorithm does not introduce any mandatory thresholds or meta-parameters for the user to set or optimize, although it does allow the user to enable or disable the individual optimizations, as shown in the next section.

3 Detailed description of package qCBA with examples in R

3.1 Overview of qcba() arguments

To build a model, qcba() needs a set of rules. As a second mandatory argument, the qcba() function takes the *raw* data frame. This can contain nominal as well as numerical columns. The remaining arguments are optional. The most important ones relating to the optimizations performed by QCBA are described in the following two subsections.

This rule model can be either the instance of customCBARuleModel or CBARuleModel class. The difference is that in the *rules* slot, in the former class, rules are represented as string objects in a data frame. This universal data frame format is convenient for loading rules from other sources or R packages that export rules as strings. The latter uses an instance of *rules* class from arules, which is more efficient, especially when the number of rules or items is larger. An important slot shared between both classes is *cutp*, which contains information on the cutpoints used to quantize the data on which the rules were learnt.

3.2 Optimizations on individual rules

Optimizations can be divided into two groups depending on whether they are performed on individual rules or on the entire model. We first describe the first group. Since these operations are independent of the other rules, they can also be parallelized.

Refitting rules. This step processes all items derived with quantization in the antecedent of a given rule. These items have boundaries that stick to a grid that corresponds to the result of discretization. The grid used by QCBA corresponds to all unique values appearing in the training data. *This is the only mandatory step.*

Attribute pruning (`attributePruning`). Attribute pruning is a step in QCBA that evaluates if the items are needed for each rule and item in its antecedent. The item is removed if a rule created without the item has at least the confidence of the original rule. *Enabled* (TRUE) by default.

Trimming (`trim_literal_boundaries`). Boundary parts of intervals into which no instances correctly classified by the rule fall are removed. *Enabled* (TRUE) by default.

Extension (`extendType`). The ranges of intervals in the antecedent of each rule are attempted to be enlarged. Currently, only extension on numerical attributes is supported. By default, the extension is accepted if it does not decrease rule confidence, but this behaviour can be controlled by setting the `minImprovement` parameter (default is 0). To overcome local minima, the extension process can provisionally accept a drop in confidence in the intermediate result of the extension. How much the confidence can temporarily decrease for the extension process not to stop is controlled by `minCondImprovement`. In the current version, the extension applies only to numerical attributes (set `extendType="numericOnly"` to enable, this is also the default value). In future versions, other extend types may be added.

3.3 Optimizations on rule list

The second group of optimizations aims at removing rules, considering the context of other rules in the list.

Data coverage pruning (`postpruning`). The purpose of this step is to remove rules that were made redundant by the previous QCBA optimizations. Possible values are none, cba and greedy. The cba option is identical to CBA's data coverage pruning: a rule is removed if it does not correctly classify any transaction in the training data after all transactions covered by retained rules with higher precedence were removed. The greedy option is an experimental modification of data coverage pruning described in the [qCBA](#) documentation. *Enabled* by default (the default value is `postpruning=cba`).

Default rule overlap pruning (`defaultRuleOverlapPruning`). Let R_p denote the set of rules that classify into the same class as the default rule $r_d : \{\} \rightarrow cons_1$. To determine if a pruning candidate, denoted as $r_p \in R_p : ant \rightarrow cons_1$, can be removed, all rules with lower precedence that have a nonempty antecedent and a different consequent $cons_2$ ($cons_1 \neq cons_2$) are identified. Let's denote the set of these rules as R_c . If the antecedents of all rules in R_c do not cover any of the transactions covered by r_p in the training data, r_p is removed. The removal of r_p will not affect the classification of training data, since the instances originally covered by $r_p : ant \rightarrow cons_1$ will be classified to the same class by $r_d : \{\} \rightarrow cons_1$. This is called the transaction-based version. In the alternative range-based version, the checks on rules in R_p involve checking the boundaries of intervals rather than overlap in matched transactions. This parameter has three possible values: `transactionBased`, `rangeBased` and `none`. According to analysis and benchmarks in ([Kliegr and Izquierdo, 2023](#)), the transaction-based version removes more rules than the range-based version, although it can sometimes affect predictive performance on unseen data. *Transaction-based pruning is the default*.

3.4 Effects on classification performance and model size

An overview of observed properties of the QCBA steps is present in Table 1. The entries denote the effect of applying the algorithm specified in the first column on the input rule list: \geq denotes that the value of the given metric will increase or will not change, $=$ the value will not change, \leq decrease or will not change, na can increase, decrease or will not change. For example, applying the refit algorithm on a rule can have the following effects according to the table: the density of the rule will improve or remain the same: the (+) symbol in the table denotes that an increase in that value is considered a favourable property, and (-) as negative. Rule confidence (`conf`), rule support (`supp`), rule length (`length`) will remain unaffected. Considering the entire rule list, the refit operation will not affect the rule count

or accuracy on training data (acc_{train}). However, the accuracy on unseen data (acc_{test}) may change.

algorithm dataset	rule (local classifier)			rule list		
	conf	supp	length	acc_{train}	acc_{test}	rule count
refit	=	=	=	=	na	=
literal pruning	\geq	$\geq (+)$	$\leq (+)$	na	na	$\leq^* (+)$
trimming	$\geq (+)$	=	=	$\geq (+)$	na	=
extension	$\geq (+)$	$\geq (+)$	=	na	na	=
postpruning	na	na	na	\geq	na	$\leq (+)$
drop - trans.	na	na	na	=	na	$\leq (+)$
drop - range	na	na	na	=	=	$\leq (+)$

Table 1: Hypothesized properties of the proposed rule tuning algorithms. *drop* denotes default rule pruning (*trans.* transaction-based; *range* range-based). * the number of rules is not directly reduced, but literal pruning can make a rule redundant (identical to another rule). The value *na* expresses that there is no unanimous effect of the preprocessing algorithm on the quality measure.

3.5 Handling missing data

Association rule learning approaches are resilient to the presence of missing data in the input data frame. The reason is that rows are viewed as transactions and combinations of column names and their values as items. A missing value (NA) in a given column is then interpreted as an item not present in a given transaction and skipped when a data frame is converted to the transactions data structure from the `arules` using the `as()` function, which will result in NA values not being present in learnt rules. Note that `qcba()` treats an empty string in the input dataframe in the same way as the NA value along with a dataframe containing NA values: the NA value is first converted to an empty string, represented using `.jarray()` from `rJava` and then passed to the Java-based core of `qCBA`, where it is treated as a `Float.NaN` value and also generally skipped.

3.6 Computational costs for large datasets

The individual postprocessing operations have distinct computational costs. These depend on several main factors: the number of rows and columns of the input data, the number of unique numerical values, and the number of input rules. A detailed experimental study of the influence of these factors is presented in Kliegr and Izquierdo (2023). This was performed on subsets of the KDD'99 Anomaly Detection dataset, with the maximum dataset size being processed using all QCBA optimizations, reaching about 40,000 rows and about the same number of unique numerical values. The results show that the most computationally intensive step is the extension step. For large datasets, the user may, therefore, consider disabling it or tuning its `minCI` parameter, which can influence the runtime. An important factor is also the number of input rules for optimization. This is often related to the chosen base learning algorithm. For example, CPAR tends to produce a large number of rules while SBRL will output condensed rule models. For details, please refer to Kliegr and Izquierdo (2023).

4 Demonstration of individual QCBA optimization steps

Compared to the example in Subsection Example, this part will use the larger iris dataset, which allows for the demonstration of all QCBA steps. To show QCBA with a different base rule learner than CBA from the `arc` package used in the previous section, we will use CPAR from the `arulesCBA` package.

4.1 Data

We use the `iris` dataset from the `datasets` R package. The dataset was shuffled and randomly split into a train set (100 rows) and a test set (50 rows). The data was then automatically discretized using the MDLP algorithm wrapped by `arc::discrNumeric()` and originally available from the R `discretization` library.

```
library(arulesCBA) #version 1.2.7
set.seed(12) # Chosen for demonstration purposes
allDataShuffled <- datasets::iris[sample(nrow(datasets::iris)), ]
trainFold <- allDataShuffled[1:100, ]
testFold <- allDataShuffled[101:nrow(datasets::iris), ]
classAtt <- "Species"

discrModel <- discrNumeric(df = trainFold, classAtt = classAtt)
train_disc <- as.data.frame(lapply(discrModel$Disc.data, as.factor))
cutPoints <- discrModel$cutp
test_disc <- applyCuts(
  testFold,
  cutPoints,
  infinite_bounds = TRUE,
  labels = TRUE
)
y_true <- testFold[[classAtt]]
```

4.2 Learn base ARC model with CPAR

In the following, we learn an ARC model using the CPAR (Classification based on Predictive Association Rules) algorithm using its default settings.

```
rmBASE <- CPAR(train_disc, formula = as.formula(paste(classAtt, "~ .")))
predictionBASE <- predict(rmBASE, test_disc)
inspect(rmBASE$rules)
cat("Number of rules: ", length(rmBASE$rules))
cat("Total conditions: ", sum(rmBASE$rules@lhs@data))
cat("Accuracy on test data: ", mean(predictionBASE == y_true))
```

In this case, the rule model is composed of seven rules. Note that the last field on the output containing the Laplace statistics was omitted for brevity.

lhs	rhs	support	confidence	lift
[1] {Petal.Length=[-Inf;2.6]}	=> {Species=setosa}	0.32	1.0000000	3.125000
[2] {Petal.Width=[-Inf;0.8]}	=> {Species=setosa}	0.32	1.0000000	3.125000
[3] {Petal.Length=(5.15; Inf], Petal.Width=(1.75; Inf]}	=> {Species=virginica}	0.25	1.0000000	2.777778
[4] {Petal.Width=(1.75; Inf]}	=> {Species=virginica}	0.33	0.9705882	2.696078
[5] {Sepal.Length=(5.55; Inf], Petal.Length=(2.6;4.75], Petal.Width=(0.8;1.75]}	=> {Species=versicolor}	0.21	1.0000000	3.125000
[6] {Petal.Length=(2.6;4.75]}	=> {Species=versicolor}	0.26	0.9629630	3.009259
[7] {Petal.Width=(0.8;1.75]}	=> {Species=versicolor}	0.31	0.9117647	2.849265

The statistics are:

```
"Number of rules: 7 Total conditions: 10 Accuracy on test data: 0.96"
```

4.3 Configuring QCBA optimizations and printing statistics

For our demonstration purposes, we will set up a generic `qCBA` configuration (variable `baseModel_arc`), which initially disables all optimizations. To avoid repeating code for passing long argument lists to `qcba()` and for printing model statistics such as model size and accuracy on test data, we will also introduce the helper function `qcba_with_summary()`.

```
baseModel_arc <- arulesCBA2arcCBAModel(
  arulesCBAModel = rmBASE,
```

```

    cutPoints = cutPoints,
    rawDataSet = trainFold,
    classAtt = classAtt
)
qcbaParams <- list(
  cbaRuleModel = baseModel_arc,
  datadf = trainFold,
  extend = "noExtend",
  attributePruning = FALSE,
  continuousPruning = FALSE,
  postpruning = "none",
  trim_literal_boundaries = FALSE,
  defaultRuleOverlapPruning = "noPruning",
  minImprovement = 0,
  minCondImprovement = 0
)

qcba_with_summary <- function(params) {
  rmQCBA <- do.call(qcba, params)
  cat("Number of rules: ", nrow(rmQCBA@rules), " ")
  cat("Total conditions: ", sum(rmQCBA@rules$condition_count), " ")
  accuracy <- CBARuleModelAccuracy(predict(rmQCBA, testFold), testFold[[classAtt]])
  cat("Accuracy on test data: ", round(accuracy, 2))
  print(rmQCBA@rules)
}

```

4.4 QCBA Refit

The following code will postoptimize the previously learnt CBA model using the refit optimization:

```
qcba_with_summary(qcbaParams)
```

This will output the following list of eight rules (note that in this and the following printouts, the columns with rule measures are omitted for brevity):

```

1          {Petal.Length=[-Inf;1.9]} => {Species=setosa}
2          {Petal.Width=[-Inf;0.6]} => {Species=setosa}
3          {Petal.Length=[5.2;Inf],Petal.Width=[1.8;Inf]} => {Species=virginica}
4  {Sepal.Length=[5.6;Inf],Petal.Length=[3.3;4.7],Petal.Width=[1;1.7]} => {Species=versicolor}
5          {Petal.Width=[1.8;Inf]} => {Species=virginica}
6          {Petal.Length=[3.3;4.7]} => {Species=versicolor}
7          {Petal.Width=[1;1.7]} => {Species=versicolor}
8          {} => {Species=virginica}

```

The intervals (except for boundaries set to infinity) were shortened. For example, for the first rule, the training data does not contain any data point with Petal.Length=2.6 (original boundary) but it does contain the value 1.9 (new boundary).

```
any(trainFold$Petal.Length == 1.9)
any(trainFold$Petal.Length == 2.6)
```

The first returns TRUE and the second FALSE.

The statistics are

```
[1] "Number of rules: 8 Total conditions: 10 Accuracy on test data: 0.96"
```

While this is one rule more than the CPAR model, this extra rule is the explicitly included default rule (rule #8). In the [arulesCBA](#) CPAR model, the default rule is included in a separate slot (`rmBASE$default`) and is the same virginica class as the one in the QCBA rule set.

Recall that rules are applied from top to bottom. A careful inspection of the rule list shows that it contains rule 4, which is a special case of rule 7 (both predicting the versicolor class). However, neither of these rules can be removed without impact on the predictions. If the more specific rule 4 was removed, some instances would be classified differently, as more instances would reach rule 5, which would classify them as virginica. Similarly, the classification would change if we replace rule 4 with rule 7 or rule 7 with rule 4.

4.5 Adjusting boundaries and attribute pruning

We will demonstrate extension, trimming and attribute pruning steps simultaneously for efficient presentation.

```
qcbaParams$attributePruning <- TRUE
qcbaParams$trim_literal_boundaries <- TRUE
qcbaParams$extend <- "numericOnly"
qcba_with_summary(qcbaParams)
```

The list of resulting rules is

1 {Petal.Length=[-Inf;1.9]}	=> {Species=setosa}
2 {Petal.Width=[-Inf;0.6]}	=> {Species=setosa}
3 {Petal.Length=[5.2;Inf]}	=> {Species=virginica}
4 {Sepal.Length=[5;Inf], Petal.Length=[3.3;4.7]}	=> {Species=versicolor}
5 {Petal.Width=[1.8;Inf]}	=> {Species=virginica}
6 {Petal.Length=[3.3;4.7]}	=> {Species=versicolor}
7 {Petal.Width=[1;1.7]}	=> {Species=versicolor}
8 {}	=> {Species=virginica}

As can be seen, the adjustment of intervals resulted in a change in the boundary for Sepal.length in rule #4. The attribute pruning removed extra conditions from rules #3 and #4 resulting in a smaller model with overall improved test accuracy:

```
"Number of rules: 8 Total conditions: 8 Accuracy on test data: 1"
```

4.6 Postpruning

The postpruning is performed on a model resulting from all previous steps.

```
qcbaParams$postpruning <- "cba"
qcba_with_summary(qcbaParams)
```

The result is a substantially reduced rule list:

1 {Petal.Length=[1;1.9]}	=> {Species=setosa}
2 {Petal.Length=[5.2;Inf]}	=> {Species=virginica}
3 {Sepal.Length=[5;Inf], Petal.Length=[3.3;4.7]}	=> {Species=versicolor}
4 {Petal.Width=[1.8;Inf]}	=> {Species=virginica}
5 {}	=> {Species=versicolor}

The statistics are:

```
"Number of rules: 5 Total conditions: 5 Accuracy on test data: 1.0"
```

As can be seen, postpruning reduced the model size significantly, and in this case, the model has even better accuracy than the base CPAR model. However, in some other cases, a small average decrease in accuracy as a result of pruning has been shown in benchmarks in [Kliegr and Izquierdo \(2023\)](#).

4.7 Default rule pruning

The following code demonstrates the standard strategy for default rule pruning. As outlined earlier, this often provides an effective way to reduce the rule count, however, sometimes at the expense of slightly lower accuracy.

```
qcbaParams$defaultRuleOverlapPruning <- "transactionBased"
qcba_with_summary(qcbaParams)
```

The result is the final reduced rule list with the removed rule #3 from the previous print-out. This rule classifies to the versicolor class. Since it is the default class, instances that were covered by this rule will be covered by the default rule. The original rule #4 covers – considering its position in the rule list – different instances of training data and, therefore, will not interfere with this.

```

1 {Petal.Length=[1;1.9]} => {Species=setosa}
2 {Petal.Length=[5.2;Inf]} => {Species=virginica}
3 {Petal.Width=[1.8;Inf]} => {Species=virginica}
4 {}                      => {Species=versicolor}

```

The statistics for the final model are:

```
Number of rules: 4 Total conditions: 3 Accuracy on test data: 1.0
```

Compared to the original CPAR model, the number of rules dropped from 8 (including the default rule) to 4, the number of conditions dropped from 10 to 3, and the accuracy increased by 0.04 points. This is an illustrative example and actual results may vary for a particular dataset. On average, the improvement reported over CPAR as measured on 22 benchmark datasets was a 2% improvement in accuracy, a 40% reduction in the number of rules and a 29% reduction in the number of conditions Kliegr and Izquierdo (2023). More details on the benchmarks are covered in Section [Built-in benchmark support](#).

5 Interoperability with Rule Learning Packages in CRAN

The **qCBA** package is able to process CBA models produced by all three CBA implementations in CRAN: **arc**, **arulesCBA**, **rCBA**. Additionally, it can process other rule models generated by **arulesCBA** such as CPAR and SBRL models generated by the package **sbrl**.

On the input, `qcba()` requires an instance of `CBARuleModel`, which has the following slots:

- `rules`: list of rules in the model (instance of `rules` from `arules` package).
- `cutp`: specification of cutpoints used to discretize numerical attributes,
- `classAtt`: name of the target attribute,
- `attTypes`: types of the attributes (numeric or factor).

As shown below, the instance of this class is created automatically when used with **arc** and through prepared helper functions for other libraries. The code examples in the following are built on the data preparation described in Subsection [Data](#).

5.1 **arc** package

As the **arc** package was specifically designed for **qCBA**, it outputs an instance of the `CBARuleModel` class, which is accepted by the `qcba()` function. The postprocessing can thus be directly applied to the result of `cba()`.

```
rmCBA <- cba(datadf = trainFold, classAtt = "Species")
rmqCBA <- qcba(cbaRuleModel = rmCBA, datadf = trainFold)
```

By default, the function `cba()` from the **arc** package learns candidate association rules using automatic threshold detection using the heuristic algorithm from (Kliegr and Kuchar, 2019). Therefore, no support and confidence thresholds have to be passed. A more complex case involving custom discretization and thresholds was demonstrated in Section [Primer on building rule-based classifiers for QCBA in R](#).

5.2 **arulesCBA** package

Compared to the previous example, there is an extra line with a call to a helper function.

```
library(arulesCBA)
arulesCBAModel <- arulesCBA::CBA(Species ~ ., data = train_disc, supp = 0.1, conf = 0.9)
CBAModel <- arulesCBA2arcCBARuleModel(arulesCBAModel, discrModel$cutp, iris, classAtt)
qCBAModel <- qcba(cbaRuleModel = CBAModel, datadf = iris)
```

Note that we passed a prediscritized data in `irisDisc` to the package `arulesCBA`. While the `arulesCBA` package allows for supervised discretization using the `arulesCBA::discretizeDF.supervised()` method, the cutpoints determined during the discretization are not exposed in a machine-readable way. Therefore, when `arulesCBA` is used in conjunction with `qCBA`, the discretization should be performed using `arc::discrNumeric()`. Since both `arc` and `arulesCBA` internally use the MDLP method from the `discretization` package, this should not influence the results.

5.3 rCBA package

The logic of the use of `rCBA` package is similar to that of the previously covered package; it is only necessary to ensure the use of a different conversion function:

```
library(rCBA)
rCBAmode1 <- rCBA::build(train_disc)
CBAmodel <- rcbaModel2CBARuleModel(rCBAmode1, discrModel$cutp, iris, "Species")
qCBAmode1 <- qcba(CBAmodel, iris)
```

Confidence and support thresholds are not specified as `rCBA` uses automatic confidence and threshold tuning using the simulated annealing algorithm from (Kliegr and Kuchar, 2019). The `rCBA` package internally uses a Java implementation of CBA, which may result in faster performance on some datasets than `arulesCBA` (Hahsler et al., 2019).

5.4 sbrl and other packages

The logic of the use of `qCBA` package for `sbrl` is similar; it is again only necessary to use a dedicated conversion function `sbrlModel2arcCBARuleModel()`.

An example for `sbrl` is contained in the `qCBA` package documentation, as `sbrl` requires additional preprocessing and postprocessing: `sbrl` requires a specially named target attribute, allows only for binary targets, and outputs probabilities rather than specific class predictions.

For compatibility with packages that do not use the `arules` data structures, there is also the `customCBARuleModel` class, which takes rules as a dataframe conforming to the format used in `arules` that can be obtained with `(as(rules, "data.frame"))`.

6 Built-in benchmark support

The `qCBA` package has built-in support for benchmarking over all supported types of algorithms covered in the previous section. This includes `arulesCBA` implementations of CBA, CMAR, CPAR, PRM and FOIL2 (Quinlan and Cameron-Jones, 1993).

By default, an average of two runs of each algorithm is performed.

```
# learn with default metaparameter values
stats <- benchmarkQCBA(train = trainFold, test = testFold, classAtt = classAtt)
print(stats)
```

The result of the last printout is

	CBA	CMAR	CPAR	PRM	FOIL2	CBA_QCBA	CMAR_QCBA	CPAR_QCBA	PRM_QCBA	FOIL2_QCBA
accuracy	1.00	0.960	0.960	0.960	0.960	1.000	1.000	1.000	1.000	0.960
rulecount	5.00	25.000	7.000	6.000	8.000	4.000	4.000	4.000	4.000	5.000
modelsiz	5.00	52.000	10.000	9.000	13.000	3.000	3.000	3.000	3.000	5.000
buildtime	0.05	0.535	0.215	0.205	0.215	0.058	0.138	0.059	0.059	0.081

The function can be easily turned into a comparison with `round((stats[,6:10] / stats[,1:5]) - 1), 3)`, the result is then:

	CBA_QCBA	CMAR_QCBA	CPAR_QCBA	PRM_QCBA	FOIL2_QCBA
accuracy	0.000	0.042	0.042	0.042	0.000
rulecount	-0.200	-0.840	-0.429	-0.333	-0.375
modelsiz	-0.400	-0.942	-0.700	-0.667	-0.615
buildtime	0.204	-0.737	-0.681	-0.722	-0.665

This shows that on the iris, depending on the base algorithm, QCBA decreased the rule count between 20% and 84%, while accuracy remained unchanged or increased by about 4%. The last row shows that the time required by QCBA is, for four out of five studied reference algorithms, lower than what it takes to train the input model by the corresponding algorithm. The `benchmarkQCBA()` function can also accept custom metaparameters and selected base rule learners. The user can also choose the number of runs (`iterations` parameter) and obtain the resulting models from the last iteration.

Since the outputs of some learners may depend on chance, the function also allows setting the random seed through the optional `seed` argument. Note that the provided seed is not used for splitting data, which needs to be performed externally. This approach provides the most control for the user, avoids replicating code in other R packages and functions aimed at splitting data, and also improves reproducibility.

```
output <- benchmarkQCBA(
  trainFold,
  testFold,
  classAtt,
  train_disc,
  test_disc,
  discrModel$cutp,
  CBA = list(support = 0.05, confidence = 0.5),
  algs = c("CPAR"),
  iterations = 10,
  return_models = TRUE,
  seed = 1
)

message("Evaluation statistics")
print(output$stats)
message("CPAR model")
inspect(output$CPAR[[1]])
message("QCBA model")
print(output$CPAR_QCBA[[1]])
```

This will produce `output` with a list of rules similar to the final CPAR model presented in Section [Demonstration of individual QCBA optimization steps](#). A more complex benchmark of computational costs is presented in [Kliegr and Izquierdo \(2023\)](#), which also includes the study of various data sizes and the effect of varying the number of unique values in the dataset. A brief overview of the main results was presented in Subsection [Computational costs for large datasets](#).

A GitHub repository <https://github.com/kliegr/arcBench> contains scripts that extend this workflow into automation across multiple datasets and materialized splits in each dataset. It also includes support for benchmarking additional rule learning algorithms, including SBRL, Python packages producing IDS models ([Lakkaraju et al., 2016](#)) and Weka libraries for RIPPER ([Cohen, 1995](#)) and FURIA ([Hühn and Hüllermeier, 2009](#)). Detailed benchmarking results are included in ([Kliegr and Izquierdo, 2023](#)).

7 Conclusions

Quantitative CBA ameliorates one of the major drawbacks of association rule classification, the adherence of rules comprising the classifier to the multidimensional grid created by discretization of numerical attributes. By working with the original continuous data, the algorithm can improve the fit of the rules and consequently reduce their count. The QCBA algorithm does not introduce any mandatory thresholds or meta-parameters for the user to set or optimize, although it does allow disabling the individual optimizations. The `qCBA` package implements QCBA, allowing the postprocessing of the output of all three CBA implementations currently in CRAN. The package can also be used in conjunction with other association rule-based models, including those producing rule sets and using multi-rule classification.

The QCBA algorithm is described in detail in Kliegr and Izquierdo (2023), the package documentation is available in Kliegr (2024), and additional information is available at <https://github.com/kliegr/qcba>, which also features an interactive RMarkdown tutorial supplementing this paper.

8 Acknowledgment

The author thanks the Faculty of Informatics and Statistics, Prague University of Economics and Business, for long-term institutional support of research activities.

References

- R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94*, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc. ISBN 1-55860-153-8. [p157]
- M. Azmi and A. Berrado. RCAR framework: building a regularized class association rules model in a categorical data space. In *Proceedings of the 13th International Conference on Intelligent Systems: Theories and Applications*, pages 1–6, 2020. [p156]
- W. W. Cohen. Fast effective rule induction. In *Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995. [p170]
- U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *13th International Joint Conference on Uncertainty in Artificial Intelligence (IJCAI93)*, pages 1022–1029, 1993. [p156]
- M. Hahsler, B. Grün, and K. Hornik. Introduction to arules—mining association rules and frequent item sets. *SIGKDD Explor*, 2(4):1–28, 2007. [p160]
- M. Hahsler, S. Chelluboina, K. Hornik, and C. Buchta. The arules r-package ecosystem: analyzing interesting patterns from large transaction data sets. *Journal of Machine Learning Research*, 12(Jun):2021–2025, 2011. [p156]
- M. Hahsler, I. Johnson, T. Kliegr, and J. Kuchař. Associative classification in R: arc, arulescba, and rcba. *R Journal*, 9(2), 2019. [p156, 169]
- J. Hühn and E. Hüllermeier. FURIA: an algorithm for unordered fuzzy rule induction. *Data Mining and Knowledge Discovery*, 19(3):293–319, 2009. [p170]
- T. Kliegr. *Association Rule Classification*, 2016. URL <https://CRAN.R-project.org/package=arc>. R package version 1.1.3. [p156]
- T. Kliegr. *qCBA: Quantitative Classification by Association Rules*, 2024. URL <https://CRAN.R-project.org/package=qCBA>. R package version 1.0. [p171]
- T. Kliegr and E. Izquierdo. QCBA: improving rule classifiers learned from quantitative data by recovering information lost by discretisation. *Applied Intelligence*, 53(18):20797–20827, 2023. [p156, 157, 158, 162, 163, 164, 167, 168, 170, 171]
- T. Kliegr and J. Kuchar. Tuning hyperparameters of classification based on associations (CBA). In *ITAT*, pages 9–16, 2019. [p158, 168, 169]
- J. Kuchař and T. Kliegr. *rCBA: CBA Classifier for R*, 2019. URL <https://CRAN.R-project.org/package=rCBA>. R package version 0.4.3. [p156]
- H. Lakkaraju, S. H. Bach, and J. Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16*, pages 1675–1684, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. [p170]

- W. Li, J. Han, and J. Pei. Cmar: Accurate and efficient classification based on multiple class-association rules. In *Proceedings of the 2001 IEEE International Conference on Data Mining, ICDM '01*, pages 369–376, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1119-8. [p157]
- B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, KDD'98*, pages 80–86. AAAI Press, 1998. [p156, 157, 159]
- T. G. Michael Hahsler, Ian Johnson. *arulesCBA: Classification Based on Association Rules*, 2024. URL <https://CRAN.R-project.org/package=arulesCBA>. R package version 1.2.7. [p156]
- J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993. [p158]
- J. R. Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. In *European conference on machine learning*, pages 1–20. Springer, 1993. [p169]
- K. Vanhoof and B. Depaire. Structure of association rule classifiers: a review. In *2010 International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pages 9–12, November 2010. [p157]
- H. Yang, C. Rudin, and M. Seltzer. Scalable Bayesian rule lists. In *International conference on machine learning*, pages 3921–3930. PMLR, 2017. [p156]
- H. Yang, C. Rudin, and M. Seltzer. *sbrl: Scalable Bayesian Rule Lists Model*, 2024. URL <https://CRAN.R-project.org/package=sbrl>. R package version 1.4. [p156, 157]
- X. Yin and J. Han. CPAR: Classification based on predictive association rules. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 331–335. SIAM, 2003. [p157]

Tomas Kliegr
Prague University of Economics and Business
Winston Churchill Sq. 4, Prague
Czech Republic
<https://nb.vse.cz/~kliet01>
ORCID: 0000-0002-7261-0380
tomas.kliegr@vse.cz

simlandr: Simulation-Based Landscape Construction for Dynamical Systems

by Jingmeng Cui, Merlijn Olthof, Anna Lichtwarck-Aschoff, Tiejun Li, and Fred Hasselman

Abstract We present the simlandr package for R, which provides a set of tools for constructing potential landscapes for dynamical systems using Monte Carlo simulation. Potential landscapes can be used to quantify the stability of system states. While the canonical form of a potential function is defined for gradient systems, generalized potential functions can also be defined for non-gradient dynamical systems. Our method is based on the potential landscape definition from the steady-state distribution, and can be used for a large variety of models. To facilitate simulation and computation, we introduce several novel features, including data structures optimized for batch simulations under varying conditions, an out-of-memory computation tool with integrated hash-based file-saving systems, and an algorithm for efficiently searching the minimum energy path. Using a multitable cell differentiation model as examples, we illustrate how simlandr can be used for model simulation, landscape construction, and barrier height calculation. The simlandr package is available at <https://CRAN.R-project.org/package=simlandr>, under GPL-3 license.

1 Introduction

To better understand a dynamical system, it is often important to know the stability of different states. The metaphor of a potential landscape consisting of hills and valleys has been used to illustrate differences in stability in many fields, including genetics (Wang et al., 2011; Waddington, 1966), ecology (Lamothe et al., 2019), and psychology (Olthof et al., 2023). In such a landscape, the stable states of the system correspond to the lowest points (minima) in the valleys of the landscape. Just like a ball that is thrown in such a landscape will eventually gravitate towards such a minimum, the dynamical system is conceptually more likely to visit its stable states in which the system is also more resilient to noise. For example, in the landscape metaphor of psychopathology (Figure 1), the valleys represent different mental health states, their relative depth represents the relative stability of the states, and the barriers between valleys represent the difficulty of transitioning between these states (Olthof et al., 2023, Hayes and Andrews (2020)). When the healthy state is more stable, the person is more likely to stay mentally healthy, whereas when the maladaptive state is more stable, the person is more likely to suffer from mental disorders.

Yet, formally quantifying the stability of states is a nontrivial question. Here we present an R package, `simlandr`, that can quantify the stability of various kinds of systems without many mathematical restrictions.

Dynamical systems are usually modeled by stochastic differential equations, which may depend on the past history (i.e., may be non-Markovian, Stumpf et al. (2021)). They take the general form of

$$dX_t = b(X_t, H_t)dt + \sigma(X_t, H_t)dW, \quad (1)$$

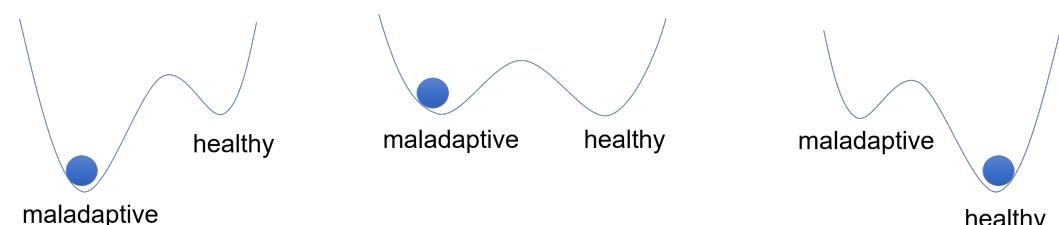


Figure 1: Illustration of the ball-and-landscape metaphor commonly used in the field of psychopathology.

where X_t is the random variable representing the current state of the system and H_t represents the past history of the system $H_t = \{X_s | s \in [0, t)\}$ ¹. The first term on the right-hand side of Eq (1) represents the deterministic part of the dynamics, which is a function of the system's current state $b(X_t, H_t)$. The second term represents the stochastic part, which is standard white noise dW multiplied by the noise strength $\sigma(X_t, H_t)$.

If the dynamical equation (Eq (1)) can be written in the following form

$$dX = -\nabla U dt + \sqrt{2} dW, \quad (2)$$

then U is the potential function of the system.². However, this is not possible for general dynamical systems. The trajectory of such a system may contain loops which are not possible to be represented by a gradient system (this issue was compared to Escher's stairs by Rodríguez-Sánchez et al. (2020)). In this case, further generalization is needed. The theoretical background of **simlandr** is the generalized potential landscape by Wang et al. (2008), which is based on the Boltzmann distribution and the steady-state distribution of the system. The Boltzmann distribution is a distribution law in physics, which states the distribution of classical particles depends on the energy level they occupy. When the energy is higher, the particle is exponentially less likely to be in such states

$$P(x) \propto \exp(-U). \quad (3)$$

This is then linked to dynamical systems by the steady-state distribution. The steady-state distribution of stochastic differential equations is the distribution that does not change over time, denoted by P_{SS} which satisfies

$$\frac{\partial P_{SS}(x, t)}{\partial t} = 0. \quad (4)$$

The steady-state distribution is important because it extracts time-invariant information from a set of stochastic differential equations. Substituting the steady-state distribution to Eq (3) gives Wang's generalized potential landscape function (Wang et al., 2008)

$$U(x) = -\ln P_{SS}(x). \quad (5)$$

If the system has ergodicity (i.e., after sufficient time it can travel to all possible states in the state space), the long-term sample distribution can be used to estimate the steady-state distribution, and the generalized potential function can be calculated.

Our approach is not the only possible way for constructing potential landscapes. Many other theoretical approaches are available, including the SDE decomposition method by Ao (2004) and the quasi-potential by Freidlin and Wentzell (2012). Various strategies to numerically compute these landscapes have been proposed (see Zhou and Li (2016) for a review). However, available realizations are still scarce. To our knowledge, besides **simlandr**, there are two existing packages specifically for computing potential landscapes: the **waydown** package (Rodríguez-Sánchez, 2020) and the **QPot** package (Moore et al., 2016; Dahiya and Cameron, 2018). The **waydown** package uses the skew-symmetric decomposition of the Jacobian, which theoretically produces landscapes that are similar to Wang et al. (2008) (but see Cui et al. (2023a) for a potential technical issue with this package.). The **QPot** package uses a path integral method that produces quasi-landscapes following the definition by Freidlin and Wentzell (2012). Because of the analytical methods used by **waydown** and **QPot**, they both require the dynamic function to be Markovian and differentiable in the whole state space. Moreover, they can only be used for systems of up to two dimensions. **simlandr**, in contrast, is based on Monte Carlo simulation and the steady-state distribution. It does not have specific requirements for the model. Even for models that are not globally

¹The corresponding variable representing positions in the state space is not a random variable, so we use lowercase x for it. This convention will be followed throughout this article.

²Under zero inertia approximation.

differentiable, have history-dependence, and are defined in a high-dimensional space, `simlandr` is still applicable (e.g., Cui et al. (2023b)). Therefore, `simlandr` can be applied to a much wider range of dynamical systems, illustrate a big picture of dominated attractors, and investigate how the stability of different attractors may be influenced by model parameters. As a trade-off, `simlandr` is not designed for rare events sampling in which the noise strength $\sigma(X)$ is extremely small, nor for the precise calculation of the tail probability and transition paths. Instead, it is better to view `simlandr` as a semi-quantitative tool that provides a broad overview of key attractors in dynamic systems, allowing for comparisons of their relative stability and the investigation of system parameter influences. We will show some typical use cases of `simlandr` in later sections. Some key terms used in this article are summarized in Table 1.

Table 1: Summary of key terms used in this article.

Term	Explanation
Potential landscape metaphor	A conceptual metaphor representing the stability of a complex dynamic system as an uneven landscape, with a ball on it representing the system's state. This can be quantitatively realized in various ways (Zhou and Li, 2016).
Gradient system	A system whose deterministic motion can be described solely by the gradient of a potential function.
Non-Markovian system	A system whose future evolution depends not only on its current state but also on its past history.
Steady-state distribution	The probability distribution of a dynamic system that remains unchanged over time.
Ergodic system	A dynamic system that, given enough time, will eventually pass through all possible states.
Minimum energy path (MEP)	A transition path linking two local minima and passing through a saddle point (Wan et al., 2024). It is always parallel to the gradient of the energy landscape, representing an efficient transition route.

2 Design and implementation

The general workflow of `simlandr` involves three steps: model simulation, landscape construction, and barrier height computation. See Figure 2 for a summary.

2.1 Step 1: model simulation

For the first step, a simulation function should be created by the user. This function should be able to simulate how the dynamical system of interest evolves over time and record its state in every time step. This can often be done with the Euler-Maruyama method. If the SDEs are up to three dimensions and Markovian, a helper function from `simlandr`, `sim_SDE()`, can also be used. This function is based on the simulation utilities from the `Sim.DiffProc` (Guidoum and Boukhetala, 2020) and the output can be directly used for later steps. Moreover, the `multi_init_simulation()` function can be used to simulate trajectories from various starting points, thus reducing the possibility for the system to be trapped in a local minimum. The `multi_init_simulation()` function also supports parallel simulation based on the `future` framework to improve time efficiency Bengtsson (2021).

For Monte Carlo methods, it is important that the simulation *converges*, which means the distribution of the system is roughly stable. The precision of the steady-state distribution estimation, according to Eq (5), determines the precision of the distribution estimation. `simlandr` provides a visual tool to compare the sample distributions in different stages

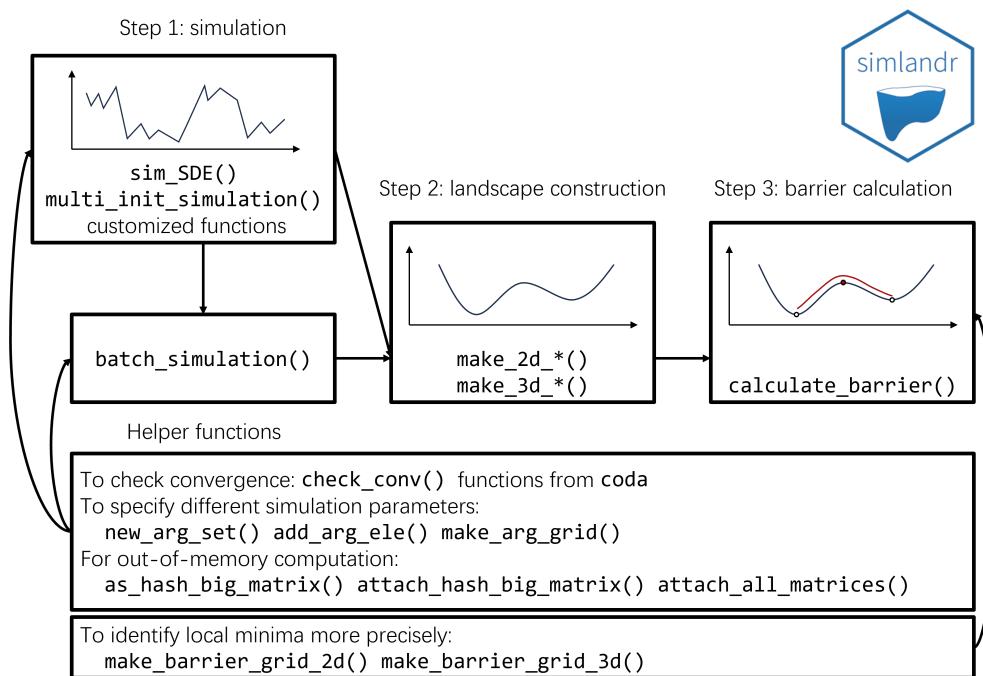


Figure 2: The structure and workflow of `simlandr`.

(`check_conv()`), whereas the `coda` package (Plummer et al., 2006) can be used for more advanced diagnostics. The output of the `sim_SDE()` and `multi_init_simulation()` functions also uses the classes from the `coda` package to enable easy convergence diagnosis. To achieve ergodicity in reasonable time, sometimes stronger noises need to be added to the system.

A simulation function is sufficient if the user is only interested in a single model setting. If the model is parameterized and the user wants to investigate the influence of parameters on the stability of the system, then multiple simulations need to be run with different parameter settings. `simlandr` provides functions to perform batch simulations and store the outputs for landscape construction as one object. This can later be used to compare the stability under different parameter settings or produce animations to show how a model parameter influences the stability of the model.

In many cases, the output of the simulation is so large that it cannot be properly stored in the memory. `simlandr` provides a `hash_big_matrix` class, which is a modification of the `big.matrix` class from the `bigmemory` package (Kane et al., 2013), that can perform out-of-memory computation and organize the data files in the disk. In an out-of-memory computation, the majority of the data is not loaded into the memory, but only the small subset of data that is used for the current computation step. Therefore, the memory occupation is dramatically reduced. The `big.matrix` class in the `bigmemory` package provides a powerful tool for out-of-memory computation. It, however, requires an explicit file name for each matrix, which can be cumbersome if there are many matrices to be handled, and this is likely to be the case in a batch simulation. The `hash_big_matrix` class automatically generates the file names using the md5 values of the matrices with the `digest` package (Eddelbuettel et al., 2021) and stores it within the object. Therefore, the file links can also be restored automatically.

2.2 Step 2: landscape construction

`simlandr` provides a set of tools to construct 2D, 3D, and 4D³ landscapes from single or multiple simulation results. The steady-state distribution for selected variables of the system

³In this package, we use the number of dimensions in landscape plots (including U to define the dimension of landscapes. The x -, y -, z -, and color- axes can all be regarded as a dimension. Therefore, the dimension of a landscape can be one more than the dimension of the kernel smooth function.

is first estimated using the kernel density estimates (KDE). The density function in R is used for 2D landscapes, whereas the `ks` package (Chacón and Duong, 2018) is used by default for 3D and 4D landscapes because of its higher efficiency. Then the potential function U is calculated from Eq (5). The landscape plots without a z-axis are created with `ggplot2` (Wickham, 2016), and those with a z-axis are created with `plotly` (Sievert, 2020). These plots can be further refined using the standard `ggplot2` or `plotly` methods. See Table 2 for an overview for the family of landscape functions.

Table 2: Overview of various landscape functions provided by `simlandr`. Dimensions in bold represent the potential U calculated by the function. Dimensions in italic represent model parameters. Dimensions in parentheses are optional.

Type of Input	Function	Dimensions
Single simulation data	<code>make_2d_static()</code>	<code>x, y</code>
	<code>make_3d_static()</code>	<code>1 x, y, z+color; (2) x, y, color</code>
	<code>make_4d_static()</code>	<code>x, y, z, color</code>
Multiple simulation data	<code>make_2d_matrix()</code>	<code>x, y, cols, (rows)</code>
	<code>make_3d_matrix()</code>	<code>x, y, z+color, cols, (rows)</code>
	<code>make_3d_animation()</code>	<code>1 x, y, z+color, fr; (2) x, y, color, fr; (3) x, y, z+color, cols</code>

2.3 Step 3: barrier height calculation

An important property of the states in a landscape is their stability, which can be indicated by the barrier height that the system has to overcome when it transitions between one stable state to another adjacent state (see Cui et al. (2023b) for further discussions about different stability indicators). The barrier height is also related to the escape time that the system transitions from one valley to another, which can be tested empirically (Wang et al., 2008). `simlandr` provides tools to calculate the barrier heights from landscapes. These functions look for the local minima in given regions and try to find the saddle point between two local minima. The potential differences between the saddle point and local minima are calculated as barrier heights.

In 2D cases, there is only one possible path connecting two local minima. The point on the path with the highest U is identified as the saddle point. For 3D landscapes, there are multiple paths between two local minima. If we treat the system *as if* it is a gradient system with Brownian noise, then the most probable path (termed as the *minimum energy path*, MEP) that the system transitions is that it first goes along the steepest *ascent* path from the starting point, and then goes along the steepest *descent* path to the end point (E and Vanden-Eijnden, 2010). We find this path by minimizing the following action using the Dijkstra (1959) algorithm (Heymann and Vanden-Eijnden, 2008)

$$\varphi_{\text{MEP}} = \arg \min_{\varphi} \int_A^B |\nabla U| |\text{d}\varphi| \left(\approx \min_{\varphi} \sum_i |\nabla U_i| |\Delta \varphi_i| \right), \quad (6)$$

where A and B are the starting and end points and φ is the path starting at A and ending in B . After that, the point with the maximum potential value on the MEP is identified as the saddle point. Note that while the barrier height still indicates the stability of local minima, the MEP may not be the true most probable path for a nongradient system to transition between stable states.

3 Examples

We use two dynamical systems to illustrate the usage of the `simlandr` package. The first one is a two-dimensional stochastic non-gradient gene expression model, which was used by

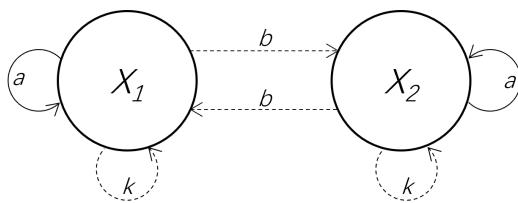


Figure 3: A graphical illustration of the relationship between the activation levels of the two genes. Solid arrows represent positive relationships (i.e., activation) and dashed arrows represent negative relationships (i.e., inhibition).

Wang et al. (2011) to represent cell development and differentiation. The second example is a dynamic model of panic disorder (Robinaugh et al., 2024) which contains many more variables and parameters, non-Markovian property, and non-differentiable formulas. We mainly use the first example to show the agreement of the results from `simlandr` with previous analytic results, and the second example as a typical use case of a complex dynamic model which is not treatable with other methods (also see Cui et al. (2023b) for more substantive discussions). Note that, both systems include more than two variables, making it impossible to perform the landscape analysis with other available R packages.

3.1 Example 1: the gene expression model

This model is built on the mutual regulations of the expressions of two genes, in which X_1 and X_2 represent the expression levels of two genes which activate themselves and inhibit each other. A graphical illustration is shown in Figure 3 (adapted from Wang et al. (2011)). Their dynamic functions can be written as

$$\frac{dX_1}{dt} = \frac{ax_1^n}{S^n + x_1^n} + \frac{bS^n}{S^n + x_2^n} - kx_1 + \sigma_1 \frac{dW_1}{dt}, \quad (7)$$

$$\frac{dX_2}{dt} = \frac{ax_2^n}{S^n + x_2^n} + \frac{bS^n}{S^n + x_1^n} - kx_2 + \sigma_2 \frac{dW_2}{dt}, \quad (8)$$

$$\frac{da}{dt} = -\lambda a + \sigma_3 \frac{dW_3}{dt}, \quad (9)$$

where a represents the strength of self-activation, b represents the strength of mutual-inhibition, and k represents the speed of degradation. The development of an organism is modeled as a decreasing at a certain speed λ . In the beginning, there is only one possible state for the cell. After a certain milestone, the cell differentiates into one of the two possible states.

This model can be simulated using the `sim_SDE()` function in `simlandr`, with the default parameter setting $b = 1$, $k = 1$, $S = 0.5$, $n = 4$, $\lambda = 0.01$, and $\sigma_1 = \sigma_2 = \sigma_3 = 0.2$.

```
# Load the package.
library(simlandr)

# Specify the simulation function.
b <- 1
k <- 1
S <- 0.5
n <- 4
lambda <- 0.01

drift_gene <- c(rlang::expr(z * x^(!!n)/((!!S)^(!!n) + x^(!!n)) + (!!b) *
  (!!S)^(!!n)/((!!S)^(!!n) + y^(!!n)) - (!!k) * x), rlang::expr(z * y^(!!n)/((!!S)^(!!n) +
  y^(!!n)) + (!!b) * (!!S)^(!!n)/((!!S)^(!!n) + x^(!!n)) - (!!k) * y),
  rlang::expr(-(!!lambda) * z)) |>
```

```

as.expression()

diffusion_gene <- expression(0.2, 0.2, 0.2)

# Perform a simulation and save the output.
set.seed(1614)
single_output_gene <- sim_SDE(drift = drift_gene, diffusion = diffusion_gene,
  N = 1e+06, M = 10, Dt = 0.1, x0 = c(0, 0, 1), keep_full = FALSE)

```

After the simulation, we perform some basic data wrangling to produce a dataset that can be used for further analysis. We create a new variable `delta_x` as the difference between X_1 (X) and X_2 (Y), and we rename the variable Z as a .

```

single_output_gene2 <- do.call(rbind, single_output_gene)
single_output_gene2 <- cbind(single_output_gene2[, "X"] - single_output_gene2[, "Y"], single_output_gene2[, "Z"])
colnames(single_output_gene2) <- c("delta_x", "a")

```

We then perform the convergence check on the simulation result. First, we convert the simulation output to the format for the `coda` package, and thin the output to speed up the convergence check.

```

single_output_gene_mcmc_thin <- as.mcmc.list(lapply(single_output_gene,
  function(x) x[seq(1, nrow(x), by = 100), ]))

```

We then show the convergence diagnosis plot to check the convergence of the simulation in Figure 4. The distribution of the two key variables in different simulation stages are converging, indicating that the simulation is long enough to provide reliable estimation of the steady-state distribution. Other convergence checks can also be readily performed using the `coda` package.

```
plot(single_output_gene_mcmc_thin)
```

We generated the 3D landscape for this model with `make_3d_single()`. Here, we use x , y to specify the variables of interest, and use `lims` to specify the limits of the x and y axes for the landscape. The `lims` argument can be left blank, then the limits will be automatically calculated.

```

l_single_gene_3d <- make_3d_single(single_output_gene2, x = "delta_x",
  y = "a", lims = c(-1.5, 1.5, 0, 1.5), Umax = 8)

```

The resulting landscape is shown in the left panel of Figure 5. In this plot, the x -axis represents $\Delta x (= x_1 - x_2)$, and the y -axis represents a . To compare with, the potential landscape obtained analytically by Wang et al. (2011) is shown in the right panel Figure 5. The result of `simlandr` appears to be very close to the result based on the analytical derivation. Note that because different normalization methods were used, the U values of the two landscapes are not directly comparable. Here, we are mainly interested in their relative shape.

```
plot(l_single_non_grad_3d)
```

We then calculate the barrier for the landscape using `calculate_barrier()`. The barrier is calculated by specifying the start and end locations, and the radii of the start and end locations. The height of the barrier from two sides can be calculated with `get_barrier_height()`.

```

b_single_gene_3d <- calculate_barrier(l_single_gene_3d, start_location_value = c(0, 1.2),
  end_location_value = c(1, 0.2), start_r = 0.3, end_r = 0.3)

get_barrier_height(b_single_gene_3d)

```

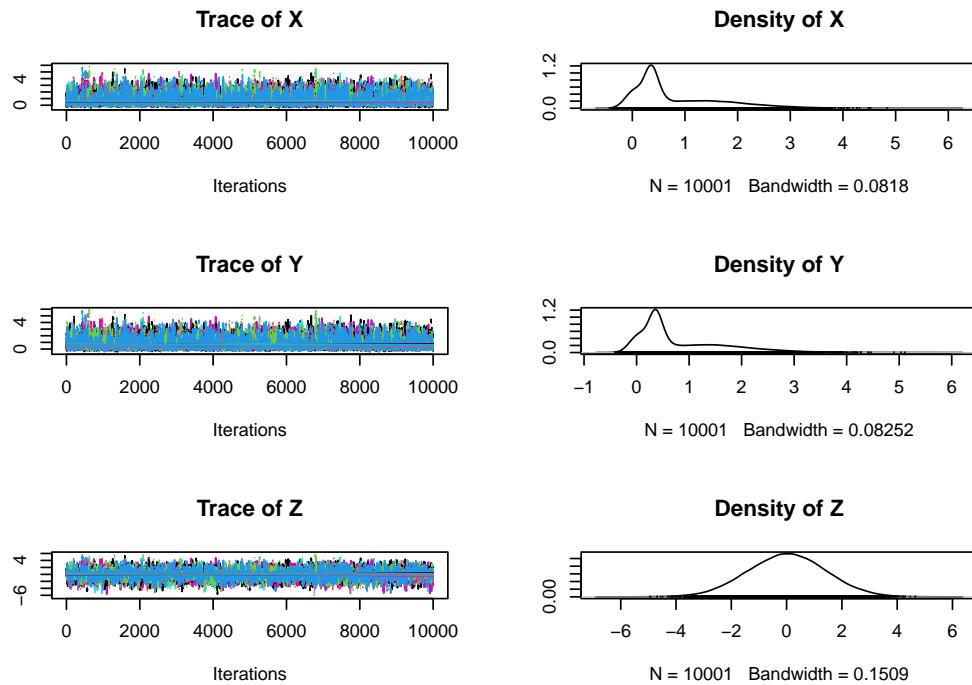


Figure 4: The convergence check result for the simulation of the gene expression model. The variables in different simulation stages did not show distributional differences, indicating that the simulation is long enough to provide a reliable estimation of the steady-state distribution.

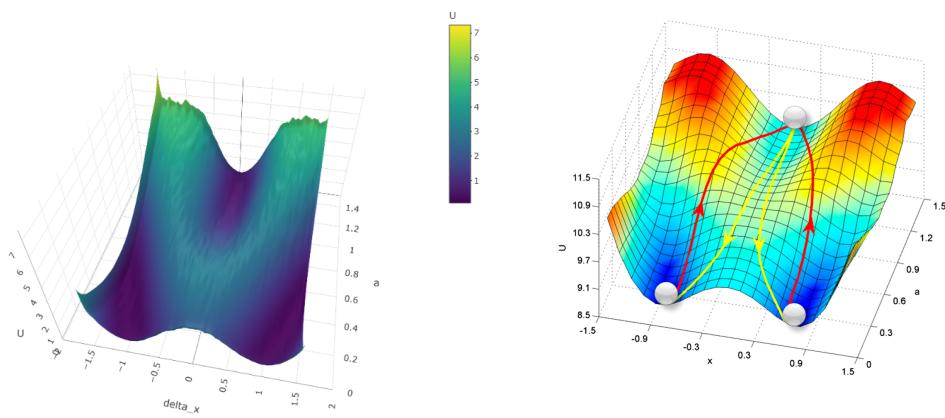


Figure 5: The 3D landscape (potential value as z-axis) for the gene expression model. The left panel is the plot produced by simlandr; the right panel is the potential landscape obtained analytically by Wang et al. (2008), reproduced with the permission of the authors and in accordance with the journal policy.

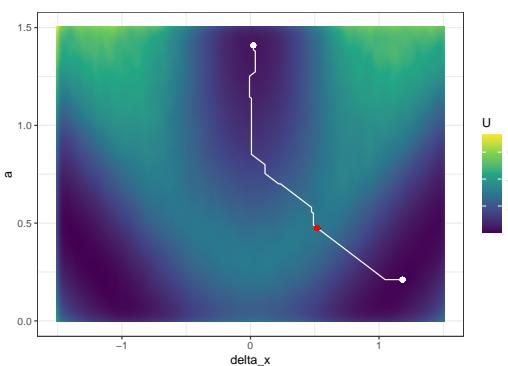


Figure 6: The landscape for the gene expression model. The local minima are marked as white dots, the saddle points are marked as red dots, and the MEPs are marked as white lines.

```
#> delta_U_start    delta_U_end
#>      2.506326     2.810604
```

The local minima, the saddle point, and the MEP can be added to the landscape with `autolayer()`, shown in Figure 6.

```
plot(l_single_gene_3d, 2) + autolayer(b_single_gene_3d)
```

Next, we use multiple simulations to investigate the influence of two parameters, k and b , on the stability of the system. As explained above, the parameter b represents the strength of mutual-inhibition between the two genes. Therefore, as b increases, we expect the differentiation is more extreme, that is, the cell is more likely to develop into one of the two cell types with very different gene expression levels. The valleys in the landscape representing the two types will become further apart and the barrier becomes clearer. The parameter k represents the speed of degradation of the gene products. As k increases, the gene products degrade faster, and this effect is more pronounced when the gene products are at high levels. Therefore, the dominant gene will express at a less extreme level, and we expect that the two valleys become closer, and the barrier will become less clear as k increases.

We use the batch simulation functions of `simlandr`. First, we create the argument set for the batch simulation. This specifies the parameters to be varied. We examined three b values, 0.5, 1, 1.5, and three k values, 0.5, 1, 1.5, which form 9 possible combinations.

```
batch_arg_set_gene <- new_arg_set()
batch_arg_set_gene <- batch_arg_set_gene |>
  add_arg_ele(arg_name = "parameter", ele_name = "b", start = 0.5, end = 1.5,
             by = 0.5) |>
  add_arg_ele(arg_name = "parameter", ele_name = "k", start = 0.5, end = 1.5,
             by = 0.5)
batch_grid_gene <- make_arg_grid(batch_arg_set_gene)
```

We then perform the batch simulation with the `batch_simulation()` function. Here, we specify the simulation function to be used, which is similar to the single simulation we showed above, together with the data wrangling procedure. The simulation function is defined with the `sim_fun` argument. We also use `bigmemory = TRUE` to store the simulation results in the `hash_big_matrix` format, which is more memory-efficient.

```
batch_output_gene <- batch_simulation(batch_grid_gene, sim_fun = function(parameter) {
  b <- parameter[["b"]]
  k <- parameter[["k"]]
  drift_gene <- c(rlang::expr(z * x^(!n)/((!!S)^(!n) + x^(!n)) + (!!b) *
```

```

(!!S)^(!!n)/((!!S)^(!!n) + y^(!!n)) - (!!k) * x), rlang::expr(z *
y^(!!n)/((!!S)^(!!n) + y^(!!n)) + (!!b) * (!!S)^(!!n)/((!!S)^(!!n) +
x^(!!n)) - (!!k) * y), rlang::expr(-(!!lambda) * z)) |>
as.expression()
set.seed(1614)
single_output_gene <- sim_SDE(drift = drift_gene, diffusion = diffusion_gene,
N = 1e+06, M = 10, Dt = 0.1, x0 = c(0, 0, 1), keep_full = FALSE)
single_output_gene2 <- do.call(rbind, single_output_gene)
single_output_gene2 <- cbind(single_output_gene2[, "X"] - single_output_gene2[, "Y"], single_output_gene2[, "Z"])
colnames(single_output_gene2) <- c("delta_x", "a")
single_output_gene2
}, bigmemory = TRUE)

```

If the output is saved in an RDS file, upon next use, it can be read as follows.

```

saveRDS(batch_output_gene, "batch_output_gene.RDS")
batch_output_gene <- readRDS("batch_output_gene.RDS") |>
attach_all_matrices()

```

We then make the 3D matrix for the batch output, using `make_3d_matrix()`.

```

l_batch_gene_3d <- make_3d_matrix(batch_output_gene, x = "delta_x", y = "a",
cols = "b", rows = "k", lims = c(-5, 5, -0.5, 2), h = 0.005, Umax = 8,
kde_fun = "ks", individual_landscape = TRUE)

```

For the barrier calculation step, the start and end points of the barrier may be different for each landscape plot. The following code shows how to create a barrier grid for each landscape plot. First, we create a barrier grid template using the function `make_barrier_grid_3d()`. Next, we modify the barrier grid template to create a barrier grid for the landscape plot.

```

make_barrier_grid_3d(batch_grid_gene, start_location_value = c(0, 1.5),
end_location_value = c(1, -0.5), start_r = 1, end_r = 1, print_template = TRUE)

#> structure(list(start_location_value = list(c(0, 1.5), c(0, 1.5),
#> ), c(0, 1.5), c(0, 1.5), c(0, 1.5), c(0, 1.5), c(0, 1.5),
#> ), start_r = list(c(1, 1), c(1, 1), c(1, 1), c(1, 1),
#> ), c(1, 1), c(1, 1), c(1, 1), c(1, 1)), end_location_value = list(
#> ), c(1, -0.5), c(1, -0.5), c(1, -0.5), c(1, -0.5), c(1, -0.5),
#> ), c(1, -0.5), c(1, -0.5), c(1, -0.5), c(1, -0.5)), end_r = list(
#> ), c(1, 1), c(1, 1), c(1, 1), c(1, 1), c(1, 1), c(1, 1),
#> ), c(1, 1), c(1, 1)), row.names = c(NA, -9L), class = c("arg_grid",
#> "data.frame"))

#>   ele_list b k start_location_value start_r end_location_value end_r
#> 1 0.5, 0.5 0.5 0.5          0.0, 1.5    1, 1          1.0, -0.5  1, 1
#> 2 1.0, 0.5 1.0 0.5          0.0, 1.5    1, 1          1.0, -0.5  1, 1
#> 3 1.5, 0.5 1.5 0.5          0.0, 1.5    1, 1          1.0, -0.5  1, 1
#> 4 0.5, 1.0 0.5 1.0          0.0, 1.5    1, 1          1.0, -0.5  1, 1
#> 5 1, 1 1.0 1.0            0.0, 1.5    1, 1          1.0, -0.5  1, 1
#> 6 1.5, 1.0 1.5 1.0          0.0, 1.5    1, 1          1.0, -0.5  1, 1
#> 7 0.5, 1.5 0.5 1.5          0.0, 1.5    1, 1          1.0, -0.5  1, 1
#> 8 1.0, 1.5 1.0 1.5          0.0, 1.5    1, 1          1.0, -0.5  1, 1
#> 9 1.5, 1.5 1.5 1.5          0.0, 1.5    1, 1          1.0, -0.5  1, 1

```

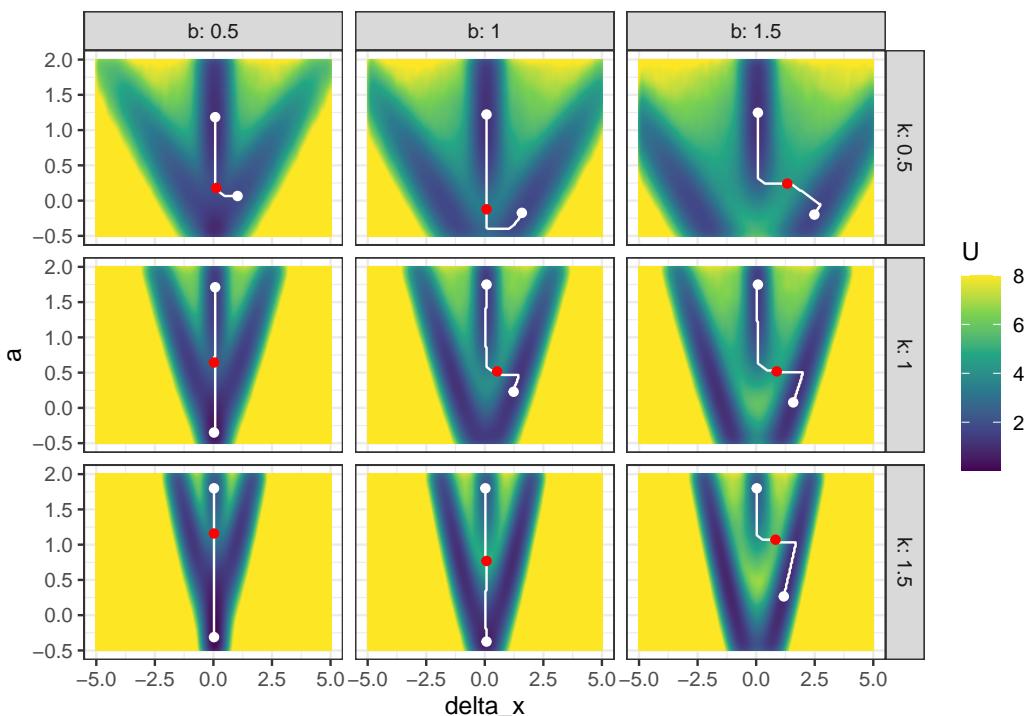


Figure 7: The landscape for the gene expression model of different b and k values. The local minima are marked as white dots, the saddle points are marked as red dots, and the MEPs are marked as white lines.

```
bg_gene <- make_barrier_grid_3d(batch_grid_gene, df = structure(list(start_location_value = list(c(0
1.5), c(0, 1.5), c(0, 1.5), c(0, 1.5), c(0, 1.5), c(0, 1.5), c(0, 1.5),
c(0, 1.5), c(0, 1.5)), start_r = list(c(0.2, 1), c(0.2, 1), c(0.2,
1), c(0.2, 0.5), c(0.2, 0.5), c(0.2, 0.3), c(0.2, 0.3),
c(0.2, 0.3)), end_location_value = list(c(2, 0), c(2, 0), c(2, 0),
c(1, 0), c(1, 0), c(1, 0), c(1, 0), c(1, 0)), end_r = list(c(1,
1), c(1, 1), c(1, 1), c(1, 1), c(1, 1), c(1, 1), c(1, 1),
c(1, 1))), row.names = c(NA, -9L), class = c("arg_grid", "data.frame")))
```

With the barrier grid template, we can calculate the barrier for each landscape plot.

```
b_batch_gene_3d <- calculate_barrier(l_batch_gene_3d, bg = bg_gene)
```

If the barrier grid was not needed, the following code can be used to calculate the barrier.

```
b_batch_gene_3d <- calculate_barrier(l_batch_gene_3d, start_location_value = c(0,
1.5), end_location_value = c(1, 0), start_r = 1, end_r = 1)
```

The resulting landscapes and the MEPs between states are shown in Figure 7.

```
plot(l_batch_gene_3d) + autolayer(b_batch_gene_3d)
```

From the landscapes, it is clear that increasing b , which represents higher strength of gene mutual prohibition, makes the two differentiated states further apart from each other. Increasing k , which represents faster degradation, makes the undifferentiated state disappear earlier, thus makes the differentiation earlier. When b is low enough and k is high enough, there is no differentiation anymore because the two differentiated states merge together and form a more stable state at $\Delta x = 0$. In this case, there is no actual differentiation in the system, but only a one-to-one conversion of cell types. Only when b is high enough and k is low enough is it possible for the cell to differentiate into two types.

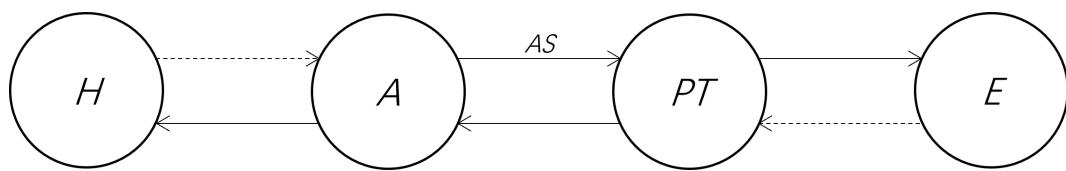


Figure 8: A graphical illustration of the relationships between several important psychological variables in the panic disorder model. Solid arrows represent positive relationships and dashed arrows represent negative relationships.

3.2 Example 2: panic disorder model

The second example we use is the panic disorder model proposed by Robinaugh et al. (2024). The model is implemented in the **PanicModel** package (<https://github.com/jmbh/PanicModel/>). It contains 12 variables and 33 parameters and also involves history dependency and non-differentiable formula (such as if-else conditions) to model the complex interplay of individual and environmental elements in different time scales. The most important variables of the model are the physical arousal (A) of a person (e.g., heart beat, muscle tension, sweating), the person's perceived threat (PT , how dangerous the person cognitively evaluates the environment), and the person's tendency to escape from the situation (E). The core theoretical idea of the model is that physical arousal and perceived threat of a person may strengthen each other in certain circumstances, leading to sudden increases in both variables, manifesting as panic attacks. The tendency that a person tends to use physical arousal as cognitive evidence of threat is represented by another variable, arousal schema S . A comprehensive introduction of the model is beyond the scope of the current article, and we would like to refer interested readers to Robinaugh et al. (2024). Here, to simplify the context, we assume that AS does not change over time, and no psychotherapy is being administered. We focus on the influence of AS on the system's stability represented by A and PT . A graphical illustration of several core variables of this model is shown in Figure 8 (adapted from Cui et al. (2023b)).

To construct the potential landscapes for this model, we first create a function that performs a simulation using the `simPanic()` function from **PanicModel**. This is required as we need to modify some default options for illustration.

```

library(PanicModel)

sim_fun_panic <- function(x0, par) {

  # Change several default parameters
  pars <- pars_default
  # Increase the noise strength to improve sampling efficiency
  pars$N$lambda_N <- 200
  # Make S constant through the simulation
  pars$TS$r_S_a <- 0
  pars$TS$r_S_e <- 0

  # Specify the initial values of A and PT according to the format
  # requirement by `multi_init_simulation()`, while the other
  # variables use the default initial values.
  initial <- initial_default
  initial$A <- x0[1]
  initial$PT <- x0[2]

  # Specify the value of S according to the format requirement by
  # `batch_simulation()`.

  initial$S <- par$S
}
  
```

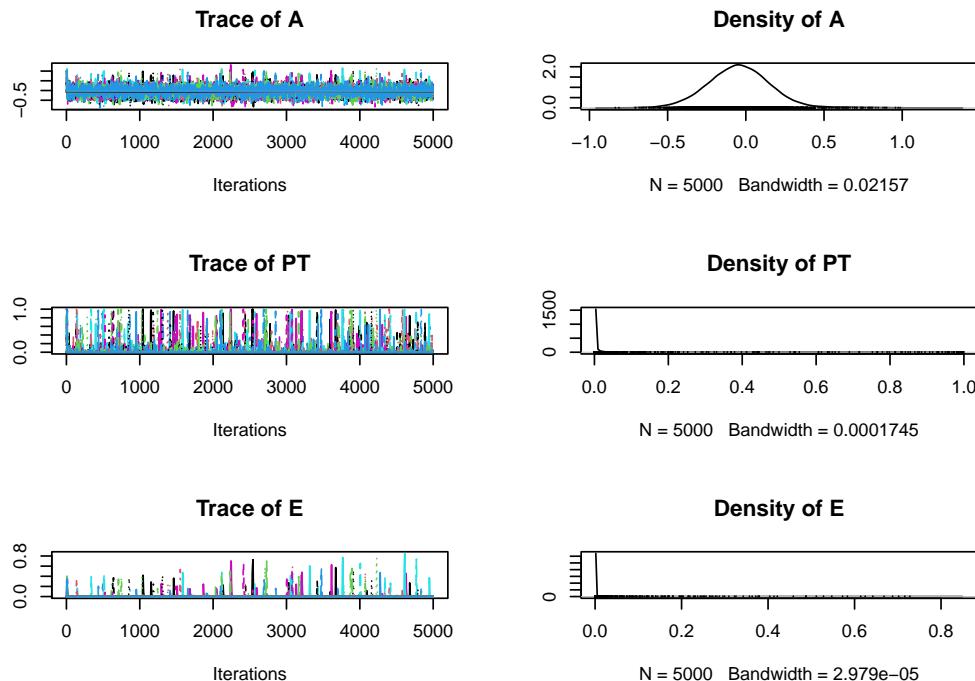


Figure 9: The convergence check result for the simulation of the panic disorder model.

```
# Extract the simulation output from the result by simPanic().
# Only keep the core variables.
return(as.matrix(simPanic(1:5000, initial = initial, parameters = pars)$outmat[,
  c("A", "PT", "E")]))
}
```

We then perform a single simulation from multiple starting points. To speed up the simulation, we use parallel computing.

```
future::plan("multisession")
set.seed(1614, kind = "L'Ecuyer-CMRG")
single_output_panic <- multi_init_simulation(sim_fun = sim_fun_panic, range_x0 = c(0,
  1, 0, 1), R = 4, par = list(S = 0.5))
```

The convergence check results of the simulation, shown in Figure 9, indicate that the time series of the first 100 data points are strongly influenced by the choice of initial value. Therefore, we remove the first 100 data points in the following analysis.

```
plot(single_output_panic)
```

We then create the 3D landscape for the panic disorder model, shown in Figure 10. The landscape shows that the system has two stable states, which are represented by the valleys in the landscape. The system can be trapped in these valleys, leading to different levels of physical arousal and perceived threat. The valley with higher potential value represents a state with higher physical arousal and perceived threat, which corresponds to a panic attack. In contrast, the valley with lower potential value represents a state with lower physical arousal and perceived threat, which corresponds to a healthy state.

We now investigate the effect of the parameter S on the potential landscape. This parameter represents the tendency that a person considers physical arousal as a sign of danger. Therefore, we expect that higher S will stabilize the panic state and destabilize the healthy state.

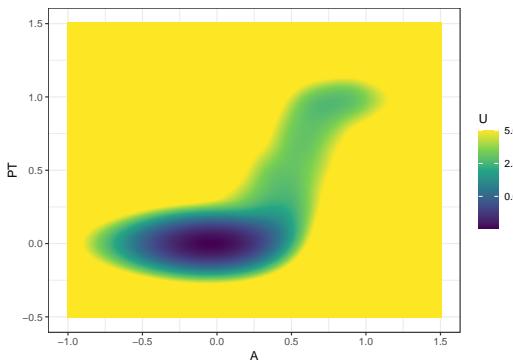


Figure 10: The 3D landscape (potential value as color) for the panic disorder model

We perform a batch simulation with varying S values to construct the potential landscapes for different S values. This, again, starts with the creation of a grid of parameter values.

```
batch_arg_grid_panic <- new_arg_set() |>
  add_arg_ele(arg_name = "par", ele_name = "S", start = 0, end = 1, by = 0.5) |>
  make_arg_grid()
```

We then perform the batch simulation using parallel computing.

```
future::plan("multisession")
set.seed(1614, kind = "L'Ecuyer-CMRG")
batch_output_panic <- batch_simulation(batch_arg_grid_panic, sim_fun = function(par) {
  multi_init_simulation(sim_fun_panic, range_x0 = c(0, 1, 0, 1), R = 4,
    par = par) |>
  window(start = 100)
})
```

The 3D landscapes for different S values are shown in Figure 11. The landscapes show that the system only has one stable state when S is low, but has two stable states when S is high. The stability of the panic state also increases when S is higher. This indicates that a higher S value corresponds to a higher risk of panic attacks.

```
l_batch_panic_3d <- make_3d_matrix(batch_output_panic, x = "A", y = "PT",
  cols = "S", h = 0.005, lims = c(-1, 1.5, -0.5, 1.5))
plot(l_batch_panic_3d)
```

4 Discussion

Potential landscapes can show the stability of states for a dynamical system in an intuitive and quantitative way. They are especially informative for multistable systems. In this article, we illustrated how to construct potential landscapes using `simlandr`. The potential landscapes generated by `simlandr` are based on the steady-state distribution of the system, which is in turn estimated using Monte Carlo simulation. Compared to analytic methods, Monte Carlo estimation is more flexible and thus more applicable for complex models. The flexibility comes together with a higher demand for time and storage, which is necessary to make the estimation precise enough. The `hash_big_matrix` class partly solved this problem by dumping the memory storage to hard disk space. Also, it is important that the simulation function itself is efficient enough. The functions `sim_SDE()` and `multi_init_simulation()` make use of the efficient simulations provided by `Sim.DiffProc` (Guidoum and Boukhetala, 2020) and the parallel computing with the `future` framework

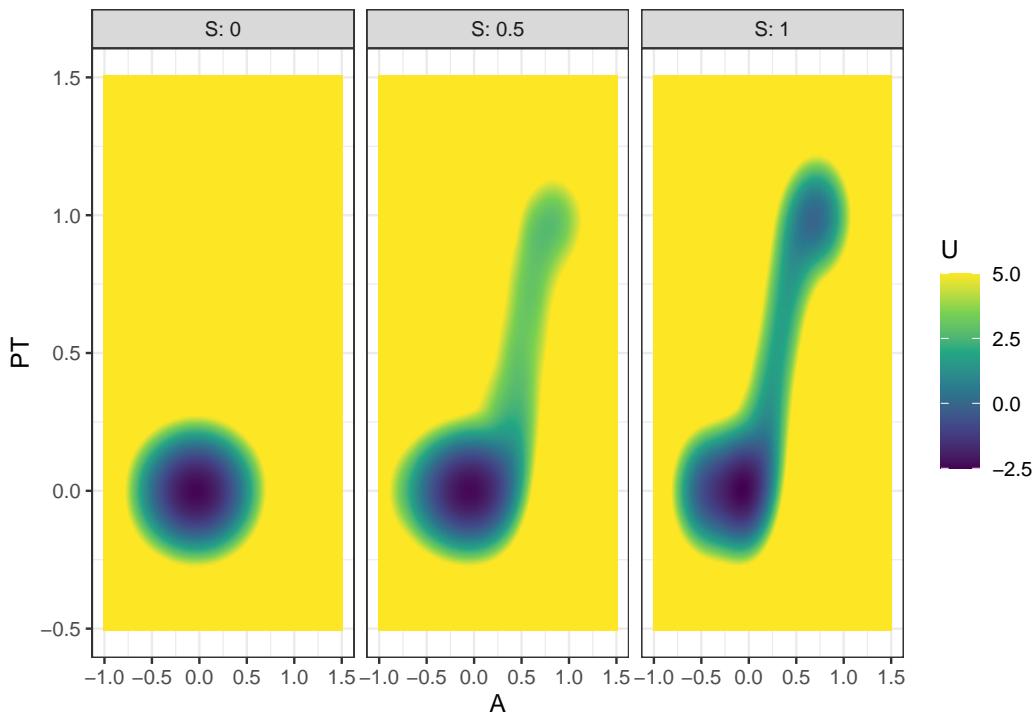


Figure 11: The landscape for the panic disorder model of different S values. Two landscapes are shown for different variable combinations, A and PT , or A and E .

(Bengtsson, 2021). For customized simulation functions, there are also multiple approaches that can be used to improve the performance, for which we refer interested readers to Wickham (2019). In Supplementary Materials A, we provide a benchmark of the typical time and memory usage of the procedures in `simlandr`. From there, we can see that time and memory usage are acceptable in most cases on a personal computer. When the transition between attractors is rare, the `multi_init_simulation()` function may help to speed up the convergence, and more advanced sampling methods like importance sampling or rare event sampling may be needed in more complex situations. The detailed ways to implement such methods are highly dependent on the specific model and are beyond the scope of this package. We direct interested readers to Rubino and Tuffin (2009) and Kloek and van Dijk (1978) for a comprehensive review of rare event simulation methods. Nevertheless, the landscape construction functions in `simlandr` allow users to provide weights for the simulation results, which can be used to adjust the sampling distribution.

In addition, the length of the simulation and the choice of noise strength may also have an important influence on the results. If the length is too short, the density estimation will be inaccurate, resulting in rugged landscapes. If the length is too long, the simulation part would require more computational resources, which is not always realistic. If the noise is too weak, the system may not be able to converge in a reasonable time, resulting in problems in convergence checks, overly noisy landscapes, or failure to show valleys that are theoretically present. If the noise is too strong, the simulation may be unstable and the boundaries between valleys may be blurry. In Supplementary Materials B, we showed the influence of simulation length and noise strength on the landscape output. With some theoretical expectation of the system's behavior, it is not difficult to spot that the simulation is too short, or the noise level might be unsuitable. In that case, some adjustments are required before the landscape can be well constructed.

All landscape construction and barrier calculation functions in `simlandr` contain both visual aids and numerical data that can be used for further processing. The html plots based on `plotly` are more suitable for interactive illustrations, while it is also possible to export

them to static plots using `plotly::orca()`. The `ggplot2` plots are readily usable for flat printing.

We also want to note some limitations of the potential landscape generated by `simlandr`. First, the generalized potential landscape is not a complete description of all dynamics in a system. It emphasizes the stability of different states by filtering out other dynamical information. Some behaviors are not possible in gradient systems (e.g., oscillations and loops), thus cannot be shown in a potential landscape (Zhou and Li, 2016). Second, since the steady-state distribution is estimated using a kernel smoothing method, which depends on stochastic simulations, the resulting potential function may not be highly accurate. Its accuracy is further affected by the choice of kernel bandwidth and noise strength. This issue is particularly pronounced at valley edges, where fewer samples are available for estimation. Similar limitations apply to MEP calculations, as they are derived from the generalized landscape rather than the original dynamics. Therefore, we do not recommend directly interpreting the potential function or barrier height results for applications requiring high precision. Instead, the potential landscape is best used as a semi-quantitative tool to gain insights into the system's overall behavior, guide further analysis, and compare system behavior under different parameter settings, provided the same simulation and kernel estimation conditions are used. The examples in this article illustrated some typical use cases we recommend.

5 Availability and future directions

This package is publicly available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/package=simlandr>, under GPL-3 license. The results in the current article were generated with `simlandr` 0.4.0 version. R script to replicate all the results in this article can be found in the supporting information.

The barrier height data calculated by `simlandr` can also be further analyzed and visualized. For example, sometimes it is helpful to look into how the barrier height changes with varying parameters (e.g., Cui et al. (2023b)). We encourage users to explore other ways of analyzing and visualizing the various results provided by `simlandr`.

The method we chose for `simlandr` is not the only possible one. The generalized landscape by Wang et al. (2008), which we implemented, is more flexible and emphasizes the possibility that the system is in a specific state, while other methods may have other strengths (e.g., the method by Rodríguez-Sánchez (2020) emphasizes the gradient part of the vector field, and the method by Moore et al. (2016) emphasizes the possibility of transition processes under small noise). We look forward to future theoretical and methodological developments in this direction.

6 Acknowledgments

TL was supported by the NSFC under Grant No. 11825102 and the Beijing Academy of Artificial Intelligence (BAAI). ALA and MO were supported by an NWO VIDI grant, Grant No. VI.Vidi.191.178.

References

- P. Ao. Potential in stochastic differential equations: Novel construction. *Journal of Physics A: Mathematical and General*, 37(3):L25–L30, 2004. ISSN 0305-4470. doi: 10.1088/0305-4470/37/3/L01. [p174]
- H. Bengtsson. A unifying framework for parallel and distributed processing in R using futures. *The R Journal*, 13(2):208–227, 2021. doi: 10.32614/RJ-2021-048. [p175, 187]

- J. E. Chacón and T. Duong. *Multivariate Kernel Smoothing and Its Applications*. Chapman and Hall/CRC, New York, May 2018. ISBN 978-0-429-48557-2. doi: 10.1201/9780429485572. [p177]
- J. Cui, A. Lichtwarck-Aschoff, and F. Hasselman. Comments on "Climbing Escher's Stairs: A way to approximate stability landscapes in multidimensional systems", Dec. 2023a. [p174]
- J. Cui, A. Lichtwarck-Aschoff, M. Olthof, T. Li, and F. Hasselman. From metaphor to computation: Constructing the potential landscape for multivariate psychological formal models. *Multivariate Behavioral Research*, 58(4):743–761, 2023b. doi: 10.1080/00273171.2022.2119927. [p175, 177, 178, 184, 188]
- D. Dahiya and M. Cameron. Ordered line integral methods for computing the quasi-potential. *Journal of Scientific Computing*, 75(3):1351–1384, 2018. ISSN 1573-7691. doi: 10.1007/s10915-017-0590-9. [p174]
- E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. doi: 10.1007/BF01386390. [p177]
- W. E and E. Vanden-Eijnden. Transition-path theory and path-finding algorithms for the study of rare events. *Annual Review of Physical Chemistry*, 61(1):391–420, 2010. ISSN 0066-426X. doi: 10.1146/annurev.physchem.040808.090412. [p177]
- D. Eddelbuettel, A. Lucas, J. Tuszyński, H. Bengtsson, S. Urbanek, M. Frasca, B. Lewis, M. Stokely, H. Muehleisen, D. Murdoch, J. Hester, W. Wu, Q. Kou, T. Onkelinx, M. Lang, V. Simko, K. Hornik, R. Neal, K. Bell, M. de Queljoe, I. Suruceanu, B. Denney, D. Schumacher, and W. Chang. *digest: Create Compact Hash Digests of R Objects*, 2021. [p176]
- M. Freidlin and A. Wentzell. *Random Perturbations of Dynamical Systems*. Springer, Berlin, 3rd edition, 2012. doi: 10.1007/978-3-642-25847-3. [p174]
- A. C. Guidoum and K. Boukhetala. Performing parallel Monte Carlo and moment equations methods for Itô and Stratonovich stochastic differential systems: R package Sim.DiffProc. *Journal of Statistical Software*, 96:1–82, Nov. 2020. ISSN 1548-7660. doi: 10.18637/jss.v096.i02. [p175, 186]
- A. M. Hayes and L. A. Andrews. A complex systems approach to the study of change in psychotherapy. *BMC Medicine*, 18(1):197, July 2020. ISSN 1741-7015. doi: 10.1186/s12916-020-01662-2. [p173]
- M. Heymann and E. Vanden-Eijnden. Pathways of maximum likelihood for rare events in nonequilibrium systems: Application to nucleation in the presence of shear. *Physical Review Letters*, 100(14):140601, 2008. doi: 10.1103/PhysRevLett.100.140601. [p177]
- M. J. Kane, J. Emerson, and S. Weston. Scalable strategies for computing with massive data. *Journal of Statistical Software*, 55(14):1–19, 2013. doi: 10.18637/jss.v055.i14. [p176]
- T. Kloek and H. K. van Dijk. Bayesian estimates of equation system parameters: An application of integration by Monte Carlo. *Econometrica*, 46(1):1–19, 1978. ISSN 0012-9682. doi: 10.2307/1913641. [p187]
- K. A. Lamothe, K. M. Somers, and D. A. Jackson. Linking the ball-and-cup analogy and ordination trajectories to describe ecosystem stability, resistance, and resilience. *Ecosphere*, 10(3):e02629, 2019. ISSN 2150-8925. doi: 10.1002/ecs2.2629. [p173]
- C. M. Moore, C. R. Stieha, B. C. Nolting, M. K. Cameron, and K. C. Abbott. QPot: An R package for stochastic differential equation quasi-potential analysis. *The R Journal*, 8(2):19–38, 2016. doi: 10.32614/RJ-2016-031. [p174, 188]
- M. Olthof, F. Hasselman, F. Oude Maatman, A. M. T. Bosman, and A. Lichtwarck-Aschoff. Complexity theory of psychopathology. *Journal of Psychopathology and Clinical Science*, 132(3):314–323, 2023. doi: 10.1037/abn0000740. [p173]

- M. Plummer, N. Best, K. Cowles, and K. Vines. Coda: Convergence diagnosis and output analysis for mcmc. *R News*, 6(1):7–11, 2006. URL <https://journal.r-project.org/articles/RN-2006-002/RN-2006-002.pdf>. [p176]
- D. J. Robinaugh, J. M. B. Haslbeck, L. J. Waldorp, J. J. Kossakowski, E. I. Fried, A. J. Millner, R. J. McNally, O. Ryan, J. de Ron, H. L. J. van der Maas, E. H. van Nes, M. Scheffer, K. S. Kendler, and D. Borsboom. Advancing the network theory of mental disorders: A computational model of panic disorder. *Psychological Review*, 131(6):1482–1508, 2024. doi: 10.1037/rev0000515. [p178, 184]
- P. Rodríguez-Sánchez. Pabrod / rolldown: Post-publication update, 2020. [p174, 188]
- P. Rodríguez-Sánchez, E. H. van Nes, and M. Scheffer. Climbing Escher’s stairs: A way to approximate stability landscapes in multidimensional systems. *PLOS Computational Biology*, 16(4):e1007788, 2020. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1007788. [p174]
- G. Rubino and B. Tuffin. *Rare Event Simulation Using Monte Carlo Methods*. John Wiley & Sons, Incorporated, Newark, UNITED KINGDOM, 2009. ISBN 978-0-470-74541-0. doi: 10.1002/9780470745403. [p187]
- C. Sievert. *Interactive Web-Based Data Visualization with R, Plotly, and Shiny*. Chapman and Hall/CRC, 2020. ISBN 978-1-138-33145-7. URL <https://plotly-r.com>. [p177]
- P. S. Stumpf, F. Arai, and B. D. MacArthur. Modeling stem cell fates using non-Markov processes. *Cell Stem Cell*, 28(2):187–190, 02 2021. doi: 10.1016/j.stem.2021.01.009. [p173]
- C. H. Waddington. *Principles of Development and Differentiation*. Macmillan, New York, 1966. [p173]
- G. Wan, S. J. Avis, Z. Wang, X. Wang, H. Kusumaatmaja, and T. Zhang. Finding transition state and minimum energy path of bistable elastic continua through energy landscape explorations. *Journal of the Mechanics and Physics of Solids*, 183:105503, Feb. 2024. ISSN 0022-5096. doi: 10.1016/j.jmps.2023.105503. [p175]
- J. Wang, L. Xu, and E. Wang. Potential landscape and flux framework of nonequilibrium networks: Robustness, dissipation, and coherence of biochemical oscillations. *Proceedings of the National Academy of Sciences*, 105(34):12271–12276, 2008. ISSN 0027-8424. doi: 10.1073/pnas.0800579105. [p174, 177, 188]
- J. Wang, K. Zhang, L. Xu, and E. Wang. Quantifying the Waddington landscape and biological paths for development and differentiation. *Proceedings of the National Academy of Sciences*, 108(20):8257–8262, 2011. ISSN 0027-8424. doi: 10.1073/pnas.1017017108. [p173, 178, 179]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p177]
- H. Wickham. Improving performance. In *Advanced R*, pages 531–546. Chapman and Hall/CRC, Boca Raton, 2nd edition, 2019. URL <https://adv-r.hadley.nz/>. [p187]
- P. Zhou and T. Li. Construction of the landscape for multi-stable systems: Potential landscape, quasi-potential, a-type integral and beyond. *The Journal of Chemical Physics*, 144(9): 094109, 2016. ISSN 0021-9606. doi: 10.1063/1.4943096. [p174, 175, 188]

Jingmeng Cui
University of GroningenRadboud University
Faculty of Behavioural and Social Sciences, Groningen, the Netherlands
Behavioural Science Institute, Nijmegen, the Netherlands
ORCID: 0000-0003-3421-8457
jingmeng.cui@rug.nl

Merlijn Olthof

University of GroningenRadboud University

Faculty of Behavioural and Social Sciences, Groningen, the Netherlands

Behavioural Science Institute, Nijmegen, the Netherlands

ORCID: 0000-0002-5975-6588

merlijn.olthof@ru.nl

Anna Lichtwarck-Aschoff

University of GroningenRadboud University

Faculty of Behavioural and Social Sciences, Groningen, the Netherlands

Behavioural Science Institute, Nijmegen, the Netherlands

ORCID: 0000-0002-4365-1538

a.lichtwarck-aschoff@rug.nl

Tiejun Li

Peking University

(corresponding author) LMAM and School of Mathematical Sciences, Beijing, China

ORCID: 0000-0002-2086-1620

tieli@pku.edu.cn

Fred Hasselman

Radboud University

(corresponding author) Behavioural Science Institute, Nijmegen, the Netherlands

ORCID: 0000-0003-1384-8361

fred.hasselman@ru.nl

rvif: a Decision Rule to Detect Troubling Statistical Multicollinearity Based on Redefined VIF

by Román Salmerón-Gómez and Catalina B. García-García

Abstract Multicollinearity is relevant in many different fields where linear regression models are applied since its presence may affect the analysis of ordinary least squares estimators not only numerically but also from a statistical point of view, which is the focus of this paper. Thus, it is known that collinearity can lead to incoherence in the statistical significance of the coefficients of the independent variables and in the global significance of the model. In this paper, the thresholds of the Redefined Variance Inflation Factor (RVIF) are reinterpreted and presented as a statistical test with a region of non-rejection (which depends on a significance level) to diagnose the existence of a degree of worrying multicollinearity that affects the linear regression model from a statistical point of view. The proposed methodology is implemented in the `rvif` package of R and its application is illustrated with different real data examples previously applied in the scientific literature.

1 Introduction

It is well known that linear relationships between the independent variables of a multiple linear regression model (multicollinearity) can affect the analysis of the model estimated by Ordinary Least Squares (OLS), either by causing unstable estimates of the coefficients of these variables or by rejecting individual significance tests of these coefficients (see, for example, [Farrar and Glauber \(1967\)](#), [Gunst and Mason \(1977\)](#), [Gujarati \(2003\)](#), [Silvey \(1969\)](#), [Willan and Watts \(1978\)](#) or [Wooldridge \(2020\)](#)). However, the measures traditionally applied to detect multicollinearity may conclude that multicollinearity exists even if it does not lead to the negative effects mentioned above (see Subsection [Effect of sample size..](#) for more details), when, in fact, the best solution in this case may be not to treat the multicollinearity (see [O'Brien \(2007\)](#)).

Focusing on the possible effect of multicollinearity on the individual significance tests of the coefficients of the independent variables (tendency not to reject the null hypothesis), this paper proposes an alternative procedure that focuses on checking whether the detected multicollinearity affects the statistical analysis of the model. For this disruptive approach, a methodology is necessary that indicates whether multicollinearity affects the statistical analysis of the model. The introduction of such methodology is the main objective of this paper. The paper also shows the use of the `rvif` package of R ([R Core Team \(2025\)](#)) in which this procedure is implemented.

To this end, we start from the Variance Inflation Factor (VIF). The VIF is obtained from the coefficient of determination of the auxiliary regression of each independent variable of linear regression model as a function of the other independent variables. Thus, there is a VIF for each independent variable except for the intercept, for which it is not possible to calculate a coefficient of determination for the corresponding auxiliary regression. Consequently, the VIF is able to diagnose the degree of essential approximate multicollinearity (strong linear relationship between the independent variables except the intercept) existing in the model but is not able to detect the non-essential one (strong relationship between the intercept and at least one of the independent variables). For more information on multicollinearity of essential and non-essential type, see [Marquardt and Snee \(1975\)](#) and [Salmerón-Gómez et al. \(2019\)](#).

However, the fact that the VIF detects a worrying level of multicollinearity does not always translate into a negative impact on the statistical analysis. This lack of specificity is due to the fact that other factors, such as sample size and the variance of the random

disturbance, can lead to high values of the VIF but not increase the variance of the OLS estimators (see O'Brien (2007)). The explanation for this phenomenon hinges on the fact that, in the orthogonal variable reference model, which is traditionally considered as the reference, the linear relationships are assumed to be eliminated, while other factors, such as the variance of the random disturbance, maintain the same values.

Then, to avoid these inconsistencies, Salmerón et al. (2025) propose a QR decomposition in the matrix of independent variables of the model in order to obtain an orthonormal matrix. By redefining the reference point, the variance inflation factor is also redefined, resulting in a new detection measure that analyzes the change in the VIF and the rest of relevant factors of the model, thereby overcoming the problems associated with the traditional VIF, as described by O'Brien (2007) among others. The intercept is also included in the detection (contrary to what happens with the traditional VIF), it is therefore able to detect both essential and non-essential multicollinearity. This new measure presented by Salmerón et al. (2025) is called Redefined Variance Inflation Factor (RVIF).

In this paper, the RVIF is associated with a statistical test for detecting troubling multicollinearity, this test is given by a region of non-rejection that depends on a significance level. Note that most of the measures used to diagnose multicollinearity are merely indicators with rules of thumb rather than statistical tests per se. To the best of our knowledge, the only existing statistical test for diagnosing multicollinearity was presented by Farrar and Glauber (1967) and has received strong criticism (see, for example, Haitovsky (1969), Kumar (1975), Wicher (1975) and O'Hagan and McCabe (1975)). Thus, for example, Haitovsky (1969) indicates that the Farrar and Glauber statistic indicates that the variables are not orthogonal to each other; it tells us nothing more. In this sense, O'Hagan and McCabe (1975) indicates that such a test simply indicates whether the null hypothesis of orthogonality is rejected by giving no information on the value of the matrix of correlations determinant above which the multicollinearity problem becomes intolerable. Therefore, the non-rejection region presented in this paper should be a relevant contribution to the field of econometrics insofar as it would fill an existing gap in the scientific literature.

The paper is structured as follows: Sections Preliminares and A first attempt of... provide preliminary information to introduce the methodology used to establish the non-rejection region described in Section A non-rejection region.... Section rvif package presents the package `rvif` of R (R Core Team (2025)) and shows its main commands by replicating the results given in Salmerón et al. (2025) and in the previous sections of this paper. Finally, Section Conclusions summarizes the main contributions of this paper.

2 Preliminaries

This section identifies some inconsistencies in the definition of the VIF and how these are reflected in the individual significance tests of the linear regression model. It also shows how these inconsistencies are overcome in the proposal presented by Salmerón et al. (2025) and how this proposal can lead to a decision rule to determine whether the degree of multicollinearity is troubling, i.e., whether it affects the statistical analysis (individual significance tests) of the model.

2.1 The original model

The multiple linear regression model with n observations and k independent variables can be expressed as:

$$\mathbf{y}_{n \times 1} = \mathbf{X}_{n \times k} \cdot \boldsymbol{\beta}_{k \times 1} + \mathbf{u}_{n \times 1}, \quad (1)$$

where the first column of $\mathbf{X} = [\mathbf{1} \ \mathbf{X}_2 \dots \mathbf{X}_i \dots \mathbf{X}_k]$ is composed of ones representing the intercept and \mathbf{u} represents the random disturbance assumed to be centered and spherical. That is, $E[\mathbf{u}_{n \times 1}] = \mathbf{0}_{n \times 1}$ and $var(\mathbf{u}_{n \times 1}) = \sigma^2 \cdot \mathbf{I}_{n \times n}$, where $\mathbf{0}$ is a vector of zeros, σ^2 is the variance of the random disturbance and \mathbf{I} is the identity matrix.

Given the original model (1), the VIF is defined as the ratio between the variance of the estimator in this model, $\text{var}(\hat{\beta}_i)$, and the variance of the estimator of a hypothetical reference model, that is, a hypothetical model in which orthogonality among the independent variables is assumed, $\text{var}(\hat{\beta}_{i,o})$. This is to say:

$$\text{var}(\hat{\beta}_i) = \frac{\sigma^2}{n \cdot \text{var}(\mathbf{X}_i)} \cdot \frac{1}{1 - R_i^2} = \text{var}(\hat{\beta}_{i,o}) \cdot \text{VIF}(i), \quad i = 2, \dots, k, \quad (2)$$

$$\frac{\text{var}(\hat{\beta}_i)}{\text{var}(\hat{\beta}_{i,o})} = \text{VIF}(i), \quad i = 2, \dots, k, \quad (3)$$

where \mathbf{X}_i is the independent variable i of the model (1) and R_i^2 the coefficient of determination of the following auxiliary regression:

$$\mathbf{X}_i = \mathbf{X}_{-i} \cdot \boldsymbol{\alpha} + \mathbf{v},$$

where \mathbf{X}_{-i} is the result of eliminating \mathbf{X}_i from the matrix \mathbf{X} .

As observed in the expression (2), a high VIF leads to a high variance. Then, since the experimental value for the individual significance test is given by:

$$t_i = \left| \frac{\hat{\beta}_i}{\sqrt{\frac{\hat{\sigma}^2}{n \cdot \text{var}(\mathbf{X}_i)} \cdot \text{VIF}(i)}} \right|, \quad i = 2, \dots, k, \quad (4)$$

a high VIF will lead to a low experimental statistic (t_i), provoking the tendency not to reject the null hypothesis, i.e. the experimental statistic will be lower than the theoretical statistic (given by $t_{n-k}(1 - \alpha/2)$, where α is the significance level).

However, this statement is full of simplifications. By following O'Brien (2007), and as can be easily observed in the expression (4), other factors, such as the estimation of the random disturbance and the size of the sample, can counterbalance the high value of the VIF to yield a low value for the experimental statistic. That is to say, it is possible to obtain VIF values greater than 10 (the threshold traditionally established as troubling, see Marquardt (1970) for example) that do not necessarily imply high estimated variance on account of a large sample size or a low value for the estimated variance of the random disturbance. This explains, as noted in the introduction, why not all models with a high value for the VIF present effects on the statistical analysis of the model.

Example 1. Thus, for example, García et al. (2019) considered an extension of the interest rate model presented by Wooldridge (2020), where $k = 3$, in which all the independent variables have associated coefficients significantly different from zero, presenting a VIF equal to 71.516, much higher than the threshold normally established as worrying. In other words, in this case, a high VIF does not mean that the individual significance tests are affected. This situation is probably due to the fact that in this case 131 observations are available, i.e. the expression (4) can be expressed as:

$$t_i = \left| \frac{\hat{\beta}_i}{\sqrt{\frac{\hat{\sigma}^2}{131 \cdot \text{var}(\mathbf{X}_i)} \cdot 71.516}} \right| = \left| \frac{\hat{\beta}_i}{\sqrt{0.546 \cdot \frac{\hat{\sigma}^2}{\text{var}(\mathbf{X}_i)}}} \right|, \quad i = 2, 3.$$

Note that in this case a high value of n compensates for the high value of VIF. In addition, the value of n will also cause $\hat{\sigma}^2$ to decrease, since $\hat{\sigma}^2 = \frac{\mathbf{e}' \mathbf{e}}{n-k}$, where \mathbf{e} are the residuals of the original model (1).

The Subsection [Effect of sample size..](#) provides an example that illustrates in more detail the effect of sample size on the statistical analysis of the model. ◇

On the other hand, considering the hypothetical orthogonal model, the value of the experimental statistic of the individual significance test, whose null hypothesis is $\beta_i = 0$ in face of the alternative hypothesis $\beta_i \neq 0$ with $i = 2, \dots, k$, is given by:

$$t_i^o = \left| \frac{\hat{\beta}_i}{\sqrt{\frac{\hat{\sigma}^2}{n \cdot \text{var}(\mathbf{X}_i)}}} \right|, \quad i = 2, \dots, k, \quad (5)$$

where the estimated variance of the estimator has been diminished due to the VIF always being greater than or equal to 1, and consequently, $t_i^o \geq t_i$. However, it has been assumed that the same estimates for the independent variable coefficients and random disturbance variance are obtained in the orthogonal and original models, which does not seem to be a plausible supposition (see [Salmerón et al. \(2025\)](#) Section 2.1 for more details).

2.2 An orthonormal reference model

In [Salmerón et al. \(2025\)](#) the following QR decomposition of the matrix $\mathbf{X}_{n \times k}$ of the model (1) is proposed: $\mathbf{X} = \mathbf{X}_o \cdot \mathbf{P}$, where \mathbf{X}_o is an orthonormal matrix of the same dimensions as \mathbf{X} and \mathbf{P} is a higher-order triangular matrix of dimensions $k \times k$. Then, the following hypothetical orthonormal reference model:

$$\mathbf{y} = \mathbf{X}_o \cdot \boldsymbol{\beta}_o + \mathbf{w}, \quad (6)$$

verifies that:

$$\hat{\boldsymbol{\beta}} = \mathbf{P}^{-1} \cdot \hat{\boldsymbol{\beta}}_o, \quad \mathbf{e} = \mathbf{e}_o, \quad \text{var}(\hat{\boldsymbol{\beta}}_o) = \sigma^2 \cdot \mathbf{I},$$

where \mathbf{e}_o are the residuals of the orthonormal reference model (6). Note that since $\mathbf{e} = \mathbf{e}_o$, the estimate of σ^2 is the same in the original model (1) and in the orthonormal reference model (6). Moreover, since the dependent variable is the same in both models, the coefficient of determination and the experimental value of the global significance test are the same in both cases.

From these values, taking into account the expressions (2) and (3), it is evident that the ratio between the variance of the estimator in the original model (1) and the variance of the estimator of the orthonormal reference model (6) is:

$$\frac{\text{var}(\hat{\beta}_i)}{\text{var}(\hat{\beta}_{i,o})} = \frac{\text{VIF}(i)}{n \cdot \text{var}(\mathbf{X}_i)}, \quad i = 2, \dots, k.$$

Consequently, [Salmerón et al. \(2025\)](#) defined the redefined VIF (RVIF) for $i = 1, \dots, k$ as:

$$\text{RVIF}(i) = \frac{\text{VIF}(i)}{n \cdot \text{var}(\mathbf{X}_i)} = \frac{\mathbf{X}_i^t \mathbf{X}_i}{\mathbf{X}_i^t \mathbf{X}_i - \mathbf{X}_i^t \mathbf{X}_{-i} \cdot (\mathbf{X}_{-i}^t \mathbf{X}_{-i})^{-1} \cdot \mathbf{X}_{-i}^t \mathbf{X}_i}, \quad (7)$$

which shows, among other questions, that it is defined for $i = 1, 2, \dots, k$. That is, in contrast to the VIF, the RVIF can be calculated for the intercept of the linear regression model.

Other considerations to be taken into account are the following:

- If the data are expressed in unit length, same transformation used to calculate the Condition Number (CN), then:

$$\text{RVIF}(i) = \frac{1}{1 - \mathbf{X}_i^t \mathbf{X}_{-i} \cdot (\mathbf{X}_{-i}^t \mathbf{X}_{-i})^{-1} \cdot \mathbf{X}_{-i}^t \mathbf{X}_i}, \quad i = 1, \dots, k.$$

- In this case (data expressed in unit length), when \mathbf{X}_i is orthogonal to \mathbf{X}_{-i} , it is verified that $\mathbf{X}_i^t \mathbf{X}_{-i} = \mathbf{0}$ and, consequently $RVIF(i) = 1$ for $i = 1, \dots, k$. That is, the RVIF is always greater than or equal to 1 and its minimum value is indicative of the absence of multicollinearity.
- Denoted by $a_i = \mathbf{X}_i^t \mathbf{X}_i \cdot (\mathbf{X}_{-i}^t \mathbf{X}_{-i})^{-1} \cdot \mathbf{X}_{-i}^t \mathbf{X}_i$, it is verified that $RVIF(i) = \frac{1}{1-a_i}$ where a_i can be interpreted as the percentage of approximate multicollinearity due to variable \mathbf{X}_i . Note the similarity of this expression to that of the VIF: $VIF(i) = \frac{1}{1-R_i^2}$ (see equation (2)).
- Finally, from a simulation for $k = 3$, Salmerón et al. (2025) show that if $a_i > 0.826$, then the degree of multicollinearity is worrying. In any case this value should be refined by considering higher values of k .

On the other hand, given the orthonormal reference model (6), the value for the experimental statistic of the individual significance test with the null hypothesis $\beta_{i,o} = 0$ (given the alternative hypothesis $\beta_{i,o} \neq 0$, for $i = 1, \dots, k$) is:

$$t_i^o = \left| \frac{\hat{\beta}_{i,o}}{\hat{\sigma}} \right| = \left| \frac{\mathbf{p}_i \cdot \hat{\beta}}{\hat{\sigma}} \right|, \quad (8)$$

where \mathbf{p}_i is the i row of the matrix \mathbf{P} .

By comparing this expression with the one given in (5), it is observed that, as expected, not only the denominator but also the numerator has changed. Thus, in addition to the VIF, the rest of the elements in expression (4) have also changed. Consequently, if the null hypothesis is rejected in the original model, it is not assured that the same will occur in the orthonormal reference model. For this reason, it is possible to consider that the orthonormal model proposed as the reference model in Salmerón et al. (2025) is more plausible than the one traditionally applied.

2.3 Possible scenarios in the individual significance tests

To determine whether the tendency not to reject the null hypothesis in the individual significance test is caused by a troubling approximate multicollinearity that inflates the variance of the estimator, or whether it is caused by variables not being statistically significantly related, the following situations are distinguished with a significance level α :

- If the null hypothesis is initially rejected in the original model (1), $t_i > t_{n-k}(1 - \alpha/2)$, the following results can be obtained for the orthonormal model:
 - the null hypothesis is rejected, $t_i^o > t_{n-k}(1 - \alpha/2)$; then, the results are consistent.
 - the null hypothesis is not rejected, $t_i^o < t_{n-k}(1 - \alpha/2)$; this could be an inconsistency.
- If the null hypothesis is not initially rejected in the original model (1), $t_i < t_{n-k}(1 - \alpha/2)$, the following results may occur for the orthonormal model:
 - the null hypothesis is rejected, $t_i^o > t_{n-k}(1 - \alpha/2)$; then, it is possible to conclude that the degree of multicollinearity affects the statistical analysis of the model, provoking not rejecting the null hypothesis in the original model.
 - the null hypothesis is also not rejected, $t_i^o < t_{n-k}(1 - \alpha/2)$; then, the results are consistent.

In conclusion, when option b.1 is given, the null hypothesis of the individual significance test is not rejected when the linear relationships are considered (original model) but is rejected when the linear relationships are not considered (orthonormal model). Consequently, it is possible to conclude that the linear relationships affect the statistical analysis

Table 1: Data set presented previously by Wissell

t	D	C	I	CP
1996	3.805	4.770	4.879	808.23
1997	3.946	4.778	5.051	798.03
1998	4.058	4.935	5.362	806.12
1999	4.191	5.100	5.559	865.65
2000	4.359	5.291	5.843	997.30
2001	4.545	5.434	6.152	1140.70
2002	4.815	5.619	6.521	1253.40
2003	5.129	5.832	6.915	1324.80
2004	5.615	6.126	7.423	1420.50
2005	6.225	6.439	7.802	1532.10
2006	6.786	6.739	8.430	1717.50
2007	7.494	6.910	8.724	1867.20
2008	8.399	7.099	8.882	1974.10
2009	9.395	7.295	9.164	2078.00
2010	10.680	7.561	9.727	2191.30
2011	12.071	7.804	10.301	2284.90
2012	13.448	8.044	10.983	2387.50

of the model. The possible inconsistency discussed in option a.2 is analyzed in detail in Appendix [Inconsistency](#), concluding that it will rarely occur in cases where a high degree of multicollinearity is assumed. The other two scenarios provide consistent situations.

3 A first attempt to obtain a non-rejection region associated with a statistical test to detect multicollinearity

3.1 From the traditional orthogonal model

Considering the expressions (4) and (5), it is verified that $t_i^o = t_i \cdot \sqrt{VIF(i)}$. Consequently, in the orthogonal case, with a significance level α , the null hypothesis $\beta_{i,o} = 0$ is rejected if $t_i^o > t_{n-k}(1 - \alpha/2)$ for $i = 2, \dots, k$. That is, if:

$$VIF(i) > \left(\frac{t_{n-k}(1 - \alpha/2)}{t_i} \right)^2 = c_1(i), \quad i = 2, \dots, k. \quad (9)$$

Thus, if the VIF associated with the variable i is greater than the upper bound $c_1(i)$, then it can be concluded that the estimator of the coefficient of that variable is significantly different from zero in the hypothetical case where the variables are orthogonal. In addition, if the null hypothesis is not rejected in the initial model, the reason for the failure to reject could be due to the degree of multicollinearity that affects the statistical analysis of the model.

Finally, note that since the interesting cases are those where the null hypothesis is not initially rejected, $t_i < t_{n-k}(1 - \alpha/2)$, the upper bound $c_1(i)$ will always be greater than one.

Example 2. Table 1 shows a dataset (previously presented by [Wissel \(2009\)](#)) with the following variables: outstanding mortgage debt (**D**, trillions of dollars), personal consumption (**C**, trillions of dollars), personal income (**I**, trillions of dollars) and outstanding consumer credit (**CP**, trillions of dollars) for the years 1996 to 2012.

Table 2: OLS estimation for the Wissel model

	Estimator	Standard Error	Experimental t	p-value
Intercept	5.469	13.017	0.420	0.681
Personal consumption	-4.252	5.135	-0.828	0.422
Personal income	3.120	2.036	1.533	0.149
Outstanding consumer credit	0.003	0.006	0.500	0.626
(Obs, Sigma Est., Coef. Det., F exp.)	17.000	0.870	0.923	52.305

Table 3: OLS estimation for part of the Wissel model

	Estimator	Standard Error	Experimental t	p-value
Intercept	-9.594	1.351	-7.102	0.000
Personal consumption	2.629	0.214	12.285	0.000
(Obs, Sigma Est., Coef. Det., F exp.)	17.000	0.890	0.910	150.925

Table 2 shows the OLS estimation of the model explaining the outstanding mortgage debt as a function of the rest of the variables. That is:

$$\mathbf{D} = \beta_1 + \beta_2 \cdot \mathbf{C} + \beta_3 \cdot \mathbf{I} + \beta_4 \cdot \mathbf{CP} + \mathbf{u}.$$

Note that the estimates for the coefficients of personal consumption, personal income and outstanding consumer credit are not significantly different from zero (a significance level of 5% is considered throughout the paper), while the model is considered to be globally valid (experimental value, F exp., higher than theoretical value).

In addition, the estimated coefficient for the variable personal consumption, which is not significantly different from zero, has the opposite sign to the simple correlation coefficient between this variable and outstanding mortgage debt, 0.953. Thus, in the simple linear regression between both variables (see Table 3), the estimated coefficient of the variable personal consumption is positive and significantly different from zero. However, adding a second variable (see Tables 4 and 5) none of the coefficients are individually significantly different from zero although both models are globally significant. This is traditionally understood as a symptom of statistically troubling multicollinearity.

By using expression (9) in order to confirm this problem, it is verified that $c_1(2) = 6.807$, $c_1(3) = 1.985$ and $c_1(4) = 18.743$, taking into account that $t_{13}(0.975) = 2.160$. Since the VIFs are equal to 589.754, 281.886 and 189.487, respectively, it is concluded that the individual significance tests for the three cases are affected by the degree of multicollinearity existing in the model. \diamond

3.2 From the alternative orthonormal model (6)

In the Subsection [An orthonormal reference model](#) the individual significance test from the expression (8) is redefined. Thus, the null hypothesis $\beta_{i,o} = 0$ will be rejected, with a

Table 4: OLS estimation for part of the Wissel model

	Estimator	Standard Error	Experimental t	p-value
Intercept	-0.117	6.476	-0.018	0.986
Personal consumption	-2.343	3.335	-0.703	0.494
Personal income	2.856	1.912	1.494	0.158
(Obs, Sigma Est., Coef. Det., F exp.)	17.000	0.823	0.922	82.770

Table 5: OLS estimation for part of the Wissel model

	Estimator	Standard Error	Experimental t	p-value
Intercept	-8.640	9.638	-0.896	0.385
Personal consumption	2.335	2.943	0.793	0.441
Outstanding consumer credit	0.001	0.006	0.100	0.922
(Obs, Sigma Est., Coef. Det., F exp.)	17.000	0.953	0.910	70.487

Table 6: OLS estimation for the orthonormal Wissel model

	Estimator	Standard Error	Experimental t	p-value
Intercept	-27.882	0.932	-29.901	0.000
Personal consumption	11.592	0.932	12.432	0.000
Personal income	-1.355	0.932	-1.453	0.170
Outstanding consumer credit	0.466	0.932	0.500	0.626
(Obs, Sigma Est., Coef. Det., F exp.)	17.000	0.870	0.923	52.305

significance level α , if the following condition is verified:

$$t_i^o > t_{n-k}(1 - \alpha/2), \quad i = 2, \dots, k.$$

Taking into account the expressions (4) and (8), this is equivalent to:

$$VIF(i) > \left(\frac{t_{n-k}(1 - \alpha/2)}{\hat{\beta}_{i,o}} \right)^2 \cdot \widehat{\text{var}}(\hat{\beta}_i) \cdot n \cdot \text{var}(\mathbf{X}_i) = c_2(i). \quad (10)$$

Thus, if the $VIF(i)$ is greater than $c_2(i)$, the null hypothesis is rejected in the respective individual significance tests in the orthonormal model (with $i = 2, \dots, k$). Then, if the null hypothesis is not rejected in the original model and it is verified that $VIF(i) > c_2(i)$, it can be concluded that the multicollinearity existing in the model affects its statistical analysis. In summary, a lower bound for the VIF is established to indicate when the approximate multicollinearity is troubling in a way that can be reinterpreted and presented as a region of non-rejection of a statistical test.

Example 3. Continuing with the dataset presented by [Wissel \(2009\)](#), Table 6 shows the results of the OLS estimation of the orthonormal model obtained from the original model.

When these results are compared with those in Table 2, the following conclusions can be obtained:

- Except for the outstanding consumer credit variable, whose standard deviation has increased, the standard deviation has decreased in all cases.
- The absolute values of the experimental statistics of the individual significance tests associated with the intercept and the personal consumption variable have increased, while the experimental statistic of the personal income variable has decreased, and the experimental statistic of the outstanding consumer credit variable remains the same. These facts show that the change from the original model to the orthonormal model does not guarantee an increase in the absolute value of the experimental statistic.
- The estimation of the coefficient of the personal consumption variable is not significantly different from zero in the original model, but it is in the orthogonal model. Thus, it is concluded that multicollinearity affects the statistical analysis of the model. Note that there is also a change in the sign of the estimate, although the purpose of the orthogonal model is not to

obtain estimates for the coefficients, but rather to provide a reference point against which to measure how much the variances are inflated. Note that an orthonormal model is an idealized construction that may lack a proper interpretation in practice.

- The values corresponding to the estimated variance for the random disturbance, the coefficient of determination and the experimental statistic (F exp.) for the global significance test remain the same.

On the other hand, considering the VIF of the independent variables except for the intercept (589.754, 281.886 and 189.487) and their corresponding bounds (17.809, 623.127 and 3545.167) obtained from the expression (10), only the variable of personal consumption verifies that the VIF is higher than the corresponding bound. These results are different from those obtained in Example 2, where the traditional orthogonal model was taken as a reference.

Finally, Tables 2 and 6 show that the experimental values of the statistic t of the variable outstanding consumer credit are the same in the original and orthonormal models. \diamond

The last fact highlighted at the end of the previous example is not a coincidence, but a consequence of the QR decomposition, see Appendix [Test of...](#). Therefore, in this case, the conclusion of the individual significance test will be the same in the original and in the orthonormal model, i.e. we will always be in scenarios a.1 or b.2.

Thus, this behavior establishes a situation where it is required to select the variable fixed in the last position. Some criteria to select the most appropriate variable for this placement could be:

- To fix the variable that is considered less relevant to the model.
- To fix a variable whose associated coefficient is significantly different from zero, since this case would not be of interest for the definition of multicollinearity given in the paper. Note that the interest will be related to a coefficient considered as zero in the original model and significantly different from zero in the orthonormal one.

These options are explored in the Subsection [Choice of the variable to be fix...](#).

4 A non-rejection region associated with a statistical test to detect multicollinearity

[Salmerón et al. \(2025\)](#) show that high values of RVIF are associated with a high degree of multicollinearity. The question, however, is how high RVIFs have to be to reflect troubling multicollinearity.

Taking into account the expressions (7) and (10), it is possible to conclude that multicollinearity is affecting the statistical analysis of the model if it can be verified that:

$$RVIF(i) > \left(\frac{t_{n-k}(1 - \alpha/2)}{\widehat{\beta}_{i,o}} \right)^2 \cdot \widehat{var}(\widehat{\beta}_i) = c_3(i), \quad (11)$$

for any $i = 1, \dots, k$. Note that the intercept is included in this proposal, in contrast to the previous section, in which it was not included.

By following [O'Brien \(2007\)](#) and taking into account that the estimation of the expression (2) can be expressed as:

$$\widehat{var}(\widehat{\beta}_i) = \widehat{\sigma}^2 \cdot RVIF(i) = \frac{\mathbf{e}^T \mathbf{e}}{n-k} \cdot RVIF(i),$$

Table 7: Data set presented previously by Klein and Goldberger

Consumption	Wage income	Non-farm income	Farm income
62.8	43.41	17.10	3.96
65.0	46.44	18.65	5.48
63.9	44.35	17.09	4.37
67.5	47.82	19.28	4.51
71.3	51.02	23.24	4.88
76.6	58.71	28.11	6.37
86.3	87.69	30.29	8.96
95.7	76.73	28.26	9.76
98.3	75.91	27.91	9.31
100.3	77.62	32.30	9.85
103.2	78.01	31.39	7.21
108.9	83.57	35.61	7.39
108.5	90.59	37.58	7.98
111.4	95.47	35.17	7.42

Table 8: OLS estimation for the Klein and Goldberger model

	Estimator	Standard Error	Experimental t	p-value
Intercept	18.702	6.845	2.732	0.021
Wage income	0.380	0.312	1.218	0.251
Non-farm income	1.419	0.720	1.969	0.077
Farm income	0.533	1.400	0.381	0.711
(Obs, Sigma Est., Coef. Det., F exp.)	14.000	36.725	0.919	37.678

there are other factors that counterbalance a high value of RVIF, thereby avoiding high estimated variances for the estimated coefficients. These factors are the sum of the squared residuals ($\text{SSR} = \mathbf{e}^t \mathbf{e}$) of the model (1) and n . Thus, an appropriate specification of the econometric model (i.e., one that implies a good fit and, consequently, a small SSR) and a large sample size can compensate for high RVIF values. However, contrary to what happens for the VIF in the traditional case, these factors are taken into account in the threshold $c_3(i)$, as established in the expression (11) in $\widehat{\text{var}}(\widehat{\beta}_i)$.

Example 4. This contribution can be illustrated with the data set previously presented by Klein and Goldberger (1955), which includes variables for consumption, C , wage incomes, I , non-farm incomes, InA , and farm incomes, IA , in United States from 1936 to 1952, as shown in Table 7 (data from 1942 to 1944 are not available because they were war years).

Table 8 shows the OLS estimations of the model explaining consumption as a function of the rest of the variables. Note that there is some incoherence between the individual significance values of the variables and the global significance of the model.

The RVIFs are calculated, yielding 1.275, 0.002, 0.014 and 0.053, respectively. The associated bounds, $c_3(i)$, are also calculated, yielding 0.002, 0.0001, 0.018 and 1.826, respectively.

Since the coefficient of the wage income variable is not significantly different from zero, and because it is verified that $0.002 > 0.0001$, from (11) it is concluded that the degree of multicollinearity existing in the model is affecting its statistical analysis. \diamond

Table 9: Theorem results of the Wissel model

	RVIFs	c0	c3	Scenario	Affects
Intercept	194.866090	7.371069	1.017198	b.1	Yes
Personal consumption	30.326281	4.456018	0.915790	b.1	Yes
Personal income	4.765888	2.399341	10.535976	b.2	No
Outstanding consumer credit	0.000038	0.000002	0.000715	b.2	No

Table 10: Theorem results of the Klein and Goldberger model

	RVIFs	c0	c3	Scenario	Affects
Intercept	1.275948	1.918383	0.002189	a.1	No
Wage income	0.002653	0.000793	0.000121	b.1	Yes
Non-farm income	0.014131	0.011037	0.018739	b.2	No
Farm income	0.053355	0.001558	1.826589	b.2	No

4.1 From the RVIF

Considering that in the original model (1) the null hypothesis $\beta_i = 0$ of the individual significance test is not rejected if:

$$RVIF(i) > \left(\frac{\hat{\beta}_i}{\hat{\sigma} \cdot t_{n-k}(1 - \alpha/2)} \right)^2 = c_0(i), \quad i = 1, \dots, k,$$

while in the orthonormal model, the null hypothesis is rejected if $RVIF(i) > c_3(i)$, the following theorem can be established:

Theorem. Given the multiple linear regression model (1), the degree of multicollinearity affects its statistical analysis (with a level of significance of $\alpha\%$) if there is a variable i , with $i = 1, \dots, k$, that verifies $RVIF(i) > \max\{c_0(i), c_3(i)\}$.

Note that Salmerón et al. (2025) indicate that the RVIF must be calculated with unit length data (as any other transformation removes the intercept from the analysis), however, for the correct application of this theorem the original data must be used as no transformation has been considered in this paper.

Example 5. Tables 9 and 10 present the results of applying the theorem to the Wissel (2009) and Klein and Goldberger (1955) models, respectively. Note that in both cases, there is a variable i that verifies that $RVIF(i) > \max\{c_0(i), c_3(i)\}$, and consequently, we can conclude that the degree of approximate multicollinearity is affecting the statistical analysis in both models (with a level of significance of 5%). \diamond

5 The rvif package

The results developed in Salmerón et al. (2025) and in this paper have been implemented in the **rvif** package of R (R Core Team (2025)). The following shows how to replicate the results presented in both papers from the existing commands **rvifs** and **multicollinearity** in **rvif**. For this reason, the code executed is shown below.

In addition, the following issues will be addressed:

- Discussion on the effect of sample size in detecting the influence of multicollinearity on the statistical analysis of the model.

- Discussion on the choice of the variable to be fixed as the last one before the orthonormalization.

The code used in these two Subsections is available at <https://github.com/rnoremblas/RVIF/tree/main/rvif%20package>. It is also interesting to consult the package vignette using the command `browseVignettes("rvif")`, as well as its web page with `browseURL(system.file("docs/index.html", package = "rvif"))` or <https://www.ugr.es/local/romansg/rvif/index.html>.

5.1 Detection of multicollinearity with RVIF: does the degree of multicollinearity affect the statistical analysis of the model?

In Salmerón et al. (2025) a series of examples are presented to illustrate the usefulness of RVIF to detect the degree of approximate multicollinearity in a multiple linear regression model. Results presented by Salmerón et al. (2025) will be reproduced by using the command `rvifs` of `rvif` package and complemented with the contribution developed in the present work by using the command `multicollinearity` of the same package. In order to facilitate the reading of the paper, this information is available in Appendix Examples of....

On the other hand, the following shows how to use the above commands to obtain the results shown in Table 9 of this paper:

```
y_W = Wissel[,2]
X_W = Wissel[,3:6]
multicollinearity(y_W, X_W)

#>          RVIFs      c0      c3 Scenario Affects
#> 1 1.948661e+02 7.371069e+00 1.017198e+00    b.1    Yes
#> 2 3.032628e+01 4.456018e+00 9.157898e-01    b.1    Yes
#> 3 4.765888e+00 2.399341e+00 1.053598e+01    b.2     No
#> 4 3.821626e-05 2.042640e-06 7.149977e-04    b.2     No
```

It is noted that the first two arguments of the `multicollinearity` command are, respectively, the dependent variable of the linear model and the design matrix containing the independent variables (intercept included as the first column).

While the results in Table 10 can be obtained using this code:

```
y_KG = KG[,1]
cte = rep(1, length(y))
X_KG = cbind(cte, KG[,2:4])
multicollinearity(y_KG, X_KG)

#>          RVIFs      c0      c3 Scenario Affects
#> 1 1.275947615 1.9183829079 0.0021892653    a.1     No
#> 2 0.002652862 0.0007931658 0.0001206694    b.1    Yes
#> 3 0.014130621 0.0110372472 0.0187393601    b.2     No
#> 4 0.053354814 0.0015584988 1.8265885762    b.2     No
```

As is known, in both cases it is concluded that the degree of multicollinearity in the model affects its statistical analysis.

The `multicollinearity` command is used by default with a significance level of 5% for the application of the Theorem set in Subsection From the RVIF. Note that if the significance level is changed to 1% (third argument of the `multicollinearity` command), in the Klein and Goldberger model it is obtained that the individual significance test of the intercept is also affected by the degree of existing multicollinearity:

```
multicollinearity(y_W, X_W, alpha = 0.01)
```

```
#>          RVIFs          c0          c3 Scenario Affects
#> 1 1.948661e+02 3.791375e+00 1.977602791     b.1    Yes
#> 2 3.032628e+01 2.291992e+00 1.780449066     b.1    Yes
#> 3 4.765888e+00 1.234122e+00 20.483705068     b.2    No
#> 4 3.821626e-05 1.050650e-06 0.001390076     b.2    No

multicollinearity(y_KG, X_KG, alpha = 0.01)

#>          RVIFs          c0          c3 Scenario Affects
#> 1 1.275947615 0.9482013897 0.0044292796     b.1    Yes
#> 2 0.002652862 0.0003920390 0.0002441361     b.1    Yes
#> 3 0.014130621 0.0054553932 0.0379131147     b.2    No
#> 4 0.053354814 0.0007703211 3.6955190555     b.2    No
```

It can be seen that the values of c_0 and c_3 change depending on the significance level used.

5.2 Effect of the sample size on the detection of the influence of multicollinearity on the statistical analysis of the model

The introduction has highlighted the idea that the measures traditionally used to detect whether the degree of multicollinearity is of concern may indicate that it is troubling while the model analysis is not affected by it. Example 1 shows that this may be due, among other factors, to the size of the sample.

To explore this issue in more detail, below is given an example where traditional measures of multicollinearity detection indicate that the existing multicollinearity is troubling while the statistical analysis of the model is not affected when the sample size is high. In particular, observations are simulated for $\mathbf{X} = [\mathbf{1} \mathbf{X}_2 \mathbf{X}_3 \mathbf{X}_4 \mathbf{X}_5 \mathbf{X}_6]$ where:

$$\begin{aligned}\mathbf{X}_2 &\sim N(5, 0.1^2), \quad \mathbf{X}_3 \sim N(5, 10^2), \quad \mathbf{X}_4 = \mathbf{X}_3 + \mathbf{p} \\ \mathbf{X}_5 &\sim N(-1, 3^2), \quad \mathbf{X}_6 \sim N(15, 2.5^2),\end{aligned}$$

where $\mathbf{p} \sim N(5, 0.5^2)$ and considering three different sample sizes: $n = 3000$ (Simulation 1), $n = 100$ (Simulation 2) and $n = 30$ (Simulation 3). In all cases the dependent variable is generated according to:

$$\mathbf{y} = 4 + 5 \cdot \mathbf{X}_2 - 9 \cdot \mathbf{X}_3 - 2 \cdot \mathbf{X}_4 + 2 \cdot \mathbf{X}_5 + 7 \cdot \mathbf{X}_6 + \mathbf{u},$$

where $\mathbf{u} \sim N(0, 2^2)$.

To set the results, a seed has been established using the command `set.seed(2024)`.

With this generation it is intended that the variable \mathbf{X}_2 is linearly related to the intercept as well as \mathbf{X}_3 to \mathbf{X}_4 . This is supported by the results shown in Table 11, which have been obtained using the `multiColl` package of R ([R Core Team \(2025\)](#)) using the commands `CV`, `VIF` and `CN`.

The results imply the same conclusions in all three simulations:

- There is a worrying degree of non-essential multicollinearity in the model relating the intercept to the variable \mathbf{X}_2 since its coefficient of variation (CV) is lower than 0.1002506.
- There is a worrying degree of essential multicollinearity in the model relating the variables \mathbf{X}_3 and \mathbf{X}_4 since the associated Variance Inflation Factors (VIF) are greater than 10.

However, does the degree of multicollinearity detected really affect the statistical analysis of the model? According to the results shown in Tables 12 to 14 this is not always the case:

Table 11: CVs, VIFs and CN for data of Simulations 1, 2 and 3

	Simulation 1	Simulation 2	Simulation 3
X2 CV	0.020	0.019	0.025
X3 CV	2.010	1.827	3.326
X4 CV	1.004	0.968	1.434
X5 CV	3.138	1.948	2.413
X6 CV	0.167	0.176	0.194
X2 VIF	1.003	1.053	1.167
X3 VIF	388.669	373.092	926.768
X4 VIF	388.696	373.280	929.916
X5 VIF	1.001	1.014	1.043
X6 VIF	1.003	1.066	1.254
CN	148.247	162.707	123.025

Table 12: Theorem results of the Simulation 1 model

	RVIFs	c0	c3	Scenario	Affects
Intercept	0.934369	1.916912	0.000001	a.1	No
X2	0.034899	1.359909	0.000168	a.1	No
X3	0.001299	5.339519	0.000000	a.1	No
X4	0.001296	0.230992	0.000004	a.1	No
X5	0.000036	0.257015	0.000000	a.1	No
X6	0.000053	3.160352	0.000000	a.1	No

- In Simulation 1, when $n = 3000$, the degree of multicollinearity in the model does not affect the statistical analysis of the model; scenario a.1 is always verified, i.e., both in the model proposed and in the orthonormal model, the null hypothesis is rejected in the individual significance tests.
- In Simulation 2, when $n = 100$, the degree of multicollinearity in the model affects the statistical analysis of the model only in the individual significance of the intercept; in all other cases scenario a.1 is verified again.
 - As will be seen below, the fact that the individual significance of the variable X_2 is not affected may be due to the number of observations in the data set. But it may also be because multicollinearity of the nonessential type affects only the intercept estimate. Thus, for example, in Salmerón et al. (2019) it is shown (see Table 2 of Example 2) that solving this type of approximate multicollinearity (by centering the variables that cause it) only modifies the estimate of the intercept and its standard deviation, with the estimates of the rest of the independent variables remaining unchanged.
- In Simulation 3, when $n = 30$, the degree of multicollinearity in the model affects the statistical analysis of the model in the individual significance of the intercept, in X_2 and in X_4 .
 - In this case, as discussed, the reduction in sample size does not prevent the individual significance of X_2 from being affected.

In conclusion, as O'Brien (2007) indicates, it can be seen that the increase in sample size prevents the statistical analysis of the model from being affected by the degree of existing multicollinearity, even though the values of the measures traditionally used to detect this problem indicate that it is troubling. To reach this conclusion, the use of the RVIF proposed by Salmerón et al. (2025) and the theorem developed in this paper is decisive.

Table 13: Theorem results of the Simulation 2 model

	RVIFs	c0	c3	Scenario	Affects
Intercept	32.965272	0.228580	0.001580	b.1	Yes
X2	1.179581	1.678248	0.014061	a.1	No
X3	0.037287	5.662562	0.000001	a.1	No
X4	0.036687	0.113376	0.000353	a.1	No
X5	0.001269	0.252728	0.000006	a.1	No
X6	0.001601	3.060976	0.000001	a.1	No

Table 14: Theorem results of the Simulation 3 model

	RVIFs	c0	c3	Scenario	Affects
Intercept	70.990340	46.793605	0.008667	b.1	Yes
X2	2.524792	0.000570	0.005083	b.1	Yes
X3	0.187896	3.892727	0.000007	a.1	No
X4	0.187317	0.168758	0.005113	b.1	Yes
X5	0.003863	0.169923	0.000325	a.1	No
X6	0.005193	2.139108	0.000013	a.1	No

5.3 Selection of the variable to be set as the last before orthonormalization

Since there are as many QR decompositions as there are possible rearrangements of the independent variables, it is convenient to test different options to determine whether the degree of multicollinearity in the regression model affects its statistical analysis.

A first possibility is to try all possible reorderings considering that the intercept must always be in first place. Thus, in the Example 2 of Salmerón et al. (2025) (see Appendix Examples of... for more details) it is considered that $\mathbf{X} = [\mathbf{1} \ \mathbf{K} \ \mathbf{W}]$ (see Table 15), but it could also be considered that $\mathbf{X} = [\mathbf{1} \ \mathbf{W} \ \mathbf{K}]$ (see Table 16).

Note that in these tables the values for each variable of RVIF and c_0 are always the same, but those of c_3 change depending on the position of each variable within the design matrix.

It is observed that in one of the two possibilities considered, the individual significance of the labor variable is affected by the degree of existing multicollinearity.

Therefore, to state that the statistical analysis of the multiple linear regression model is not affected by the multicollinearity present in the model, it is necessary to check all the possible QR decompositions and to determine in all cases that the statistical analysis is not affected. However, to determine that the statistical analysis of the model is affected by the presence of multicollinearity, it is sufficient to find one of the possible rearrangements in which the situation b.1 occurs.

Another possibility is to set in the last position of \mathbf{X} a particular variable following a specific criterion. Thus, for example, in Example 3 of Salmerón et al. (2025) (see Appendix Examples of... for more details) it is verified that the variable FA has a coefficient significantly different from zero. Fixing this variable in third place since the individual significance will not be modified yields the results shown in Table 17.

Table 15: Theorem results of the Example 2 of Salmerón et al. (2025)

	RVIFs	c0	c3	Scenario	Affects
Intercept	6388.887975	88495.933700	1.649518	a.1	No
Capital	4.136993	207.628058	0.050431	a.1	No
Work	37.336378	9.445619	147.582132	b.2	No

Table 16: Theorem results of the Example 2 of Salmerón et al. (2025) (reordination 2)

	RVIFs	c0	c3	Scenario	Affects
Intercept	6388.882446	88495.933700	1.649518	a.1	No
Work	37.336378	9.445619	1.163201	b.1	Yes
Capital	4.136993	207.628058	0.082430	a.1	No

Table 17: Theorem results of the Example 3 of Salmerón et al. (2025) reordination

	RVIFs	c0	c3	Scenario	Affects
OI	1.696454e-12	9.594942e-13	1.775244e-13	b.1	Yes
S	1.718535e-12	1.100437e-12	1.012113e-12	b.1	Yes
FA	1.829200e-16	2.307700e-16	1.449800e-16	a.1	No

It can be seen that in this case the degree of multicollinearity in the model affects the individual significance of the OI and S variables.

6 Conclusions

In this paper, following Salmerón et al. (2025), we propose an alternative orthogonal model that leads to a lower bound for the RVIF, indicating whether the degree of multicollinearity present in the model affects its statistical analysis. These thresholds serve as complements to the results presented by O'Brien (2007), who stated that the estimated variances depend on other factors that can counterbalance a high value of the VIF, for example, the size of the sample or the estimated variance of the independent variables. Thus, the thresholds presented for the RVIF also depend on these factors meeting a threshold associated with each independent variable (including the intercept). Note that these thresholds will indicate whether the degree of multicollinearity affects the statistical analysis.

As these thresholds are derived from the individual significance tests of the model, it is possible to reinterpret them as a statistical test to determine whether the degree of multicollinearity in the linear regression model affects its statistical analysis. This analytic tool allows researchers to conclude whether the degree of multicollinearity is statistically troubling and whether it needs to be treated. We consider this to be a relevant contribution since, to the best of our knowledge, the only existing example of such a measure, presented by Farrar and Glauber (1967), has been strongly criticized (in addition to the limitations highlighted in the introduction, it should be noted that it completely ignores approximate non-essential multicollinearity since the correlation matrix does not include information on the intercept); consequently, this new statistical test with a non-rejection region will fill a gap in the scientific literature.

On the other hand, note that the position of each of the variables in the matrix \mathbf{X} uniquely determines the reference orthonormal model \mathbf{X}_0 . It is to say, there are as many reference models given by the proposed QR decomposition as there are possible rearrangements of the variables within the matrix \mathbf{X} .

In this sense, as has been shown, in order to affirm that the statistical analysis of the model is not affected by the degree of multicollinearity existing in the model (with the degree of significance used in the application of the proposed theorem), it is necessary to state that in all the possible rearrangements of \mathbf{X} it is concluded that scenario b.1 does not occur. On the other hand, when there is a rearrangement in which this scenario appears, it can be stated (to the degree of significance used when applying the proposed theorem) that the degree of existing multicollinearity affects the statistical analysis of the model.

Finally, as a future line of work, it would be interesting to complete the analysis presented here by studying when the degree of multicollinearity in the model affects its numerical analysis.

7 Acknowledgments

This work has been supported by project PP2019-EI-02 of the University of Granada (Spain) and by project A-SEJ-496-UGR20 of the Andalusian Government's Counseling of Economic Transformation, Industry, Knowledge and Universities (Spain).

8 Appendix

8.1 Inconsistency in hypothesis tests: situation a.2

From a numerical point of view it is possible to reject $H_0 : \beta_i = 0$ while $H_0 : \beta_{i,o} = 0$ is not rejected, which implies that $t_i^o < t_{n-k}(1 - \alpha/2) < t_i$. Or, in other words, $t_i/t_i^o > 1$.

However, from expression (8) it is obtained that $\hat{\sigma} = |\hat{\beta}_{i,o}|/t_i^o$. By substituting $\hat{\sigma}$ in expression (4), taking into account expression (7), it is obtained that

$$\frac{t_i}{t_i^o} = \frac{|\hat{\beta}_i|}{|\hat{\beta}_{i,o}|} \cdot \frac{1}{\sqrt{RVIF(i)}}.$$

From this expression it can be concluded that in situations with high collinearity, $RVIF(i) \rightarrow +\infty$, the ratio t_i/t_i^o will tend to zero, and the condition $t_i/t_i^o > 1$ will rarely occur. That is to say, the inconsistency in situation a.2, commented on in the preliminaries of the paper, will not appear.

On the other hand, if the variable i is orthogonal to the rest of independent variables, it is verified that $\hat{\beta}_{i,o} = \hat{\beta}_i$ since $p_i = (0 \dots \underbrace{1 \dots 0}_{(i)} \dots 0)$. At the same time, $RVIF(i) = \frac{1}{SST_i}$ where

SST denotes the sum of total squares. If there is orthonormality, as proposed in this paper, $SST_i = 1$ and, as consequence, it is verified that $t_i = t_i^o$. Thus, the individual significance test for the original data and for the orthonormal data are the same.

8.2 Test of individual significance of coefficient k

Taking into account that it is verified that $\beta_o = P\beta$ where:

$$\beta_o = \begin{pmatrix} \beta_{1,o} \\ \beta_{2,o} \\ \vdots \\ \beta_{k,o} \end{pmatrix}, \quad P = \begin{pmatrix} p_{11} & p_{12} & \dots & p_{1k} \\ 0 & p_{22} & \dots & p_{2k} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & p_{kk} \end{pmatrix}, \quad \beta = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \end{pmatrix},$$

it is obtained that $\beta_{k,o} = p_{kk}\beta_k$. Then, the null hypothesis $H_0 : \beta_{k,o} = 0$ is equivalent to $H_0 : \beta_k = 0$. Due to this fact, Tables 2 and 6 showed an expectable behaviour. However, this behaviour will be analyzed with more detail.

The experimental value to be considered to take a decision in the test with null hypothesis $H_0 : \beta_{k,o} = 0$ and alternative hypothesis $H_1 : \beta_{k,o} \neq 0$ is given by the following expression:

$$t_k^o = \left| \frac{\hat{\beta}_{k,o}}{\sqrt{var(\hat{\beta}_{k,o})}} \right|.$$

Taking into account that $\widehat{\beta}_o = \mathbf{P}\widehat{\beta}$ and $var(\widehat{\beta}_o) = \mathbf{P}var(\widehat{\beta})\mathbf{P}^t$, it is verified that $\widehat{\beta}_{k,o} = p_{kk}\widehat{\beta}_k$ and $var(\widehat{\beta}_{k,o}) = p_{kk}^2 var(\widehat{\beta}_k)$. Then:

$$t_k^o = \left| \frac{p_{kk}\widehat{\beta}_k}{p_{kk}\sqrt{var(\widehat{\beta}_k)}} \right| = \left| \frac{\widehat{\beta}_k}{\sqrt{var(\widehat{\beta}_k)}} \right| = t_k,$$

where t_k is the experimental value to take a decision in the test with null hypothesis $H_0 : \beta_k = 0$ and alternative hypothesis $H_1 : \beta_k \neq 0$.

8.3 Examples of Salmerón et al. (2025)

Example 1 of Salmerón et al. (2025): Detection of traditional nonessential multicollinearity. Using data from a financial model in which the Euribor (E) is analyzed from the Harmonized Index of Consumer Prices (HICP), the balance of payments to net current account (BC) and the government deficit to net nonfinancial accounts (GD), we illustrate the detection of approximate multicollinearity of the non-essential type, i.e. where the intercept is related to one of the remaining independent variables (for details see Marquardt and Snee (1975)). For more information on this data set use `help(euribor)`.

Note that Salmerón-Gómez et al. (2019) establishes that an independent variable with a coefficient of variation less than 0.1002506 indicates that this variable is responsible for a non-essential multicollinearity problem.

Thus, first of all, the approximate multicollinearity detection is performed using the measures traditionally applied for this purpose: the Variance Inflation Factor (VIF) and the Condition Number (CN). Values higher than 10 for the VIF (see, for example, Marquardt (1970)) and 30 for the CN (see, for example, Belsley (1991) or Belsley et al. (1980)), imply that the degree of existing multicollinearity is troubling. Moreover, according to Salmerón-Gómez et al. (2019), the VIF is only able to detect essential multicollinearity (relationship between independent variables excluding the intercept, see Marquardt and Snee (1975)), while the CN detects both essential and non-essential multicollinearity.

Therefore, the values calculated below (using the VIF, CN and CVs commands from the `multiColl` package, see Salmerón et al. (2021) and Salmerón et al. (2022) for more details on this package) indicate that the degree of approximate multicollinearity existing in the model of the essential type is not troubling, while that of the non-essential type is troubling due to the relationship of HIPC with the intercept.

```
E = euribor[,1]
data1 = euribor[,-1]

VIF(data1)

#>      HIPC        BC        GD
#> 1.349666 1.058593 1.283815

CN(data1)

#> [1] 39.35375

CVs(data1)

#> [1] 0.06957876 4.34031035 0.55015508
```

This assumption is confirmed by calculating the RVIF values, which point to a strong relationship between the second variable and the intercept:

```
rvifs(data1, ul = T, intercept = T)

#>          RVIF      %
#> Intercept 250.294157 99.6005
#> Variable 2 280.136873 99.6430
#> Variable 3  1.114787 10.2967
#> Variable 4  5.525440 81.9019
```

The output of the `rvifs` command provides the values of the Redefined Variance Inflation Factor (RVIF) and the percentage of multicollinearity due to each variable (denoted as a_i in the [An orthonormal...](#) section).

In this case, three of the four arguments available in the `rvifs` command are used:

- The first of these refers to the design matrix containing the independent variables (the intercept, if any, being the first column).
- The second argument, *ul*, indicates that the data is to be transformed into unit length. This transformation makes it possible to establish that the RVIF is always greater than or equal to 1, having as a reference a minimum value that indicates the absence of worrying multicollinearity.
- The third argument, *intercept*, indicates whether there is an intercept in the design matrix.

Note that these results can also be obtained after using the `lm` and `model.matrix` commands as follows:

```
reg_E = lm(euribor[,1]~as.matrix(euribor[,-c(1,2)]))
rvifs(model.matrix(reg_E))

#>          RVIF      %
#> Intercept 250.294157 99.6005
#> Variable 2 280.136873 99.6430
#> Variable 3  1.114787 10.2967
#> Variable 4  5.525440 81.9019
```

Finally, the application of the Theorem established in Subsection [From the RVIF](#) detects that the individual inference of the second variable (HIPC) is affected by the degree of multicollinearity existing in the model. These results are obtained using the `multicollinearity` command from the `rvif` package:

```
multicollinearity(E, data1)

#>          RVIFs      c0      c3 Scenario Affects
#> 1 5.325408e+00 1.575871e+01 2.166907e-02    a.1     No
#> 2 5.357830e-04 3.219456e-06 4.249359e-05    b.1     Yes
#> 3 5.109564e-11 1.098649e-09 2.586237e-12    a.1     No
#> 4 1.631439e-11 3.216522e-10 8.274760e-13    a.1     No
```

Therefore, it can be established that the existing multicollinearity affects the statistical analysis of the Euribor model.

Example 2 of Salmerón et al. (2025): Detection of generalized nonessential multicollinearity. Using data from a Cobb-Douglas production function in which the production (P) is analyzed from the capital (K) and the work (W), we illustrate the detection of approximate multicollinearity of the generalized non-essential type, i.e., that in which at least two independent variables with very little variability (excluding the intercept) are related to each

other (for more details, see [Salmerón et al. \(2020\)](#)). For more information on this dataset use `help(CDpf)`.

Using the `rvifs` command, it can be determined that both capital and labor are linearly related to each other with high RVIF values below the threshold established as a worrying value:

```
P = CDpf[,1]
data2 = CDpf[,2:4]

rvifs(data2, ul = T)

#>          RVIF      %
#> Intercept 178888.71 99.9994
#> Variable 2 38071.44 99.9974
#> Variable 3 255219.74 99.9996
```

However, the application of the Theorem established in Subsection [From the RVIF](#) does not detect that the degree of multicollinearity in the model affects the statistical analysis of the model:

```
multicollinearity(P, data2)

#>          RVIFs      c0      c3 Scenario Affects
#> 1 6388.887975 88495.933700 1.64951764    a.1     No
#> 2 4.136993   207.628058  0.05043083    a.1     No
#> 3 37.336378   9.445619 147.58213164    b.2     No
```

Now then, if we rearrange the design matrix \mathbf{X} we obtain that:

```
data2 = CDpf[,c(2,4,3)]
multicollinearity(P, data2)

#>          RVIFs      c0      c3 Scenario Affects
#> 1 6388.882446 88495.933700 1.64951764    a.1     No
#> 2 37.336378   9.445619 1.16320125    b.1     Yes
#> 3 4.136993   207.628058 0.08242979    a.1     No
```

Therefore, it can be established that the existing multicollinearity does affect the statistical analysis of the Cobb-Douglas production function model.

Example 3 of Salmerón et al. (2025): Detection of essential multicollinearity. Using data from a model in which the number of employees of Spanish companies (NE) is analyzed from the fixed assets (FA), operating income (OI) and sales (S), we illustrate the detection of approximate multicollinearity of the essential type, i.e., that in which at least two independent variables (excluding the intercept) are related to each other (for more details, see [Marquardt and Snee \(1975\)](#)). For more information on this dataset use `help(employees)`.

In this case, the `rvifs` command shows that variables three and four (OI and S) have a high VIF value, so they are highly linearly related:

```
NE = employees[,1]
data3 = employees[,2:5]

rvifs(data3, ul = T)

#>          RVIF      %
#> Intercept 2.984146 66.4896
#> Variable 2 5.011397 80.0455
#> Variable 3 15186.744870 99.9934
#> Variable 4 15052.679178 99.9934
```

Note that if in `rvifs(data3, ul = T)` the unit length transformation is avoided, which is done in the `multicollinearity` command, the RVIF cannot be calculated since the system is computationally singular. For this reason, the intercept is eliminated below since it has been shown above that it does not play a relevant role in the linear relationships of the model.

Finally, the application of the Theorem established in Subsection [From the RVIF](#) detects that the individual inference of the third variable (OI) is affected by the degree of multicollinearity existing in the model:

```
multicollinearity(NE, data3[,-1])

#>          RVIFs      c0      c3 Scenario Affects
#> 1 1.829154e-16 2.307712e-16 4.679301e-17    a.1     No
#> 2 1.696454e-12 9.594942e-13 2.129511e-13    b.1    Yes
#> 3 1.718535e-12 1.100437e-12 2.683809e-12    b.2     No
```

Therefore, it can be established that the existing multicollinearity affects the statistical analysis of the model of the number of employees in Spanish companies.

Example 4 of Salmerón et al. (2025): The special case of simple linear model. The simple linear regression model is an interesting case because it has a single independent variable and the intercept. Since the intercept is not properly considered as an independent variable of the model in many cases (see introduction of [Salmerón-Gómez et al. \(2019\)](#) for more details), different software (including R, [R Core Team \(2025\)](#)) do not consider that there can be worrisome multicollinearity in this type of model.

To illustrate this situation, [Salmerón et al. \(2025\)](#) randomly generates observations for the following two simple linear regression models $y_1 = \beta_1 + \beta_2 V + u_1$ and $y_2 = \alpha_1 + \alpha_2 Z + u_2$, according to the following code:

```
set.seed(2022)
obs = 50
cte4 = rep(1, obs)
V = rnorm(obs, 10, 10)
y1 = 3 + 4*V + rnorm(obs, 0, 2)
Z = rnorm(obs, 10, 0.1)
y2 = 3 + 4*Z + rnorm(obs, 0, 2)

data4.1 = cbind(cte4, V)
data4.2 = cbind(cte4, Z)
```

For more information on these data sets use `help(SLM1)` and `help(SLM2)`.

As mentioned above, the R package ([R Core Team \(2025\)](#)) denies the existence of multicollinearity in this type of model. Thus, for example, when using the `vif` command of the `car` package on `reg=lm(y1~V)` the following message is obtained: *Error in vif.default(reg): model contains fewer than 2 terms.*

Undoubtedly, this message is coherent with the fact that, as mentioned above, the VIF is not capable of detecting non-essential multicollinearity (which is the only multicollinearity that exists in this type of model). However, the error message provided may lead a non-specialized user to consider that the multicollinearity problem does not exist in this type of model. These issues are addressed in more depth in [Salmerón et al. \(2022\)](#).

On the other hand, the calculation of the RVIF in the first model shows that the degree of multicollinearity is not troubling, since it presents very low values:

```
rvifs(data4.1, ul = T)

#>          RVIF      %
#> Intercept 2.015249 50.3783
#> Variable  2 2.015249 50.3783
```

While in the second model they are very high, indicating a problem of non-essential multicollinearity:

```
rvifs(data4.2, ul = T)

#>          RVIF      %
#> Intercept 9390.044 99.9894
#> Variable 2 9390.044 99.9894
```

By using the `multicollinearity` command, it is found that the individual inference of the intercept of the second model is affected by the degree of multicollinearity in the model:

```
multicollinearity(y1, data4.1)

#>          RVIFs      c0      c3 Scenario Affects
#> 1 0.0403049717 0.6454323 1.045802e-05     a.1      No
#> 2 0.0002675731 0.8383436 8.540101e-08     a.1      No

multicollinearity(y2, data4.2)

#>          RVIFs      c0      c3 Scenario Affects
#> 1 187.800878 21.4798003 0.03277691     b.1      Yes
#> 2 1.879296  0.3687652 9.57724567     b.2      No
```

Therefore, it can be established that the multicollinearity existing in the first simple linear regression model does not affect the statistical analysis of the model, while in the second one it does.

References

- D. Belsley. A guide to using the collinearity diagnostics. *Computational Science in Economics and Management*, 4:33–50, 1991. doi: 10.1007/BF00426854. URL <https://doi.org/10.1007/BF00426854>. [p209]
- D. Belsley, E. Kuh, and R. Welsch. *Regression diagnostics: Identifying influential data and sources of collinearity*. John Wiley and Sons, 1980. URL <https://onlinelibrary.wiley.com/doi/book/10.1002/0471725153>. [p209]
- D. E. Farrar and R. R. Glauber. Multicollinearity in regression analysis: the problem revisited. *The Review of Economic and Statistics*, 49(1):92–107, 1967. doi: 10.2307/1937887. URL <https://doi.org/10.2307/1937887>. [p192, 193, 207]
- C. B. García, R. Salmerón, C. García-García, and J. García. Residualization: justification, properties and application. *Journal of Applied Statistics*, 47(11):1990–2010, 2019. doi: 10.1111/insr.12575. URL <https://doi.org/10.1111/insr.12575>. [p194]
- D. Gujarati. *Basic Econometrics*. McGraw-Hill (fourth edition), 2003. URL <https://highered.mheducation.com/sites/0072335424/>. [p192]
- R. Gunst and R. Mason. Advantages of examining multicollinearities in regression analysis. *Biometrics*, 33(1):249–260, 1977. doi: 10.2307/2529320. URL <https://doi.org/10.2307/2529320>. [p192]
- Y. Haitovsky. Multicollinearity in regression analysis: Comment. *The Review of Economics and Statistics*, 51(4):486–489, 1969. doi: 10.2307/1926450. URL <https://doi.org/10.2307/1926450>. [p193]

- L. R. Klein and A. S. Goldberger. *An Economic Model of the United States 1929-1952*. Amsterdam: North-Holland Publishing Company, 1955. doi: 10.2307/2227976. URL <https://doi.org/10.2307/2227976>. [p201, 202]
- T. K. Kumar. Multicollinearity in regression analysis. *The Review of Economics and Statistics*, 57(3):365–366, 1975. doi: 10.2307/1923925. URL <https://doi.org/10.2307/1923925>. [p193]
- D. Marquardt. Generalized inverses, ridge regression, biased linear estimation and nonlinear estimation. *Technometrics*, 12(3):591–612, 1970. doi: 10.2307/1267205. URL <https://doi.org/10.2307/1267205>. [p194, 209]
- D. Marquardt and R. Snee. Ridge regression in practice. *The American Statistician*, 29(1):3–20, 1975. doi: 10.2307/2683673. URL <https://doi.org/10.2307/2683673>. [p192, 209, 211]
- R. O'Brien. A caution regarding rules of thumb for variance inflation factors. *Quality & quantity*, 41(5):673–690, 2007. URL <https://link.springer.com/article/10.1007/s11135-006-9018-6>. [p192, 193, 194, 200, 205, 207]
- J. O'Hagan and B. McCabe. Tests for the severity of multicollinearity in regression analysis: A comment. *The Review of Economics and Statistics*, 57(3):368–370, 1975. doi: 10.2307/1923927. URL <https://doi.org/10.2307/1923927>. [p193]
- R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 4.5.1 edition, 2025. URL <https://www.R-project.org/>. [p192, 193, 202, 204, 212]
- R. Salmerón, C. García, and J. García. Comment on “A note on collinearity diagnostics and centering” by Velilla (2018). *The American Statistician*, 74(1):68–71, 2019. doi: 10.1080/00031305.2019.1635527. URL <https://doi.org/10.1080/00031305.2019.1635527>. [p205]
- R. Salmerón, C. García, and J. García. Detection of near-multicollinearity through centered and noncentered regression. *Mathematics*, 8(6):931, 2020. doi: 10.3390/math8060931. URL <https://doi.org/10.3390/math8060931>. [p211]
- R. Salmerón, C. García, and J. García. A guide to using the R package multicoll for detecting multicollinearity. *Computational Economics*, 57:529–536, 2021. doi: 10.1007/s10614-019-09967-y. URL <https://doi.org/10.1007/s10614-019-09967-y>. [p209]
- R. Salmerón, C. García, and J. García. The multicoll package versus other existing packages in R to detect multicollinearity. *Computational Economics*, 60:439–450, 2022. doi: 10.1007/s10614-021-10154-1. URL <https://doi.org/10.1007/s10614-021-10154-1>. [p209, 212]
- R. Salmerón, C. B. García, and J. García. A redefined variance inflation factor: overcoming the limitations of the variance inflation factor. *Computational Economics*, 65:337–363, 2025. doi: 10.1007/s10614-024-10575-8. URL <https://doi.org/10.1007/s10614-024-10575-8>. [p193, 195, 196, 200, 202, 203, 205, 206, 207, 209, 210, 211, 212]
- R. Salmerón-Gómez, A. Rodríguez-Sánchez, and C. García-García. Diagnosis and quantification of the non-essential collinearity. *Computational Statistics*, 35:647–666, 2019. doi: 10.1007/s00180-019-00922-x. URL <https://doi.org/10.1007/s00180-019-00922-x>. [p192, 209, 212]
- S. Silvey. Multicollinearity and imprecise estimation. *Journal of the Royal Statistical Society. Series B (Methodological)*, 31(3):539–552, 1969. URL <https://www.jstor.org/stable/2984357>. [p192]
- C. R. Wicher. The detection of multicollinearity: A comment. *The Review of Economics and Statistics*, 57(3):366–368, 1975. doi: 10.2307/1923926. URL <https://doi.org/10.2307/1923926>. [p193]
- A. Willan and D. Watts. Meaningful multicollinearity measures. *Technometrics*, 20(4):407–412, 1978. doi: 10.1080/00401706.1978.10489694. URL <https://doi.org/10.1080/00401706.1978.10489694>. [p192]

J. Wissel. *A new biased estimator for multivariate regression models with highly collinear variables*. 2009. URL <https://opus.bibliothek.uni-wuerzburg.de/frontdoor/index/index/docId/2949>. [p197, 199, 202]

J. Wooldridge. *Introductory Econometrics. A Modern Approach*. South-Western, CENGAGE Learning (7th edition), 2020. URL <https://www.cengage.uk/c/introductory-econometrics-a-modern-approach-7e-wooldridge/9781337558860PF/>. [p192, 194]

Román Salmerón-Gómez

University of Granada

Department of Quantitative Methods for Economics and Business

Campus Universitario de La Cartuja, Universidad de Granada. 18071 Granada (España)

<https://www.ugr.es/~romansg/web/index.html>

ORCID: 0000-0003-2589-4058

romansg@ugr.es

Catalina B. García-García

University of Granada

Department of Quantitative Methods for Economics and Business

Campus Universitario de La Cartuja, Universidad de Granada. 18071 Granada (España)

<https://metodoscuantitativos.ugr.es/informacion/directorio-personal/catalina-garcia-garcia>

ORCID: 0000-0003-1622-3877

cbgarcia@ugr.es

ASML: An R Package for Algorithm Selection with Machine Learning

by Ignacio Gómez-Casares, Beatriz Pateiro-López, Brais González-Rodríguez, and Julio González-Díaz

Abstract For extensively studied computational problems, it is commonly acknowledged that different instances may require different algorithms for optimal performance. The R package ASML focuses on the task of efficiently selecting from a given portfolio of algorithms, the most suitable one for each specific problem instance, based on significant instance features. The package allows for the use of the machine learning tools available in the R package caret and additionally offers visualization tools and summaries of results that make it easier to interpret how algorithm selection techniques perform, helping users better understand and assess their behavior and performance improvements.

1 Introduction

Selecting from a set of algorithms the most appropriate one for solving a given problem instance (understood as an individual problem case with its own specific characteristics) is a common issue that comes up in many different situations, such as in combinatorial search problems (Kotthoff, 2016; Drake et al., 2020), planning and scheduling problems (Speck et al., 2021; Messelis and De Causmaecker, 2014), or in machine learning (ML), where the multitude of available techniques often makes it challenging to determine the best approach for a particular dataset (Vanschoren, 2019). For an extensive survey on automated algorithm selection and application areas, we refer to Kerschke et al. (2019).

Figure 1 presents a general scheme, adapted from Figure 1 in Kerschke et al. (2019), illustrating the use of ML for algorithm selection. A set of problem instances is given, each described by associated features, together with a portfolio of algorithms that have been evaluated on all instances. The instance features and performance results are then fed into a ML framework, which is trained to produce a selector capable of predicting the best-performing algorithm for an unseen instance. Note that we are restricting attention to *offline* algorithm selection, in which the selector is constructed using a training set of instances and then applied to new problem instances.

Algorithm selection tools also demonstrate significant potential in the field of optimization, enhancing performance at solving problems where multiple solving strategies are often available. For example, a key factor in the efficiency of state-of-the-art global solvers in mixed integer linear programming and also in nonlinear optimization is the design of branch-and-bound algorithms and, in particular, of their branching rules. There isn't a single branching rule that outperforms all others on every problem instance. Instead, different branching rules exhibit optimal performance on different types of problem instances. Developing methods for the automatic selection of branching rules based on instance features has proven to be an effective strategy toward solving optimization problems more efficiently (Lodi and Zarpellon, 2017; Bengio et al., 2021; Ghaddar et al., 2023).

In algorithm selection, not only do the problem domain to which it applies and the algorithms for addressing problem instances play a crucial role, but also the metrics used to assess algorithm effectiveness —referred to in this work as Key Performance Indicators (KPIs). KPIs are used in different fields to assess and measure the performance of specific objectives or goals. In a business context, these indicators are quantifiable metrics that provide valuable insights into how well an individual, team, or entire organization is progressing towards achieving its defined targets. In the context of algorithms, KPIs serve as quantifiable measures used to evaluate the effectiveness and efficiency of algorithmic processes. For instance, in the realm of computer science and data analysis, KPIs can include measures like execution time, accuracy, and scalability. Monitoring these KPIs allows for a

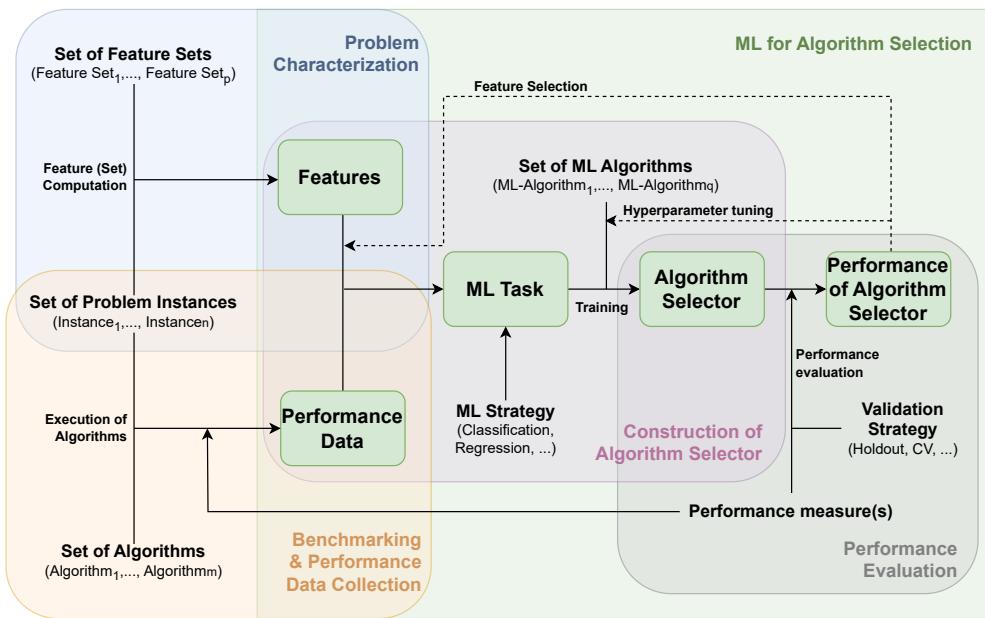


Figure 1: Schematic overview of the interplay between problem instance features (top left), algorithm performance data (bottom left), selector construction (center), and the assessment of selector performance (bottom right). Adapted from Kerschke et al. (2019).

comprehensive assessment of algorithmic performance, aiding in the selection of the most appropriate algorithm for a given instance and facilitating continuous improvement in algorithmic design and implementation.

Additionally, in many applications, normalizing the KPI to a standardized range like $[0, 1]$ provides a more meaningful basis for comparison. The KPI obtained through this process, which we will refer to as instance-normalized KPI, reflects the performance of each algorithm relative to the best-performing one for each specific instance. For example, if we have multiple algorithms and we are measuring execution time that can vary across instances, normalizing the execution time for each instance relative to the fastest algorithm within that same instance allows for a fairer evaluation. This is particularly important when the values of execution time might not directly reflect the relative performance of the algorithms due to wide variations in the scale of the measurements. Thus, normalizing puts all algorithms on an equal footing, allowing a clearer assessment of their relative efficiency.

Following the general framework illustrated in Figure 1, the R package **ASML** (González-Rodríguez et al., 2025b) provides a wrapper for ML methods to select from a portfolio of algorithms based on the value of a given KPI. It uses a set of features in a training set to learn a regression model for the instance-normalized KPI value for each algorithm. Then, the instance-normalized KPI is predicted for unseen test instances, and the algorithm with the best predicted value is chosen. As learning techniques for algorithm selection, the user can invoke any regression method from the **caret** package (Kuhn and Max, 2008) or use a custom function defined by the user. This makes our package flexible, as it automatically supports new methods when they are added to **caret**. Although initially designed for selecting branching rules in nonlinear optimization problems, its versatility allows the package to effectively address algorithm selection challenges across a wide range of domains. It can be applied to a broad spectrum of disciplines whenever there is a diverse set of instances within a specific problem domain, a suite of algorithms with varying behaviors across instances, clearly defined metrics for evaluating the performance of the available algorithms, and known features or characteristics of the instances that can be computed and are ideally correlated with algorithm performance. The visualization tools implemented in the package allow for an effective evaluation of the performance of the algorithm selection techniques. A

key distinguishing element of **ASML** is its learning-phase approach, which uses instance-normalized KPI values and trains a separate regression model for each algorithm to predict its normalized KPI on unseen instances.

2 Background

The algorithm selection problem was first outlined in the seminal work by Rice (1976). In simple terms, for a given set of problem instances (problem space) and a set of algorithms (algorithm space), the goal is to determine a selection model that maps each problem instance to the most suitable algorithm for it. By *most suitable*, we mean the best according to a specific metric that associates each combination of instance and algorithm with its respective performance. Formally, let \mathcal{P} denote the problem space or set of problem instances. The algorithm space or set of algorithms is denoted by \mathcal{A} . The metric $p : \mathcal{P} \times \mathcal{A} \rightarrow \mathbb{R}^n$ measures the performance $p(x, A)$ of any algorithm $A \in \mathcal{A}$ on instance $x \in \mathcal{P}$. The goal is to construct a selector $S : \mathcal{P} \rightarrow \mathcal{A}$ that maps any problem instance $x \in \mathcal{P}$ to an algorithm $S(x) = A \in \mathcal{A}$ in such a way that its performance is optimal.

As discussed in the Introduction, many algorithm selection methods in the literature use ML tools to model the relationship between problem instances and algorithm performance, using features derived from these instances. The pivotal step in this process is defining appropriate features that can be readily computed and are likely to impact algorithm performance. That is, given $x \in \mathcal{P}$, we make use of informative features $f(x) = (f_1(x), \dots, f_k(x)) \in \mathbb{R}^k$. In this framework, the selector S maps the simpler feature space \mathbb{R}^k into the algorithm space \mathcal{A} . A scheme of the algorithm selection problem, as described in Rice (1976), is shown in Figure 2.

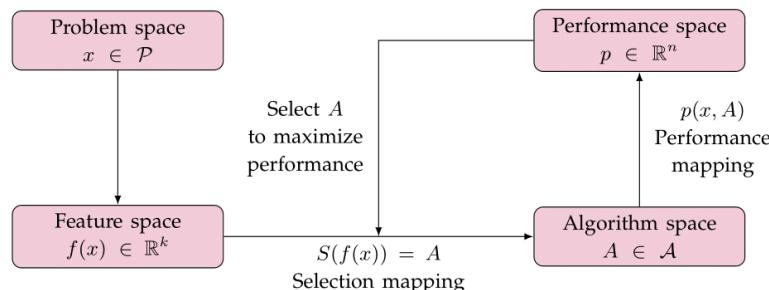


Figure 2: Scheme of the algorithm selection problem by Rice (1976).

For the practical derivation of the selection model S , we use training data consisting of features $f(x)$ and performances $p(x, A)$, where $x \in \mathcal{P}' \subset \mathcal{P}$ and $A \in \mathcal{A}$. The task is to learn the selector S based on the training data. The model allows us to forecast the performance on unobserved instance problems based on their features and subsequently select the algorithm with the highest predicted performance. A comprehensive discussion of various aspects of algorithm selection techniques can be found in Kotthoff (2016) and Pulatov et al. (2022).

3 Algorithm selection tools in R

The task of algorithm selection has seen significant advancements in recent years, with R packages facilitating this process. Here we present some of the existing tools that offer a range of functionalities, including flexible model-building frameworks, automated workflows, and standardized scenario formats, providing valuable resources for both researchers and end-users in algorithm selection.

The **llama** package (Kotthoff et al., 2021) provides a flexible implementation within R for evaluating algorithm portfolios. It simplifies the task of building predictive models to solve algorithm selection scenarios, allowing users to apply ML models effectively. In **llama**, ML algorithms are defined using the **mlr** package (Bischl et al., 2016b), offering a structured approach to model selection. On the other hand, the Algorithm Selection Library (ASlib) (Bischl et al., 2016a) proposes a standardized format for representing algorithm selection scenarios and introduces a repository that hosts an expanding collection of datasets from the literature. It serves as a benchmark for evaluating algorithm selection techniques under consistent conditions. It is accessible to R users through the **aslib** package. This integration simplifies the process for those working within the R environment. Furthermore, **aslib** interfaces with the **llama** package, facilitating the analysis of algorithm selection techniques within the benchmark scenarios it provides.

Our **ASML** package offers an approach to algorithm selection based on the powerful and flexible **caret** framework. By using **caret**'s ability to work with many different ML models, along with its model tuning and validation tools, **ASML** makes the selection process easy and effective, especially for users already familiar with **caret**. Thus, while **ASML** shares some conceptual similarities with **llama**, it distinguishes itself through its interface to the ML models in **caret** instead of **mlr**, which is currently considered retired by the mlr-org team, potentially leading to compatibility issues with certain learners, and has been succeeded by the next-generation **mlr3** (Lang et al., 2019). In addition, **ASML** automates the normalization of KPIs based on the best-performing algorithm for each instance, addressing the challenges that arise when performance metrics vary significantly across instances. **ASML** further provides new visualization tools that can be useful for understanding the results of the learning process. A comparative overview of the main features and differences between these packages can be seen in Table 1.

Table 1: Comparative overview of ASML and llama for algorithm selection.

Aspect	ASML	llama
Input data	features KPIs split by families supported	features KPIs feature costs supported
Normalized KPIs	✓	✗
ML backend	caret	mlr
Hyperparameter tuning	ASML::AStrain() supports arguments passed to caret (trainControl(), tuneGrid)	llama::cvFolds llama::tuneModel
Parallelization	✓ with snow	✓ with parallelMap
Results summary	Per algorithm Best overall and per instance ML-selected	Virtual best and single best per instance Aggregated scores (PAR, count, successes)
Visualization	Boxplots (per algorithm and ML-selected) Ranking plots Barplots (best vs ML-selected)	Scatter plots comparing two algorithm selectors
Model interpretability tools	✓ with DALEX	✗
ASlib integration	basic support	extended support
Latest release	CRAN 1.1.0 (2025)	CRAN 0.10.1 (2021)

There are also automated approaches that streamline the process of selecting and optimizing ML models within the R environment. Tools like **h2o** provide robust functionalities specifically designed for R users, facilitating an end-to-end ML workflow. These frameworks automate various tasks, including algorithm selection, hyperparameter optimization, and feature engineering, thereby simplifying the process for users of all skill levels. By integrating these automated solutions into R, users can efficiently explore a wide range of models

and tuning options without needing extensive domain knowledge or manual intervention. This automation not only accelerates the model development process but also improves the overall performance of ML projects by allowing a systematic evaluation of different approaches and configurations. However, while `h2o` excels at automating the selection of ML models and hyperparameter tuning, it does not perform algorithm selection based on instance-specific features, which is the primary focus of our approach. Instead, it evaluates multiple algorithms in parallel and selects the best-performing one based on predetermined metrics.

4 Using the ASML package

Here, we illustrate the usage of the `ASML` package with an example within the context of algorithm selection for spatial branching in polynomial optimization, aligning with the problem discussed in Ghaddar et al. (2023) and further explored in González-Rodríguez et al. (2025a). Table 2 provides an overview of the problem and a summary of the components that we will discuss in detail below.

Table 2: Summary of the branching rule selection problem.

Algorithms	max, sum, dual, range, eig-VI, eig-CMI
KPI	pace
Number of instances	407
Number of instances per library	180 (DS), 164 (MINLPLib), 63 (QPLIB)
Number of features	33

A well-known approach for finding global optima in polynomial optimization problems is based on the use of the Reformulation Linearization Technique (RLT) (Sherali and Tuncbilek, 1992). Without delving into intricate details, RLT operates by creating a linear relaxation of the original polynomial problem, which is then integrated into a branch-and-bound framework. The branching process involves assigning a score to each variable, based on the violations of the RLT identities it participates in, after solving the corresponding relaxation at each node. Subsequently, the variable with the highest score is selected for branching. The computation of these scores is a critical aspect and allows for various approaches, leading to distinct branching rules that constitute our algorithm selection portfolio. Specifically, in our example, we will examine six distinct branching rules (referred to interchangeably as branching rules or algorithms), labeled as max, sum, dual, range, eig-VI, and eig-CMI rules. For the definitions and a comprehensive understanding of the rationale behind these rules, refer to Ghaddar et al. (2023).

Measuring the performance of different algorithms in the context of optimization is crucial for evaluating their effectiveness and efficiency. Two common metrics for this evaluation are running time and optimality gap, measured as a function of the lower and upper bounds for the objective function value at the end of the algorithm (a small optimality gap indicates that the algorithm is producing solutions close to the optimal). Both metrics are important and are often considered together to evaluate algorithm performance. For instance, it is meaningful to consider the time required to reduce the optimality gap by one unit as KPI. In our example, and to ensure it is well-defined, we make use of a slightly different metric, which we refer to as pace, defined as the time required to increase the lower bound by one unit. For the pace, a smaller value is preferred, as it indicates better performance.

As depicted in Figure 2, a crucial aspect of the methodology involves selecting input variables (features) that facilitate the prediction of the KPI for each branching rule. We consider 33 features representing global information of the polynomial optimization problems, such as relevant characteristics of variables, constraints, monomials, coefficients, or other attributes. A detailed description of the considered features can be found in Table 3. Although we won't delve into these aspects, determining appropriate features is often complex, and using feature-selection methods can be beneficial for choosing the most relevant ones.

Table 3: Features from the branching dataset.

Index	Description
3	Number of variables
4	Number of constraints
5	Degree
6	Number of monomials
7	Density
8	Density of VIG
9	Modularity of VIG
10	Treewidth of VIG
11	Density of CMIG
12	Modularity of CMIG
13	Treewidth of CMIG
14	Pct. of variables not present in any monomial with degree greater than one
15	Pct. of variables not present in any monomial with degree greater than two
16	Number of variables divided by number of constraints
17	Number of variables divided by degree
18	Pct. of equality constraints
19	Pct. of linear constraints
20	Pct. of quadratic constraints
21	Number of monomials divided by number of constraints
22	Number of RLT variables divided by number of constraints
23	Pct. of linear monomials
24	Pct. of quadratic monomials
25	Pct. of linear RLT variables
26	Pct. of quadratic RLT variables
27	Variance of the ranges of the variables
28	Variance of the coefficients
29	Variance of the density of the variables
30	Variance of the no. of appearances of each variable
31	Average of the ranges of the variables
32	Average of the coefficients
33	Average pct. of monomials in each constraint and in the objective function
34	Average of the no. of appearances of each variable
35	Median of the ranges of the variables

Note:

Index refers to columns of branching\$x.

To assess the performance of the algorithm selection methods in this context, we have a diverse set of 407 instances from different optimization problems, taken from three well-known benchmarks (Bussieck et al., 2003; Dalkiran and Sherali, 2016; Furini et al., 2018), corresponding respectively to the MINLPLib, DS, and QPLIB libraries. Details are given in Table 2. The data for this analysis is contained within the branching dataset included in the package. We begin by defining two data frames. The features data frame includes two initial columns that provide the instance names and the corresponding family (library in our example) for each instance. The remaining columns consist of the features listed in Table 3.

We also define the KPI data frame, which is derived from `branching$y`. This data frame contains the pace values for each of the six branching rules considered in this study (specified by the labels in the `lab_rules` vector). These data frames will serve as the input for our subsequent analyses.

```
set.seed(1234)
library(ASML)
data(branching)
features <- branching$x
KPI <- branching$y
lab_rules <- c("max", "sum", "dual", "range", "eig-VI", "eig-CMI")
```

4.1 Pre-Processing the data

As with any analysis, the first step involves preprocessing the data. This includes using the function `partition_and_normalize`, which not only divides the dataset into training and test sets but also normalizes the KPI relative to the best result for each instance. The argument `better_smaller` specifies whether a lower KPI value is preferred (such as in our case where the KPI represents pace, with smaller values indicating better performance) or if a higher value is desired, when larger KPI values are considered more advantageous.

```
data <- partition_and_normalize(features, KPI, family_column = 1, split_by_family = TRUE,
                                 better_smaller = TRUE)
names(data)

#> [1] "x.train"           "y.train"           "y.train.original" "x.test"
#> [5] "y.test"            "y.test.original"   "families.train"    "families.test"
#> [9] "better_smaller"
```

When using the function `partition_and_normalize` the resulting object is of class `as_data` and contains several key components essential for our study. Specifically, the object includes `x.train` and `x.test`, representing the feature sets for the training and test datasets, respectively. Additionally, it contains `y.train` and `y.test`, with the instance-normalized KPI corresponding to each dataset, along with their original counterparts, `y.train.original` and `y.test.original`. This structure allows us to retain the original KPI values while working with the instance-normalized data. Furthermore, when the parameter `split_by_family` is set to `TRUE`, as in the example, the object also includes `families.train` and `families.test`, indicating the family affiliation for each observation within the training and test sets. Figure 3 illustrates how the split preserves the proportions of instances for each library.

As a tool for visualizing the performance of the considered algorithms, the `boxplots` function operates on objects of class `as_data` and generates boxplots for the instance-normalized KPI. This visualization facilitates the comparison of performance differences across instances. The function can be applied to both training and test observations and can also group the results by family. Additionally, it accepts common arguments typically used in R functions. Figure 4 shows the instance-normalized KPI of the instances in the train set. What becomes evident from the boxplots is that there is no branching rule that outperforms the others

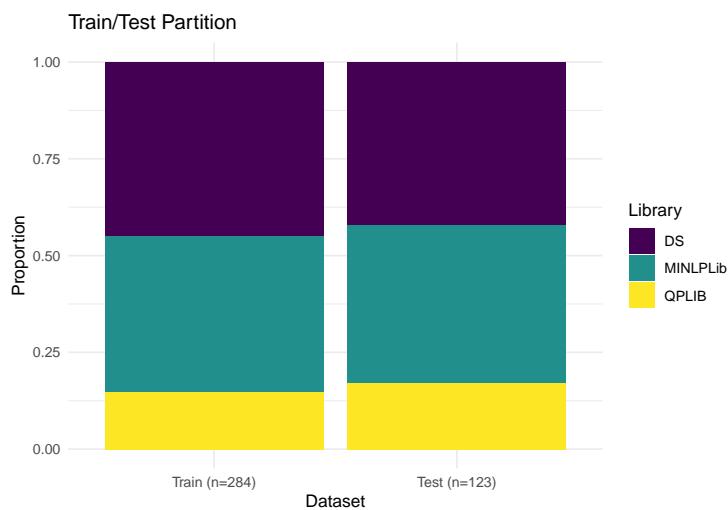


Figure 3: Train/Test partition preserving the percentage of instances for each library.

across all instances, and making a wrong choice of criteria in certain problems can lead to very poor performance.

```
boxplots(data, test = FALSE, by_families = FALSE, labels = lab_rules)
```

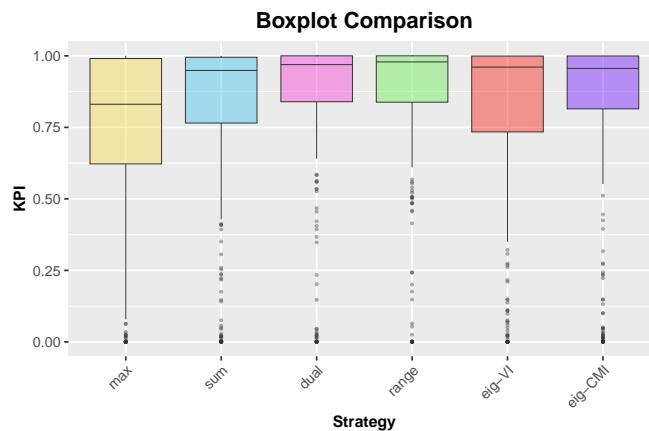


Figure 4: Boxplots of instance-normalized KPI for each algorithm across instances in the train set.

The ranking function, specifically designed for the [ASML](#) package, is also valuable for visualizing the differing behaviors of the algorithms under investigation, depending on the analyzed instances. After ranking the algorithms for each instance, based on the instance-normalized KPI, the function generates a bar chart for each algorithm, indicating the percentage of times it occupies each ranking position. The numbers displayed within the bars represent the mean value of the instance-normalized KPI for the problems associated with that specific ranking position. Again, the representation can be made both for the training and test sets, as well as by family. In Figure 5, we present the chart corresponding to the training sample and categorized by family. In particular, it is observed that certain rules, when not the best choice for a given instance, can perform quite poorly in terms of instance-normalized KPI (see, for example, the results on the MINLPLib library). This highlights the importance of not only selecting the best algorithm for each instance but also ensuring that the chosen algorithm does not perform too poorly when it isn't optimal. In some cases, even if an algorithm isn't the best-performing option, it may still provide reasonably good results, whereas a wrong choice can result in significantly worse outcomes.

```
ranking(data, test = FALSE, by_families = TRUE, labels = lab_rules)
```

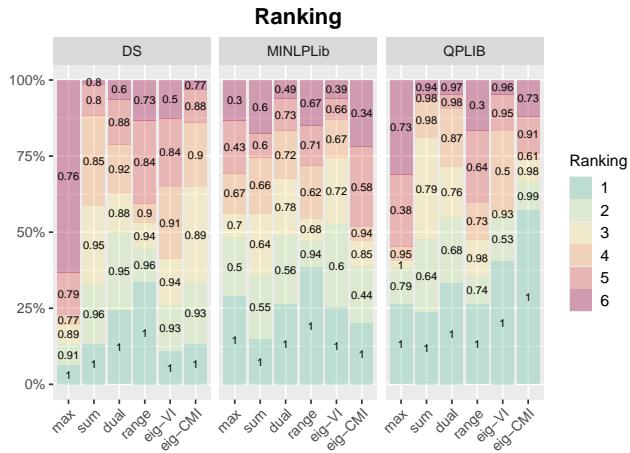


Figure 5: Ranking of algorithms based on the instance-normalized KPI for the training sample, categorized by family. The bars represent the percentage of times each algorithm appeared in different ranking positions, with the numbers indicating the mean value of the KPI.

Additionally, functions from the `caret` package can be applied if further operations on the predictors are needed. Here we show an example where the Yeo-Johnson transformation is applied to the training set, and the same transformation is subsequently applied to the test set to ensure consistency across both datasets. The flexibility of `caret` also allows for the inclusion of advanced techniques, such as feature selection and dimensionality reduction, to improve the quality of the algorithm selection process.

```
preProcValues <- caret::preProcess(data$x.train, method = "YeoJohnson")
data$x.train <- predict(preProcValues, data$x.train)
data$x.test <- predict(preProcValues, data$x.test)
```

4.2 Training models and predicting the performance of the algorithms

The approach in **ASML** to algorithm selection is based on building regression models that predict the instance-normalized KPI of each considered algorithm. To this end, users can take advantage of the wide range of ML models available in the `caret` package, which provides a unified interface for training and tuning various types of models. Models trained with `caret` can be seamlessly integrated into the **ASML** workflow using the `AStrain` function from **ASML**, as shown in the next example. Just for illustrative purposes, we use quantile random forest (Meinshausen, 2006) to model the behavior of the instance-normalized KPI based on the features. This is done with the `qrf` method in the `caret` package, which relies on the `quantregForest` package (Meinshausen, 2024).

```
library(quantregForest)
tune_grid <- expand.grid(mtry = 10)
training <- AStrain(data, method = "qrf", tuneGrid = tune_grid)
```

Additional arguments for `caret::train` can also be passed directly to `ASML::AStrain`. This allows users to take advantage of the flexibility of the `caret` package, including specifying control methods (such as cross-validation), tuning parameters, or any other relevant settings provided by `caret::train`. This integration ensures that the **ASML** workflow can fully make use of the modeling capabilities offered by `caret`. To make the execution faster (it is not our intention here to delve into the choice of the best model), we use a `tune_grid` that

sets a fixed value for `mtry`. This avoids the need for an exhaustive search for this hyperparameter, speeding up the model training process. Other modeling approaches should also be considered, as they may offer better performance depending on the specific characteristics of the data and the problem at hand. For more computationally intensive models or larger datasets, the `ASML::AStrain` function includes the argument `parallel`, which can be set to `TRUE` to enable parallel execution using the `snow` package (Tierney et al., 2021). This allows the training step to be distributed across multiple cores, reducing computation time. A detailed example on a larger dataset is provided in the following section, showing the scalability of the workflow and the effect of parallelization on training time.

The function `caret::train` returns a trained model along with performance metrics, predictions, and tuning parameters, providing insights into the model's effectiveness. In a similar manner, `ASML::AStrain` offers the same type of output but for each algorithm under consideration, allowing straightforward comparison within the `ASML` framework. The `ASML::ASpredict` function generates the predictions for new data by using the models created during the training phase for each algorithm under evaluation. Thus, predictions for the algorithms are obtained simultaneously, facilitating a direct comparison of their performance. By using `ASML::ASpredict` as follows, we obtain a matrix where each row corresponds to an instance from the test set, and each column represents the predicted instance-normalized KPIs for the six branching rules using the `qrf` method.

```
predict_test <- ASpredict(training, newdata = data$x.test)
```

4.3 Evaluating and visualizing the results

One of the key strengths of the `ASML` package lies in its ability to evaluate results collectively and provide intuitive visualizations. This approach not only aids in identifying the most effective algorithms but also contributes to the interpretability of the results, making it easier for users to make informed decisions based on the performance metrics and visual representations provided. For example, the function `KPI_table` returns a table showing the arithmetic and geometric mean of the KPI (both instance-normalized and not normalized) obtained on the test set for each algorithm, as well as for the algorithm selected by the learning model (the one with the largest instance-normalized predicted KPI for each instance). In Table 4, the results for our case study are shown. It is important to note that larger values are better in the columns for the arithmetic and geometric mean of the instance-normalized KPI (where values close to 1 indicate the best performance). Conversely, in the columns for non-normalized values, lower numbers reflect better outcomes. In all cases, the best results are obtained for the ML algorithm. Note also that in this case, the differences in the performance of the algorithms are likely better reflected by the geometric mean because it gives a better representation of relative differences.

```
KPI_table(data, predictions = predict_test)
```

Table 4: Arithmetic and geometric mean of the KPI (both instance-normalized and non-normalized) for each algorithm on the test set, along with the results for the algorithm selected by the learning model (first row).

	Arith. mean inst-norm KPI	Geom. mean inst-norm KPI	Arith. mean non-norm KPI	Geom. mean non-norm KPI
ML	0.911	0.887	88114.19	1.035
max	0.719	0.367	158716.13	2.574
sum	0.791	0.537	104402.53	1.780
dual	0.842	0.581	104393.92	1.634
range	0.879	0.644	107064.29	1.432
eig-VI	0.781	0.474	131194.49	2.007
eig-CMI	0.800	0.591	88197.74	1.616

Additionally, the function `KPI_summary_table` generates a concise comparative table displaying values for three different choices: single best, ML, and optimal, see Table 5. The single best choice refers to selecting the same algorithm for all instances based on the lowest geometric mean of the non-normalized KPI (in this case the range rule). This approach evaluates the performance of each algorithm across all instances and chooses the one that consistently performs best overall, rather than optimizing for individual instances. The ML choice represents the algorithm selected by the quantile random forest model. The optimal choice corresponds to solving each instance with the algorithm that performs best for that specific instance. The ML choice shows promising results, with a mean KPI close to the optimal choice, demonstrating its capability to select algorithms that yield competitive performance.

```
KPI_summary_table(data, predictions = predict_test)
```

Table 5: Arithmetic and geometric mean of the non-normalized KPI for single best choice, ML choice, and optimal choice.

	Arith. mean non-norm KPI	Geom. mean non-norm KPI
single best	107064.29	1.432
ML	88114.19	1.035
optimal	88085.37	0.911

The following code generates several visualizations that help us compare how well the algorithms perform according to the response variable (instance-normalized KPI) and also illustrate the behavior of the learning process. These plots give us good insights into how effective the algorithm selection process is and how it behaves in comparison to using the same branching rule for all instances. Figure 6 shows the boxplots comparing the performance of each algorithm in terms of the instance-normalized KPI, including the instance-normalized KPI of the rules selected by the ML process for the test set. In Figure 7, the performance is presented by family, allowing for a more detailed comparison across the different sets of instances. In Figure 8, we show the ranking of algorithms based on the instance-normalized KPI for the test sample, including the ML rule, categorized by family. Finally, in Figure 9, the right-side bar in the stacked bar plot (optimal) illustrates the proportion of instances in which each of the original rules is identified as the best-performing option. In contrast, the left-side bar (ML) depicts the frequency with which ML selects each rule as the top choice. Although the rule chosen by ML in each instance doesn't always match the best one for that case, ML tends to select the different rules in a similar proportion to how often those rules are the best across the test set. This means it does not consistently favor a particular rule or ignore any that are the best a significant percentage of instances.

```
boxplots(data, predictions = predict_test, labels = c(lab_rules, "ML"))
boxplots(data, predictions = predict_test, labels = c(lab_rules, "ML"), by_families = TRUE)
ranking(data, predictions = predict_test, labels = c("ML", lab_rules), by_families = TRUE)
figure_comparison(data, predictions = predict_test, by_families = FALSE, labels = lab_rules)
```

4.4 Custom user-defined methods

While `caret` provides a range of built-in methods for model training and prediction, there may be situations where researchers want to explore additional methods not directly integrated into the package. Considering alternative methods can improve the analysis and provide greater flexibility in modeling choices.

In this section, we present an example of how to modify the quantile random forest `qrf` method. The `qrf` implementation in `caret` does not allow users to specify the conditional quantile to predict, which is set to the median by default. In this case, rather than creating

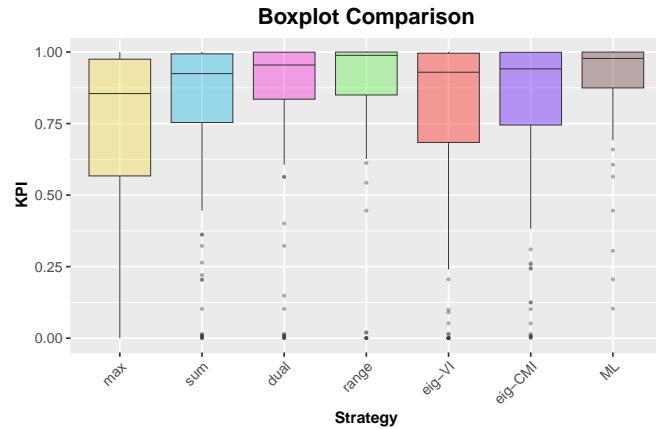


Figure 6: Boxplots of instance-normalized KPI for each algorithm, including the ML algorithm, across instances in the test set.

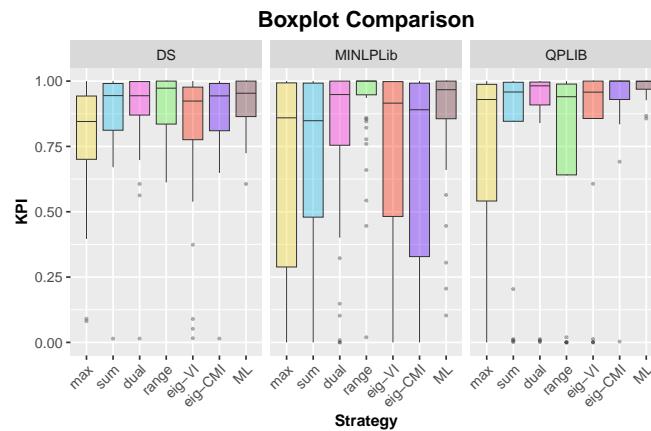


Figure 7: Boxplots of instance-normalized KPI for each algorithm, including the ML algorithm, across instances in the test set, categorized by family.

an entirely new method, we only need to adjust the prediction function to include the `what` argument, allowing us to specify the desired conditional quantile for prediction. In this execution example, we base the algorithm selection method on the predictions of the α -conditional quantile of the instance-normalized KPI for $\alpha = 0.25$.

```
qrf_q_predict <- function(modelFit, newdata, what = 0.5, submodels = NULL) {
  out <- predict(modelFit$finalModel, newdata, what = what)
  if (is.matrix(out)) {
    out <- out[, 1]
  }
  out
}

predict_test_Q1 <- ASpredict(training, newdata = data$x.test, f = "qrf_q_predict",
                               what = 0.25)
KPI_summary_table(data, predictions = predict_test_Q1)
```

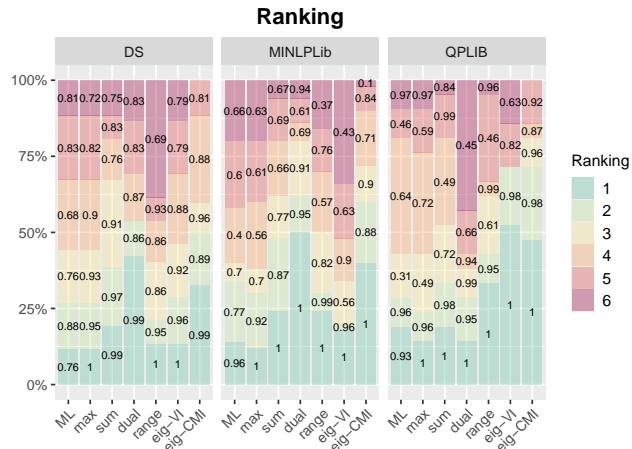


Figure 8: Ranking of algorithms, including the ML algorithm, based on the instance-normalized KPI for the test sample, categorized by family. The bars represent the percentage of times each algorithm appeared in different ranking positions, with the numbers indicating the mean value of the normalized KPI.

4.5 Model interpretability

Predictive modeling often relies on flexible but complex methods. These methods typically involve many parameters or hyperparameters, which can make the models difficult to interpret. To address this, interpretable ML techniques provide tools for exploring *black-box* models. **ASML** integrates seamlessly with the package **DALEX** (moDel Agnostic Language for Exploration and eXplanation), see [Biecek \(2018\)](#). With **DALEX**, users can obtain model performance metrics, evaluate feature importance, and generate partial dependence plots (PDPs), among other analyses.

To simplify the use of **DALEX** within our framework, **ASML** provides the function `ASexplainer`. This function automatically creates **DALEX** explainers for the models trained with **Astrain** (one for each algorithm in the portfolio). Once the explainers are created, users can easily apply **DALEX** functions to explore and compare the behavior of each model. The following example shows how to obtain a plot of the reversed empirical cumulative distribution function of the absolute residuals, from the performance metrics computed with `DALEX::model_performance`, see Figure 10.

```
# Create DALEX explainers for each trained model
explainers_qrf <- ASexplainer(training, data = data$x.test, y = data$y.test, labels = lab_rules)
# Compute model performance metrics for each explainer
mp_qrf <- lapply(explainers_qrf, DALEX::model_performance)
# Plot the performance metrics
do.call(plot, unname(mp_qrf))
```

The code below illustrates how to obtain feature importance (via `DALEX::model_parts`) and a PDP for the predictor variable degree (via `DALEX::model_profile`). Plots are not displayed in this manuscript, but they can be generated by executing the code.

```
# Compute feature importance for each model in the explainers list
vi_qrf <- lapply(explainers_qrf, DALEX::model_parts)
# Plot the top 5 most important variables for each model
do.call(plot, c(unname(vi_qrf), list(max_vars = 5)))
# Compute PDP for the variable 'degree' for each model
pdp_qrf <- lapply(explainers_qrf, DALEX::model_profile, variable = "degree", type = "partial")
# Plot the PDPs generated
do.call(plot, unname(pdp_qrf))
```

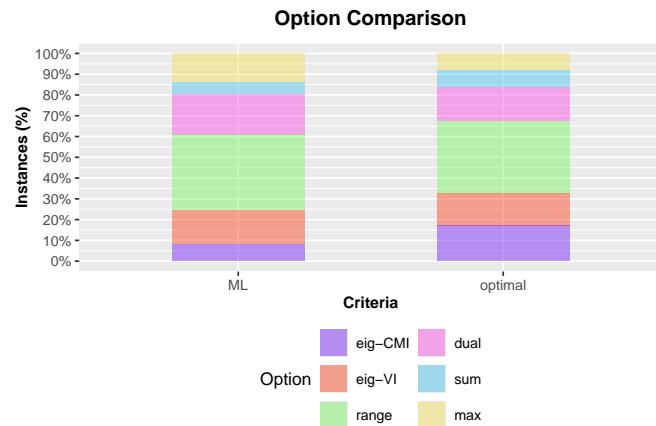


Figure 9: Comparison of the best-performing rules: The right stack shows the proportion of times each of the original rules is identified as the best-performing option, while the left stack presents the frequency of selection by ML.

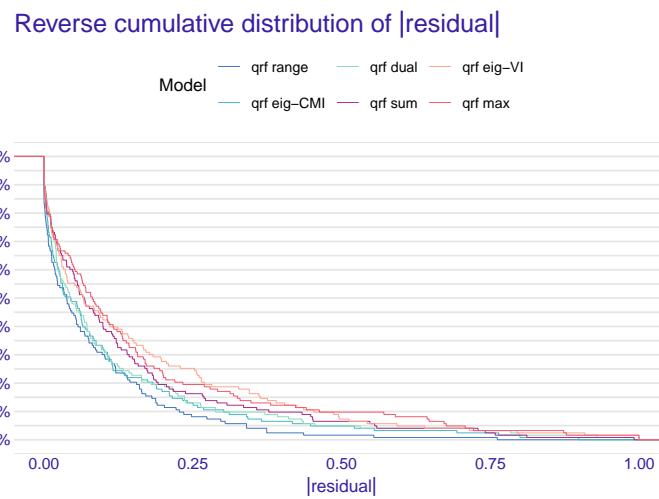


Figure 10: Reversed empirical cumulative distribution function of the absolute residuals of the trained models.

5 Example on a larger dataset

To analyze the scalability of [ASML](#), we now consider an example of algorithm selection in the field of high-performance computing (HPC), specifically in the context of the automatic selection of the most suitable storage format for sparse matrices on GPUs. This is a well-known problem in HPC, since the storage format has a decisive impact on the performance of many scientific kernels such as the sparse matrix–vector multiplication (SpMV). For this study, we use the dataset introduced by [Pichel and Pateiro-López \(2018\)](#), which contains 8111 sparse matrices and is available in the [ASML](#) package under the name SpMVformat. Each matrix is described by a set of nine structural features, and the performance of the single-precision SpMV kernel was measured on a NVIDIA GeForce GTX TITAN GPU, under three storage formats: compressed row storage (CSR), ELLPACK (ELL), and hybrid (HYB). For each matrix and format, performance is expressed as the average GFLOPS (billions of floating-point operations per second), over 1000 SpMV operations. This setup allows us to study how matrix features relate to the most efficient storage format.

The workflow follows the standard **ASML** pipeline: data are partitioned and normalized, preprocessed, and models are trained using **ASML::AStrain**. We considered different learning methods available in **caret** and evaluated execution times both with and without parallel processing, which is controlled via the `parallel` argument in **ASML::AStrain**. The selected methods were run with their default configurations in **caret**, without additional hyperparameter tuning. All experiments were performed on a machine equipped with a 12th Gen Intel(R) Core(TM) i7-12700 (12 cores), 2.11 GHz processor and 32 GB of RAM. The execution times are summarized in Tables 6 and 7.

Table 6: Execution times (in seconds) on the SpMVformat dataset for the main preprocessing stages.

Stage	Execution time (seconds)
<code>ASML::partition_and_normalize</code>	0.03
<code>caret::preProcess</code>	1.55

Table 7: Training times (in seconds) on the SpMVformat dataset for different methods using **ASML::AStrain**. The first column shows execution without parallelization (`parallel = FALSE`) and the second column shows execution with parallelization (`parallel = TRUE`).

Method	Execution times (in seconds) of ASML::AStrain	
	<code>parallel = FALSE</code>	<code>parallel = TRUE</code>
<code>nnet</code>	236.58	50.75
<code>svmRadial</code>	881.03	263.60
<code>rf</code>	4753.00	1289.68

The majority of the computational cost is associated with model training, which depends on the learning method in **caret**. We observe that training times vary across methods: `nnet` (a simple feed-forward neural network), `svmRadial` (support vector machines with radial kernel), and `rf` (random forest). Parallel execution substantially reduces training times for all selected methods, demonstrating that the workflow scales efficiently to larger datasets while keeping preprocessing overhead minimal.

Apart from the execution times, we also take this opportunity to provide a brief commentary on the outcome of the algorithm selection in this application example. In particular, we illustrate the model's ability to identify the most efficient storage format by reporting the results obtained with the `nnet` method, see Figure 11. The trained model selects the best-performing format in more than 85% of the test cases, and even when it does not, the chosen format still achieves high performance, with mean value of the normalized KPI (normalized average GFLOPS) around 0.9.

```
set.seed(1234)
data(SpMVformat)
features <- SpMVformat$x
KPI <- SpMVformat$y
data <- partition_and_normalize(features, KPI, better_smaller = FALSE)
preProcValues <- caret::preProcess(data$x.train, method = "YeoJohnson")
data$x.train <- predict(preProcValues, data$x.train)
data$x.test <- predict(preProcValues, data$x.test)
training <- AStrain(data, method = "nnet", parallel = TRUE)
pred <- ASpredict(training, newdata = data$x.test)
ranking(data, predictions = pred)
```

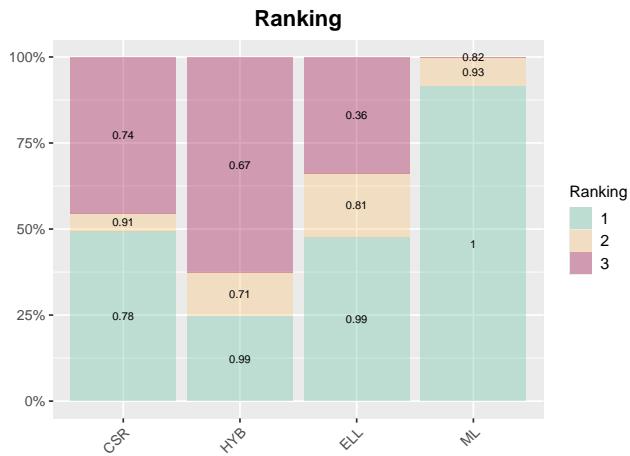


Figure 11: Ranking of storage formats, including the ML selected, based on the instance-normalized KPI for the test sample. The bars represent the percentage of times each storage format appeared in different ranking positions, with the numbers indicating the mean value of the normalized KPI.

6 Using ASML for algorithm selection on ASlib scenarios

While the primary purpose of the **ASML** package is not to systematically conduct algorithm selection studies like those found in ASlib (an area for which the **llama** toolkit is especially helpful), it does offer a complementary approach for reproducing results from the ASlib benchmark (<https://coseal.github.io/aslib-r/scenario-pages/index.html>). Our method allows for a comparative analysis using instance-normalized KPIs, which, as demonstrated in the following example, can sometimes yield improved performance results. Additionally, it can be useful for evaluating algorithm selection approaches based on methods that are not available in the **mlr** package used by **llama** but are accessible in **caret**.

6.1 Data download and preparation

First, we identify the specific scenario from ASlib we are interested in, in this case, CPMP-2015. Using the scenario name, we construct a URL that points to the corresponding page on the ASlib website. Then, we fetch the HTML content of the page and create a local directory to store the downloaded files¹.

```
set.seed(1234)
library(tidyverse)
library(rvest)
scen <- "CPMP-2015"
url <- paste0("https://coseal.github.io/aslib-r/scenario-pages/", scen, "/data_files")
page <- read_html(paste0(url, ".html"))
file_links <- page %>%
  html_nodes("a") %>%
  html_attr("href")

# Create directory for downloaded files
dir_data <- paste0(scen, "_data")
dir.create(dir_data, showWarnings = FALSE)
```

¹A more direct approach would be to use the `getCosealASScenario` function from the **aslib** package; however, this function seems to be currently not working, likely due to changes in the directory structure of the scenarios.

```
# Download files
for (clink in file_links) {
  full_link <- ifelse(grepl("^http", link), link, paste0(url, "/", link))
  file_name <- basename(link)
  dest_file <- file.path(dir_data, file_name)
  if (!is.na(full_link)) {
    download.file(full_link, dest_file, mode = "wb", quiet = TRUE)
  }
}
```

6.2 Data preparation with aslib

Now, we use the **aslib** package to parse the scenario data and extract the relevant features and performance metrics. The `parseASScenario` function from **aslib** creates a structured object `ASScen` that contains information regarding the algorithms and instances being evaluated. We then transform this data into cross-validation folds using the `cvFolds` function from **llama**. This conversion facilitates a fair evaluation of algorithm performance across different scenarios, allowing us to compare the results with those published².

```
library(aslib)
ASScen <- aslib::parseASScenario(dir_data)
llamaScen <- aslib::convertToLlama(ASScen)
folds <- llama::cvFolds(llamaScen)
```

Then we extract the key performance indicator (KPI) and features from the `folds` object. In this case, KPI refers to runtime. As described in the ASlib documentation, `KPI_pen` measures the penalized runtime. If an instance is solved within the timeout (`cutoff`) by the selected algorithm, the actual runtime is used. However, if a timeout occurs, the timeout value is multiplied by 10 to penalize the algorithm's performance. We also define `nins` as the number of instances and `ID` as unique identifiers for each instance.

```
KPI <- folds$data[, folds$performance]
features <- folds$data[, folds$features]
cutoff <- ASScen$desc$algorithm_cutoff_time
is.timeout <- ASScen$algo.runstatus[, -c(1, 2)] != "ok"
KPI_pen <- KPI * ifelse(is.timeout, 10, 1)
nins <- length(getInstanceNames(ASScen))
ID <- 1:nins
```

6.3 Quantile random forest using ASML on instance-normalized KPI

We use the **ASML** package to perform quantile random forest on instance-normalized KPI. We have already established the folds beforehand, and we want to use those partitions to maintain consistency with the original ASlib scenario design. Therefore, we provide `x.test` and `y.test` as arguments directly to the `partition_and_normalize` function.

```
data <- partition_and_normalize(x = features, y = KPI, x.test = features, y.test = KPI,
                                better_smaller = TRUE)
train_control <- caret::trainControl(index = folds$train, savePredictions = "final")
training <- AStrain(data, method = "qrf", trControl = train_control)
```

In this code block, we process the predictions made by the models trained using **ASML** and calculate the same performance metrics used in ASlib, namely, the percentage of solved instances (`succ`), penalized average runtime (`par10`), and misclassification penalty (`mcp`), as detailed in the ASlib documentation (Bischl et al., 2016a).

²Available at: <https://coseal.github.io/aslib-r/scenario-pages/CPMP-2015/llama.html> (Accessed October 25, 2024).

```

pred_list <- lapply(training, function(model) {
  model$pred %>%
    arrange(rowIndex) %>%
    pull(pred)
})

pred <- do.call(cbind, pred_list)
alg_sel <- apply(pred, 1, which.max)

succ <- mean(!is.timeout[cbind(ID, alg_sel)])
par10 <- mean(KPI_pen[cbind(ID, alg_sel)])
mcp <- mean(KPI[cbind(ID, alg_sel)]) - apply(KPI, 1, min))

```

In Table 8, we present the results. We observe that, in this example, using instance-normalized KPI along with the quantile random forest model offers an alternative modeling option in addition to the standard regression models employed in the original ASlib study (linear model, regression trees and regression random forest), resulting in improved performance outcomes.

Table 8: Performance results of various models on the CPMP-2015 dataset. The last row represents the performance of the quantile random forest model based on instance-normalized KPI using the **ASML** package. The preceding rows detail the results (all taken from original ASlib study) of the virtual best solver (vbs), single best solver (singleBest), and the considered regression methods (linear model, regression trees and regression random forest).

Model	succ	par10	mcp
baseline vbs	1.000	227.605	0.000
baseline singleBest	0.812	7002.907	688.774
regr.lm	0.843	5887.326	556.875
regr.rpart	0.843	5916.120	585.669
regr.randomForest	0.846	5748.065	540.574
ASML qrf	0.873	4807.633	460.863

It's important to note that this is merely an example of execution and there are other scenarios in ASlib where replication may not be feasible in the same manner, due to factors not considered in **ASML** (for more robust behavior across the ASlib benchmark, we refer to **llama**). Despite these limitations, **ASML** provides a flexible framework that allows researchers to explore various methodologies, including those not directly applicable with **llama** through **mlr**, and improve algorithm selection processes across different scenarios, ultimately contributing to improve understanding and performance in algorithm selection tasks.

7 Summary and discussion

In this work, we present **ASML**, an R package to select the best algorithm from a portfolio of candidates based on a chosen KPI. **ASML** uses instance-specific features and historical performance data to estimate, via a model selected by the user, including any regression method from the **caret** package or a custom function, how well each algorithm is likely to perform on new instances. This helps the automatic selection of the most suitable one. The use of instance-normalized KPIs for algorithm selection is a novel aspect of this package, allowing a unified comparison across different algorithms and problem instances.

While the motivation and examples presented in this work focus on optimization problems, particularly the automatic selection of branching rules and decision strategies in polynomial optimization, the **ASML** framework is inherently flexible and can be applied more broadly. In particular, in the context of ML, selecting the right algorithm is a crucial factor for the success of ML applications. Traditionally, this process involves empirically

assessing potential algorithms with the available data, which can be resource-intensive. In contrast, in the so-called Meta-Learning, the aim is to predict the performance of ML algorithms based on features of the learning problems (meta-examples). Each meta-example contains details about a previously solved learning problem, including features, and the performance achieved by the candidate algorithms on that problem. A common Meta-Learning approach involves using regression algorithms to forecast the value of a selected performance metric (such as classification error) for the candidate algorithms based on the problem features. This method is commonly referred to as Meta-Regression in the literature. Thus, **ASML** could also be used in this context, providing a flexible tool for algorithm selection across a variety of domains.

8 Acknowledgments

The authors would like to thank María Caseiro-Arias, Antonio Fariña-Elorza and Manuel Timiraos-López for their contributions to the development of the **ASML** package. This work is part of the R&D projects PID2024-158017NB-I00, PID2020-116587GB-I00 and PID2021-124030NB-C32 granted by MICIU/AEI/10.13039/501100011033. This research was also funded by Grupos de Referencia Competitiva ED431C-2021/24 and ED431C 2025/03 from the Consellería de Educación, Ciencia, Universidades e Formación Profesional, Xunta de Galicia. Brais González-Rodríguez acknowledges the support from MICIU, through grant BG23/00155.

References

- Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021. ISSN 0377-2217. doi: <https://doi.org/10.1016/j.ejor.2020.07.063>. URL <https://www.sciencedirect.com/science/article/pii/S0377221720306895>. [p216]
- P. Biecek. DALEX: Explainers for complex predictive models in R. *Journal of Machine Learning Research*, 19(84):1–5, 2018. URL <http://jmlr.org/papers/v19/18-416.html>. [p228]
- B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and J. Vanschoren. ASlib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016a. ISSN 0004-3702. doi: <https://doi.org/10.1016/j.artint.2016.04.003>. [p219, 232]
- B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. mlr: Machine learning in R. *Journal of Machine Learning Research*, 17(170):1–5, 2016b. URL <https://jmlr.org/papers/v17/15-066.html>. [p219]
- M. R. Bussieck, A. S. Drud, and A. Meeraus. MINLPLib-a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15:114–119, 2003. ISSN 1091-9856. doi: 10.1287/ijoc.15.1.114.15159. [p222]
- E. Dalkiran and H. D. Sherali. RLT-POS: Reformulation-linearization technique-based optimization software for solving polynomial programming problems. *Mathematical Programming Computation*, 8:337–375, 2016. ISSN 1867-2949. doi: 10.1007/s12532-016-0099-5. [p222]
- J. H. Drake, A. Kheiri, E. Özcan, and E. K. Burke. Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, 285(2):405–428, 2020. ISSN 0377-2217. doi: <https://doi.org/10.1016/j.ejor.2019.07.073>. URL <https://www.sciencedirect.com/science/article/pii/S0377221719306526>. [p216]
- F. Furini, E. Traversi, P. Belotti, A. Frangioni, A. Gleixner, N. Gould, L. Liberti, A. Lodi, R. Misener, H. Mittelmann, N. Sahinidis, S. Vigerske, and A. Wiegele. QPLIB: a library

- of quadratic programming instances. *Mathematical Programming Computation*, 1:237–265, 2018. ISSN 1867-2957. doi: 10.1007/s12532-018-0147-4. [p222]
- B. Ghaddar, I. Gómez-Casares, J. González-Díaz, B. González-Rodríguez, B. Pateiro-López, and S. Rodríguez-Ballesteros. Learning for spatial branching: An algorithm selection approach. *INFORMS Journal on Computing*, 35(5):1024–1043, 2023. doi: 10.1287/ijoc.2022.0090. URL <https://doi.org/10.1287/ijoc.2022.0090>. [p216, 220]
- B. González-Rodríguez, I. Gómez-Casares, B. Ghaddar, J. González-Díaz, and B. Pateiro-López. Learning in spatial branching: Limitations of strong branching imitation. *INFORMS Journal on Computing*, 2025a. [p220]
- B. González-Rodríguez, I. Gómez-Casares, B. Pateiro-López, and J. González-Díaz. *ASML: Algorithm Portfolio Selection with Machine Learning*, 2025b. URL <https://CRAN.R-project.org/package=ASML>. R package version 1.1.0. [p217]
- P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann. Automated Algorithm Selection: Survey and Perspectives. *Evolutionary Computation*, 27(1):3–45, 03 2019. ISSN 1063-6560. doi: 10.1162/evco_a_00242. URL https://doi.org/10.1162/evco_a_00242. [p216]
- L. Kotthoff. *Algorithm Selection for Combinatorial Search Problems: A Survey*, pages 149–190. Springer International Publishing, Cham, 2016. ISBN 978-3-319-50137-6. [p216, 218]
- L. Kotthoff, B. Bischl, B. Hurley, T. Rahwan, and D. Pulatov. *llama: Leveraging Learning to Automatically Manage Algorithms*, 2021. URL <https://CRAN.R-project.org/package=llama>. R package version 0.10.1. [p219]
- Kuhn and Max. Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5):1–26, 2008. doi: 10.18637/jss.v028.i05. URL <https://www.jstatsoft.org/index.php/jss/article/view/v028i05>. [p217]
- M. Lang, M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, Q. Au, G. Casalicchio, L. Kotthoff, and B. Bischl. mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*, dec 2019. doi: 10.21105/joss.01903. URL <https://joss.theoj.org/papers/10.21105/joss.01903>. [p219]
- A. Lodi and G. Zarpellon. On learning and branching: a survey. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 25(2):207–236, July 2017. doi: 10.1007/s11750-017-0451-6. URL https://ideas.repec.org/a/spr/topjn/v25y2017i2d10.1007_s11750-017-0451-6.html. [p216]
- N. Meinshausen. Quantile regression forests. *Journal of Machine Learning Research*, 7:983–999, Dec. 2006. ISSN 1532-4435. [p224]
- N. Meinshausen. *quantregForest: Quantile Regression Forests*, 2024. URL <https://CRAN.R-project.org/package=quantregForest>. R package version 1.3-7.1. [p224]
- T. Messelis and P. De Causmaecker. An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, 233(3):511–528, 2014. ISSN 0377-2217. doi: <https://doi.org/10.1016/j.ejor.2013.08.021>. URL <https://www.sciencedirect.com/science/article/pii/S0377221713006863>. [p216]
- J. C. Pichel and B. Pateiro-López. A new approach for sparse matrix classification based on deep learning techniques. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 46–54, 2018. doi: 10.1109/CLUSTER.2018.00017. [p229]
- D. Pulatov, M. Anastacio, L. Kotthoff, and H. Hoos. Opening the black box: Automated software analysis for algorithm selection. In I. Guyon, M. Lindauer, M. van der Schaar, F. Hüllermeier, and R. Garnett, editors, *Proceedings of the First International Conference on Automated Machine Learning*, volume 188 of *Proceedings of Machine Learning Research*, pages 6/1–18. PMLR, 25–27 Jul 2022. URL <https://proceedings.mlr.press/v188/pulatov22a.html>. [p218]

- J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976. [p218]
- H. D. Sherali and C. H. Tuncbilek. A global optimization algorithm for polynomial programming problems using a reformulation-linearization technique. *Journal of Global Optimization*, 2(1):101–112, 1992. doi: 10.1007/bf00121304. [p220]
- D. Speck, A. Biedenkapp, F. Hutter, R. Mattmüller, and M. Lindauer. Learning heuristic selection with dynamic algorithm configuration. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS21)*, aug 2021. URL <https://arxiv.org/abs/2006.08246>. [p216]
- L. Tierney, A. J. Rossini, N. Li, and H. Sevcikova. *snow: Simple Network of Workstations*, 2021. URL <https://CRAN.R-project.org/package=snow>. R package version 0.4-4. [p225]
- J. Vanschoren. *Meta-Learning*, pages 35–61. Springer International Publishing, 2019. ISBN 978-3-030-05318-5. [p216]

Ignacio Gómez-Casares

Universidade de Santiago de Compostela

Department of Statistics, Mathematical Analysis and Optimization

Santiago de Compostela, Spain

ignaciogomez.casares@usc.es

Beatriz Pateiro-López

Universidade de Santiago de Compostela

Department of Statistics, Mathematical Analysis and Optimization

CITMAga (Galician Center for Mathematical Research and Technology)

Santiago de Compostela, Spain

ORCID: 0000-0002-7714-1835

beatriz.pateiro@usc.es

Brais González-Rodríguez

Universidade de Vigo

Department of Statistics and Operational Research

SiDOR Research Group

Vigo, Spain

brais.gonzalez.rodriguez@uvigo.gal

Julio González-Díaz

Universidade de Santiago de Compostela

Department of Statistics, Mathematical Analysis and Optimization

CITMAga (Galician Center for Mathematical Research and Technology)

Santiago de Compostela, Spain

ORCID: 0000-0002-4667-4348

julio.gonzalez@usc.es

elhmc: An R Package for Hamiltonian Monte Carlo Sampling in Bayesian Empirical Likelihood

by Neo Han Wei, Dang Trung Kien, and Sanjay Chaudhuri

Abstract In this article, we describe an R package for sampling from an empirical likelihood-based posterior using a Hamiltonian Monte Carlo method. Empirical likelihood-based methodologies have been used in the Bayesian modeling of many problems of interest in recent times. This semiparametric procedure can easily combine the flexibility of a nonparametric distribution estimator together with the interpretability of a parametric model. The model is specified by estimating equation-based constraints. Drawing inference from a Bayesian empirical likelihood (BayesEL) posterior is challenging. The likelihood is computed numerically, so no closed-form expression of the posterior exists. Moreover, for any sample of finite size, the support of the likelihood is non-convex, which hinders fast mixing of many Markov Chain Monte Carlo (MCMC) procedures. It has been recently shown that using the properties of the gradient of the log empirical likelihood, one can devise an efficient Hamiltonian Monte Carlo (HMC) algorithm to sample from a BayesEL posterior. The package requires the user to specify only the estimating equations, the prior, and their respective gradients. An MCMC sample drawn from the BayesEL posterior of the parameters, with various details required by the user, is obtained.

1 Introduction

Empirical likelihood has several advantages over a traditional parametric likelihood. Even though a correctly specified parametric likelihood is usually the most efficient for parameter estimation, semiparametric methods like empirical likelihood, which use a nonparametric estimate of the underlying distribution, are often more efficient when the model is misspecified. Empirical likelihood incorporates parametric model-based information as constraints in estimating the underlying distribution, which makes the parametric estimates interpretable. Furthermore, it allows easy incorporation of known additional information not involving the parameters in the analysis.

Bayesian empirical likelihood (BayesEL) (Lazar, 2003) methods employ empirical likelihood in the Bayesian paradigm. Given some information about the model parameters in the form of a prior distribution and estimating equations obtained from the model, a likelihood is constructed from a constrained empirical estimate of the underlying distribution. The prior is then used to define a posterior based on this estimated likelihood. Inference on the parameter is drawn based on samples generated from the posterior distribution.

BayesEL methods are quite flexible and have been found useful in many areas of statistics. The examples include small area estimation, quantile regression, analysis of complex survey data, etc.

BayesEL procedures, however, require an efficient Markov Chain Monte Carlo (MCMC) procedure to sample from the resulting posterior. It turns out that such a procedure is not easily specified. For many parameter values, it may not be feasible to compute the constrained empirical distribution function, and the likelihood is estimated to be zero. That is, the estimated likelihood is not supported over the whole space. Moreover, this support is non-convex and impossible to determine in most cases. Thus, a naive random walk MCMC would quite often propose parameters outside the support and get stuck.

Many authors have encountered this problem in frequentist applications. Such "empty set" problems are quite common (Grendár and Judge, 2009) and become more frequent in problems with a large number of parameters (Bergsma et al., 2012). Several authors (Chen et al., 2008; Emerson et al., 2009; Liu et al., 2010) have suggested the addition of

extra observations generated from the available data designed specifically to avoid empty sets. They show that such observations can be proposed without changing the asymptotic distribution of the corresponding Wilks' statistics. Some authors (([Tsao, 2013](#); [Tsao and Wu, 2013, 2014](#))) have used a transformation so that the contours of the resultant empirical likelihood could be extended beyond the feasible region. However, in most Bayesian applications, the data are finite in size and not large, for which the asymptotic arguments have little use.

With the availability of user-friendly software packages like STAN ([Carpenter et al., 2017](#)), gradient-assisted MCMC methods like Hamiltonian Monte Carlo (HMC) are becoming increasingly popular in Bayesian computation. When the estimating equations are smooth with respect to the parameters, gradient-based methods would have a huge advantage in sampling from a BayesEL posterior. This is because [Chaudhuri et al. \(2017\)](#) have shown that under mild conditions, the gradient of the log-posterior would diverge to infinity at the boundary of its support. Due to this phenomenon, if an HMC chain approaches the boundary of the posterior support, it would be reflected towards its center.

There is no software to implement HMC sampling from a BayesEL posterior with smooth estimating equations and priors. We describe such a library called `e1hmc` written for the R platform. The main function in the library only requires the user to specify the estimating equations, prior, and respectively their Hessian and gradient with respect to the parameters as functions. Outputs with user-specified degree of detail can be obtained.

The `e1hmc` package has been used by practitioners since it was made available on CRAN. In recent times, various other libraries for sampling from a BayesEL posterior have been made available. Among them, the library `VBe1` ([Yu and Lim, 2024](#)) deserves special mention. The authors compute a variational approximation of the BayesEL posterior from which samples can be easily drawn. However, most of the time `e1hmc` is considered to be the benchmark.

The rest of the article is structured as follows. We start with the theoretical background behind the software package. In section 2.2 we first define the empirical likelihood and construct a Bayesian empirical likelihood from it. The next part of this section is devoted to a review of the properties of the log empirical likelihood gradient. A review of the HMC method with special emphasis on BayesEL sampling is provided next (Section 2.2.3). Section 2.3 mainly contains the description of the `e1hmc` library. Some illustrative examples with artificial and real data sets are presented in Section 2.4.

2 Theoretical background

2.1 Basics of Bayesian Empirical Likelihood

Suppose $x = (x_1, \dots, x_n) \in \mathbb{R}^p$ are n observations from a distribution F^0 depending on a parameter vector $\theta = (\theta^{(1)}, \dots, \theta^{(d)}) \in \Theta \subseteq \mathbb{R}^d$. We assume that both F^0 and the true parameter value θ^0 are unknown. However, certain smooth functions $g(\theta, x) = (g_1(\theta, x), \dots, g_q(\theta, x))^T$ are known to satisfy

$$E_{F^0}[g(\theta^0, x)] = 0. \quad (1)$$

Additionally, information about the parameter is available in the form of a prior density $\pi(\theta)$ supported on Θ . We assume that it is neither possible nor desirable to specify F^0 in a parametric form. On the other hand, it is not beneficial to estimate F^0 completely nonparametrically without taking into account the information from (1) in the estimation procedure.

Empirical likelihood provides a semiparametric procedure to estimate F^0 , by incorporating information contained in (1). A likelihood can be computed from the estimate. Moreover, if some information about the parameter is available in the form of a prior distribution, the same likelihood can be employed to derive a posterior of the parameter given the observations.

Let $F \in \mathcal{F}_\theta$ be a distribution function depending on the parameter θ . The empirical likelihood is the maximum of the “nonparametric likelihood”

$$L(F) = \prod_{i=1}^n \{F(x_i) - F(x_i-)\} \quad (2)$$

over $\mathcal{F}_\theta, \theta \in \Theta$, under constraints depending on $g(\theta, x)$.

More specifically, by defining $\omega_i = F(x_i) - F(x_i-)$, the empirical likelihood for θ is defined by,

$$L(\theta) := \max_{\omega \in \mathcal{W}_\theta} \prod_{i=1}^n \omega_i \quad (3)$$

where

$$\mathcal{W}_\theta = \left\{ \omega : \sum_{i=1}^n \omega_i g(\theta, x_i) = 0 \right\} \cap \Delta_{n-1}$$

and Δ_{n-1} is the $n-1$ dimensional simplex, i.e. $\omega_i \geq 0, \forall i$ and $\sum_{i=1}^n \omega_i = 1$. For any θ , if the problem in (3) is infeasible, i.e. $\mathcal{W}_\theta = \emptyset$, we define $L(\theta) := 0$.

Using the empirical likelihood $L(\theta)$ and the prior $\pi(\theta)$ we can define a posterior as:

$$\Pi(\theta|x) = \frac{L(\theta)\pi(\theta)}{\int L(\theta)\pi(\theta)d\theta} \propto L(\theta)\pi(\theta). \quad (4)$$

In Bayesian empirical likelihood (BayesEL), $\Pi(\theta|x)$ is used as the posterior to draw inferences on the parameter.

Returning back to (3) above, suppose we denote:

$$\hat{\omega}(\theta) = \operatorname{argmax}_{\omega \in \mathcal{W}_\theta} \prod_{i=1}^n \omega_i. \quad \left(\text{i.e. } L(\theta) = \prod_{i=1}^n \hat{\omega}_i(\theta) \right) \quad (5)$$

Each $\hat{\omega}_i \geq 0$ if and only if the origin in \mathbb{R}^q can be expressed as a convex combination of $g(\theta, x_1), \dots, g(\theta, x_n)$. Otherwise, the optimisation problem is infeasible, and $\mathcal{W}_\theta = \emptyset$. Furthermore, when $\hat{\omega}_i > 0, \forall i$ is feasible, the solution $\hat{\omega}$ of (5) is unique.

The estimate of F^0 is given by:¹

$$\hat{F}^0(x) = \sum_{i=1}^n \hat{\omega}_i(\theta) 1_{\{x_i \leq x\}}.$$

The distribution \hat{F}^0 is a step function with a jump of $\hat{\omega}_i(\theta)$ on x_i . If $\mathcal{W}_\theta = \Delta_{n-1}$, i.e. no information about $g(\theta, x)$ is present, it easily follows that $\hat{\omega}_i(\theta) = n^{-1}$, for each $i = 1, 2, \dots, n$ and \hat{F}^0 is the well-known empirical distribution function.

By construction, $\Pi(\theta|x)$ can only be computed numerically. No analytic form is available. Inferences are drawn through the observations from $\Pi(\theta|x)$ sampled using Markov chain Monte Carlo techniques.

Adaptation of Markov chain Monte Carlo methods to BayesEL applications poses several challenges. First of all, it is not possible to determine the full conditional densities in a closed form. So techniques like Gibbs sampling (Geman and Geman, 1984) cannot be used. In most cases, random walk Metropolis procedures, with carefully chosen step sizes, are attempted. However, the nature of the support of $\Pi(\theta|x)$, which we discuss in detail below, makes the choice of an appropriate step size extremely difficult.

Provided that the prior is positive over the whole Θ , which is true in most applications, the support of $\Pi(\theta|x)$ is a subset of the support of the likelihood $L(\theta)$ which can be defined as (see Figure 1):

$$\Theta_1 = \{\theta : L(\theta) > 0\}. \quad (6)$$

¹By convention, $x_i = (x_{i1}, x_{i2}, \dots, x_{ip})^T \leq x = (x_1, x_2, \dots, x_p)^T$ iff $x_{ij} \leq x_j \forall j$.

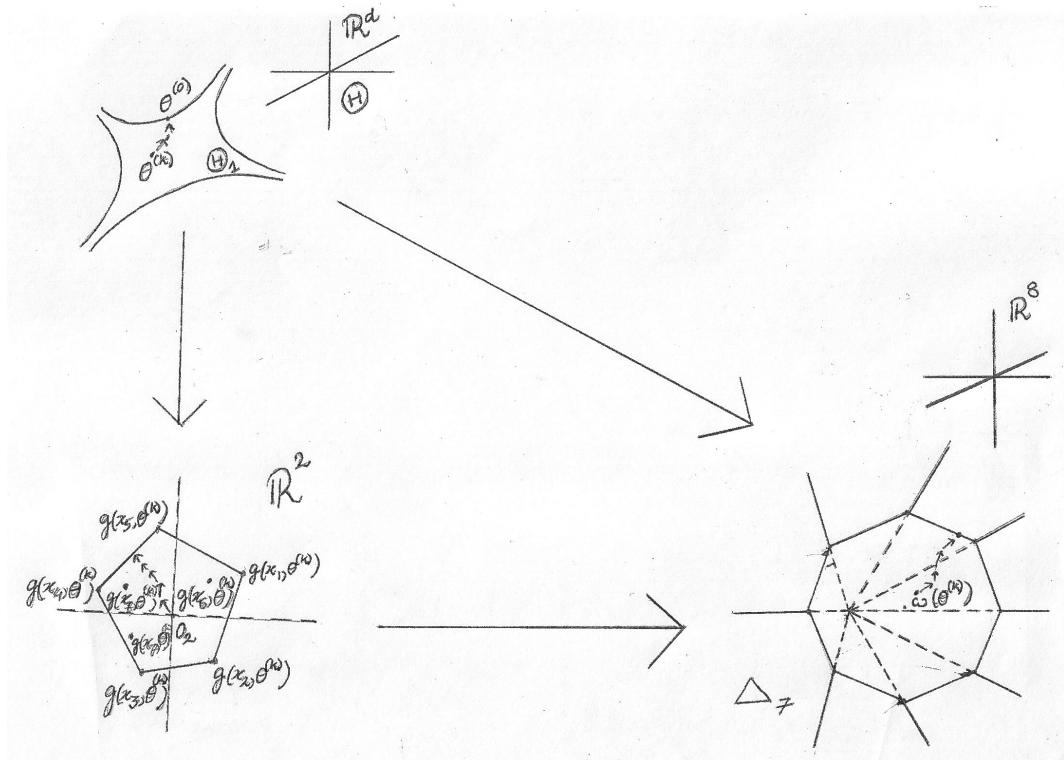


Figure 1: Schematic illustration of the Empirical likelihood problem. The support of the empirical likelihood is Θ_1 , a subset of \mathbb{R}^d . We take $n = 8$ observations. The estimating equations $g(x, \theta)$ are $q = 2$ dimensional. Note that Θ_1 is non-convex and may not be bounded. The convex hull of the q -dimensional vectors, i.e., $\mathcal{C}(\theta, x)$, is a pentagon in \mathbb{R}^2 . The largest faces of $\mathcal{C}(\theta, x)$ are the one-dimensional sides of the pentagon. It follows that, $\theta^{(k)} \in \Theta_1$ iff the origin of \mathbb{R}^2 , denoted 0_2 is in the interior $\mathcal{C}^0(\theta, x)$ of $\mathcal{C}(\theta, x)$. This also implies that the optimal empirical likelihood weights $\hat{\omega}(\theta^{(k)})$ are strictly positive and lie in the interior of the $n - 1$, i.e. 7-dimensional simplex. There is no easy way to determine Θ_1 . We check if $0_2 \in \mathcal{C}^0(\theta, x)$ or equivalently if $\hat{\omega}(\theta^{(k)})$ are in the interior of Δ_7 in order to determine if $\theta^{(k)} \in \Theta_1$. As the sequence $\theta^{(k)}$ approaches the boundary of Θ_1 , the convex polytope $\mathcal{C}(\theta^{(k)}, x)$ changes in such a way, so that 0_2 converges to its boundary. The sequence of optimal weights $\hat{\omega}(\theta^{(k)})$, will converge to the boundary of Δ_7 . The current software is based on Chaudhuri et al. (2017), who show that, under simple conditions, along almost every sequence $\theta^{(k)}$ converging to the boundary of Θ_1 , at least one component of the gradient of log-empirical likelihood based posterior diverges to positive or negative infinity.

Thus, the efficiency of the MCMC algorithm would depend on Θ_1 and the behaviour of $\Pi(\theta|x)$ on it.

By definition, Θ_1 is closely connected to the set

$$\mathcal{C}(\theta, x) = \left\{ \sum_{i=1}^n \omega_i g(\theta, x_i) \mid \omega \in \Delta_{n-1} \right\}, \quad (7)$$

which is the closed convex hull of the q dimensional vectors $G(x, \theta) = \{g(\theta, x_1), \dots, g(\theta, x_n)\}$ in \mathbb{R}^q (the pentagon in Figure 1). Suppose $\mathcal{C}^0(\theta, x)$ and $\partial\mathcal{C}(\theta, x)$ are respectively the interior and boundary of $\mathcal{C}(\theta, x)$. By construction, $\mathcal{C}(\theta, x)$ is a convex polytope. Since the data x is fixed, the set $\mathcal{C}(\theta, x)$ is a set-valued function of θ . For any $\theta \in \Theta$, the problem in (3) is feasible (i.e. $\mathcal{W}_\theta \neq \emptyset$) if and only if the origin of \mathbb{R}^q , denoted by 0_q , is in $\mathcal{C}(\theta, x)$. That is, $\theta \in \Theta_1$ if and only if the same $0_q \in \mathcal{C}^0(\theta, x)$. It is not possible to determine Θ_1 in general. The only way is to check if, for any potential θ , the origin 0_q is in $\mathcal{C}^0(\theta, x)$. There is no quick numerical way to check the latter either. Generally, an attempt is made to solve (3). The existence of such a solution indicates that $\theta \in L(\theta)$.

Examples show (Chaudhuri et al., 2017) that even for simple problems, Θ_1 may not be a convex set. Designing an efficient random walk Markov chain Monte Carlo algorithm on a potentially non-convex support is an extremely challenging task. Unless the step sizes and the proposal distributions are adapted well to the proximity of the current position to the boundary of Θ_1 , the chain may repeatedly propose values outside the likelihood support and, as a result, converge very slowly. Adaptive algorithms like the one proposed by Haario et al. (1999) do not tackle the non-convexity problem well.

Hamiltonian Monte Carlo methods solve well-known equations of motion from classical mechanics to propose new values of $\theta \in \Theta$. Numerical solutions of these equations of motion are dependent on the gradient of the log posterior. The norm of the gradient of the log empirical likelihood used in BayesEL procedures diverges near the boundary of Θ_1 . This property makes the Hamiltonian Monte Carlo procedures very efficient for sampling a BayesEL posterior. It ensures that once in Θ_1 , the chain would rarely step outside the support and repeatedly sample from the posterior.

2.2 A Review of Some Properties of the Gradient of Log Empirical Likelihood

Various properties of log-empirical likelihood have been discussed in the literature. However, the properties of its gradients with respect to the model parameters are relatively unknown. Our main goal in this section is to review the behaviour of gradients of log-empirical likelihood on the support of the empirical likelihood. We only state the relevant results here. The proofs of these results can be found in Chaudhuri et al. (2017).

Recall that, (see Figure 1) the support Θ_1 can only be specified by checking if $0_q \in \mathcal{C}^0(x, \theta_0)$ for each individual $\theta_0 \in \Theta$. If for some $\theta_0 \in \Theta$, the origin lies on the boundary of $\mathcal{C}(x, \theta_0)$, i.e. $0_q \in \partial\mathcal{C}(x, \theta_0)$, the problem in (3) is still feasible, however, $L(\theta_0) = 0$ and the solution of (5) is not unique. Below we discuss how, under mild conditions, for any $\theta_0 \in \Theta$, for a large subset $S \subseteq \partial\mathcal{C}(x, \theta_0)$, if $0_q \in S$, the absolute value of at least one component of the gradient of $\log(L(\theta_0))$ would be large.

Before we proceed, we make the following assumptions:

- (A0) Θ is an open set.
- (A1) g is a continuously differentiable function of θ in Θ , $q \leq d$ and Θ_1 is non-empty.
- (A2) The sample size $n > q$. The matrix $G(x, \theta)$ has full row rank for any $\theta \in \Theta$.
- (A3) For any fixed x , let $\nabla g(x_i, \theta)$ be the $q \times d$ Jacobian matrix for any $\theta \in \Theta$. Suppose $w = (w_1, \dots, w_n) \in \Delta_{n-1}$ and there are at least q elements of w that are greater than 0. Then, for any $\theta \in \Theta$, the matrix $\sum_{i=1}^n w_i \nabla g(x_i, \theta)$ has full row rank.

Under the above assumptions, several results about the log empirical likelihood and its gradient can be deduced.

First of all, since the properties of the gradient of the log empirical likelihood at the boundary of the support are of interest, some topological properties of the support need to be investigated. Under the standard topology of \mathbb{R}^q , since $\mathcal{C}(x, \theta)$ is a convex polytope with a finite number of faces and extreme points, using the smoothness of g , it is easy to see that, for any $\theta_0 \in \Theta_1$ one can find a real number $\delta > 0$, such that the open ball centred at θ_0 with radius δ is contained in Θ_1 . That is, Θ_1 is an open subset of Θ .

Now, since Θ_1 is an open set, the boundary $\partial\Theta_1$ of Θ_1 is not contained in Θ_1 . Let $\theta^{(0)}$ lie within Θ and on the boundary of Θ_1 (i.e. $\partial\Theta_1$). Then it follows that the primal problem (3) is feasible at $\theta^{(0)}$ and 0_q lies on the boundary of $\mathcal{C}(x, \theta^{(0)})$ (i.e. $\partial\mathcal{C}(x, \theta^{(0)})$).

Our main objective is to study the utility of Hamiltonian Monte Carlo methods for drawing samples from a BayesEL posterior. The sampling scheme will produce a sequence of sample points in $\theta^{(k)} \in \Theta_1$ (see Figure 1). It would be efficient as long as $\log L(\theta^{(k)})$ is large. The sampling scheme could potentially become inefficient if some $\theta^{(k)}$ is close to the

boundary $\partial\Theta_1$. Thus, it is sufficient to consider the properties of the log empirical likelihood and its gradient along such a sequence converging to a point $\theta^{(0)} \in \partial\Theta_1$.

From the discussion above it is evident that when $\theta^{(0)} \in \partial\Theta_1$ the problem in (3) is feasible but the likelihood $L(\theta^{(0)})$ will always be zero and (5) will not have a unique solution. Since $\mathcal{C}(x, \theta^{(0)})$ is a polytope, and 0_q lies on one of its faces, there exists a subset \mathcal{I}_0 of the observations and 0 belongs to the interior of the convex hull generated by all $g(x_i, \theta^{(0)})$ for $i \in \mathcal{I}_0$ (in Figure 1, $\mathcal{I}_0 = \{x_4, x_5\}$). It follows from the supporting hyperplane theorem (Boyd and Vandenberghe, 2004) that there exists a unit vector $a \in \mathbb{R}^q$ such that

$$a^T g(x_i, \theta^{(0)}) = 0 \quad \text{for } i \in \mathcal{I}_0, \quad \text{and} \quad a^T g(x_i, \theta^{(0)}) > 0 \quad \text{for } i \in \mathcal{I}_0^c.$$

From some algebraic manipulation it easily follows that any $\omega \in \mathcal{W}_{\theta^{(0)}}$ (\mathcal{W}_θ as defined in (3) with $\theta = \theta^{(0)}$) must satisfy²

$$\omega_i = 0 \quad \text{for } i \in \mathcal{I}_0^c \quad \text{and} \quad \omega_i > 0 \quad \text{for } i \in \mathcal{I}_0.$$

It is well known that the solution of (5) i.e. $\hat{w}(\theta)$ is smooth for all $\theta \in \Theta_1$ (Qin and Lawless, 1994). As $\theta^{(k)}$ converges to $\theta^{(0)}$, the properties of $\hat{w}(\theta^{(k)})$ need to be considered. To that goal, we first make a specific choice of $\hat{w}(\theta^{(0)})$.

First, we consider a restriction of problem (5) to \mathcal{I}_0 .

$$\hat{v}(\theta) = \operatorname{argmax}_{\nu \in \mathcal{V}_\theta} \prod_{i \in \mathcal{I}_0} \nu_i \quad (8)$$

where

$$\mathcal{V}_\theta = \left\{ \nu : \sum_{i \in \mathcal{I}_0} \nu_i g(x_i, \theta) = 0 \right\} \cap \Delta_{|\mathcal{I}_0|-1}.$$

We now define

$$\hat{\omega}_i(\theta^{(0)}) = \hat{v}(\theta^{(0)}), \quad i \in \mathcal{I}_0 \quad \text{and} \quad \hat{\omega}_i(\theta^{(0)}) = 0, \quad i \in \mathcal{I}_0^c,$$

and

$$L(\theta^{(0)}) = \prod_{i=1}^n \hat{\omega}_i(\theta^{(0)}).$$

Since $\theta^{(0)}$ is in the interior of \mathcal{I}_0 , the problem (8) has a unique solution. For each $\theta^{(k)} \in \Theta_1$, $\hat{w}(\theta^{(k)})$ is continuous taking values in a compact set. Thus as $\theta^{(k)}$ converges to $\theta^{(0)}$, $\hat{w}(\theta^{(k)})$ converges to a limit. Furthermore, this limit is a solution of (5) at $\theta^{(0)}$. However, counterexamples show (Chaudhuri et al., 2017) that the limit may not be $\hat{\omega}_i(\theta^{(0)})$ as defined above. That is, the vectors $\hat{w}(\theta^{(k)})$ do not extend continuously to the boundary $\partial\Theta_1$ as a whole. However, we can show that:

$$\lim_{k \rightarrow \infty} \hat{\omega}_i(\theta^{(k)}) = \hat{\omega}_i(\theta^{(0)}) = 0, \quad \text{for all } i \in \mathcal{I}_0^c. \quad (9)$$

That is, the components of $\hat{w}(\theta^{(k)})$ which are zero in $\hat{w}(\theta^{(0)})$ are continuously extendable. Furthermore,

$$\lim_{k \rightarrow \infty} L(\theta^{(k)}) = L(\theta^{(0)}) = 0. \quad (10)$$

That is, the likelihood is continuous at $\theta^{(0)}$.

However, this is not true for the components $\hat{\omega}_i(\theta^{(k)})$, $i \in \mathcal{I}_0$ for which $\hat{\omega}_i(\theta^{(k)}) > 0$.

²In Figure 1, $\omega_1 = \omega_2 = \omega_3 = 0$, $\omega_4 > 0$, and $\omega_5 > 0$.

Since the set $\mathcal{C}(x, \theta)$ is a convex polytope in \mathbb{R}^q , the maximum dimension of any of its faces is $q - 1$, which would have exactly q extreme points.³ Furthermore, any face with a smaller dimension can be expressed as an intersection of such $q - 1$ dimensional faces.

In certain cases, however, the whole vector $\hat{\omega}(\theta^{(k)})$ extends continuously to $\hat{\omega}(\theta^{(0)})$. In order to argue that, we define

$$\mathcal{C}(x_{\mathcal{I}}, \theta) = \left\{ \sum_{i \in \mathcal{I}} \omega_i g(x_i, \theta) \mid \omega \in \Delta_{|\mathcal{I}|-1} \right\} \quad (11)$$

and

$$\partial\Theta_1^{(q-1)} = \left\{ \theta : 0 \in \mathcal{C}^0(x_{\mathcal{I}}, \theta) \text{ for some } \mathcal{I} \text{ s.t. } \mathcal{C}(x_{\mathcal{I}}, \theta) \text{ has exactly } q \text{ extreme points} \right\} \cap \partial\Theta_1. \quad (12)$$

Thus $\partial\Theta_1^{(q-1)}$ is the set of all boundary points $\theta^{(0)}$ of Θ_1 such that 0 belongs to a $(q - 1)$ -dimensional face of the convex hull $\mathcal{C}(x, \theta^{(0)})$. Now for any $\theta^{(0)} \in \partial\Theta_1^{(q-1)}$, there is a unique set of weight $\nu \in \Delta_{|\mathcal{I}|-1}$ such that, $\sum_{i \in \mathcal{I}} \nu_i g(x_i, \theta^{(0)}) = 0$. That is the set of feasible solutions of (8) is a singleton set. This, after taking note that $\hat{\omega}$ takes values in a compact set, an argument using convergent subsequences, implies that for any sequence $\theta^{(k)} \in \Theta_1$ converging to $\theta^{(0)}$, the whole vector $\hat{\omega}(\theta^{(k)})$ converges to $\hat{\omega}(\theta^{(0)})$. That is, the whole vector $\hat{\omega}(\theta^{(k)})$ extends continuously to $\hat{\omega}(\theta^{(0)})$.

We now consider the behaviour of the gradient of the log empirical likelihood near the boundary of Θ_1 . First, note that, for any $\theta \in \Theta_1$, the gradient of the log empirical likelihood is given by

$$\nabla \log L(\theta) = -n \sum_{i=1}^n \hat{\omega}_i(\theta) \hat{\lambda}(\theta)^T \nabla g(x_i, \theta).$$

where $\hat{\lambda}(\theta)$ is the estimated Lagrange multiplier satisfying the equation:

$$\sum_{i=1}^n \frac{g(x_i, \theta)}{\{1 + \hat{\lambda}(\theta)^T g(x_i, \theta)\}} = 0. \quad (13)$$

Note that, the gradient depends on the value of the Lagrange multiplier but not on the value of its gradient.

Now, Under assumption A3, it follows that the gradient of the log empirical likelihood diverges on the set of all boundary points $\partial\Theta_1^{(q-1)}$. More specifically one can show:

1. As $\theta^{(k)} \rightarrow \theta^{(0)}$, $\| \hat{\lambda}(\theta^{(k)}) \| \rightarrow \infty$.
2. If $\theta^{(0)} \in \partial\Theta_1^{(q-1)}$, under A3 as $\theta^{(k)} \rightarrow \theta^{(0)}$, $\| \nabla \log L(\theta^{(k)}) \| \rightarrow \infty$.

Therefore, it follows that at every boundary point $\theta^{(0)}$ of Θ_1 such that 0 belongs to one of the $(q - 1)$ -dimensional faces of $\mathcal{C}(x, \theta^{(0)})$, at least one component of the estimated Lagrange multiplier and the gradient of the log empirical likelihood diverges to positive or negative infinity. The gradient of the negative log empirical likelihood represents the direction of the steepest increase of the negative log empirical likelihood. Since the value of the log empirical likelihood should typically be highest around the center of the support Θ_1 , the gradient near the boundary of Θ_1 should point towards its center. This property can be exploited in forcing candidates of θ generated by HMC proposals to bounce back towards the interior of Θ_1 from its boundaries and in consequence reducing the chance of them getting out of the support.

³In Figure 1, $q = 2$, and the faces of maximum dimension are the sides of the pentagon. They have $q = 2$ end i.e. extreme points.

2.3 Hamiltonian Monte Carlo Sampling for Bayesian Empirical Likelihood

Hamiltonian Monte Carlo algorithm is a Metropolis algorithm where the successive steps are proposed by using Hamiltonian dynamics. One can visualise these dynamics as a cube sliding without friction under gravity in a bowl with a smooth surface. The total energy of the cube is the sum of the potential energy $U(\theta)$, defined by its position θ (in this case its height) and kinetic energy $K(p)$, which is determined by its momentum p . The total energy of the cube will be conserved and it will continue to slide up and down on the smooth surface of the bowl forever. The potential and the kinetic energy would, however, vary with the position of the cube.

In order to use the Hamiltonian dynamics to sample from the posterior $\Pi(\theta | x)$ we set our potential and kinetic energy as follows:

$$U(\theta) = -\log \Pi(\theta | x) \quad \text{and} \quad K(p) = \frac{1}{2} p^T M^{-1} p.$$

Here, the momentum vector $p = (p_1, p_2, \dots, p_d)$ is a totally artificial construct usually generated from a $N(0, M)$ distribution. Most often the covariance matrix M is chosen to be a diagonal matrix with diagonal (m_1, m_2, \dots, m_d) , in which case each m_i is interpreted as the mass of the i th parameter. The Hamiltonian of the system is the total energy

$$\mathcal{H}(\theta, p) = U(\theta) + K(p). \quad (14)$$

In Hamiltonian mechanics, the variation in the position θ and momentum p with time t is determined by the partial derivatives of \mathcal{H} with p and θ respectively. In particular, the motion is governed by the pair of so-called Hamiltonian equations:

$$\frac{d\theta}{dt} = \frac{\partial \mathcal{H}}{\partial p} = M^{-1} p, \quad (15)$$

$$\frac{dp}{dt} = -\frac{\partial \mathcal{H}}{\partial \theta} = -\frac{\partial U(\theta)}{\partial \theta}. \quad (16)$$

It is easy to show that (Neal, 2011) Hamiltonian dynamics is reversible, invariant, and volume preserving, which makes it suitable for MCMC sampling schemes.

In HMC we propose successive states by solving the Hamiltonian equations (15) and (16). Unfortunately, they cannot be solved analytically (except of course for a few simple cases), and they must be approximated numerically at discrete time points. There are several ways to numerically approximate these two equations in the literature (Leimkuhler and Reich, 2004). For the purpose of MCMC sampling, we need a method that is reversible and volume-preserving.

Leapfrog integration (Birdsall and Langdon, 2004) is one such method to numerically integrate the pair of Hamiltonian equations. In this method, a step-size ϵ for the time variable t is first chosen. Given the value of θ and p at the current time point t (denoted here by $\theta(t)$ and $p(t)$ respectively), the leapfrog updates the position and the momentum at time $t + \epsilon$ as follows

$$p\left(t + \frac{\epsilon}{2}\right) = p(t) - \frac{\epsilon}{2} \frac{\partial U(\theta(t))}{\partial \theta}, \quad (17)$$

$$\theta(t + \epsilon) = \theta(t) + \epsilon M^{-1} p\left(t + \frac{\epsilon}{2}\right), \quad (18)$$

$$p(t + \epsilon) = p\left(t + \frac{\epsilon}{2}\right) - \frac{\epsilon}{2} \frac{\partial U(\theta(t + \epsilon))}{\partial \theta}. \quad (19)$$

Theoretically, due to its symmetry, the leapfrog integration satisfies the reversibility and preserves the volume. However, because of the numerical inaccuracies, the volume is not preserved. This is similar to the Langevin-Hastings algorithm (Besag, 2004), which is a special case of HMC. Fortunately, the lack of invariance in volume is easily corrected. The

accept-reject step in the MCMC procedure ensures that the chain converges to the correct posterior.

At the beginning of each iteration of the HMC algorithm, the momentum vector p is randomly sampled from the $N(0, M)$ distribution. Starting with the current state (θ, p) , the leapfrog integrator described above is used to simulate Hamiltonian dynamics for T steps with a step size of ϵ . At the end of this T -step trajectory, the momentum p is negated so that the Metropolis proposal is symmetric. At the end of this T -step iteration, the proposed state (θ^*, p^*) is accepted with probability

$$\min\{1, \exp(-\mathcal{H}(\theta^*, p^*) + \mathcal{H}(\theta, p))\}.$$

The gradient of the log-posterior used in the leapfrog is a sum of the gradient of the log empirical likelihood and the gradient of the log prior. The prior is user-specified and it is hypothetically possible that even though at least one component of the gradient of the log empirical likelihood diverges at the boundary $\partial\Theta_1$, the log prior gradient may behave in a way so that the effect is nullified and the log posterior gradient remains finite over the closure of Θ_1 . We make the following assumption on the prior mainly to avoid this possibility (see Chaudhuri et al. (2017) for more details).

- (A4) Consider a sequence $\{\theta^{(k)}\}$, $k = 1, 2, \dots$, of points in Θ_1 such that $\theta^{(k)}$ converges to a boundary point $\theta^{(0)}$ of Θ_1 . Assume that $\theta^{(0)}$ lies within Θ and $L(\theta^{(k)})$ strictly decreases to $L(\theta^{(0)})$. Then, for some constant $b(n, \theta^{(0)}) > -1$, we have

$$\liminf_{k \rightarrow \infty} \frac{\log \pi(\theta^{(k-1)}) - \log \pi(\theta^{(k)})}{\log L(\theta^{(k-1)}) - \log L(\theta^{(k)})} \geq b(n, \theta^{(0)}). \quad (20)$$

The assumption implies that near the boundary of the support, the main contribution in the gradient of the log-posterior with respect to any parameter appearing in the argument of the estimating equations comes from the corresponding gradient of the log empirical likelihood. This is in most cases expected, especially if the sample size is large. For a large sample size, the log-likelihood should be the dominant term in the log-posterior. We are just assuming here that the gradients behave the same way. It would also ensure that at the boundary, the gradient of the log-likelihood and the log-posterior do not cancel each other, which is crucial for the proposed Hamiltonian Monte Carlo to work.

Under these assumptions, Chaudhuri et al. (2017) show that the gradient of the log-posterior diverges along almost every sequence as the parameter values approach the boundary $\partial\Theta_1$ from the interior of the support. More specifically, they prove that:

$$\|\nabla \log \pi(\theta^{(k)} | x)\| \rightarrow \infty, \quad \text{as } k \rightarrow \infty. \quad (21)$$

Since the $q - 1$ dimensional faces of $\mathcal{C}(x, \theta^{(0)})$ have larger volume than its faces with lower dimension (see Figure 1), a random sequence of points from the interior to the boundary would converge to a point on $\partial\Theta_1^{(q-1)}$ with probability 1. Thus under our assumptions, the gradient of the log-posterior would diverge to infinity for these sequences with a high probability. The lower dimensional faces of the convex hull (a polytope) are an intersection of $q - 1$ dimensional faces. Although, it is not clear if the norm of the gradient of the posterior will diverge on those faces. It is conjectured that this would happen. However, even if the conjecture is not true, from the setup, it is clear that the sampler would rarely move to the region where the origin belongs to the lower dimensional faces of the convex hull.

As has been pointed out above, the gradient vector would always point towards the mode of the posterior. From our results, since the gradient is large near the support boundary, whenever the HMC sampler approaches the boundary due to the high value of the gradient it would reflect towards the interior of the support and not get out of it. The leapfrog parameters can be controlled to increase efficiency of sampling.

3 Package description

The main function of the package is ELHMC. It draws samples from an empirical likelihood Bayesian posterior of the parameter of interest using Hamiltonian Monte Carlo once the estimating equations involving the parameters, the prior distribution of the parameters, the gradients of the estimating equations, and the log priors are specified. Some other parameters which control the HMC process can also be specified.

Suppose that the data set consists of observations $x = (x_1, \dots, x_n)$ where each x_i is a vector of length p and follows a probability distribution F of family \mathcal{F}_θ . Here $\theta = (\theta_1, \dots, \theta_d)$ is the d -dimensional parameter of interest associated with F . Suppose there exist smooth functions $g(\theta, x_i) = (g_1(\theta, x_i), \dots, g_q(\theta, x_i))^T$ which satisfy $E_F[g(\theta, x_i)] = 0$. As we have explained above, ELHMC is used to draw samples of θ from its posterior defined by an empirical likelihood.

initial	A vector containing the initial values of the parameter
data	A matrix containing the data
fun	The estimating function g . It takes in a parameter vector <code>params</code> as the first argument and a data point vector <code>x</code> as the second parameter. This function returns a vector.
dfun	A function that calculates the gradient of the estimating function g . It takes in a parameter vector <code>params</code> as the first argument and a data point vector <code>x</code> as the second argument. This function returns a matrix.
prior	A function with one argument <code>x</code> that returns the log joint prior density of the parameters of interest.
dprior	A function with one argument <code>x</code> that returns the gradients of the log densities of the parameters of interest
n.samples	Number of samples to draw
lf.steps	Number of leap frog steps in each Hamiltonian Monte Carlo update (defaults to 10).
epsilon	The leap frog step size (defaults to 0.05).
p.variance	The covariance matrix of a multivariate normal distribution used to generate the initial values of momentum <code>p</code> in Hamiltonian Monte Carlo. This can also be a single numeric value or a vector (defaults to 0.1).
tol	EL tolerance
detailed	If this is set to TRUE, the function will return a list with extra information.
print.interval	The frequency at which the results would be printed on the terminal. Defaults to 1000.
plot.interval	The frequency at which the drawn samples would be plotted. The last half of the samples drawn are plotted after each plot.interval steps. The acceptance rate is also plotted. Defaults to 0, which means no plot.
which.plot	The vector of parameters to be plotted after each plot.interval. Defaults to NULL, which means no plot.
FUN	the same as <code>fun</code> but takes in a matrix <code>X</code> instead of a vector <code>x</code> and returns a matrix so that <code>FUN(params, X)[i,]</code> is the same as <code>fun(params, X[i,])</code> . Only one of <code>FUN</code> and <code>fun</code> should be provided. If both are then <code>fun</code> is ignored.
DFUN	the same as <code>dfun</code> but takes in a matrix <code>X</code> instead of a vector <code>x</code> and returns an array so that <code>DFUN(params, X[, , i])</code> is the same as <code>dfun(params, X[i,])</code> . Only one of <code>DFUN</code> and <code>dfun</code> should be provided. If both are then <code>dfun</code> is ignored.

Table 1: Arguments for function ELHMC

Table 1 enlists the full list of arguments for ELHMC. Arguments `data` and `fun` define the problem. They are the data set x and the collection of smooth functions in g . The user-specified starting point for θ is given in `initial`, whereas `n.samples` is the number of samples of θ to be drawn. The gradient matrix of g with respect to the parameter θ (i.e. $\nabla_\theta g$) has to be specified in `dfun`. At the moment the function does not compute the gradient numerically by itself. The prior `prior` represents the joint density functions of $\theta_1, \dots, \theta_q$, which for the purpose of this description we denote by π . The gradient of the log prior function is specified in `dprior`. The function returns a vector containing the values

<code>samples</code>	A matrix containing the parameter samples
<code>acceptance.rate</code>	The acceptance rate
<code>call</code>	The matched call

Table 2: Elements of the list returned by ELHMC if detailed = FALSE

of $\frac{\partial}{\partial \theta_1} \pi(\theta), \dots, \frac{\partial}{\partial \theta_d} \pi(\theta)$. Finally, the arguments `epsilon`, `lf.steps`, `p.variance` and `tol` are hyper-parameters which control the Hamiltonian Monte Carlo algorithm.

The arguments `print.interval`, `plot.interval`, and `which.plot` can be used to tune the HMC samplers. They can be used for printing and plotting the sampled values at specified intervals while the code is running. The argument `which.plot` allows the user to only plot the variables whose convergence needs to be checked.

Given the data and a value of θ , ELHMC computes the optimal weights using the `e1.test` function from `emplik` library (Zhou, 2014). The `e1.test` provides $\hat{\lambda}(\theta^{(k)})$ from which the gradient of the log-empirical likelihood can be computed.

If $\theta \notin \Theta_1$, i.e. problem 5 is not feasible, then `e1.test` converges to weights all close to zero which do not sum to one. Furthermore, the norm of $\hat{\lambda}(\theta^{(k)})$ will be large. In such cases, the empirical likelihood will be zero. This means that, whenever the optimal weights are computed, we need to check if they sum to one (within numerical errors) or not.

The function ELHMC returns a list. If argument `detailed` is set to FALSE, the list contains samples of the parameters of interest θ , the Monte Carlo acceptance rate as listed in table 2. If `detailed` is set to TRUE, additional information such as the trajectories of θ and the momentum is included in the returned list (see Table 3).

At the moment ELHMC only allows a diagonal covariance matrix for the momentum p . The default value for the stepsize `epsilon` and step number `lf.steps` are 0.05 and 10 respectively. For a specific problem they need to be determined by trial and error, using the outputs from `plot.interval`, and `print.interval` commands.

<code>samples</code>	A matrix containing the parameter samples
<code>acceptance.rate</code>	The acceptance rate
<code>proposed</code>	A matrix containing the proposed values at <code>n.samaples - 1</code> Hamiltonian Monte Carlo updates
<code>acceptance</code>	A vector of TRUE/FALSE values indicates whether each proposed value is accepted
<code>trajectory</code>	A list with 2 elements <code>trajectory.q</code> and <code>trajectory.p</code> . These are lists of matrices containing position and momentum values along trajectory in each Hamiltonian Monte Carlo update.
<code>call</code>	The matched call

Table 3: Elements of the list returned by ELHMC if detailed = TRUE

4 Examples

In this section, we present two examples of usage of the package. Both examples in some sense supplement the conditions considered by Chaudhuri et al. (2017). In each case, it is seen that the function can sample from the resulting empirical likelihood-based posterior quite efficiently.

4.1 Sample the mean of a simple data set

In the first example, suppose the data set consists of eight data points $v = (v_1, \dots, v_8)$:

```
R> v <- rbind(c(1, 1), c(1, 0), c(1, -1), c(0, -1),
+               c(-1, -1), c(-1, 0), c(-1, 1), c(0, 1))
```

```
R> print(v)
   [,1] [,2]
[1,]    1    1
[2,]    1    0
[3,]    1   -1
[4,]    0   -1
[5,]   -1   -1
[6,]   -1    0
[7,]   -1    1
[8,]    0    1
```

The parameters of interest are the mean $\theta = (\theta_1, \theta_2)$. Since $E[\theta - v_i] = 0$, the smooth function is $g = \theta - v_i$ with $\nabla_\theta g = ((1, 0), (0, 1))$:

```
Function: fun
R> g <- function(params, x) {
+   params - x
+ }

Function: dfun
R> dlg <- function(params, x) {
+   rbind(c(1, 0), c(0, 1))
+ }
```

Functions `g` and `dlg` are supplied to arguments `fun` and `dfun` in `ELHMC`. These two functions must have `params` as the first argument and `x` as the second. `params` represents a sample of θ whereas `x` represents a data point v_i or a row in the matrix `v`. `fun` should return a vector and `dfun` a matrix whose (i, j) entry is $\partial g_i / \partial \theta_j$.

We assume that both θ_1 and θ_2 have independent standard normal distributions as priors. Next, we define the functions that calculate the prior densities and gradients of log prior densities as `pr` and `dpr` in the following ways:

```
Function: prior
R> pr <- function(x) {
+   -.5*(x[1]^2+x[2]^2)-log(2*pi)
+ }
Function: dprior
R> dpr <- function(x) {
+   -x
+ }
```

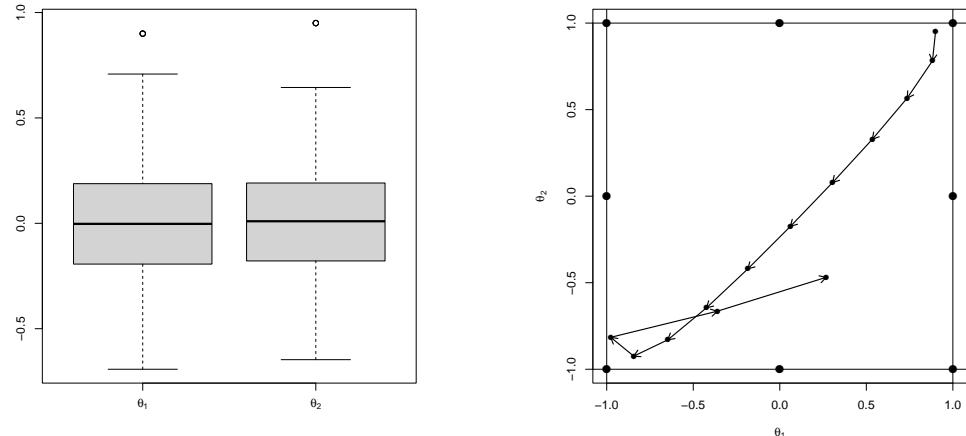
Functions `pr` and `dpr` are assigned to `prior` and `dprior` in `ELHMC`. `prior` and `dprior` must take in only one argument `x` and return a vector of the same length as θ .

We can now use `ELHMC` to draw samples of θ . Let us draw 1000 samples, with starting point $(0.9, 0.95)$ using 12 leapfrog steps with step size 0.06 for both θ_1 and θ_2 for each Hamiltonian Monte Carlo update:

```
R> library(elhmc)
R> set.seed(476)
R> thetas <- ELHMC(initial = c(0.9, 0.95), data = v, fun = g, dfun = dlg,
+                     prior = pr, dprior = dpr, n.samples = 1000,
+                     lf.steps = 12, epsilon = 0.06, detailed = TRUE)
```

We extract and visualise the distribution of the samples using a boxplot (Figure 2a):

```
R> boxplot(thetas$samples, names = c(expression(theta[1]), expression(theta[2])))
```

(a) Posterior distribution of θ_1 and θ_2 samples.(b) Trajectory of θ during the first Monte Carlo update.Figure 2: Samples of θ drawn from ELHMC.

Since we set `detailed = TRUE`, we have data on the trajectory of θ as well as momentum p . They are stored in element `trajectory` of `thetas` and can be accessed by `thetas$trajectory`. `thetas$trajectory` is a list with two elements named `trajectory.q` and `trajectory.p` denoting trajectories for θ and momentum p . `trajectory.q` and `trajectory.p` are both lists with elements $1, \dots, n.\text{samples} - 1$. Each of these elements is a matrix containing trajectories of θ (`trajectory.q`) and p (`trajectory.p`) at each Hamiltonian Monte Carlo update.

We illustrate by extracting the trajectories of θ at the first update and plotting them (Figure 2b):

```
R> q <- thetas$trajectory$trajectory.q[[1]]
R> plot(q, xlab = expression(theta[1]), ylab = expression(theta[2]),
+       xlim = c(-1, 1), ylim = c(-1, 1), cex = 1, pch = 16)
R> points(v[,1],v[,2],type="p",cex=1.5,pch=16)
R> abline(h=-1); abline(h=1); abline(v=-1); abline(v=1)
R> arrows(q[-nrow(q), 1], q[-nrow(q), 2], q[-1, 1], q[-1, 2],
+          length = 0.1, lwd = 1.5)
```

The specialty in this example is in the choice of the data points in v . Chaudhuri et al. (2017) show that the chain will reflect if the one-dimensional boundaries of the convex hull (in this case the unit square) have two observations, which happens with probability one for continuous distributions. In this example, however, there is more than one point in two one-dimensional boundaries. However, we can see that the HMC method works very well here.

4.2 Logistic regression with an additional constraint

In this example, we consider a constrained logistic regression of one binary variable on another, where the expectation of the response is known. The frequentist estimation problem using empirical likelihood was considered by Chaudhuri et al. (2008). It has been shown that empirical likelihood-based formulation has a major applicational advantage over the fully parametric formulation. Below we consider a Bayesian extension of the proposed empirical likelihood-based formulation and use ELHMC to sample from the resulting posterior.

	$x = 0$	$x = 1$
$y = 0$	5903	5157
$y = 1$	230	350

The data set v consists of n observations of two variables and two columns X and Y . In the i th row y_i represents the indicator of whether a woman gave birth between time $t - 1$ and t while x_i is the indicator of

Table 4: The dataset used in Example

whether she had at least one child at time $t - 1$. The data can be found in Table 4 above. In addition, it was known that the prevalent general fertility rate in the population was 0.06179.⁴

We are interested in fitting a logistic regression model to the data with X as the independent variable and Y as the dependent variable. However, we also would like to constrain the sample general fertility rate to its value in the population. The logistic regression model takes the form of:

$$P(Y = 1|X = x) = \frac{\exp(\beta_0 + \beta_1 x)}{1 + \exp(\beta_0 + \beta_1 x)}.$$

From the model, using conditions similar to zero-mean residual and exogeneity, it is clear that:

$$E\left[y_i - \frac{\exp(\beta_0 + \beta_1 x_i)}{1 + \exp(\beta_0 + \beta_1 x_i)}\right] = 0, \quad E\left[x_i \left\{y_i - \frac{\exp(\beta_0 + \beta_1 x_i)}{1 + \exp(\beta_0 + \beta_1 x_i)}\right\}\right] = 0.$$

Furthermore from the definition of general fertility rate, we get:

$$E[y_i - 0.06179] = 0.$$

Following Chaudhuri et al. (2008), we define the estimating equations g as follows:

$$g(\beta, v_i) = \begin{bmatrix} y_i - \frac{\exp(\beta_0 + \beta_1 x_i)}{1 + \exp(\beta_0 + \beta_1 x_i)} \\ x_i \left[y_i - \frac{\exp(\beta_0 + \beta_1 x_i)}{1 + \exp(\beta_0 + \beta_1 x_i)} \right] \\ y_i - 0.06179 \end{bmatrix}$$

The gradient of g with respect to β is given by:

$$\nabla_{\beta} g = \begin{bmatrix} \frac{-\exp(\beta_0 + \beta_1 x_i)}{(\exp(\beta_0 + \beta_1 x_i) + 1)^2} & \frac{-\exp(\beta_0 + \beta_1 x_i)x_i}{(\exp(\beta_0 + \beta_1 x_i) + 1)^2} \\ \frac{-\exp(\beta_0 + \beta_1 x_i)x_i}{(\exp(\beta_0 + \beta_1 x_i) + 1)^2} & \frac{-\exp(\beta_0 + \beta_1 x_i)x_i^2}{(\exp(\beta_0 + \beta_1 x_i) + 1)^2} \\ 0 & 0 \end{bmatrix}$$

In R, we create functions g and dg to represent g and $\nabla_{\beta} g$:

```
Function: fun
R> g <- function(params, X) {
+   result <- matrix(0, nrow = nrow(X), ncol = 3)
+   a <- exp(params[1] + params[2] * X[, 1])
+   a <- a / (1 + a)
+   result[, 1] <- X[, 2] - a
+   result[, 2] <- (X[, 2] - a) * X[, 1]
+   result[, 3] <- X[, 2] - 0.06179
+   result
}
Function: dfun
R> dg <- function(params, X) {
+   result <- array(0, c(3, 2, nrow(X)))
+   a <- exp(params[1] + params[2] * X[, 1])
+   a <- -a / (a + 1) ^ 2
+   result[1, 1, ] <- a
```

⁴The authors are grateful to Prof. Michael Rendall, Department of Sociology, University of Maryland, College Park, for kindly sharing the data on which this example is based.

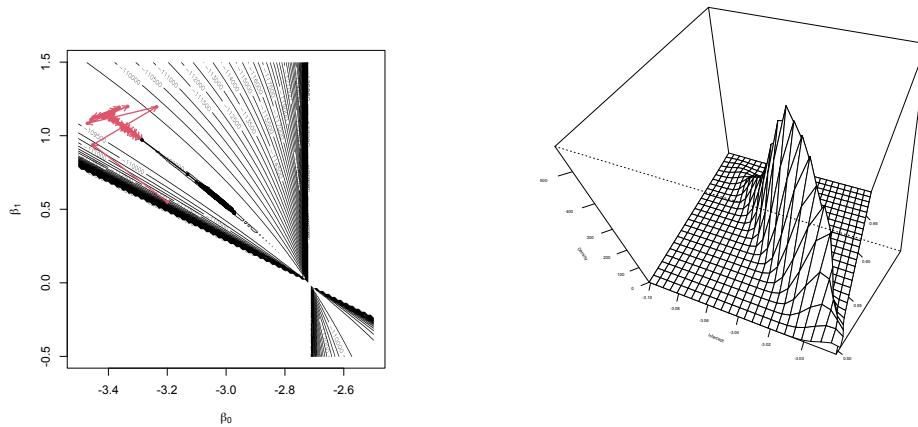


Figure 3: Contour plot of the non-normalised log posterior with the HMC sampling path (left) and density plot (right) of the samples for the constrained logistic regression problem.

```
+ result[1, 2, ] <- result[1, 1, ] * X[, 1]
+ result[2, 1, ] <- result[1, 2, ]
+ result[2, 2, ] <- result[1, 2, ] * X[, 1]
+ result[3, , ] <- 0
+
}
```

We choose independent $N(0, 100)$ priors for both β_0 and β_1 :

```
Function: prior
R> pr <- function(x) {
+   - 0.5*t(x)%*%x/10^4 - log(2*pi*10^4)
+ },
Function: dprior
R> dpr <- function(x) {
+   -x * 10 ^ (-4)
+ },
```

where `pr` is the prior and `dpr` is the gradient of the log prior for β .

Our goal is to use ELHMC to draw samples of $\beta = (\beta_0, \beta_1)$ from their resulting posterior based on empirical likelihood.

We start our sampling from $(-3.2, 0.55)$ and use two stages of sampling. In the first stage, 50 points are sampled with $\epsilon = 0.001$, $T = 15$, and the momentum generated from a $N(0, 0.02 \cdot I_2)$ distribution. The acceptance rate at this stage is very high, but it is designed to find a good starting point for the second stage, where the acceptance rate can be easily controlled.

```
R> bstart.init=c(-3.2,.55)
R> betas.init <- ELHMC(initial = bstart.init, data = data, FUN = g, DFUN = dg,
+                           n.samples = 50, prior = pr, dprior = dpr, epsilon = 0.001,
+                           lf.steps = 15, detailed = T, p.variance = 0.2)
```

In this second stage, we draw 500 samples of β with starting values as the last value from the first stage. The number of leapfrog steps per Monte Carlo update is set to 30, with a step size of 0.004 for both β_0 and β_1 . We use $N(0, 0.02(I_2))$ as prior for the momentum.

```
R> bstart=betas.init$samples[50,]
```

```
R> betas <- ELHMC(initial = bstart, data = data, fun = g, dfun = dg,
+                     n.samples = 500, prior = pr, dprior = dpr, epsilon = 0.004,
+                     lf.steps = 30, detailed = FALSE, p.variance = 0.2, print.interval=10,
+                     plot.interval=1, which.plot=c(1))
```

Based on our output, we can make inferences about β . As an example, the autocorrelation plots and the density plot of the last 1000 samples of β are shown in Figure 3.

```
R> library(MASS)
R> beta.density <- kde2d(betas$sample[, 1], betas$sample[, 2])
R> persp(beta.density, phi = 50, theta = 20,
+         xlab = 'Intercept', ylab = '', zlab = 'Density',
+         ticktype = 'detailed', cex.axis = 0.35, cex.lab = 0.35, d = 0.7)
R> acf(betas$sample[round(n.samp/2):n.samp, 1],
+       main=expression(paste("Series ",beta[0])))
R> acf(betas$sample[round(n.samp/2):n.samp, 2],
+       main=expression(paste("Series ",beta[1])))
```

It is well known (Chaudhuri et al., 2008) that the constrained estimates of β_0 and β_1 have very low standard error. The acceptance rate is close to 78%. It is evident that our software can sample from such a narrow ridge with ease. Furthermore, the autocorrelation of the samples seems to decrease very quickly with the lag, which would not be the case for most other MCMC procedures.

Acknowledgement

Dang Trung Kien would like to acknowledge the support of MOE AcRF R-155-000-140-112 from the National University of Singapore. Sanjay Chaudhuri acknowledges the partial support from NSF-DMS grant 2413491 from the National Science Foundation USA. The authors are grateful to Professor Michael Rendall, Department of Sociology, University of Maryland, College Park for kindly sharing the data set on which the second example is based.

References

- W. Bergsma, M. Croon, L. A. van der Ark, et al. The empty set and zero likelihood problems in maximum empirical likelihood estimation. *Electronic Journal of Statistics*, 6:2356–2361, 2012. [p237]
- J. Besag. Markov chain monte carlo methods for statistical inference. 2004. [p244]
- C. K. Birdsall and A. B. Langdon. *Plasma physics via computer simulation*. CRC Press, 2004. [p244]
- S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004. [p242]

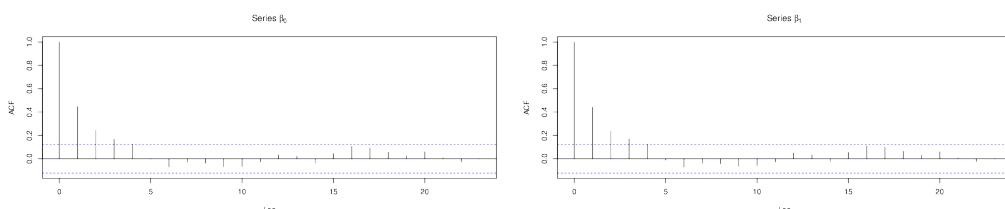


Figure 4: The autocorrelation function of the samples drawn from the posterior of β .

- 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. doi: 10.18637/jss.v076.i01. URL <https://www.jstatsoft.org/v076/i01>. [p238]
- S. Chaudhuri, M. S. Handcock, and M. S. Rendall. Generalized linear models incorporating population level information: an empirical-likelihood-based approach. *Journal of the Royal Statistical Society series B*, 70:311–328, 2008. [p249, 250, 252]
- S. Chaudhuri, D. Mondal, and T. Yin. Hamiltonian monte carlo sampling in Bayesian empirical likelihood computation. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79(1):293–320, 2017. ISSN 1467-9868. doi: 10.1111/rssb.12164. URL <http://dx.doi.org/10.1111/rssb.12164>. [p238, 240, 241, 242, 245, 247, 249]
- J. Chen, A. Variyath, and B. Abraham. Adjusted empirical likelihood and its properties. *Journal of Computational and Graphical Statistics*, 17(2):426–443, 2008. [p237]
- S. C. Emerson, A. B. Owen, et al. Calibration of the empirical likelihood method for a vector mean. *Electronic Journal of Statistics*, 3:1161–1192, 2009. [p237]
- S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6): 721–741, 1984. [p239]
- M. Grendár and G. Judge. Empty set problem of maximum empirical likelihood methods. *Electronic Journal of Statistics*, 3:1542–1555, 2009. [p237]
- H. Haario, E. Saksman, and J. Tamminen. Adaptive proposal distribution for random walk Metropolis algorithm. *Computational Statistics*, 14(3):375–396, 1999. [p241]
- N. Lazar. Bayesian empirical likelihood. *Biometrika*, 90(2):319–326, 2003. [p237]
- B. Leimkuhler and S. Reich. *Simulating hamiltonian dynamics*, volume 14. Cambridge University Press, 2004. [p244]
- Y. Liu, J. Chen, et al. Adjusted empirical likelihood with high-order precision. *The Annals of Statistics*, 38(3):1341–1362, 2010. [p237]
- R. Neal. MCMC for using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, pages 113–162, 2011. [p244]
- J. Qin and J. Lawless. Empirical likelihood and general estimating equations. *The Annals of Statistics*, pages 300–325, 1994. [p242]
- M. Tsao. Extending the empirical likelihood by domain expansion. *Canadian Journal of Statistics*, 41(2):257–274, 2013. [p238]
- M. Tsao and F. Wu. Empirical likelihood on the full parameter space. *The Annals of Statistics*, 41(4):2176–2196, 2013. [p238]
- M. Tsao and F. Wu. Extended empirical likelihood for estimating equations. *Biometrika*, 101(3):703–710, 2014. doi: 10.1093/biomet/asu014. URL <http://dx.doi.org/10.1093/biomet/asu014>. [p238]
- W. Yu and J. Lim. *VBel: Variational Bayes for Fast and Accurate Empirical Likelihood Inference*, 2024. URL <https://CRAN.R-project.org/package=VBel>. R package version 1.1.0. [p238]
- M. Zhou. *emplik: Empirical likelihood ratio for censored/truncated data*, 2014. URL <http://CRAN.R-project.org/package=emplik>. R package version 0.9-9-2. [p247]

Neo Han Wei
Citibank, Singapore
nhanwei@gmail.com

Dang Trung Kien
Independent Consultant
trungkiendang@hotmail.com
<http://www.stat.nus.edu.sg/>

Sanjay Chaudhuri
University of Nebraska-Lincoln
Department of Statistics
840 Hardin Hall North Wing, Lincoln, NE, USA
schaudhuri2@nebraska.edu
<http://www.stat.nus.edu.sg/>

GeRnika: An R Package for the Simulation, Visualization and Comparison of Tumor Phylogenies

by A. Sánchez-Ferrera*, M. Tellaetxe-Abete*, B. Calvo-Molinos

Abstract The development of methods to study intratumoral heterogeneity and tumor phylogenies is a highly active area of research. However, the advancement of these approaches often necessitates access to substantial amounts of data, which can be challenging and expensive to acquire. Moreover, the assessment of results requires tools for visualizing and comparing tumor phylogenies. In this paper, we introduce GeRnika, an R package designed to address these needs by enabling the simulation, visualization, and comparison of tumor evolution data. In summary, GeRnika provides researchers with a user-friendly tool that facilitates the analysis of their approaches aimed at studying tumor composition and evolutionary history.

1 Introduction

The development of tumors is a complex and dynamic process characterized by a succession of events where DNA mutations accumulate over time. These mutations give rise to genetic diversity within the tumor, leading to the emergence of distinct clonal subpopulations or, simply, clones (Nowell, 1976). Each of these subpopulations exhibits unique mutational profiles, resulting in varied phenotypic and behavioral characteristics among the cancer cells (Marass et al., 2016). This phenomenon, known as intratumoral heterogeneity (ITH), significantly hinders the design of effective medical therapies, since different clones within the same tumor may respond differently to treatments, ultimately leading to therapy resistance and disease recurrence (Burrell et al., 2013). To address this challenge, several innovative approaches are being developed to study the tumor composition in greater detail and to reconstruct its evolutionary history.

One common approach to studying tumor composition and phylogeny involves using bulk DNA sequencing data from multiple tumor biopsies. This data is relatively straightforward to obtain and provides a broad overview of the genetic alterations within the tumor. However, using bulk sequencing data in the study of ITH faces the challenge that each sample potentially contains a mixture of different clonal populations rather than just one clone. Consequently, the observed mutation frequencies—measured as variant allele frequencies (VAFs)—do not directly estimate the fraction of individual clones. Instead, the VAF values represent a composite signal: the sum of the fractions of all clones that harbor each mutation in a given sample.

This complexity implies that reconstructing the tumor's evolutionary history requires deconvolving these clonal admixtures within the samples. This task is precisely the focus of the Clonal Deconvolution and Evolution Problem (CDEP) (Strino et al., 2013; El-Kebir et al., 2015), which can be summarized as determining the tumor's clonal structure—that is, identifying the number, proportion, and mutational composition of clones in each sample—as well as reconstructing the clonal phylogenetic tree that leads to the observed clonal mosaic. In this context, one of the most prominent approaches to addressing the CDEP is the Variant Allele Frequency Factorization Problem (VAFFP) (El-Kebir et al., 2015).

Given s tumor samples and n mutations identified across these samples, we define a matrix $F \in [0, 1]^{s \times n}$, where each element f_{ij} represents the variant VAF value, or equivalently, the fraction of cells that carry mutation j in sample i . The VAFFP seeks to decompose this input matrix F into two matrices: a matrix $B \in \{0, 1\}^{n \times n}$ that represents the clonal

*These authors contributed equally to this work.

phylogeny, and a matrix $\mathbf{U} \in [0, 1]^{s \times n}$ that captures the clone proportions in each tumor sample:

$$\mathbf{F} = \mathbf{U} \cdot \mathbf{B} \quad (1)$$

The \mathbf{B} matrix is a binary square matrix of size n , where $b_{ij} = 1$ iff clone i contains the mutation j (Gusfield, 1991). The matrix \mathbf{U} is an $s \times n$ matrix where u_{ij} is the fraction of clone j in sample i .

The VAFFP operates under two key assumptions: tumors have a monoclonal origin, meaning they arise from a single abnormal cell, and the infinite sites assumption (ISA), which states that mutations occur at most once and cannot disappear over time (Kimura, 1969). Under these assumptions, the tumor's clonal structure can be modeled as a perfect phylogeny (Gusfield, 1991). This model imposes two key constraints: (1) if two clones share a mutation, they must either be identical or ancestrally related, and (2) once a clone acquires a mutation, that mutation is inherited by all its descendants.

Over the years, numerous methods have been developed to solve the CDEP (Sengupta et al., 2014; Yuan et al., 2015; Deshwar et al., 2015; El-Kebir et al., 2015; Malikic et al., 2015; Popic et al., 2015; El-Kebir et al., 2016; Jiang et al., 2016; Husić et al., 2019; Fu et al., 2022; Grigoriadis et al., 2024), with recent advancements addressing reformulations of the problem that incorporate single-cell sequencing-derived information or variants in metastases, and account for temporal resolution (Ross and Markowetz, 2016; El-Kebir et al., 2018; Malikic et al., 2019; Satas et al., 2020; Sollier et al., 2023). These methods primarily focus on reconstructing tumor phylogenies and clonal compositions using both real and simulated data. However, the simulation tools they employ often have limitations. For instance, while MiPUP uses simulated data, it allows only basic parameter adjustments—such as the number of mutations, samples, and reads—and lacks the flexibility to fine-tune biological parameters. BitPhylogeny creates highly realistic and complex simulated datasets representing different modes of evolution, but these simulations are manually crafted and limited in number, posing scalability issues.

Several tools specifically designed for simulating data for the CDEP have also been introduced. Pearsim, written in Python, allows control over parameters like read depth, number of subclones, samples, and mutations (Kulman et al., 2022). OncoLib, a C++ library, facilitates the simulation of tumor heterogeneity and the reconstruction of NGS sequencing data of metastatic tumors, offering control over parameters such as driver mutation probability, per-base sequencing error rate, migration rate, and mutation rate (Qi et al., 2019). Machina, also in C++, provides a framework for simulating metastatic tumors and visualizing their phylogenetic trees and migration graphs (El-Kebir et al., 2018). HeteroGenesis, implemented in Python, simulates heterogeneous tumors at the level of clone genomes, but it is not specifically tailored for CDEP instances and requires additional processing to generate suitable datasets (Tanner et al., 2019).

Visualization of tumor phylogenies is another crucial aspect of studying tumor evolution. Tools like CALDER, Clonewol, SPRUCE, **fishplot**, ClonArch, and **clevRvis** offer solutions for visualizing clonal structures and evolutionary trajectories (Myers et al., 2019; Dang et al., 2017; El-Kebir et al., 2016; Miller et al., 2016; Wu and El-Kebir, 2020; Sandmann et al., 2023). Among these, **fishplot** and **clevRvis** are available as R packages. Additionally, methods for generating consensus trees, such as TuELIP and ConTreeDP, have been developed to summarize multiple phylogenetic trees into a single representative tree (Guang et al., 2023; Fu and Schwartz, 2021).

Despite the availability of existing tools, there remains a lack of options in the R programming environment for realistically simulating tumor evolution in a way that is both flexible and user-friendly, while also enabling effective visualization and comparison.

In this paper, we introduce **GeRnika**, an R package that provides a comprehensive solution for simulating, visualizing, and comparing tumor evolution data. Although **GeRnika**'s data simulation functionality was primarily devised to create instances for solving the CDEP, the simulated data are not restricted to this purpose and can also be used for exploring

evolutionary dynamics in broader contexts. To accommodate diverse research needs, we have implemented the procedures to be highly customizable, allowing users to adjust a wide range of parameters such as the number of clones, selective pressures, mutation rates, and sequencing noise levels. Unlike existing tools that may offer limited customization or are implemented in other programming languages, **GeRnika** is fully integrated into the R environment, making it easy to use alongside other bioinformatics packages. By combining simulation capabilities with visualization and comparison tools in a user-friendly interface, **GeRnika** offers an accessible and flexible option within the R ecosystem for researchers studying tumor evolution. It is important to note that **GeRnika** does not implement algorithms for inferring clonal composition or reconstructing phylogenies from experimental datasets. Instead, it provides a controlled framework for generating, visualizing, and comparing data that can be used to benchmark such methods.

2 Simulation of tumor evolution

The main contribution of this work is the introduction of a novel approach for simulating biologically plausible instances of the VAFFP that accounts for several key factors, including the number of clones, selective pressures, and sequencing noise. In this section, we provide a detailed description of the approach.

Broadly speaking, each problem instance consists of a matrix F containing the VAF values of a set of mutations in a set of samples, as described previously. This matrix is built from a pair of matrices B and U that represent a tumor phylogeny fulfilling the ISA and the proportions of the clones in the samples, respectively, following Equation (1).

In order to simulate the B and U matrices, we have devised two models: a tumor model that simulates the evolutionary history and current state of the tumor, and a sampling model that represents the tumor sampling process. A third model, namely the sequencing noise model, has been devised to optionally introduce sequencing noise to the VAF values in the F matrix, if noisy data is desired. The following subsections describe these models in detail.

2.1 Tumor model

The tumor model generates a clonal tree T and an associated matrix B , together with the clone proportions c and tumor blend at the moment of sampling. Briefly, for a tumor with a set of n mutations denoted by M , T is a rooted tree on an n -sized vertex set $V_n = \{v_1, \dots, v_n\}$, where v_i represents clone i and simultaneously corresponds to the first clone containing mutation M_i . This one-to-one correspondence between clones and mutations allows us to refer to them interchangeably. The tree is further defined by an $(n - 1)$ -sized edge set E_T , where each edge $e_{ij} \in E_T$ represents a direct ancestral relationship from vertex v_i to vertex v_j .

In our tumor model, T is iteratively generated with a random topology, as follows. First, the root node of T , $\mathcal{R}(T)$, is set, and a random mutation $M_i \in M$ is assigned to it. For each of the remaining $M_j \in M - \{M_i\}$ mutations, a new node v_j is created and the mutation M_j is assigned to this node. The node v_j is then attached as a child to one of the nodes already included in T . To adhere to the ISA model, each newly added node inherits all the mutations present in its parent node.

The attachment of nodes to the tree is not uniformly random. Instead, the nodes in the growing tree T have different probabilities of being selected as parents for the new nodes, depending on the number of descendants, $\mathcal{A}(v_i)$, they have. Specifically, $\forall v_j \neq \mathcal{R}(T)$, the parent node of v_j is sampled from a multinomial distribution where the probabilities are calculated as:

$$p(v_i; k) = \frac{k^{\frac{|\mathcal{A}(v_i)|+1}{\delta}}}{\sum_{v_l \in V'} k^{\frac{|\mathcal{A}(v_l)|+1}{\delta}}}; \quad v_i \in V' \quad (2)$$

Here, δ represents the depth of the growing tree, i.e., the number of levels or layers in the tree structure. $k \in (0, +\infty)$ is the topology parameter that determines whether the topology tends to be branched, with a decreasing probability for increasing numbers of ascendants ($k < 1$), or linear, with an increasing probability for increasing numbers of ascendants ($k > 1$).

Once T has been generated, it is represented in the form of a B matrix, constructed by initializing an identity matrix B_n and setting b_{ji} to 1 for each pair of nodes v_i and v_j where node v_j is a descendant of node v_i in T .

After obtaining B , the proportions of the clones in the whole tumor, denoted as $c = \{c_1, \dots, c_n\}$, are simulated. It is important to note that these proportions are not the same as those appearing in the U matrix, which represent the *sampled* clone proportions and depend not only on the global clone proportions but also on the spatial distribution of the clones and the sampling sites.

These clone proportions c are calculated by sequentially sampling a Dirichlet distribution at each multifurcation in T , starting from the root. For instance, for a node v_i with children $\mathcal{K}(v_i) = \{v_j, v_k\}$, we draw a sample (x_i, x_j, x_k) that represents the proportions of the parent clone and its two children, respectively, from a Dirichlet distribution $Dir(\alpha_i, \alpha_j, \alpha_k)$. When this sampling is performed at a node $v_i \neq \mathcal{R}(T)$, these proportions are scaled relative to the original proportion of the parent clone. This ensures that the sum rule is met, and that once all multifurcations have been visited, the proportions of all clones in T sum up to one. While several approaches can exist to determine these proportions, this method provides a natural approximation to the problem that can be interpreted as the distribution of the mass or proportion of each clone between itself and its descendants.

The parameters of the Dirichlet distribution depend on the tumor's evolution model. In this work, we consider two fundamental cases: positive selection-driven evolution and neutral evolution. In positive selection-driven evolution, certain mutations confer a growth advantage, while most mutations do not. As a result, the clones carrying these advantageous mutations outcompete other clones and dominate the tumor. Consequently, tumors are predominantly composed of a few dominant clones, with the remaining clones present in too small proportions. Under neutral evolution, instead, there is no significant number of mutations that provide a fitness advantage, and clones accumulate solely due to tumor progression. As a result, all clones are present in similar proportions (Davis et al., 2017).

Based on this, all the parameters for the Dirichlet distribution for positive selection-driven evolution are set to 0.3. For neutral evolution, the parameter corresponding to the parent node (α_p) is set to 5, and the parameters corresponding to the children node(s) (α_c) are set to 10. Different alpha values are used for parent and children nodes in neutral evolution to ensure that clones arising late in the evolution do not end up with proportions that are too small solely due to their position in the topology, preventing the deviation from the expected clone proportion distribution for this type of evolution model. These values have been chosen empirically, and their effect is illustrated in Figure 1, which shows how 5,000 random samples from the mentioned Dirichlet distributions (for the particular case of 3 dimensions, i.e., one parent and two children nodes) are distributed. As observed, in the case of positive selection ($\alpha_i = \alpha_j = \alpha_k = 0.3$), the (x_i, x_j, x_k) values are equally pushed towards the three corners of the simplex. In other words, the samples tend to be sparse, with typically one component having a large value and the rest close to 0. Instead, when neutral selection is adopted ($\alpha = (5, 10, 10)$), the (x_i, x_j, x_k) values concentrate close to the center of the simplex, but with a tendency to deviate towards those components with larger α value. This means that samples (x_i, x_j, x_k) are less sparse in neutral evolution, with larger values for x_2 and x_3 in this case, which represent the children nodes.

Taking into account that marginalizing the Dirichlet distribution results in a Beta distribution, the proportion of the clone $v_i \in V_n \mid v_i \neq \mathcal{R}(T)$ in the tumor, denoted as C_i , follows the distribution:

$$C_i \sim C_{\mathcal{P}(v_i)} \cdot \Gamma_i \cdot \Gamma'_i \quad (3)$$

where

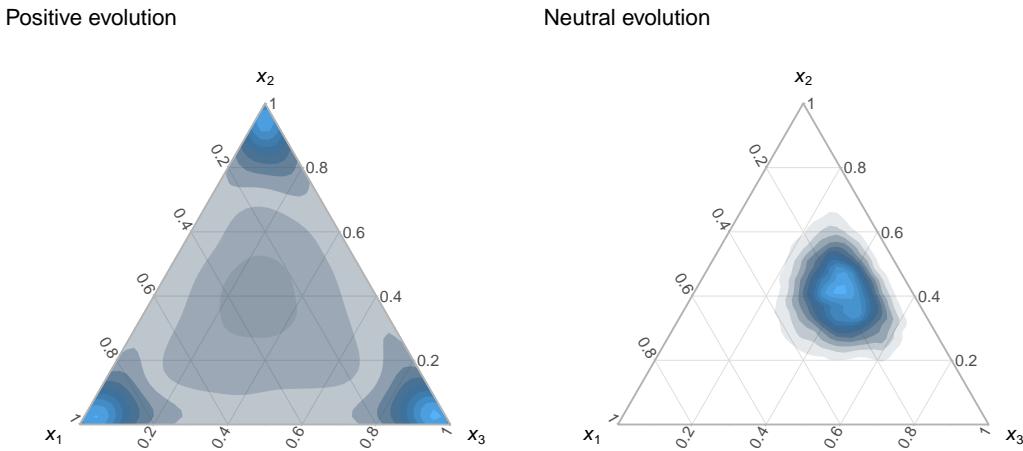


Figure 1: Ternary density plots of 5,000 samples drawn from two 3-dimensional Dirichlet distributions. The parameters of the Dirichlet distribution on the left are $\alpha = (0.3, 0.3, 0.3)$ and the distribution is used to represent positive selection-driven evolution. The distribution on the right has parameters $\alpha = (5, 10, 10)$ and is used to represent neutral evolution. Samples drawn from these distributions (or their generalization to higher spaces) are used to calculate clone proportions in each tree multifurcation.

$$\Gamma_i \sim Beta(\alpha_c, \alpha_p + \alpha_c \cdot (|\mathcal{K}(\mathcal{P}(v_i))| - 1)) \quad (4)$$

and

$$\Gamma'_i = 1 \quad \text{if } |\mathcal{K}(v_i)| = 0 \quad (5)$$

$$\Gamma'_i \sim Beta(\alpha_p, \alpha_c \cdot |\mathcal{K}(v_i)|) \quad \text{if } |\mathcal{K}(v_i)| \neq 0 \quad (6)$$

For the case where $v_i = \mathcal{R}(T)$, the root node, C_i , follows:

$$C_i \sim Beta(\alpha_p, \alpha_c \cdot |\mathcal{K}(v_i)|) \quad (7)$$

Here, α_p and α_c are the parameters of the Dirichlet distribution assigned to parent and child nodes, respectively.

To complete the tumor model, the tumor blend is simulated, which represents the degree of physical mixing between the tumor clones. In order to do this, we simplify the spatial distribution to one dimension and model the tumor as a Gaussian mixture model with n components, where each component G_i represents a tumor clone, and the mixture weights are given by c . The variance for all components is set to 1, while the mean values are random variables.

Specifically, we start by selecting a random clone, and its component's mean value is set to 0. Then, the mean values of the remaining $n - 1$ components are calculated sequentially by adding d units to the mean value of the previous component. To introduce variability in the tumor blend, the value of d is chosen from the set $\{0, 0.1, \dots, 4\}$. For $d = 0$, the two clones are completely mixed, while for $d = 4$, they are physically far apart from each other. The choice of the upper limit for d has been determined empirically, considering that with this value, the overlapping area between the two clones becomes negligible.

To ensure the separation between the clones is random and that most of the time the separation is small, we use an exponential-like distribution with the form $Beta(\alpha = 1, \beta = X)$ to sample the values of d . Specifically, we set $\beta = 5$ to ensure that the samples obtained from the mixture are not excessively sparse. We can express this mathematically as:

$$D \sim 4 \cdot Beta(\alpha = 1, \beta = 5) \quad (8)$$

2.2 Sampling simulation

So far, we have described how the clones of a tumor are modelled by the tumor model. However, in real practice, there is no easy way of observing these global properties of a tumor. Instead, we typically have access to information provided by samples or biopsies. This means that certain tumor characteristics, such as the real clone proportions c , cannot be directly obtained. Instead, we can only determine the *sampled* clone proportions, which depend on the specific sampling procedure employed. Unless there is a perfectly uniform mixture of the clone cells, their sampled proportions will not match the global proportions. These sampled clone proportions are, in fact, the u_i . elements in the \mathbf{U} matrix.

The sampling simulation we have devised simulates the physical sampling of the tumor and allows us to construct the \mathbf{U} matrix of the problem. This procedure operates on the data simulated using the tumor model. Specifically, it simulates a sampling procedure carried out in a grid manner over the tumor Gaussian mixture model described in the previous section. Let G_1 and G_n be the components with the lowest and largest mean values, respectively, in the Gaussian mixture model. The 1st and m^{th} sampling points in the grid are always set to $\mu_{G_1} - 2.8 \cdot \sigma_{G_1}$ and $\mu_{G_n} + 2.8 \cdot \sigma_{G_n}$, respectively, and the remaining $m-2$ sampling points are determined by dividing the range between these two endpoints into $m-1$ equal intervals.

The densities of the Gaussian distributions at each sampling point are multiplied by the global proportion of the clones sampled from the Dirichlet distributions, so that for each sampling point i , the fraction of clone j , p_{ij} , is proportional to their product:

$$p_{ij} \propto c_j \cdot \phi_{ij} \quad (9)$$

where c_j is the global proportion of clone j and ϕ_{ij} is the density of the Gaussian component associated with clone j at sampling point i .

Finally, to account for the effect of cell count in the samples, a multinomial distribution is used to sample a given number of cells n_c for each tumor sample. In that distribution, the probability of selecting each clone at sampling site i is given by (p_{i1}, \dots, p_{in}) . The resulting values determine the final tumor clone composition in sample i , which are represented in the matrix \mathbf{U} :

$$U_i \sim \frac{M(n = n_c, p = (p_{i1}, \dots, p_{in}))}{100} \quad (10)$$

Note that selecting a relatively low value for n in the multinomial distribution can lead to clones with very low frequencies being modeled as absent in the sample, with composition values equal to 0. This is indeed more realistic than truly observing them with such low frequencies.

2.3 Sequencing noise simulation

Up to this point, the \mathbf{B} and \mathbf{U} matrices of an instance have been simulated. In case we are simulating noise-free data, the simulation is complete once Equation (1) is applied to obtain the \mathbf{F} matrix.

As a brief reminder, each element f_{ij} in \mathbf{F} denotes the frequency or VAF of the mutation M_j in sample i or, in other words, the proportion of sequencing reads that carry the mutation M_j in that particular sample. This also means that the proportion of reads in that sample that do not observe the mutation but instead contain the reference nucleotide is $1 - f_{ij}$.

However, empirical factors can artificially alter the VAF value, leading it to deviate from the true ratio between the variant and total allele molecule counts. One of these factors is the noise introduced during the DNA sequencing process itself, which can arise in two

main ways. First, limitations of the sequencing instrument can lead to incorrect nucleotide readings of DNA fragments. For example, a position that actually contains nucleotide A may be read as a T. Second, there can be a biased number of reads produced for a particular site, which can result from chemical reaction peculiarities or simply because not all fragments are sequenced. These limitations can, however, be mitigated to some extent. For instance, it has been shown that a high depth of coverage, which refers to the average number of reads that cover each position, can lead to more accurate VAF values (Petrackova et al., 2019).

In order to incorporate the effect of sequencing noise in the data instances, we have developed a procedure to simulate sequencing noise. This procedure introduces noise to the F matrix and generates a noisy matrix $F^{(n)}$, where $F^{(n)} \neq U \cdot B$. The procedure simulates noise at the level of the sequencing reads and recalculates the new $f_{ij}^{(n)}$ values, as follows.

The sequencing depth r at the genomic position where M_j occurs in sample i is distributed according to a negative binomial distribution:

$$r_{ij} \sim NB(\mu = \mu_{sd}, \alpha = 5) \quad (11)$$

where μ_{sd} represents the mean sequencing depth, which is the average number of reads covering the genomic position of mutation M_j in the sample, and α is the dispersion parameter, which controls the variability of the sequencing depth around the mean and is fixed at 5.

The number of reads supporting the alternate allele r_{ij}^a is then modeled by a binomial distribution:

$$r_{ij}^a \sim B(n = r_{ij}, p = f_{ij}) \quad (12)$$

In sequencing data, errors can occur due to limitations inherent to the sequencing methodology. These errors vary depending on the technology used.

To simulate the effect of these errors on the VAF values, the number of reads $r_{ij}^{a'}$ that, despite originally supporting the alternate allele, contain a different allele as a result of a sequencing error, is modeled using a binomial distribution:

$$r_{ij}^{a'} \sim B(n = r_{ij}^a, p = \varepsilon), \quad (13)$$

where ε represents the sequencing error rate.

We also need to consider the situation where the reads contain the reference nucleotide but are read with the alternate allele as a result of this error. This can be better understood with an example. Let's imagine that at a certain genomic position, the normal cells have a T, but in some cells, there is a mutation where the T has changed to an A. In this case, for the normal cells, with a rate of ε , a sequencing error may occur, resulting in a read of C, G, or A instead of T, each with an equal chance. Therefore, in approximately $\frac{\varepsilon}{3}$ of the cases, reads with the mutation of interest will arise from normal reads:

$$r_{ij}^{r'} \sim B(n = r_{ij} - r_{ij}^a, p = \frac{\varepsilon}{3}) \quad (14)$$

Thus, taking all these into consideration, the final noisy VAF values $f_{ij}^{(n)}$ are simulated as:

$$f_{ij}^{(n)} = \frac{r_{ij}^a - r_{ij}^{a'} + r_{ij}^{r'}}{r_{ij}} \quad (15)$$

By default, the sequencing error rate ε is set to 0.001, following commonly reported values for Illumina data (Loman et al., 2012).

As an illustration of the effect of the noise model, in Figure 2, we have depicted the density of the mean absolute error between the $F^{(n)}$ matrix and its corresponding noise-free

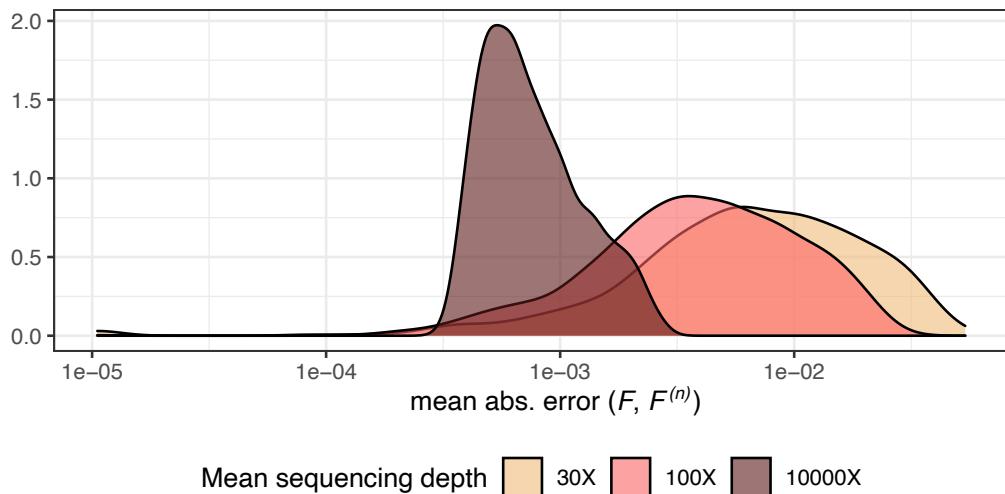


Figure 2: Density of the mean absolute error in noisy F matrices for different μ_{sd} values that correspond to different noise levels.

F matrix for a collection of noisy instances. As can be seen, as μ_{sd} increases, the error introduced to the $F^{(n)}$ matrix decreases. This is expected because the r_{ij}^a values follow a binomial distribution as described in Equation (12), where the number of trials is determined by μ_{sd} as shown in Equation (11), and the event probability corresponds to the $f_{ij}^{(n)}$ value. Therefore, the larger the number of trials, the closer the noisy VAF value is to the noise-free VAF value.

As a final remark, it is important to note that although our data simulation procedure follows the ISA, the addition of noise may cause the resulting data to break this assumption.

3 The package

GeRnika provides three main functionalities for studying tumor evolution data: (I) simulating artificial tumor evolution, (II) visualizing tumor phylogenies, and (III) comparing tumor phylogenies. This section explains the functions that support these features. Additionally, we describe extra data provided by **GeRnika** that users can use to try the methods in the package.

3.1 Simulation methods

To enable users to simulate tumor evolution data, **GeRnika** provides various functions inspired by the methods described in section 2.2. GeRnika offers two options: a single method for streamlined simulations and separate methods for performing each step individually, allowing users to customize or replace specific parts of the process.

`create_instance`

The main function for streamlined tumor data simulation is `create_instance`. This function provides a convenient way to perform the entire simulation process in a single step. The following command demonstrates how to use it to generate the artificial data:

```
create_instance(n, m, k, selection, noisy = TRUE, depth = 30, seed = Sys.time())
```

where each argument of the method is described as follows:

- `n`: An integer representing the number of clones.

- m : An integer representing the number of samples.
- k : A numeric value that determines the linearity of the tree topology. Also referred to as the topology parameter. Increasing values of this parameter increase the linearity of the topology. When k is set to 1, all nodes have equal probabilities of being chosen as parents, resulting in a uniformly random topology.
- **selection**: A character string representing the evolutionary mode the tumor follows. This should be either "positive" or "neutral".
- **noisy**: A logical value (TRUE by default) indicating whether to add noise to the frequency matrix. If TRUE, noise is added to the frequency matrix. If FALSE, no noise is added.
- **depth**: A numeric value (30 by default) representing the mean depth of sequencing.
- **seed**: A numeric value (Sys.time() by default) used to set the seed for the random number generator.

The `create_instance` function returns a list containing the following components:

- **F_{noisy}** : A matrix representing the noisy frequencies of each mutation across samples. If the `noisy` parameter is set to FALSE, this matrix is equal to F_{true} .
- **B** : A matrix representing the relationships between mutations and clones in the tumor.
- **U** : A matrix representing the frequencies of the clones across the set of samples.
- **F_{true}** : A matrix representing the noise-free frequencies of each mutation across the samples.

As explained, the `create_instance` function generates all matrices representing frequencies, proportions, and the phylogeny of the simulated tumor data in a single step. However, **GeRnika** also provides individual functions for simulating each of these elements independently, providing users with greater control over the characteristics of the simulated tumor data.

create_B, create_U, create_F and add_noise

These methods provide specialized functions to generate each matrix involved in representing tumor evolution data. These functions include the following:

- **`create_B`**: This function generates a mutation matrix (B matrix) for a tumor phylogenetic tree with a given number of nodes and a value k determining the linearity of the tree topology.
- **`create_U`**: This function calculates the U matrix, containing the frequencies of each clone in a set of samples, based on a B matrix, the number of samples considered, the number of cells in each sample, and the evolutionary mode of the tumor.
- **`create_F`**: This function generates the F matrix, which contains mutation frequency values for a series of mutations across a collection of tumor biopsies or samples. The matrix is computed based on a pair of matrices, U and B , and considers whether the mutations are heterozygous.
- **`add_noise`**: This function introduces sequencing noise into the noise-free F matrix generated by the `create_F` method. Users can specify the mean sequencing depth and the overdispersion parameter, which are used to simulate sequencing depth based on a negative binomial distribution.

The reader is encouraged to refer to the package documentation for more information about these functions and their parameters.

3.2 Visualization methods

The following functions enable the visualization of tumor evolution data by generating phylogenetic trees based on the data under analysis.

Phylotree S4 class

To simplify the execution of its functionalities, **GeRnika** utilizes the "Phylotree" class. The "Phylotree" S4 class is a data structure specifically designed to represent phylogenetic trees, facilitating the use of the package's methods and ensuring their computational efficiency. The attributes of the "Phylotree" class are as follows:

- **B**: A `data.frame` containing the square matrix that represents the ancestral relationships among the clones in the phylogenetic tree (**B** matrix).
- **clones**: A vector representing the indices of the clones in the **B** matrix.
- **genes**: A vector indicating the index of the gene that firstly mutated in each clone within the **B** matrix.
- **parents**: A vector indicating the parent clones for each clone in the phylogenetic tree.
- **tree**: A "Node" class object representing the phylogenetic tree (this class is inherited from the `data.tree` package).
- **labels**: A vector containing the gene tags associated with the nodes in the phylogenetic tree.

A customized "Phylotree" class object can be instantiated with custom attributes using the `create_instance` method. This method takes all the attributes according to the "Phylotree" class as arguments. Alternatively, **GeRnika** provides a function that automatically generates a "Phylotree" class object on the basis of a given **B**.

B_to_phylotree

In order to instantiate an object of the "Phylotree" class, the following command can be used:

```
B_to_phylotree(B, labels = NA)
```

where each argument of the method is described as follows:

- **B**: A square **B** matrix that represents the phylogenetic tree.
- **labels**: An optional vector containing the tags of the genes in the phylogenetic tree. **NA** by default.

This function returns an object of the "Phylotree" class, automatically generating its attributes based on **B**, which represents the phylogenetic tree of the tumor under analysis.

Once instantiated, the phylogenetic tree in a "Phylotree" class object can be visualized using the generic `plot` function, which takes the "Phylotree" object as its argument. The `plot` function also includes a `labels` argument that can be set to `TRUE` to display node labels on the phylogenetic tree, using the gene tags stored within the "Phylotree" object.

The **GeRnika** package provides the `plot_proportions` function for visualizing phylogenetic trees, with node sizes and colors reflecting the proportions of each clone. This function requires two inputs: a "Phylotree" class object representing the phylogenetic tree and a numeric vector or matrix specifying clone proportions. If a vector is provided, a single tree is plotted, with the node sizes and colors determined by the values in the vector. Instead, if

a matrix is provided, such as the \mathbf{U} matrix that represents the frequencies of clones across samples, the function plots one tree for each row of the matrix. Each tree is generated based on the clone proportions specified in the corresponding row. Additionally, users can enable node labeling by setting the `labels` argument to TRUE, which annotates the tree nodes with gene tags from the "Phylotree" object.

3.3 Comparison methods

This section describes the methods included in **GeRnika** that facilitate the comparison of tumor phylogenies.

A fundamental approach for comparing two phylogenetic trees is to determine if their evolutionary histories are equivalent. The `equals` function performs this comparison by accepting two "Phylotree" class objects as arguments. This function returns a boolean value indicating whether the provided phylogenetic trees are equivalent.

To analyze similarities and differences between two phylogenetic trees, the `find_common_subtrees` function identifies and plots all maximal common subtrees between them. In addition to visualizing these subtrees, the function outputs the number of shared and unique edges (those present in only one of the trees) and calculates the distance between the trees, defined as the sum of their unique edges. This method also includes an option to label the maximal common subtrees with gene tags by setting `labels = TRUE`.

The `combine_trees` function generates a *consensus tree* by combining the nodes and edges of two "Phylotree" class objects. The consensus tree highlights the nodes and edges that form common subtrees between the original trees, as well as the independent edges unique to each tree, which are displayed with reduced opacity. This method also allows labeling the nodes with gene tags by setting `labels = TRUE` and customizing the colors of the consensus tree by passing a 3-element hexadecimal vector to the `palette` argument.

3.4 Exported data

GeRnika provides various exported data instances to help users easily explore the package's functionalities. These are as follows:

- `B_mats`: A list of 10 trios of \mathbf{B} matrices. Each trio includes a real \mathbf{B} matrix and two \mathbf{B} matrices generated using different algorithms that infer evolutionary relationships from a given \mathbf{F} matrix. These matrices serve as illustrative examples for testing and exploring the package's functionalities.
- `palettes`: A data frame containing three predefined palettes for use with methods in **GeRnika** that require color palettes.

4 Examples

In this section we show examples of the use of the methods explained in section 2.3. Please note that in the following examples, we will set the seeds of non-deterministic methods to a predefined value to ensure reproducibility.

4.1 Simulating tumor evolution data

The simulation of tumor clonal data involves generating the matrices \mathbf{B} , \mathbf{U} , \mathbf{F} , and $\mathbf{F}^{(n)}$ associated with a specific instance. For example, we can simulate a noisy instance of a tumor composed of 5 clones/mutations, which has evolved under neutral evolution with a k value of 0.5, and from which 3 samples have been taken in a single line of code, as follows:

```
> I <- create_instance(n = 5, m = 3, k = 0.5, selection = "neutral", seed = 1)
```

```

> I

$F_noisy
      mut1      mut2      mut3      mut4      mut5
sample1 1.000 0.09090909 0.0000000 0.2777778 0.3548387
sample2 1.000 0.20000000 0.2631579 0.8536585 0.2000000
sample3 0.975 0.03846154 1.0000000 1.0000000 0.0000000

$B
      mut1 mut2 mut3 mut4 mut5
clone1 1     0     0     0     0
clone2 1     1     0     1     0
clone3 1     0     1     1     0
clone4 1     0     0     1     0
clone5 1     0     0     1     1

$U
      clone1 clone2 clone3 clone4 clone5
sample1 0.59   0.13   0.00   0.01   0.27
sample2 0.13   0.27   0.24   0.19   0.17
sample3 0.00   0.04   0.89   0.07   0.00

$F_true
      mut1 mut2 mut3 mut4 mut5
sample1 1 0.13 0.00 0.41 0.27
sample2 1 0.27 0.24 0.87 0.17
sample3 1 0.04 0.89 1.00 0.00

```

Using this approach, the previously mentioned four matrices are simulated. Note that the noise-free F matrix is referred to as `F_true` in the package's code, while the noisy $F^{(n)}$ is denoted as `F_noisy`.

The previous method allows users to generate instances easily and quickly. However, some users may require more precise control over the data, which can be achieved using the `create_B`, `create_U`, `create_F`, and `add_noise` methods. For examples on how to use these methods, please refer to the package documentation.

4.2 Visualizing tumor phylogenies

Once the matrices associated with our tumor instance have been generated, we can create a "Phylotree" class object, as follows:

```

> phylotree <- B_to_phylotree(B = I$B)
> phylotree

```

An object of class "Phylotree"

Slot "B":

```

      mut1 mut2 mut3 mut4 mut5
clone1 1     0     0     0     0
clone2 1     1     0     1     0
clone3 1     0     1     1     0
clone4 1     0     0     1     0
clone5 1     0     0     1     1

```

Slot "clones":

```
mut1 mut2 mut3 mut4 mut5
```

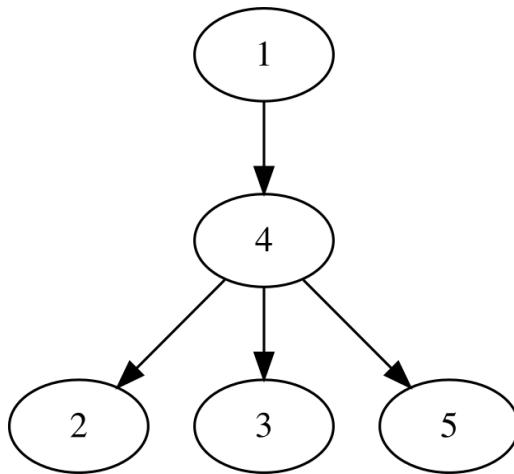


Figure 3: Phylogenetic tree associated to the generated "Phylotree" class object.

```

1   2   3   4   5

Slot "genes":
mut1 mut2 mut3 mut4 mut5
1   2   3   4   5

Slot "parents":
[1] -1  4  4  1  4

Slot "tree":
  levelName
1 1
2  °--4
3      |--2
4      |--3
5      °--5

Slot "labels":
[1] "mut1" "mut4" "mut2" "mut3" "mut5"
  
```

Since no list of tags is provided to the `labels` parameter, a default set of labels is automatically assigned to the instantiated "Phylotree" class object.

Afterwards, we can visualize the tumor phylogeny associated to the simulated B matrix by using the generic `plot` method, as follows:

```
> plot(phylotree)
```

The resulting plot is shown in Figure 3. Instead of clone numbers, the user can utilize the predefined tags in the "Phylotree" class object to label the nodes in the tree by setting the `labels = TRUE` parameter in the `plot` function.

When plotting a "Phylotree" class object, its nodes can be resized according to the proportions of the clones that compose the tumor samples. To achieve this, we can use the U matrix from the previously generated instance to determine the proportions of the clones, as shown below:

```
> plot_proportions(phylotree, I$U, labels = TRUE)
```

The resulting plot is shown in Figure 4. This method plots the proportions of the U matrix, where each tree represents a sample from the U matrix, illustrating the proportion

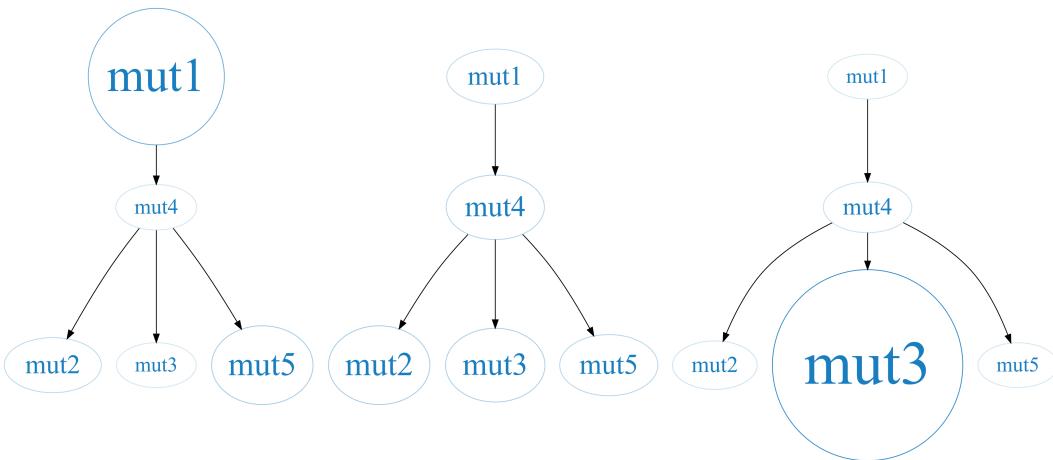


Figure 4: Phylogenetic trees associated to the generated "Phylotree" class object, using the proportions associated to the previously generated \mathbf{U} matrix.

of each clone within that specific sample. In this case, we have set the `labels` parameter to `TRUE` to label the nodes in the tree using the predefined tags "mut1", "mut4", "mut2", "mut3", and "mut5".

4.3 Comparing tumor phylogenies

Now, we present examples showing the use of **GeRnika**'s functionalities for comparing tumor phylogenies. For this purpose, we will use the `B_mats` object included in the **GeRnika** package, which contains 10 \mathbf{B} matrix trios. Specifically, we will use the first trio of matrices, and set a predefined set of tags for the clones in the trees:

```

> B_mats <- GeRnika::B_mats

> B_real <- B_mats[[1]]$B_real
> B_alg1 <- B_mats[[1]]$B_alg1
> B_alg2 <- B_mats[[1]]$B_alg2

> tags <- c("TP53", "KRAS", "PIK3CA", "APC", "EGFR", "BRCA1", "PTEN", "BRAF",
           "MYC", "CDKN2A")

> phylotree_real <- B_to_phylotree(B = B_real, labels = tags)
> phylotree_alg2 <- B_to_phylotree(B = B_alg2, labels = tags)
> phylotree_alg1 <- B_to_phylotree(B = B_alg1, labels = tags)

> plot(phylotree_real, labels=TRUE)
> plot(phylotree_alg1, labels=TRUE)
> plot(phylotree_alg2, labels=TRUE)
  
```

The plots of the three instantiated `Phylotree` class objects are depicted in Figure 5.

We can check if the phylogenies of two tumors are equivalent using the `equals` method:

```

> equals(phylotree_1 = phylotree_real, phylotree_2 = phylotree_alg1)
[1] FALSE

> equals(phylotree_1 = phylotree_real, phylotree_2 = phylotree_real)
[1] TRUE
  
```

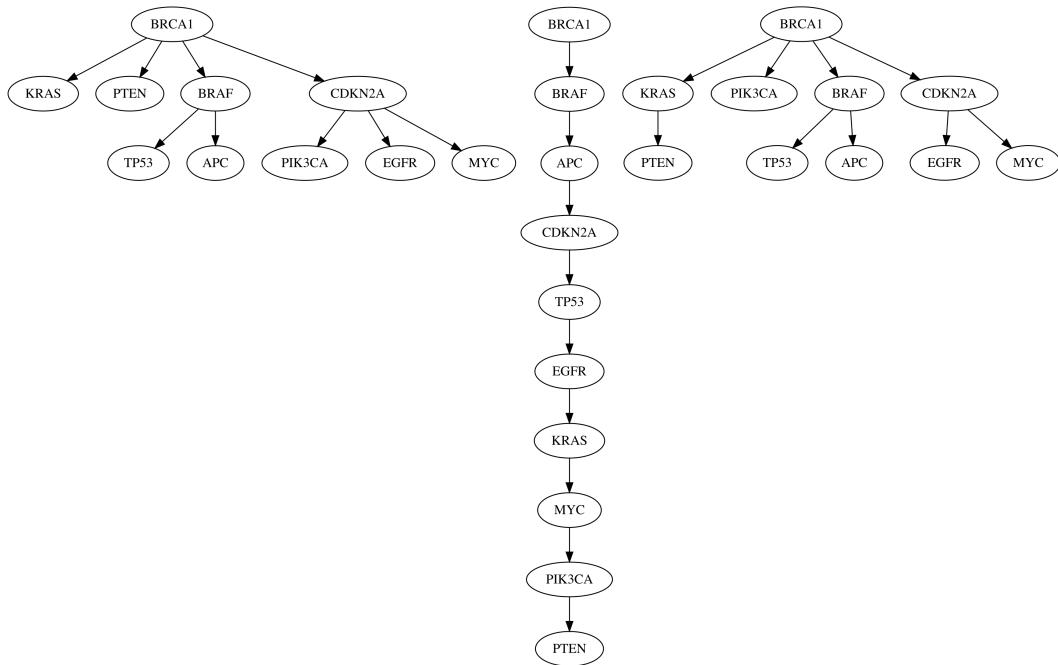


Figure 5: phylotree_real, phylotree_alg2 and phylotree_alg1, from the left to the right.

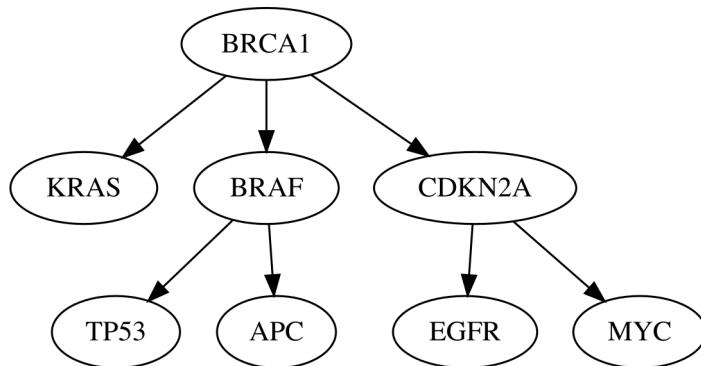


Figure 6: Maximal common subtrees between phylotree_real and phylotree_alg2 using predefined tags. In this case, there exists a single common subtree, but there may exist more in other cases.

In this case, phylotree_real and phylotree_alg1 are not identical, as some edges present in phylotree_real are absent in phylotree_alg1, and vice versa. However, a phylogenetic tree will always be identical to itself, as shown when comparing phylotree_real to itself.

To find the maximal common subtrees between two phylogenetic trees, we do it using the following command:

```
> find_common_subtrees(phylotree_1 = phylotree_real, phylotree_2 = phylotree_alg2,
                      labels = TRUE)
```

Independent edges of tree1: 2

Independent edges of tree2: 2

Common edges: 7

Distance: 4

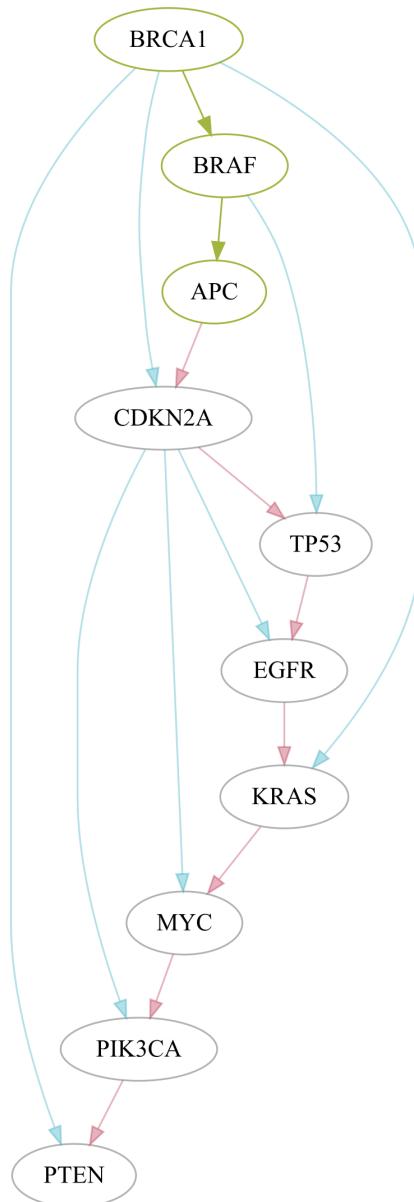


Figure 7: Consensus tree between phylotree_real and phylotree_alg1 using the *Lancet* palette and predefined tags.

The maximal common subtrees (in this case, one subtree) between phylotree_real and phylotree_alg2 are shown in Figure 6. Note that the clones in the maximal common subtree are represented by the predefined tags in the Phylotree class objects as we have set labels = TRUE. Additionally, this method prints the number of common and independent edges of the trees, along with the distance between them.

Finally, we generate the consensus tree between two phylogenetic trees using one of the custom palettes offered by **GeRnika**, specifically the *Lancet* palette, and the predefined tags for the clones as follows:

```

> palette <- GeRnika::palettes$Lancet

> consensus_real_alg1 <- combine_trees(phylotree_1 = phylotree_real,
                                         phylotree_2 = phylotree_alg1,
                                         labels = TRUE,
                                         palette = palette)
  
```

```
> DiagrammeR::render_graph(consensus_real_alg1)
```

The consensus tree between `phylotree_real` and `phylotree_alg1` is depicted in Figure 7. Here, the nodes and the edges that compose the common subtrees between the original trees are green. In addition, pink edges denote the independent edges of the tree passed as the first parameter of the method, while blue edges represent the independent edges of the second tree. Note that the independent edges of both trees are presented with translucent colors.

5 Conclusions

GeRnika is a comprehensive R package designed to address a critical gap in the tools available for studying tumor evolution within the R environment. To this end, it provides researchers with an integrated suite for simulating, visualizing, and comparing tumor phylogenies. Unlike many existing tools, **GeRnika** is fully implemented in R, making it particularly accessible for the bioinformatics community, which widely relies on R for data analysis and visualization.

One of **GeRnika**'s key contributions is providing tools to generate biologically plausible datasets for studying intratumoral heterogeneity and clonal dynamics. With varied, easily customizable parameters—controlling features such as clonal tree topology, selective pressures, and sequencing noise—**GeRnika** enables exploration of a wide range of evolutionary patterns and complexities that provide a valuable resource for testing new methods and hypotheses in tumor heterogeneity research.

Beyond its core simulation features, **GeRnika** includes tools for visualizing and comparing tumor phylogenies, offering a unified solution that eliminates the need for multiple packages or complex data processing workflows. Future work will focus on enhancing **GeRnika**'s compatibility with other R packages related to tumor evolution, allowing for easier integration with existing resources.

Overall, **GeRnika** provides an accessible, user-friendly tool that supports research into intratumoral heterogeneity, with the potential to substantially advance tumor phylogeny research. While it does not perform clonal deconvolution or phylogeny inference from real sequencing data, **GeRnika** serves as a valuable platform for simulation, visualization, and benchmarking, supporting the development and evaluation of such algorithms.

6 R software

The R package **GeRnika** is now available on CRAN.

References

- R. Burrell, N. Mcgranahan, J. Bartek, and C. Swanton. The causes and consequences of genetic heterogeneity in cancer evolution. *Nature*, 501:338–45, 09 2013. doi: 10.1038/nature12625. [p255]
- H. Dang, B. White, S. Foltz, C. Miller, J. Luo, R. Fields, and C. Maher. ClonEvol: clonal ordering and visualization in cancer sequencing. *Annals of Oncology*, 28(12):3076–3082, 2017. doi: 10.1093/annonc/mdx517. [p256]
- A. Davis, R. Gao, and N. Navin. Tumor evolution: Linear, branching, neutral or punctuated? *Biochimica et Biophysica Acta (BBA)-Reviews on Cancer*, 1867(2):151–161, 2017. doi: 10.1016/j.bbcan.2017.01.003. [p258]

- A. G. Deshwar, S. Vembu, C. K. Yung, G. H. Jang, L. Stein, and Q. Morris. PhyloWGS: reconstructing subclonal composition and evolution from whole-genome sequencing of tumors. *Genome Biology*, 16(1):1–20, 2015. doi: 10.1186/s13059-015-0602-8. [p256]
- M. El-Kebir, L. Oesper, H. Acheson-Field, and B. J. Raphael. Reconstruction of clonal trees and tumor composition from multi-sample sequencing data. *Bioinformatics*, 31(12):62–70, 06 2015. ISSN 1367-4803. doi: 10.1093/bioinformatics/btv261. [p255, 256]
- M. El-Kebir, G. Satas, L. Oesper, and B. J. Raphael. Inferring the mutational history of a tumor using multi-state perfect phylogeny mixtures. *Cell Systems*, 3(1):43–53, 2016. doi: 10.1016/j.cels.2016.07.004. [p256]
- M. El-Kebir, G. Satas, and B. J. Raphael. Inferring parsimonious migration histories for metastatic cancers. *Nature Genetics*, 50(5):718–726, 2018. doi: 10.1038/s41588-018-0106-z. [p256]
- X. Fu and R. Schwartz. ConTreeDP: A consensus method of tumor trees based on maximum directed partition support problem. In *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 125–130. IEEE, 2021. doi: 10.1101/2021.10.13.463978. [p256]
- X. Fu, H. Lei, Y. Tao, and R. Schwartz. Reconstructing tumor clonal lineage trees incorporating single-nucleotide variants, copy number alterations and structural variations. *Bioinformatics*, 38(Supplement_1):i125–i133, 2022. doi: 10.1093/bioinformatics/btac253. [p256]
- K. Grigoriadis, A. Huebner, A. Bunkum, E. Colliver, A. M. Frankell, M. S. Hill, K. Thol, N. J. Birkbak, C. Swanton, S. Zaccaria, et al. CONIPHER: a computational framework for scalable phylogenetic reconstruction with error correction. *Nature Protocols*, 19(1):159–183, 2024. doi: 10.1038/s41596-023-00913-9. [p256]
- Z. Guang, M. Smith-Erb, and L. Oesper. A weighted distance-based approach for deriving consensus tumor evolutionary trees. *Bioinformatics*, 39(Supplement_1):i204–i212, 2023. doi: 10.1093/bioinformatics/btad230. [p256]
- D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21(1):19–28, 1991. doi: 10.1002/net.3230210104. [p256]
- E. Husić, X. Li, A. Hujdurović, M. Mehine, R. Rizzi, V. Mäkinen, M. Milanič, and A. I. Tomescu. MIPUP: minimum perfect unmixed phylogenies for multi-sampled tumors via branchings and ILP. *Bioinformatics*, 35(5):769–777, 2019. doi: 10.1093/bioinformatics/bty683. [p256]
- Y. Jiang, Y. Qiu, A. J. Minn, and N. R. Zhang. Assessing intratumor heterogeneity and tracking longitudinal and spatial clonal evolutionary history by next-generation sequencing. *Proceedings of the National Academy of Sciences*, 113(37):E5528–E5537, 2016. doi: 10.1073/pnas.1522203113. [p256]
- M. Kimura. The number of heterozygous nucleotide sites maintained in a finite population due to steady flux of mutations. *Genetics*, 61(4):893, 1969. doi: 10.1093/genetics/61.4.893. [p256]
- E. Kulman, J. Wintersinger, and Q. Morris. Reconstructing cancer phylogenies using Pairtree, a clone tree reconstruction algorithm. *STAR Protocols*, 3(4):101706, 2022. doi: 10.1016/j.xpro.2022.101706. [p256]
- N. J. Loman, R. V. Misra, T. J. Dallman, C. Constantinidou, S. E. Gharbia, J. Wain, and M. J. Pallen. Performance comparison of benchtop high-throughput sequencing platforms. *Nature Biotechnology*, 30(5):434–439, 2012. doi: 10.1038/nbt.2198. [p261]
- S. Malikic, A. W. McPherson, N. Donmez, and C. S. Sahinalp. Clonality inference in multiple tumor samples using phylogeny. *Bioinformatics*, 31(9):1349–1356, 2015. doi: 10.1093/bioinformatics/btv003. [p256]

- S. Malikic, F. R. Mehrabadi, S. Ciccolella, M. K. Rahman, C. Ricketts, E. Haghshenas, D. Seidman, F. Hach, I. Hajirasouliha, and S. C. Sahinalp. PhISCS: a combinatorial approach for subperfect tumor phylogeny reconstruction via integrative use of single-cell and bulk sequencing data. *Genome research*, 29(11):1860–1877, 2019. doi: 10.1101/gr.234435.118. [p256]
- F. Marass, F. Mouliere, K. Yuan, N. Rosenfeld, and F. Markowetz. A phylogenetic latent feature model for clonal deconvolution. *The Annals of Applied Statistics*, 10:2377–2404, 2016. doi: 10.1214/16-AOAS986. [p255]
- C. A. Miller, J. McMichael, H. X. Dang, C. A. Maher, L. Ding, T. J. Ley, E. R. Mardis, and R. K. Wilson. Visualizing tumor evolution with the fishplot package for R. *BMC Genomics*, 17: 1–3, 2016. doi: 10.1186/s12864-016-3195-z. [p256]
- M. A. Myers, G. Satas, and B. J. Raphael. CALDER: Inferring phylogenetic trees from longitudinal tumor samples. *Cell Systems*, 8(6):514–522, 2019. doi: 10.1016/j.cels.2019.05.010. [p256]
- P. C. Nowell. The clonal evolution of tumor cell populations. *Science*, 194(4260):23–28, 1976. doi: 10.1126/science.959840. [p255]
- A. Petrackova, M. Vasinek, L. Sedlarikova, T. Dyskova, P. Schneiderova, T. Novosad, T. Pajik, and E. Kriegova. Standardization of sequencing coverage depth in NGS: recommendation for detection of clonal and subclonal mutations in cancer diagnostics. *Frontiers in Oncology*, 9:851, 2019. doi: 10.3389/fonc.2019.00851. [p261]
- V. Popic, R. Salari, I. Hajirasouliha, D. Kashef-Haghghi, R. B. West, and S. Batzoglou. Fast and scalable inference of multi-sample cancer lineages. *Genome Biology*, 16(1):91, 2015. doi: 10.1186/s13059-015-0647-8. [p256]
- Y. Qi, Y. Luo, and M. El-Kebir. OncoLib. <https://github.com/elkebir-group/OncoLib/>, 2019. URL <https://github.com/elkebir-group/OncoLib/>. [p256]
- E. M. Ross and F. Markowetz. OncoNEM: inferring tumor evolution from single-cell sequencing data. *Genome biology*, 17:1–14, 2016. doi: 10.1186/s13059-016-0929-9. [p256]
- S. Sandmann, C. Inserte, and J. Varghese. clevRvis: visualization techniques for clonal evolution. *GigaScience*, 12:giad020, 2023. doi: 10.1093/gigascience/giad020. [p256]
- G. Satas, S. Zaccaria, G. Mon, and B. J. Raphael. SCARLET: single-cell tumor phylogeny inference with copy-number constrained mutation losses. *Cell systems*, 10(4):323–332, 2020. doi: 10.1016/j.cels.2020.04.001. [p256]
- S. Sengupta, J. Wang, J. Lee, P. Müller, K. Gulukota, A. Banerjee, and Y. Ji. Bayclone: Bayesian nonparametric inference of tumor subclones using NGS data. In *Pacific Symposium on Biocomputing Co-Chairs*, pages 467–478. World Scientific, 2014. doi: 10.1142/9789814644730_0044. [p256]
- E. Sollier, J. Kuipers, K. Takahashi, N. Beerenswinkel, and K. Jahn. COMPASS: joint copy number and mutation phylogeny reconstruction from amplicon single-cell sequencing data. *Nature communications*, 14(1):4921, 2023. doi: 10.1038/s41467-023-40378-8. [p256]
- F. Strino, F. Parisi, M. Micsinai, and Y. Kluger. TrAp: a tree approach for fingerprinting subclonal tumor composition. *Nucleic Acids Research*, 41:e165 – e165, 2013. doi: 10.1093/nar/gkt641. [p255]
- G. Tanner, D. R. Westhead, A. Droop, and L. F. Stead. Simulation of heterogeneous tumour genomes with heterogenesis and in silico whole exome sequencing. *Bioinformatics*, 35(16): 2850–2852, 2019. doi: 10.1093/bioinformatics/bty1063. [p256]
- J. Wu and M. El-Kebir. ClonArch: visualizing the spatial clonal architecture of tumors. *Bioinformatics*, 36(Supplement_1):i161–i168, 2020. doi: 10.1093/bioinformatics/btaa471. [p256]

K. Yuan, T. Sakoparnig, F. Markowetz, and N. Beerenwinkel. BitPhylogeny: a probabilistic framework for reconstructing intra-tumor phylogenies. *Genome Biology*, 16(1):1–16, 2015.
doi: 10.1186/s13059-015-0592-6. [p256]

Aitor Sánchez-Ferrera

Intelligent Systems Group, Computer Science Faculty, University of the Basque Country
Paseo Manuel Lardizabal, Donostia/San Sebastian, 20018
Spain
ORCID: 0000-0001-6127-0686
aitor.sanchezf@ehu.eus

Maitena Tellaeche-Abete

Intelligent Systems Group, Computer Science Faculty, University of the Basque Country
Paseo Manuel Lardizabal, Donostia/San Sebastian, 20018
Spain
ORCID: 0000-0003-1894-4547
maitena.tellaetxe@ehu.eus

Borja Calvo-Molinos

Intelligent Systems Group, Computer Science Faculty, University of the Basque Country
Paseo Manuel Lardizabal, Donostia/San Sebastian, 20018
Spain
ORCID: 0000-0001-9969-9664
borja.calvo@ehu.eus

MultiATSM: An R Package for Arbitrage-Free Macrofinance Multicountry Affine Term Structure Models

by Rubens Moura

Abstract The MultiATSM package provides estimation tools and a wide range of outputs for eight macrofinance affine term structure model (ATSM) classes, supporting practitioners, academics, and policymakers. All models extend the single-country framework of Joslin et al. (2014) to multicountry settings, with additional adaptations from Jotikasthira et al. (2015) and Candelon and Moura (2024). These model extensions incorporate, respectively, the presence of a dominant (global) economy and adopt a global vector autoregressive (GVAR) setup to capture the joint dynamics of risk factors. The package generates diverse outputs for each ATSM, including graphical representations of model fit, risk premia, impulse response functions, and forecast error variance decompositions. It also implements bootstrap methods for confidence intervals and produces bond yield forecasts.

0.1 Introduction

The term structure of interest rates (or yield curve) describes the relationship between bond yields and investment maturities. As Piazzesi (2010) emphasizes, understanding its dynamics is essential for several reasons. First, long-term yields incorporate market expectations of future short-term rates, making the yield curve a handy forecasting tool for macroeconomic aggregates like output and inflation. As such, this supports optimal consumption-saving decisions and capital allocation by economic agents. Second, it plays a key role in the transmission of monetary policy, linking short-term policy rates to long-term borrowing costs. Third, it guides fiscal authorities in shaping debt maturities to balance refinancing risk and interest rate exposure. Fourth, it is essential for pricing and hedging interest rate derivatives, which rely on accurate yield curve modelling.

Affine Term Structure Models (ATSMs) are the workhorse in yield curve modelling. Based on the assumption of no arbitrage, ATSMs offer a flexible framework to assess how investors price risks and generate predictions for the price of any bond (see Piazzesi (2010); Gürkaynak and Wright (2012) for comprehensive reviews). Early ATSMs gained popularity for their ability to capture nearly all term structure fluctuations, appealing to both academics and practitioners (Vasicek, 1977; Duffie and Kan, 1996; Dai and Singleton, 2002). While these models produce accurate statistical descriptions of the yield curve, they are silent on the deeper economic determinants that policymakers require for causal inference.

In response to this limitation, a large body of research has emerged to explore the interplay between the term structure and macroeconomic developments (seminal contributions include Ang and Piazzesi (2003) and Rudebusch and Wu (2008)). A prominent contribution in this area is the unspanned economic risk framework developed by Joslin et al. (2014) (henceforth JPS, 2014). In essence, this model assumes an arbitrage-free bond market and considers a linear state space representation to describe the dynamics of the yield curve. Compared to earlier macrofinance ATSMs, JPS (2014) offers a tractable estimation approach that integrates traditional yield curve factors (spanned factors) with macroeconomic variables (unspanned factors). As a result, the model delivers a strong cross-sectional fit while explicitly linking bond yield responses to the state of the economy.

The work of JPS (2014) lays the foundational framework for the modelling tools included in the **MultiATSM** package (Moura, 2025). In addition to the original single-country setup proposed by JPS (2014), the package incorporates multicountry extensions developed by Jotikasthira et al. (2015) (henceforth JLL, 2015) and Candelon and Moura (2024) (henceforth CM, 2024). Altogether, the package offers functions to build eight types of ATSMs, covering the original versions and several variants of these three frameworks.

Beyond complete routines for model estimation, **MultiATSM** produces a wide range of analytical outputs. In particular, it generates graphical representations such as model-implied bond yields, bond risk premia, and both orthogonalized and generalized versions of: (i) impulse response functions, and (ii) forecast error variance decompositions for yields and risk factors. Confidence intervals for the two latter outputs can be computed using three bootstrap methods: residual-based, block, or wild bootstrap. Moreover, the package supports out-of-sample forecasting of bond yields across the maturity spectrum. This paper provides detailed guidance on how to use the **MultiATSM** package effectively.

There are a few notable packages for term structure modelling in the R programming environment. **YieldCurve** (Guirri, 2015) and **fBonds** (Setz, 2017) provide a collection of functions to build term structures based on the frameworks of Nelson and Siegel (1987) and Svensson (1994). These yield curve methods have gained popularity for their parsimonious parameterization and good empirical fit. However, these models do not rule out arbitrage opportunities, a limitation addressed by ATSMs. Moreover, the focus of **YieldCurve** and **fBonds** is restricted to parameter estimation and yield curve fitting, without offering additional model outputs such as those provided by **MultiATSM**.

Several other R packages support time series modeling (Hyndman and Killick, 2025), particularly within state space and vector autoregressive (VAR) frameworks. State space packages are relatively few and tend to focus on either estimation, **statespacer** (Beijers, 2023), or simulation, **simStateSpace**(Pesigan, 2025). VAR-based tools are more numerous. For instance, **vars** (Pfaff and Stigler, 2024) and **MTS** (Tsay et al., 2022) provide extensive functionality for estimation, diagnostics, and forecasting, while **svars** (Lange et al., 2023) adds structural identification methods. High-dimensional VARs are handled by packages like **bigtime** (Wilms et al., 2023) and **BigVAR** (Nicholson et al., 2025), and cross-country spillovers are modeled by **Spillover** (Urbina, 2024) and **BGVAR** (Boeck et al., 2024).

Although these tools share some features with **MultiATSM**, they are tailored to standard state space or VAR analysis. In contrast, **MultiATSM** embeds VAR dynamics within a state space representation that is explicitly grounded in arbitrage-free asset pricing theory. As such, **MultiATSM** fills a specific gap in the R ecosystem by combining the structure of ATSMs with the flexibility of modern time series tools.

The remainder of the paper is organized as follows. Section 2 outlines the theoretical foundations of the ATSMs implemented in the **MultiATSM** package, and Section 3 details each model's features. The subsequent sections focus on the practical implementation of ATSMs. Section 4 presents the dataset included in the package. Section 5 explains the user inputs required for model estimation. Section 6 explains the estimation procedure, and Section 7 shows how to estimate ATSMs from scratch using **MultiATSM**. Replications of published academic studies are provided in the Appendix.

0.2 ATSMs with unspanned economic risks: theoretical background

In this section, I outline several arbitrage-free ATSMs with unspanned macroeconomic risks available in the **MultiATSM** package. A key appealing feature of these setups is their ability to disentangle the yield curve into a cross-sectional component, governed by the risk-neutral (\mathbb{Q}) dynamics, and a time-series component, driven by the physical (\mathbb{P}) dynamics. In light of this characteristic of the models, I present the single and the multicountry \mathbb{Q} -dynamics model dimensions in Section 2.1. Next, I expose the specific features of the risk factor dynamics under the \mathbb{P} -measure of the various restricted and unrestricted VARs settings in Section 2.2. Section 2.3 describes the model estimation procedures.

Model cross-sectional dimension (Q-dynamics)

Single-country specifications (individual Q-dynamics model classes) The model cross-sectional structure is based on two central equations. The first one assumes that the country

i short-term interest rate at time t , $r_{i,t}$, is an affine function of N unobserved (latent) country-specific factors, $X_{i,t}$:

$$\begin{array}{lcl} r_{i,t} & = & \delta_{i,0} + \delta_{i,1}^\top X_{i,t}, \\ & (1 \times 1) & (1 \times 1) \quad (1 \times N)(N \times 1) \end{array} \quad (1)$$

where $\delta_{i,0}$ and $\delta_{i,1}$ are time-invariant parameters.

The second equation assumes that the unobserved factor dynamics for each country i follow a maximally flexible, first-order, N -dimensional multivariate Gaussian (\mathcal{N}) VAR model under the Q-measure:

$$\begin{array}{lcl} X_{i,t} & = & \mu_{i,X}^Q + \Phi_{i,X}^Q X_{i,t-1} + \Gamma_{i,X} \varepsilon_{i,t}^Q, \\ & (N \times 1) & (N \times 1) \quad (N \times N) \quad (N \times 1) \quad (N \times N)(N \times 1) \end{array} \quad (2)$$

where $\mu_{i,X}^Q$ contains intercepts, $\Phi_{i,X}^Q$, the feedback matrix, and $\Gamma_{i,X}$ a lower triangular matrix.

Based on Equations (1) and (2), [Dai and Singleton \(2000\)](#) show that the country-specific zero-coupon bond yield with maturity of n periods, $y_{i,t}^{(n)}$, is affine in $X_{i,t}$:

$$\begin{array}{lcl} y_{i,t}^{(n)} & = & a_{i,n}(\Theta_n) + b_{i,n}(\Theta_n)^\top X_{i,t}, \\ & (1 \times 1) & (1 \times 1) \quad (1 \times N) \quad (N \times 1) \end{array} \quad (3)$$

where $a_{i,n}(\Theta_n)$ and $b_{i,n}(\Theta_n)$ are constrained to eliminate arbitrage opportunities within this bond market, as dictated by the well-known Riccati equations.¹ For notational simplicity, we collect J bond yields into the vector $Y_{i,t} = [y_{i,t}^{(1)}, y_{i,t}^{(2)}, \dots, y_{i,t}^{(J)}]^\top$, the J intercepts into $A_X(\Theta_i) = [a_{i,1}(\Theta_1), a_{i,2}(\Theta_2), \dots, a_{i,J}(\Theta_J)]^\top \in \mathbb{R}^J$, and the N slope coefficients into a $J \times N$ matrix $B_X(\Theta_i) = [b_{i,1}(\Theta_1)^\top, b_{i,2}(\Theta_2)^\top, \dots, b_{i,J}(\Theta_J)^\top]^\top$. Accordingly, the yield curve cross-section dimension of country i is:

$$\begin{array}{lcl} Y_{i,t} & = & A_X(\Theta_i) + B_X(\Theta_i) X_{i,t}. \\ & (J \times 1) & (J \times 1) \quad (J \times N) \quad (N \times 1) \end{array} \quad (4)$$

It follows from Equations (1) and (2) that the parameter set $\Theta_i = \{\mu_{i,X}^Q, \Phi_{i,X}^Q, \Gamma_{i,X}, \delta_{i,0}, \delta_{i,1}\}$ fully characterizes the cross-section of country's i term structure. Importantly, [Dai and Singleton \(2000\)](#) demonstrate that this system is not identified without additional restrictions, since $X_{i,t}$ and any invertible affine transformation of $X_{i,t}$ yield observationally equivalent representations. To circumvent this problem, [JPS \(2014\)](#) adopt the three sets of (minimal) restrictions proposed by [Joslin et al. \(2011\)](#). First, they impose the latent factors to be zero-mean processes, forcing $\mu_{i,X}^Q = \mathbf{0}_N$. Second, they choose $\delta_{i,1}$ to be a N -dimensional vector whose entries are all equal to one. Lastly, $\Phi_{i,X}^Q$ is a diagonal matrix, the elements of which are the real and distinct eigenvalues, λ_i^Q , of the matrix of eigenvectors of $\Phi_{i,X}^Q$. Based on this restriction set, [Joslin et al. \(2011\)](#) show that no additional invariant rotation is possible.

[Joslin et al. \(2011\)](#) also show that a rotation from $X_{i,t}$ to portfolios of yields, the spanned factors $P_{i,t}$, leads to an observationally equivalent model representation. This invariant transformation implies that N portfolios of yields are perfectly priced and observed without errors, while the remaining $J - N$ portfolios are priced and observed imperfectly. Specifically, the spanned factors are computed as $P_{i,t} = V_i Y_{i,t}$, for a full-rank matrix V_i . Based on this definition, Equation (4) can be rearranged as an affine function of $P_{i,t}$

¹Specifically, the referred loadings are $a_{i,n+1}(\Theta_{n+1}) = a_{i,n}(\Theta_n) + b_{i,n}(\Theta_n) \mu_{i,X}^Q + \frac{1}{2} b_{i,n}(\Theta_n) \Gamma_{i,X} \Gamma_{i,X}' b_{i,n}(\Theta_n)' - \delta_{i,0}$ and $b_{i,n+1} = b_{i,n} \Phi_{i,X}^Q - \delta_{i,1}$, considering that the boundary conditions are $a_{i,1}(\Theta_1) = -\delta_{i,0}$ and $b_{i,1}(\Theta_1) = -\delta_{i,1}$. These expressions assume that the Radon–Nikodym derivative, which maps the risk-neutral measure to the physical measure, follows a log-normal process, and that the market price of risk is time-varying and affine in X_t . See [Ang and Piazzesi \(2003\)](#) for a detailed derivation of these expressions.

$$\begin{matrix} \mathbf{Y}_{i,t} \\ (J \times 1) \end{matrix} = \begin{matrix} \mathbf{A}_P(\Theta_i) \\ (J \times 1) \end{matrix} + \begin{matrix} \mathbf{B}_P(\Theta_i) \\ (J \times N) \end{matrix} \begin{matrix} \mathbf{P}_{i,t} \\ (N \times 1) \end{matrix} . \quad (5)$$

where $\mathbf{A}_P(\Theta_i) = \mathbf{I}_n - \mathbf{B}_X(\Theta_i) [\mathbf{V}_i \mathbf{B}_X(\Theta_i)]^{-1} \mathbf{V}_i \mathbf{A}_X(\Theta_i)$ and $\mathbf{B}_P(\Theta_i) = \mathbf{B}_X(\Theta_i) [\mathbf{V}_i \mathbf{B}_X(\Theta_i)]^{-1}$.

The rotation from $\mathbf{X}_{i,t}$ to $\mathbf{P}_{i,t}$ is convenient for two key reasons. First, $\mathbf{P}_{i,t}$ contains directly observable yield curve factors (unlike $\mathbf{X}_{i,t}$), with its N elements mapping to traditional yield curve components. For instance, for $N = 3$ and \mathbf{V}_i being the weight matrix that results from a principal component analysis, the portfolios of yields $\mathbf{P}_{i,t}$ are commonly referred to as the level, slope, and curvature factors (see Section 6.1). Second, it enables a convenient decomposition of the likelihood function, facilitating both estimation and the interpretation of model parameters.

Multicountry specifications (joint Q-dynamics model classes) The cross-section multicountry extension is formed by stacking the country yields, spanned factors, and intercepts from Equation (5) into, respectively, $\mathbf{Y}_t = [\mathbf{Y}_{1,t}^\top, \mathbf{Y}_{2,t}^\top, \dots, \mathbf{Y}_{C,t}^\top]^\top$, $\mathbf{P}_t = [\mathbf{P}_{1,t}^\top, \mathbf{P}_{2,t}^\top, \dots, \mathbf{P}_{C,t}^\top]^\top$, and $\mathbf{A}_P(\Theta) = [\mathbf{A}_P^\top(\Theta_1), \mathbf{A}_P^\top(\Theta_2), \dots, \mathbf{A}_P^\top(\Theta_C)]^\top$, where C denotes the number of countries in this economic system. Additionally, we set $\mathbf{B}_P(\Theta)$ as block diagonal, $\mathbf{B}_P(\Theta) = \mathbf{B}_P(\Theta_1) \oplus \mathbf{B}_P(\Theta_2) \oplus \dots \oplus \mathbf{B}_P(\Theta_C)$, where \oplus refers to the direct sum symbol. Accordingly,

$$\begin{matrix} \mathbf{Y}_t \\ (CJ \times 1) \end{matrix} = \begin{matrix} \mathbf{A}_P(\Theta) \\ (CJ \times 1) \end{matrix} + \begin{matrix} \mathbf{B}_P(\Theta) \\ (CJ \times CN) \end{matrix} \begin{matrix} \mathbf{P}_t \\ (CN \times 1) \end{matrix} . \quad (6)$$

Model time series dimension (P-dynamics)

In the modelling frameworks implemented in the **MultiATSM** package, the risk factor dynamics under the \mathbb{P} -measure must include at least N domestic spanned factors ($\mathbf{P}_{i,t}$) and M domestic unspanned factors ($\mathbf{M}_{i,t}$), and may optionally include G global unspanned factors (\mathbf{M}_t^W), depending on the specification. The dynamics of these risk factors evolves as either an unrestricted or a restricted VAR models. The unrestricted case corresponds to the JPS specification, while the restricted setup encompasses the GVAR and JLL frameworks.

It is worth stressing the role of unspanned factors in shaping the yield curve developments. Although these factors are absent in the cross-section dimension of the models, they influence the dynamics of the spanned factors which, in turn, affect directly bond yields.

JPS-based models The country-specific state vector, $\mathbf{Z}_{i,t}$, is formed from stacking the global and domestic (unspanned and spanned) risk factors: $\mathbf{Z}_{i,t} = [\mathbf{M}_t^{W\top}, \mathbf{M}_{i,t}^\top, \mathbf{P}_{i,t}^\top]^\top$. As such, $\mathbf{Z}_{i,t}$ is a R -dimensional vector, where $R = G + K$ and $K = M + N$. In JPS-based setups, $\mathbf{Z}_{i,t}$ follows a standard unrestricted Gaussian VAR(1):

$$\begin{matrix} \mathbf{Z}_{i,t} \\ (R \times 1) \end{matrix} = \begin{matrix} \mathbf{C}_i^{\mathbb{P}} \\ (R \times 1) \end{matrix} + \begin{matrix} \Phi_i^{\mathbb{P}} \\ (R \times R) \end{matrix} \begin{matrix} \mathbf{Z}_{i,t-1} \\ (R \times R) \end{matrix} + \begin{matrix} \Gamma_i \\ (R \times 1) \end{matrix} \begin{matrix} \varepsilon_{\mathbf{Z},t}^{\mathbb{P}} \\ (R \times R)(R \times 1) \end{matrix} , \quad \varepsilon_{\mathbf{Z},t}^{\mathbb{P}} \sim \mathcal{N}_R(\mathbf{0}_R, \mathbf{I}_R), \quad (7)$$

where $\mathbf{C}_i^{\mathbb{P}}$ denotes the vector of intercepts; $\Phi_i^{\mathbb{P}}$, the feedback matrix; and Γ_i , the Cholesky factor (a lower triangular matrix).

GVAR-based models In the **MultiATSM** package, the GVAR setup is formed from two parts: the marginal and the VARX* models. The former captures the joint dynamics of the global economy, whereas the latter describes the developments from the domestic factors. For a thorough description of GVAR models, See [Chudik and Pesaran \(2016\)](#).

The marginal model is an unrestricted VAR(1) featuring exclusively the global factors:

$$\begin{aligned} \mathbf{M}_t^W &= \mathbf{C}^W + \Phi^W \mathbf{M}_{t-1}^W + \Gamma^W \varepsilon_t^W, & \varepsilon_t^W &\sim \mathcal{N}_G(\mathbf{0}_G, \mathbf{I}_G). \end{aligned} \quad (8)$$

(G×1) (G×1) (G×G) (G×1) (G×G) (G×1)

The VARX* setups are country-specific small-scale VAR models containing global factors and weakly exogenous ‘star’ variables — weighted averages of foreign variables — built as

$$\mathbf{Z}_{i,t}^{*\top} = \sum_{j=1}^C w_{i,j} \mathbf{Z}_{j,t}^\top, \quad \sum_{j=1}^C w_{i,j} = 1, \quad w_{i,i} = 0 \quad \forall i \in \{1, 2, \dots, C\}, \quad (9)$$

where $\mathbf{Z}_{j,t}$ is a K -dimension vector of domestic factors $\mathbf{Z}_{j,t} = [\mathbf{M}_{j,t}^\top, \mathbf{P}_{j,t}^\top]^\top$ and $w_{i,j}$ is a scalar that measures the degree of connectedness of country i with country j .

These models follow a VARX*(p, q, r) specification, where p, q and r are the number of lags from, respectively, the domestic, the star, and the global risk factors. The **MultiATSM** package provides the estimates for the case $p = q = r = 1$. In such a case, the dynamics of $\mathbf{Z}_{i,t}$ is described as a VARX* of the following form:

$$\begin{aligned} \mathbf{Z}_{i,t} &= \mathbf{C}_i^X + \Phi_i^X \mathbf{Z}_{i,t-1} + \Phi_i^{X*} \mathbf{Z}_{i,t-1}^* + \Phi_i^{X^W} \mathbf{M}_{t-1}^W + \Gamma_i^X \varepsilon_{i,t}^X, & \varepsilon_{i,t}^X &\sim \mathcal{N}_K(\mathbf{0}_K, \mathbf{I}_K). \end{aligned} \quad (10)$$

(K×1) (K×1) (K×K) (K×1) (K×K) (K×1) (K×G) (G×1) (K×K) (K×1)

Additionally, GVAR models require, as an intermediate step, the specification of country-specific $2K \times CK$ -link matrices, W_i , to unify the individual VARX* models. Formally,

$$\begin{bmatrix} \mathbf{Z}_{i,t} \\ \mathbf{Z}_{i,t}^* \end{bmatrix}_{2K \times 1} \equiv \begin{bmatrix} W_i \\ (2K \times CK) \end{bmatrix} \begin{bmatrix} \mathbf{Z}_{1,t} \\ \mathbf{Z}_{2,t} \\ \vdots \\ \mathbf{Z}_{C,t} \end{bmatrix}_{CK \times 1}. \quad (11)$$

Last, to compose the F -dimensional state vector for $F = G + CK$, we gather the global economic variables and the country-specific risk factors, as $\mathbf{Z}_t = [\mathbf{M}_t^{W\top}, \mathbf{Z}_{1,t}^\top, \mathbf{Z}_{2,t}^\top, \dots, \mathbf{Z}_{C,t}^\top]^\top$. As such, we can form a first order GVAR process as

$$\begin{aligned} \mathbf{Z}_t &= \mathbf{C}_y + \Phi_y \mathbf{Z}_{t-1} + \Gamma_y \varepsilon_{y,t}, & \varepsilon_{y,t} &\sim \mathcal{N}_F(\mathbf{0}_F, \mathbf{I}_F), \end{aligned} \quad (12)$$

(F×1) (F×1) (F×F)(F×F) (F×F)(F×1)

where $\mathbf{C}_y = [\mathbf{C}^{W\top}, \mathbf{C}_1^{X\top}, \mathbf{C}_2^{X\top}, \dots, \mathbf{C}_C^{X\top}]^\top$, $\varepsilon_{y,t} = [\varepsilon_t^{W\top}, \varepsilon_{1,t}^{X\top}, \varepsilon_{2,t}^{X\top} \dots, \varepsilon_{C,t}^{X\top}]^\top$, $\Gamma_y = \Gamma^W \oplus \Gamma_1^X \oplus \Gamma_2^X \oplus \dots \oplus \Gamma_C^X$, and

$$\Phi_y = \begin{bmatrix} \Phi^W & 0_{G \times CK} \\ \Phi^{X^W} & G_1 \end{bmatrix}_{F \times F}, \quad (13)$$

where $\Phi^{X^W} = \begin{bmatrix} \Phi_1^{X^W} \\ \Phi_2^{X^W} \\ \vdots \\ \Phi_C^{X^W} \end{bmatrix}_{CK \times G}$ and $G_1 = \begin{bmatrix} \Phi_1 W_1 \\ \Phi_2 W_2 \\ \vdots \\ \Phi_C W_C \end{bmatrix}_{CK \times CK}$, for $\Phi_i = [\Phi_i^X, \Phi_i^{X*}]$ and $\forall i \in \{1, 2, \dots, C\}$.

JLL-based models JLL-based models incorporate three components: (i) the global economy, (ii) a dominant large economy,² and (iii) a set of smaller economies. The state vector is formed from a number of linear projections to build domestic risk factors that are free of the influence of the variables from other countries and/or from the global economy.

The construction of the domestic spanned factors proceeds in two steps. First, for each economy i , $P_{i,t}$ is projected on $M_{i,t}$ of this same country

$$\begin{matrix} P_{i,t} = b_i & M_{i,t} + P_{i,t}^e, \\ (N \times 1) & (N \times M)(M \times 1) & (N \times 1) \end{matrix} \quad (14)$$

where the residuals $P_{i,t}^e$ are orthogonal to the economic fundamentals of the country i .

Second, for the non-dominant economies, $P_{i,t}^e$ is additionally projected on the orthogonalized spanned factors of the dominant country, indexed by D , as follows:

$$\begin{matrix} P_{i,t}^e = c_i^D & P_{D,t}^e + P_{i,t}^{e*}, & i \neq D, \\ (N \times 1) & (N \times N)(N \times 1) & (N \times 1) \end{matrix} \quad (15)$$

where $P_{i,t}^{e*}$ corresponds to the non-dominant country i set of residuals.

The design of the domestic unspanned factors also features two steps: for the dominant economy, $M_{D,t}$ is projected on the global economic factors

$$\begin{matrix} M_{D,t} = a_D^W & M_t^W + M_{D,t}^e, \\ (M \times 1) & (M \times G)(G \times 1) & (M \times 1) \end{matrix} \quad (16)$$

and, for the other economies, the residuals of the previous regression are used to compute

$$\begin{matrix} M_{i,t} = a_i^W & M_t^W + a_i^D & M_{D,t}^e + M_{i,t}^{e*}. \\ (M \times 1) & (M \times G)(G \times 1) & (M \times M)(M \times 1) & (M \times 1) \end{matrix} \quad (17)$$

Accordingly, the state vector is formed by $Z_t^e = [M_t^{W^\top}, M_{D,t}^{e^\top}, P_{D,t}^{e^\top}, M_{2,t}^{e*^\top}, P_{2,t}^{e*^\top} \dots M_{C,t}^{e*^\top}]^\top$ and its dynamics evolve as a restricted VAR(1),

$$\begin{matrix} Z_t^e = C_Y^e & + \Phi_Y^e Z_{t-1}^e + \Gamma_Y^e \varepsilon_{Z,t}^e, & \varepsilon_{Z,t}^e \sim \mathcal{N}_F(\mathbf{0}_F, I_F). \\ (F \times 1) & (F \times 1) & (F \times F)(F \times 1) & (F \times F)(F \times 1) \end{matrix} \quad (18)$$

JLL (2015) impose a set of zero restrictions on Φ_Y^e and Γ_Y^e , with their detailed structure provided in the original study.

Estimation procedures

The approach proposed by JPS (2014) enables an efficient estimation procedure through its structural design. Specifically, the parameters governing the Q- and P-measures can be estimated independently. The only exception is the variance-covariance matrix, Σ , which appears in both likelihood functions and, therefore, must be estimated jointly.

In JLL (2015), however, the authors adopt a simplified estimation procedure by estimating the Σ matrix exclusively under the P-measure. While they acknowledge that this approach is not fully efficient, they argue that the empirical implications are limited in their application.

²Noticeably, in the context of the [MultiATSM](#) package, the model type JLL_No_DomUnit is the only exception (see Section 3).

0.3 The ATSMs available at the MultiATSM package

As outlined in the previous section, the ATSMs implemented in the **MultiATSM** package differ in the specification of their Q- and P-measure dynamics. In short, under the Q-measure, models can be specified either on a country-by-country basis (JPS, 2014) or jointly across countries (JLL, 2015; CM, 2024). Under the P-measure, risk factor dynamics follow a VAR(1) process, which may be unrestricted, as in the JPS-related frameworks, or restricted, as in the JLL and GVAR specifications.

MultiATSM provides support for eight different classes of ATSMs based on these modelling approaches. These classes vary along several dimensions: the specification of the P- and Q-dynamics, the estimation approach, and whether a dominant economy is included. Table 1 summarizes the defining features of each model class available in the package. A brief overview of these specifications follows below.

Table 1: Summary of model features

	P-dynamics				Q-dynamics		Sigma estimation		Dom. Eco.
	Single		Joint		Single	Joint	P	P and Q	
	UR	R	UR	R					
JLL GVAR									
Unrestricted VAR									
JPS original	x				x			x	
JPS global		x			x			x	
JPS multi	x					x		x	
Restricted VAR (GVAR)									
GVAR single			x		x			x	
GVAR multi			x			x		x	
Restricted VAR (JLL)									
JLL original		x				x	x		x
JLL No DomUnit	x					x	x		
JLL joint Sigma	x				x		x		x

Note: Risk factor dynamics under the P-measure may follow either an unrestricted (UR) or a restricted (R) specification. The set of restrictions present in the JLL-based and GVAR-based models are described in [Jotikasthira et al. \(2015\)](#) and [Candelon and Moura \(2024\)](#), respectively. The estimation of the Σ matrix is done either exclusively with the other parameters of the P-dynamics (P column) or jointly under both P- and Q-parameters (P and Q column). *Dom. Eco.* relates to the presence of a dominant economy. The entries featuring x indicate that the referred characteristic is part of the model.

The ATSMs in which the estimation is performed separately for each country are labeled as JPS original, JPS global and GVAR single. In the JPS original setup, the set of risk factors includes exclusively each country's domestic variables and the global unspanned factors, whereas JPS global and GVAR single also incorporate domestic risk factors of the other countries of the economic system. Noticeably, the difference between JPS global and GVAR single stem from the set of restrictions imposed under the P-dynamics.

Within the multicountry frameworks, certain features are worth noting. The JLL original model reproduces the setup in JLL (2015), assuming an economic cohort composed of a globally dominant economy and a set of smaller countries, and estimating the Σ matrix exclusively under the P-measure. The two alternative versions assume the absence of a dominant country (JLL No DomUnit) and the estimation of Σ under both the P and Q measures (JLL joint Sigma), as in JPS (2014). The remaining specifications differ in their P-dynamics: either by an unrestricted VAR model (JPS multi) or by a GVAR setup (GVAR multi), as proposed in CM (2024).

0.4 Package dataset

The **MultiATSM** package provides datasets that approximate those used in the GVAR-based ATSMs of [Candelon and Moura \(2023\)](#) and [CM \(2024\)](#). The data requirements for estimating GVAR models encompass those of all other model classes, making them suitable

for generating outputs across all models supported by the package. As such, the examples in the following sections use the dataset from CM (2024).

The `LoadData()` function provides access to the datasets included in the package. To load the data from CM (2024), set the argument to `CM_2024`:

```
LoadData("CM_2024")
```

This function returns three sets of data. The first contains time series of zero-coupon bond yields for four emerging market economies: China, Brazil, Mexico, and Uruguay. The data spans monthly intervals from June 2004 to January 2020. For the purpose of model estimation, the package requires that (i) bond yield maturities are the same across all countries;³ and (ii) yields must be expressed in annualized percentage terms (not basis points). Note that the `MultiATSM` package does not provide routines for bootstrapping zero-coupon yields from coupon bonds, so any such treatment must be handled by the user.

The second dataset comprises time series for unspanned risk factors — specifically, the macroeconomic indicators economic growth and inflation — covering the same period as the bond yield data. These data cover both (i) domestic variables for each of the four countries in the sample and (ii) corresponding global indicators. The construction of unspanned risk factors, like that of bond yields, must be carried out externally by the user.

The final dataset contains measures of interconnectedness, proxied by trade flows, which are specifically required for estimating the GVAR-based models. The trade flow data report the annual value of goods imported and exported between each pair of countries in the sample, starting from 1948. All values are expressed in U.S. dollars on a free-on-board basis. These data are used to construct the transition matrix in the GVAR framework.

0.5 Required user inputs

Fundamental inputs

To estimate any model, the user must specify several general inputs, which can be grouped into the following categories:

1. Desired ATSM class (`ModelType`): a character vector containing the label of the model to be estimated as described in Table 1;
2. Risk Factor Features. This includes the following list of elements:
 - Set of economies (`Economies`): a character vector containing the names of the economies which are part of the economic system;
 - Global variables (`GlobalVar`): a character vector containing the labels of the G global unspanned factors. Studies examining the impact of global developments on bond prices could include proxy measures of global inflation and global economic activity in this category (Jotikasthira et al., 2015; Abbritti et al., 2018; Candelon and Moura, 2024);
 - Domestic variables (`DomVar`): a character vector containing the labels of the M domestic unspanned factors. These typically correspond to measures of domestic inflation and economic activity, the standard macroeconomic indicators monitored by central banks (Ang and Piazzesi, 2003; Joslin et al., 2014; Jotikasthira et al., 2015; Candelon and Moura, 2024);
 - Number of spanned factors (N): a scalar representing the number of country-specific spanned factors. Although, in principle, N could vary across countries, the models provided in the package assume a common value of N for all countries. A common

³It is worth emphasizing that, although the `DataForEstimation()` and `InputsForOpt()` functions in the package accept inputs with differing maturities, their outputs are standardized to a common set of yields.

choice in the literature is $N = 3$, as in JPS (2014) and CM (2024), since this produces an excellent cross-sectional fit of bond yields (Litterman and Scheinkman (1991)). Other studies, such as Adrian et al. (2013), extend the specification to $N = 5$, arguing that it improves the performance of model-implied term premia. Further intuition on the role and interpretation of spanned factors is provided in Section 6.1.

3. Sample span:
 - Initial sample date (t_0): the start of the sample period in the format *dd-mm-yyyy*;
 - End sample date (t_F): the end of the sample period in the format *dd-mm-yyyy*.
4. Data Frequency (DataFreq): a character vector specifying the frequency of the time series data. The available options are: Annually, Quarterly, Monthly, Weekly, Daily Business Days, and Daily All Days;
5. Stationarity constraint under the Q-dynamics (StatQ): a logical that takes TRUE if the user wishes to impose that the largest eigenvalue under the Q-measure, λ_i^Q , is strictly less than 1. While enforcing this stationarity constraint may increase estimation time, it can improve convergence and numerical stability. Moreover, by inducing near-cointegration, the eigenvalue restriction helps to pin down more plausible dynamics for bond risk premia (Bauer et al., 2012; Joslin et al., 2014);
6. Selected folder to save the graphical outputs (Folder2Save): path where the selected graphical outputs will be saved. If set to NULL, the outputs are stored in the user's temporary directory (accessible via `tempdir()`);
7. Output label (OutputLabel): A single-element character vector containing the name used in the file name that stores the model outputs.

The following provides an example of the basic model input specification:

```
ModelType <- "JPS original"
Economies <- c("Brazil", "Mexico", "Uruguay")
GlobalVar <- c("G1_Eco_Act", "G1_Inflation")
DomVar <- c("Eco_Act", "Inflation")
N <- 3
t0 <- "01-07-2005"
tF <- "01-12-2019"
DataFreq <- "Monthly"
StatQ <- FALSE
Folder2Save <- NULL
OutputLabel <- "Model_demo"
```

Model-specific inputs

GVARlist and JLLlist The inputs described above are sufficient for estimating all variants of the JPS models presented in Table 1. However, estimating the GVAR or JLL setups requires additional elements. For clarity, these extra inputs should be organized into separate lists for each model. This section outlines the general structure of both lists, while Section 6.2 provide a more detailed explanations of their components and available options, reflecting the broader scope of each setup.

For GVAR models, the required inputs are twofold. First, the user must specify the dynamic structure of each country's VARX model. For example:

```
VARXtype <- "unconstrained"
```

Next, provide the desired inputs to build the transition matrix. For instance:

```

data('TradeFlows')
W_type <- "Sample Mean"
t_First_Wgvar <- "2000"
t_Last_Wgvar <- "2015"
DataConnectedness <- TradeFlows

```

Based on these inputs, a complete instance of the GVARlist object is

```

GVARlist <- list(VARXtype = "unconstrained", W_type = "Sample Mean",
                  t_First_Wgvar = "2000", t_Last_Wgvar = "2015",
                  DataConnectedness = TradeFlows)

```

For the JLL frameworks, if the chosen model is either JLL original or JLL joint Sigma, it suffices to specify the name of the dominant economy. Otherwise, for the JLL No DomUnit class, the user must set None. For instance:

```

## Example for "JLL original" and "JLL joint Sigma" models
JLLlist <- list(DomUnit = "China")

## For "JLL No DomUnit" model
JLLlist <- list(DomUnit = "None")

```

BRWlist In an influential paper, [Bauer et al. \(2012\)](#) (henceforth BRW, 2012) show that estimates from traditional ATSMs often suffer from severe small-sample bias. This can lead to unrealistically stable expectations for future short-term interest rates and, consequently, distort term premium estimates for long-maturity bonds. To address this issue, BRW (2012) propose an indirect inference estimator based on a stochastic approximation algorithm, which corrects for bias and enhances the persistence of short-term interest rates, resulting in more plausible term premium dynamics.

It is worth noting that this framework serves as a complementary feature to the core ATSMs and can therefore be applied to any of the model types supported by the **MultiATSM** package. If the user intends to implement a model following the BRW (2012) approach, a few additional inputs must be specified. These include:

- Mean or median of physical dynamic estimates (Cent_Measure): compute the mean or the median of the \bar{P} -dynamics estimates after each bootstrap iteration by setting the option to Mean (for the mean) or Median (for the median);
- Adjustment parameter (gamma): this parameter controls the degree of shrinkage applied to the difference between the estimates prior to the bias correction and the bootstrap-based estimates after each iteration. It remains fixed across iterations and must lie in the interval (0, 1);
- Number of iteration (N_iter) : total number of iterations used in the stochastic approximation algorithm after burn-in;
- Number of bootstrap samples (B): quantity of simulated samples used in each burn-in or actual iteration;
- Perform closeness check (checkBRW): indicates whether the user wishes to compute the root mean square distance between the model estimates obtained through the bias-correction method and those generated via no bias-correction. The default is TRUE;
- Number of bootstrap samples used in the closeness check (B_check): default is equal to 100,000 samples;
- Eigenvalue restriction (Eigen_rest): impose a restriction on the largest eigenvalue under the P -measure after applying the bias correction procedure. Default is 1;

- Number of burn-in iteration (N_burn): quantity of the iterations to be discarded in the first stage of the bias-correction estimation process. The recommended number is 15% of the total number of iterations. In practice, this resembles the burn-in concept in Markov chain Monte Carlo methods. Particularly, the BRW (2012) stochastic approximation algorithm is iterative, and for a sufficiently large number of iterations, the parameters converge to their true values. As such, discarding early iterations avoids the need for assessing a computationally costly exit condition.

```
BRWlist <- within(list(Cent_Measure = "Mean", gamma = 0.2, N_iter = 500, B = 50,
                        checkBRW = TRUE, B_check = 1000, Eigen_rest = 1),
                        N_burn <- round(N_iter * 0.15))
```

Additional inputs for numerical and graphical outputs

Once the desired features are selected and the parameters of the chosen ATSM have been estimated, the **MultiATSM** package provides tools to generate the following numerical and graphical outputs via the `NumOutputs()` function:

- Time-series dynamics of the risk factors;
- Model fit of the bond yields;
- Orthogonalized impulse response functions (IRFs);
- Orthogonalized forecast error variance decompositions (FEVDs);
- Generalized impulse response functions (GIRFs);
- Generalized forecast error variance decompositions (GFEVDs);
- Decomposition of bond yields into expected and term premia components.

These outputs are organized into distinct analytical components, each offering different insights into model behavior and its economic interpretation.

The time-series dynamics of the risk factors are displayed in separate subplots: one for each global factor, and one subplot per domestic risk factor showing all countries in the economic system. The model fit of the bond yields is provided through two measures of model-implied yields. The first is a fitted measure derived solely from the cross-sectional component, as in Equation (5) for single-country models and Equation (6) for multicountry setups. This measure reflects the fit based exclusively on the parameters governing the Q-dynamics. The second incorporates both the physical and risk-neutral dynamics, combining the cross-sectional equations with the state evolution specified by each ATSM.

The impulse response functions and variance decompositions are available in both orthogonalized and generalized forms. The orthogonalized outputs (IRFs and FEVDs) are computed using a short-run recursive identification scheme, meaning they depend on the ordering of the selected risk factors. Specifically, the package is structured to place global unspanned factors first, followed by its domestic unspanned and spanned factors within each country, in the order in which countries are listed in the `Economies` vector. In contrast, the generalized versions (GIRFs and GFEVDs) are robust to factor ordering but allow for correlated shocks across risk factors (Pesaran and Shin, 1998). For the numerical computation of these outputs, a horizon of analysis has to be specified, *e.g.*, `Horiz <- 100`.

The bond yield decomposition can be performed with respect to two measures of risk compensation: term premia and forward premia. While the term premium is derived directly from the bond yield levels, the forward premium is obtained from the decomposition of forward rates. A more formal presentation of both measures is provided in the Appendix.

Users must specify the desired graph types in a character vector. Available options include: `RiskFactors`, `Fit`, `IRF`, `FEVD`, `GIRF`, `GFEVD`, and `TermPremia`. For example:

```
DesiredGraphs <- c("Fit", "GIRF", "GFEVD", "TermPremia")
```

Moreover, for all models, users must indicate the types of variables of interest (yields, risk factors, or both). For JLL-type models specifically, users must also specify whether to

include the orthogonalized versions. Each of these options should be set to TRUE to generate the corresponding graphs, and FALSE otherwise.

```
WishGraphRiskFac <- FALSE
WishGraphYields <- TRUE
WishOrthoJLLgraphs <- FALSE
```

The desired graphical outputs are stored in the selected folder, `Folder2Save`. Alternatively, users can display the desired plots directly in the console without saving them to `Folder2Save` by using the `autoplot()` method.

Bootstrap settings [Horowitz \(2019\)](#) shows that bootstrap methods generally produce more accurate statistical inference than those based on asymptotic distribution theory. To generate confidence intervals using bootstrap, via the `Bootstrap()` function, an additional list of inputs must be provided:

- Desired bootstrap procedure (`methodBS`): the user must select one of the following options: (i) standard residual bootstrap (`bs`); (ii) wild bootstrap (`wild`); or (iii) block bootstrap (`block`). If the block bootstrap is selected, the block length must also be specified. The residual bootstrap is a conventional method that is straightforward to implement when a parametric model, such as a VAR model, is available. The block bootstrap makes weaker assumptions about the data-generating process and is well-suited to handling both weak and strong serial dependence. The wild bootstrap is particularly appropriate for data exhibiting heteroskedasticity ([Horowitz, 2019](#));
- Number of bootstrap draws (`ndraws`): [Kilian and Lütkepohl \(2017\)](#) suggest that, in VAR specifications, `ndraws` can range from a few hundred to several thousand, depending on factors such as sample size, lag order, and the desired quantiles of the distribution. Illustrating this, CM (2024) set `ndraws = 1,000` in their ATSM to construct confidence intervals for IRFs;
- Confidence level expressed (`pctg`): the desired confidence level should be expressed in percentage points. Common choices in VAR-related setups include 68%, 90% and 95% ([Kilian and Lütkepohl, 2017](#)).

```
Bootlist <- list(methodBS = 'block', BlockLength = 4, ndraws = 1000, pctg = 95)
```

Out-of-sample forecast settings To generate bond yield forecasts, use `ForecastYields()` with the following inputs:

- Forecast horizon (`ForHoriz`): Number of forecast horizons in periods;
- Index of the first observation (`t0Sample`): time index of the first observation included in the information set;
- Index of the last observation (`t0Forecast`): time index of the last observation in the information set used to generate the first forecast;
- Method used for forecast computation (`ForType`): forecasts can be generated using either a rolling or expanding window. To use a rolling window, set this parameter to `Rolling`. In this case, the sample length for each forecast is fixed and defined by `t0Sample`. For expanding window forecasts, set this input to `Expanding`, allowing the information set to increase at each forecast iteration.

```
ForecastList <- list(ForHoriz = 12, t0Sample = 1, t0Forecast = 70, ForType = "Rolling")
```

0.6 Model estimation

Using the dataset described in Section 4, the estimation of the ATSM proceeds in three main steps. First, the country-specific spanned factors are estimated, which, along with

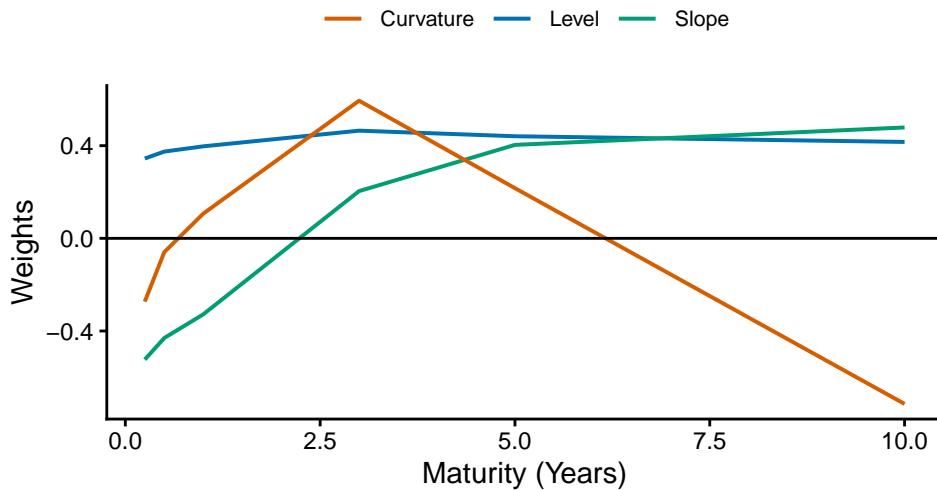


Figure 1: Yield loadings on the spanned factors. Example using bond yield data for Uruguay. Graph was generated using the `ggplot2` package (Wickham, 2016).

the global and domestic unspanned factors, form the complete set of risk factors used in the subsequent estimation steps. Second, the package estimates the parameters governing the dynamics of the risk factors under the \mathbb{P} -measure. Finally, it optimizes the full ATSM specification, including the parameters under the \mathbb{Q} -measure.

As will be made clear in Section 7, although the functions introduced in this section can be used individually, they are primarily designed to be used together with the broader set of functions available in the `MultiATSM` package. However, as these functions play a central role in the package structure, they warrant a dedicated section.

Spanned factors

The spanned factors for country i , denoted by $P_{i,t}$, are typically obtained as the first N principal components (PCs) of the observed bond yields. The PC method provides orthogonal linear combinations of the original variables, ordered by their ability to capture the variance in the data. Formally, $P_{i,t}$ is computed as $P_{i,t} = w_i Y_{i,t}$, where yields are ordered by increasing maturity in $Y_{i,t}$, and w_i is the matrix of eigenvectors derived from the covariance matrix of $Y_{i,t}$.

In the case of $N = 3$, the spanned factors are traditionally interpreted as level, slope, and curvature components of the yield curve (Litterman and Scheinkman, 1991). This interpretation stems from the properties of the w_i matrix, as illustrated below:

```
data('Yields')
w <- pca_weights_one_country(Yields, Economy = "Uruguay")
```

In matrix w , each row holds the weights for constructing a spanned factor. The first row relates to the level factor, with weights loading roughly equally across maturities. As such, high (low) values of the level factor indicate an overall high (low) value of yields across all maturities. The second row features increasing weights with maturity, capturing the slope of the yield curve: high values indicate steep curves, while low values reflect flat or inverted curves. The third row corresponds to the curvature factor, with weights emphasizing medium-term maturities. This captures the ‘hump-shaped’ features of the yield curve typically associated with changes in its curvature. These concepts are also graphically illustrated in Figure 1.

The user can directly obtain the time series of the country-specific spanned factors by calling `Spanned_Factors()`, as shown below:

```
data('Yields')
Economies <- c("China", "Brazil", "Mexico", "Uruguay")
N <- 2
SpaFact <- Spanned_Factors(Yields, Economies, N)
```

The P-dynamics estimation

As presented in Table 1 and explained in detail in Section 2, the dynamics of the risk factors under the \mathbb{P} -measure in the available models follow a VAR(1) process. This specification can be fully unrestricted, as in the JPS-related models, or subject to restrictions, as in the GVAR and JLL frameworks. This subsection illustrates how each of these model configurations is implemented.

VAR To use `VAR()`, the user needs to select the appropriate set of risk factors for the model being estimated and specify unconstrained in the argument `VARtype`. In the two examples presented below, the outputs are the intercept vector, the feedback matrix, and the variance–covariance matrix for a VAR(1) model under the \mathbb{P} -measure:

```
## Example 1: "JPS global" and "JPS multi" models
data("RiskFacFull")
PdynPara <- VAR(RiskFacFull, VARtype = "unconstrained")

## Example 2: "JPS original" model for China
FactorsChina <- RiskFacFull[1:7, ]
PdynPara <- VAR(FactorsChina, VARtype = "unconstrained")
```

GVAR The `GVAR()` function estimates a GVAR(1) model constructed from country-specific $\text{VARX}^*(1, 1, 1)$ specifications. It requires two main inputs: the number of domestic spanned factors (N) and a set of elements grouped in the `GVARinputs` list. The latter consists of four components:

1. Economies: a C –dimensional character vector containing the names of the economies present in the economic system;
2. GVAR list of risk factors: a list of risk factors sorted by country in addition to the global variables. An example of the expected data structure is:

```
data("GVARFactors")
```

To assist in formatting the data accordingly, users may use the `DatabasePrep()` function;

3. VARX type: a character vector specifying the desired structure of the VARX^* model. Two general options are available:
 - Fully unconstrained: specify as `unconstrained`. This option estimates each equation in the system separately via ordinary least squares, without imposing any restrictions.
 - With constraints: imposes specific set of zero restrictions on the feedback matrix. This category includes two sub-options: (a) `constrained`: `Spanned_Factors` prevents foreign spanned factors from affecting any domestic risk factor; (b) `constrained: [factor name]` restricts the specified risk factor to be influenced only by its own lags and the lags of its associated star variables. In both cases, the VARX^* is estimated using restricted least squares.

```
data('GVARFactors')
GVARinputs <- list(Economies = Economies, GVARFactors = GVARFactors,
                    VARXtype ="constrained: Inflation")
```

4. Transition matrix: a $C \times C$ matrix that captures the degree of interdependence across the countries in the system. Each entry (i, j) represents the strength of the dependence of economy i on economy j . As an example, the matrix below is computed from bilateral trade flow data, averaged over the period 2006–2019, for a system comprising China, Brazil, Mexico, and Uruguay. The rows are normalized so that the weights sum to 1 for each country (i.e., each row of the matrix sums to 1). The transition matrix can be generated using `Transition_Matrix()`, as illustrated in the Appendix :

```
#>           China Brazil Mexico Uruguay
#> China    0.0000 0.6549 0.3155 0.0296
#> Brazil   0.8269 0.0000 0.1234 0.0497
#> Mexico   0.8596 0.1326 0.0000 0.0078
#> Uruguay  0.3811 0.5498 0.0691 0.0000
```

With inputs specified, the user can estimate a GVAR model using:

```
data("GVARFactors")
GVARinputs <- list(Economies = Economies, GVARFactors = GVARFactors,
                     VARXtype = "unconstrained", Wgvar = W_gvar)
N <- 3
GVARpara <- GVAR(GVARinputs, N, CheckInputs = TRUE)
```

Note that the `CheckInputs` parameter should be set to `TRUE` to perform a consistency check on the inputs specified in `GVARinputs` prior to the \mathbb{P} -dynamics estimation.

JLL The `JLL()` function estimates the physical parameters. Required inputs are:

1. Risk factors: a time series matrix of the risk factors in their non-orthogonalized form;
2. Number of spanned factors (N): a scalar representing the number of country-specific spanned factors;
3. `JLLinputs`: a list object containing the following elements:
 - Economies: a C -dimensional character vector listing the economies;
 - Dominant Economy: a character vector indicating either the name of the country assigned as the dominant economy (for `JLL` original and `JLL jointSigma` models), or `None` (for the `JLL No DomUnit` case);
 - Estimate Sigma Matrices: a logical equal to `TRUE` if the user wishes to estimate the full set of `JLL` sigma matrices (i.e., variance-covariance and Cholesky factor matrices), and `FALSE` otherwise. Since this numerical estimation is costly, it may significantly increase computation time;
 - Precomputed Variance-Covariance Matrix: in some instances, a precomputed variance-covariance matrix from the non-orthogonalized dynamics can be supplied here to save time and memory. If no such matrix is available, this input should be set to `NULL`;
 - `JLL` type: a character string specifying the chosen `JLL` model, following the classification described in Table 1.

```
## First set the JLLinputs
ModelType <- "JLL original"
JLLinputs <- list(Economies = Economies, DomUnit = "China", WishSigmas = TRUE,
                  SigmaNonOrtho = NULL, JLLModelType = ModelType)

## Then, estimate the desired the  $\mathbb{P}$ -dynamics from the desired JLL model
data("RiskFacFull")
N <- 3
JLLpara <- JLL(RiskFacFull, N, JLLinputs, CheckInputs = TRUE)
```

The `CheckInputs` input is set to TRUE to perform a consistency check on the inputs specified in `JLLinputs` before running the P-dynamics estimation.

ATSM estimation

Estimating the ATSM parameters involves maximizing the log-likelihood function to obtain the best-fitting model parameters using `Optimization()`. The unspanned risk factor framework of JPS (2014) (and, therefore, all its multicountry extensions) follows a model parameterization similar to that proposed in Joslin et al. (2011). Particularly, it requires estimating a set of six parameter blocks:

1. The risk-neutral long-run mean of the short rate (r_0);
2. The risk-neutral feedback matrix (K_{1XQ});
3. Standard deviation of measurement errors for yields observed with error (se);
4. The variance-covariance matrix from the VAR process (SSZ);
5. The intercept matrix of the physical dynamics (K_{0Z});
6. The feedback matrix of the physical dynamics (K_{1Z}).

The parameters K_{0Z} and K_{1Z} have closed-form solutions. Similarly, r_0 and se are derived analytically and are factored out of the log-likelihood function. In contrast, the remaining parameters, K_{1XQ} and SSZ , must be estimated numerically.

The optimization routine in `MultiATSM` combines the Nelder–Mead and L-BFGS-B algorithms, executed sequentially and repeated until convergence is achieved. At each iteration, the parameter vector yielding the highest likelihood is retained, enhancing robustness to local optima without resorting to full multi-start procedures. Convergence is achieved when the absolute change in the mean log-likelihood falls below a user-defined tolerance (default 10^{-4}). For the bootstrap replications, the same optimization procedure is applied; however, only the Nelder–Mead algorithm is used to reduce computation time.

0.7 Full implementation of ATSMs

Package workflow

The complete workflow of the `MultiATSM` package is built around seven core functions, which together support a streamlined and modular process. An overview of these functions is provided below:

1. `LabFac()`: returns a list of risk factor labels used throughout the package. In particular, these labels assist in structuring sub-function inputs and generating variable and graph labels in a parsimonious manner;
2. `InputsForOpt()`: collects and processes the inputs needed to build the likelihood function as specified in Section 5. It estimates the model's P-dynamics and returns an object of class `ATSMModelInputs`, which includes `print()` and `summary()` S3 methods. The `print()` method summarizes model inputs and system features, while `summary()` reports statistics on risk factors and bond yields;
3. `Optimization()`: performs the estimation of the model parameters, primarily the Q-dynamics, using numerical optimization. This function returns a comprehensive list of the model's point estimates and can be computationally intensive;
4. `InputsForOutputs()`: an auxiliary function that compiles the necessary elements for producing numerical and graphical outputs. It also creates separate folders in the user's `Folder2Save` directory to store the generated figures;

5. `NumOutputs()`: produces the numerical outputs as selected in Section 5.3, based on the model's point estimates. The function returns an object of class `ATSMNumOutputs`, for which an `autoplot()` S3 method is available. This method provides a convenient way to visualize the selected graphical outputs;
6. `Bootstrap()`: computes confidence bounds for the numerical outputs using the bootstrap procedures defined in Section 5.3 (subsection "Bootstrap settings"). The function returns an `ATSMModelBoot` object, which can be accessed via the `autoplot()` S3 method to generate the desired graphical outputs with confidence intervals. As this step involves repeated model estimation, it may require several hours (possibly days) to complete;
7. `ForecastYields()`: generates bond yield forecasts and the corresponding forecast errors according to the specifications outlined in Section 5.3 (subsection "Out-of-sample forecast settings"). This function returns an object of class `ATSMModelForecast`, accessible via the `plot()` S3 method, which displays Root Mean Squared Errors (RMSEs) by country and forecast horizon.

Complete implementation

This section illustrates how to fully implement ATSMs using the `MultiATSM` package. A simplified two-country JPS original framework serves as the example. The implementation steps are outlined below, and a sample of graphical outputs are presented in Figures 2 – 5.

```
library(MultiATSM)
# 1) USER INPUTS
# A) Load database data
LoadData("CM_2024")

# B) GENERAL model inputs
ModelType <- "JPS original"
Economies <- c("China", "Brazil")
GlobalVar <- c("G1_Eco_Act")
DomVar <- c("Eco_Act")
N <- 2
t0_sample <- "01-05-2005"
tF_sample <- "01-12-2019"
OutputLabel <- "Test"
DataFreq <-"Monthly"
Folder2Save <- NULL
StatQ <- FALSE

# B.1) SPECIFIC model inputs
# GVAR-based models
GVARlist <- list( VARXtype = "unconstrained", W_type = "Sample Mean", t_First_Wgvar = "2005",
                   t_Last_Wgvar = "2019", DataConnectedness = TradeFlows )

# JLL-based models
JLLlist <- list(DomUnit = "China")

# BRW inputs
WishBC <- FALSE
BRWlist <- within(list(Cent_Measure = "Mean", gamma = 0.05, N_iter = 250, B = 50, checkBRW = TRUE,
                        B_check = 1000, Eigen_rest = 1), N_burn <- round(N_iter * 0.15))

# C) Decide on Settings for numerical outputs
WishFPremia <- TRUE
```

```

FPmatLim <- c(60,120)

Horiz <- 30
DesiredGraphs <- c()
WishGraphRiskFac <- FALSE
WishGraphYields <- FALSE
WishOrthoJLLgraphs <- FALSE

# D) Bootstrap settings
WishBootstrap <- TRUE
BootList <- list(methodBS = 'bs', BlockLength = 4, ndraws = 5, pctg = 95)

# E) Out-of-sample forecast
WishForecast <- TRUE
ForecastList <- list(ForHoriz = 12, t0Sample = 1, t0Forecast = 162, ForType = "Rolling")

#####
# NO NEED TO MAKE CHANGES FROM HERE:
# The sections below automatically process the inputs provided above, run the model
# estimation, generate the numerical and graphical outputs, and save results.

# 2) Minor preliminary work: get the sets of factor labels
FactorLabels <- LabFac(N, DomVar, GlobalVar, Economies, ModelType)

# 3) Prepare the inputs of the likelihood function
ATSMInputs <- InputsForOpt(t0_sample, tf_sample, ModelType, Yields, GlobalMacro,
                           DomMacro, FactorLabels, Economies, DataFreq, GVARlist,
                           JLLlist, WishBC, BRWlist)

# 4) Optimization of the ATSM (Point Estimates)
ModelParaList <- Optimization(ATSMInputs, StatQ, DataFreq, FactorLabels, Economies, ModelType)

# 5) Numerical and graphical outputs
# a) Prepare list of inputs for graphs and numerical outputs
InputsForOutputs <- InputsForOutputs(ModelType, Horiz, DesiredGraphs, OutputLabel, StatQ,
                                      DataFreq, WishGraphYields, WishGraphRiskFac,
                                      WishOrthoJLLgraphs, WishFPremia,
                                      FPMatLim, WishBootstrap, BootList,
                                      WishForecast, ForecastList)

# b) Fit, IRF, FEVD, GIRF, GFEVD, and Term Premia
NumericalOutputs <- NumOutputs(ModelType, ModelParaList, InputsForOutputs,
                                 FactorLabels, Economies, Folder2Save)

# c) Confidence intervals (bootstrap analysis)
BootstrapAnalysis <- Bootstrap(ModelType, ModelParaList, NumericalOutputs, Economies,
                                 InputsForOutputs, FactorLabels, JLLlist, GVARlist,
                                 WishBC, BRWlist, Folder2Save)

# 6) Out-of-sample forecasting
Forecasts <- ForecastYields(ModelType, ModelParaList, InputsForOutputs, FactorLabels,
                             Economies, JLLlist, GVARlist, WishBC, BRWlist,
                             Folder2Save)

```

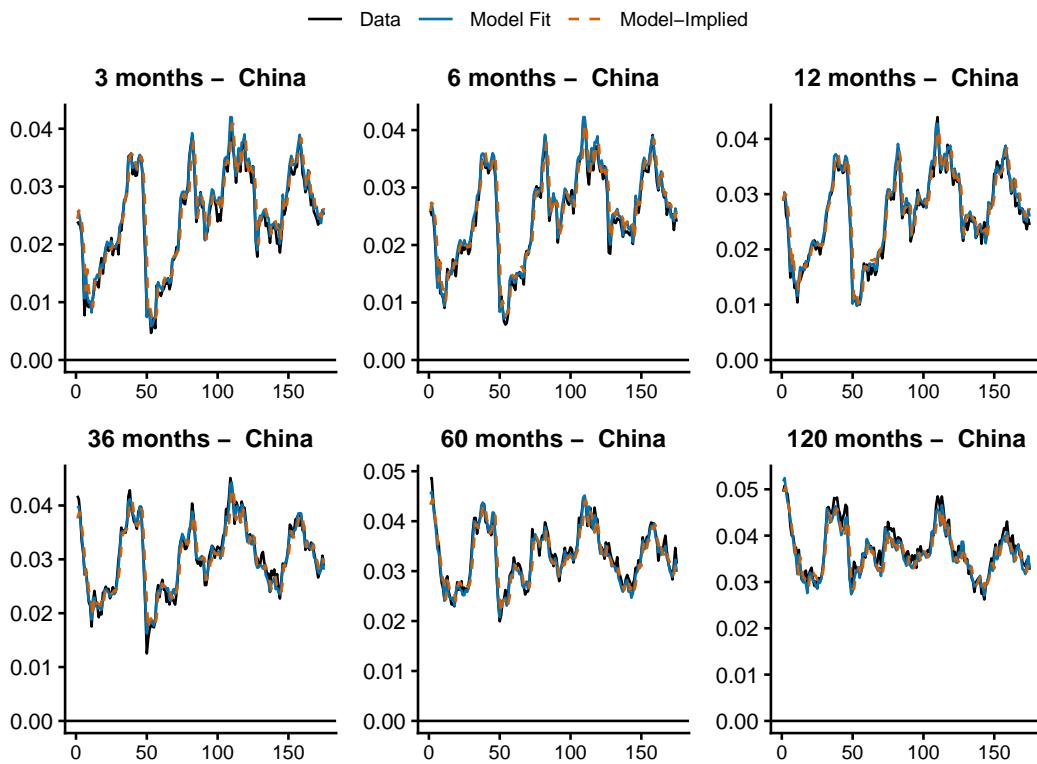


Figure 2: Chinese bond yield maturities with model fit comparisons. *Model-fit* reflects estimation using only risk-neutral (Q) dynamics parameters, while *Model-implied* incorporates both physical (P) and risk-neutral (Q) dynamics. The x-axes represent time in months and the y-axis is in natural units.

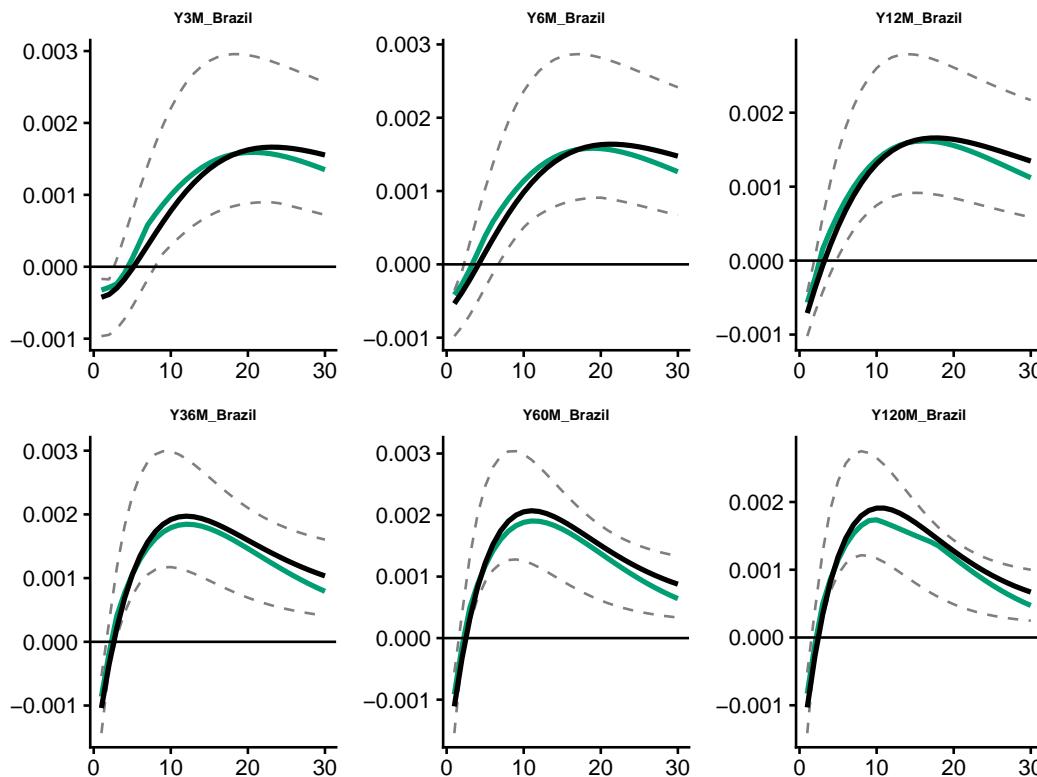


Figure 3: IRFs from the Brazilian bond yields to global economic activity. Size of the shock is one-standard deviation. The black lines are the point estimates. Gray dashed lines are the bounds of the 95% confidence intervals and the green lines correspond to the median of these intervals. The x-axes are expressed in months and the y-axis is in natural units.

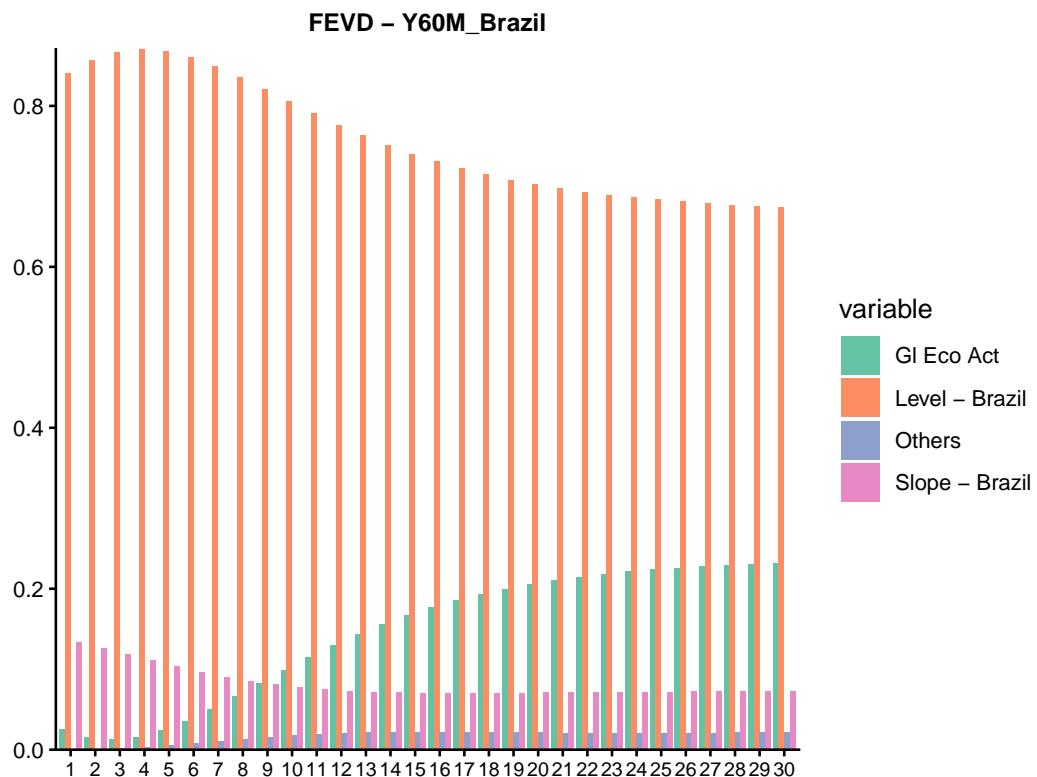


Figure 4: FEVD from the Brazilian bond yield with maturity 60 months. The x -axis represents the forecast horizon in months and the y -axis is in natural units.

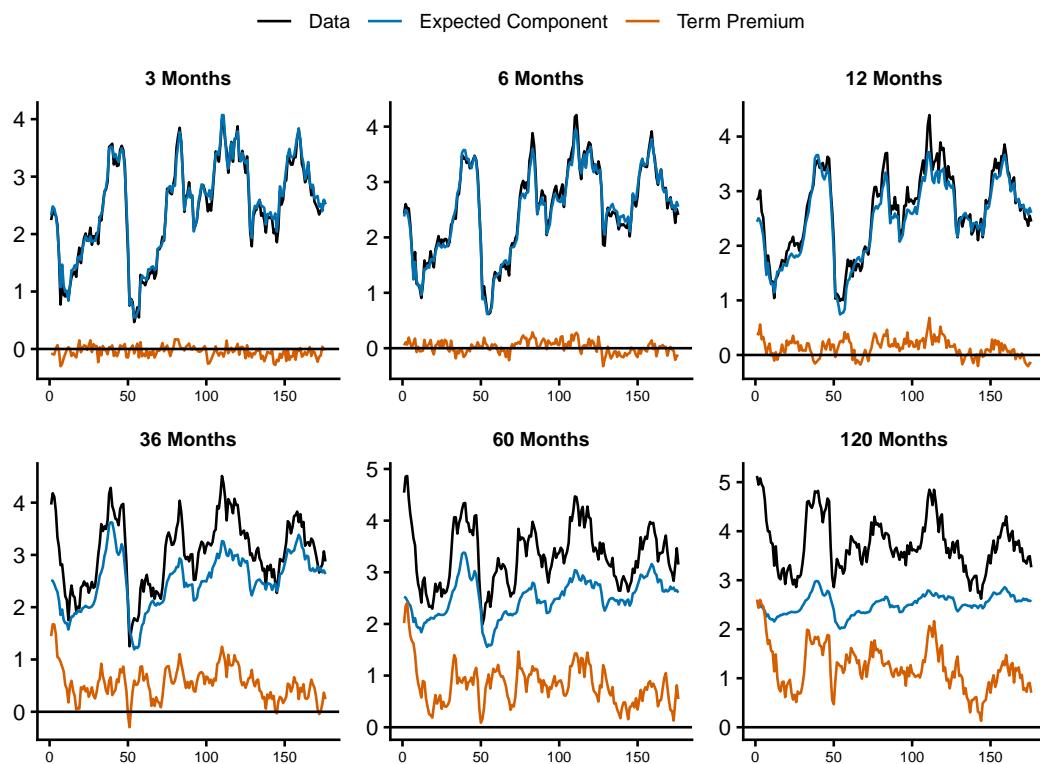


Figure 5: Chinese sovereign yield curve decomposition showing (i) expected future short rates and (ii) term premia components. The x -axis represents time in months and the y -axis is expressed in percentage points.

0.8 Concluding remarks

The **MultiATSM** package aims to advance yield curve (term structure) modelling within the R programming environment. It provides a comprehensive yet user-friendly toolkit for practitioners, academics, and policymakers, featuring estimation routines and generating detailed outputs across several macrofinance model classes. This allows for an in-depth exploration of the relationship between the real economy developments and the fixed income markets.

The package covers eight classes of macrofinance term structure models, all built upon the single-country unspanned macroeconomic risk framework of [Joslin et al. \(2014\)](#), which is also extended to a multicountry setting. Additional multicountry variants based on [Jotikasthira et al. \(2015\)](#) and [Candelon and Moura \(2024\)](#) are included, incorporating, respectively, a dominant economy and a GVAR structure to model cross-country interdependence.

Each model class provides analytical outputs that offer insight into term structure dynamics, including plots of model fit, risk premia, impulse responses, and forecast error variance decompositions. The **MultiATSM** package also offers bootstrap procedures for confidence interval construction and out-of-sample forecasting of bond yields.

Acknowledgments

I thank the editor, Rob Hyndman, and an anonymous referee for several helpful comments. I am also grateful to Bertrand Candelon, Adhir Dhole and Gustavo Torregrosa for many insightful discussions. An earlier version of this paper circulated under the title *MultiATSM: An R Package for Arbitrage-Free Multicountry Affine Term Structure of Interest Rate Models with Unspanned Macroeconomic Risk* and was part of the author's PhD dissertation at UCLouvain ([Moura, 2022](#)). The views expressed in this paper are those of the author and do not necessarily reflect those of Banco de Mexico.

0.9 Appendix

A: Supplementary functions

Importing data from Excel files The **MultiATSM** package also provides an automated procedure for importing data from Excel files via `Load_Excel_Data()` and preparing the risk factor database used directly in the model estimation. To ensure compatibility with the package functions, the following requirements must be met:

1. Databases must be organized in separate Excel files: one for unspanned factors and another for term structure data. For GVAR-based models, a third file containing the interdependence measures is also required;
2. Each Excel file should include one tab per country. In the case of unspanned factors, an additional tab must be included for the global variables if the user opts to incorporate them;
3. Variable names must be identical across all tabs within each file.

An example Excel file meeting these requirements is provided with the package. Below is an example of how to import the data from excel and construct the input list to be supplied:

```
MacroData <- Load_Excel_Data(system.file("extdata", "MacroData.xlsx",
                                         package = "MultiATSM"))
YieldsData <- Load_Excel_Data(system.file("extdata", "YieldsData.xlsx",
                                         package = "MultiATSM"))

ModelType <- "JPS original"
Initial_Date <- "2006-09-01"
Final_Date <- "2019-01-01"
```

```
DataFrequency <- "Monthly"
GlobalVar <- c("GBC", "VIX")
DomVar <- c("Eco_Act", "Inflation", "Com_Prices", "Exc_Rates")
N <- 3
Economies <- c("China", "Mexico", "Uruguay", "Brazil", "Russia")
```

These inputs are used to construct the *RiskFactorsSet* variable, which holds the full collection of risk factors required by the model.

```
FactorLabels <- LabFac(N, DomVar, GlobalVar, Economies, ModelType)
RiskFactorsSet <- DataForEstimation(Initial_Date, Final_Date, Economies, N, FactorLabels,
ModelType, DataFrequency, MacroData, YieldsData)
```

Transition matrix and star factors To construct the transition matrix for GVAR specifications, the user can employ *Transition_Matrix()*. This function requires:

1. Data selection: choose proxies for cross-country interdependence.
2. Time frame: specify the sample's start and end dates.
3. Dependence measure: select from:
 - Time-varying (dynamic weights)
 - Sample Mean (static average)
 - A numeric scalar (fixed-year snapshot).

```
data("TradeFlows")
t_First <- "2006"
t_Last <- "2019"
Economies <- c("China", "Brazil", "Mexico", "Uruguay")
type <- "Sample Mean"
W_gvar <- Transition_Matrix(t_First, t_Last, Economies, type, TradeFlows)
```

Note that if data is missing for any country in a given year, the corresponding transition matrix will contain only NAs.

A more flexible approach to modelling interdependence is to allow the transition matrix to vary over time. In this case, the star factors are constructed using trade flow weights specific to each year, adjusting the corresponding year's risk factors accordingly. To enable this feature, users must set the *type* argument to *Time-varying* and specify the same year for both the initial and final periods in the transition matrix. This indicates that the trade weights from that particular year is used when solving the GVAR system (i.e., in the construction of the link matrices, see Equation (11)).

B: Additional theoretical considerations

Bond yield decomposition The **MultiATSM** package allows for the calculation of two risk compensation measures: term premia and forward premia. Assume that an n -maturity bond yield can be decomposed into two components: the expected short-rate ($\text{Exp}_{i,t}^{(n)}$) and term premia ($\text{TP}_{i,t}^{(n)}$). Technically:

$$y_{i,t}^{(n)} = \text{Exp}_{i,t}^{(n)} + \text{TP}_{i,t}^{(n)}.$$

In the package's standard form, the expected short rate term is computed from time t to $t+n$, which represents the bond's maturity: $\text{Exp}_{i,t}^{(n)} = \sum_{h=0}^n E_t[y_{i,t+h}^{(1)}]$. Alternatively, the decomposition for the forward rates ($f_{i,t}^{(n)}$) is $f_{i,t}^{(n)} = \sum_{h=m}^n E_t[y_{i,t+h}^{(1)}] + \text{FP}_{i,t}^{(n)}$ where

$\text{FP}_{i,t}^{(n)}$ corresponds to the forward premia. In this case, the user must specify TRUE if the computation of forward premia is desired, or FALSE otherwise. If set to TRUE, the user must also provide a two-element numerical vector containing the maturities corresponding to the starting and ending dates of the bond maturity. Example:

```
WishFPremia <- TRUE
FPmatLim <- c(60, 120)
```

C: Replication of existing research

Joslin, Priebsch and Singleton (2014) The dataset used in this replication was constructed by [Bauer and Rudebusch \(2017\)](#) (henceforth BR, 2017) and is available on Bauer's website. In their paper, BR (2017) investigate whether macrofinance term structure models are better suited to the unspanned macro risk framework of JPS (2014) or to earlier, traditional spanned settings such as [Ang and Piazzesi \(2003\)](#). To that end, BR (2017) replicate selected empirical results from JPS (2014). The corresponding R code is also available on Bauer's website.

Using the dataset from BR (2017), the code below applies the [MultiATSM](#) package to estimate the key ATSM parameters following the JPS original modelling setup.

```
# 1) INPUTS
# A) Load database data
LoadData("BR_2017")

# B) GENERAL model inputs
ModelType <- "JPS original"

Economies <- c("US")
GlobalVar <- c()
DomVar <- c("GRO", "INF")
N <- 3
t0_sample <- "January-1985"
tF_sample <- "December-2007"
DataFreq <- "Monthly"
StatQ <- FALSE

# 2) Minor preliminary work
FactorLabels <- LabFac(N, DomVar, GlobalVar, Economies, ModelType)
Yields <- t(BR_jps_out$Y)
DomesticMacroVar <- t(BR_jps_out$M.o)
GlobalMacroVar <- c()

# 3) Prepare the inputs of the likelihood function
ATSMInputs <- InputsForOpt(t0_sample, tF_sample, ModelType, Yields, GlobalMacroVar,
                           DomesticMacroVar, FactorLabels, Economies, DataFreq)

# 4) Optimization of the model
ModelPara <- Optimization(ATSMInputs, StatQ, DataFreq, FactorLabels, Economies, ModelType)
```

The tables below compare the ATSM parameter estimates generated from BR (2017) and the [MultiATSM](#). Table 2 reports the risk-neutral parameters. While the values presented do not match exactly, the differences are well within convergence tolerance and are arguably economically negligible. Table 3, by contrast, contains parameters related to the model's time-series dynamics. As these are derived in closed form, the estimates are exactly the same under both specifications.

Note: λ 's are the eigenvalues from the risk-neutral feedback matrix and r_0 is the long-run mean of the short rate under \mathbb{Q} .

Table 2: *Q*-dynamics parameters

	MultiATSM	BR (2017)
r_0	0.0006	-0.0002
λ_1	0.9967	0.9968
λ_2	0.9149	0.9594
λ_3	0.9149	0.8717

Table 3: *P*-dynamics parameters

	K1Z				
	PC1	PC2	PC3	GRO	INF
BR (2017)					
PC1	0.0781	0.9369	-0.0131	-0.0218	0.1046
PC2	0.0210	0.0058	0.9781	0.1703	-0.1672
PC3	0.1005	-0.0104	-0.0062	0.7835	-0.0399
GRO	0.0690	-0.0048	0.0180	-0.1112	0.8818
INF	0.0500	0.0018	0.0064	-0.0592	0.0277
MultiATSM					
PC1	0.0781	0.9369	-0.0131	-0.0218	0.1046
PC2	0.0210	0.0058	0.9781	0.1703	-0.1672
PC3	0.1005	-0.0104	-0.0062	0.7835	-0.0399
GRO	0.0690	-0.0048	0.0180	-0.1112	0.8818
INF	0.0500	0.0018	0.0064	-0.0592	0.0277

Note:

K0Z is the intercept and *K1Z* is the feedback matrix from the *P*-dynamics.

For replicability, it is important to note that the physical dynamics results reported in Table 3 using **MultiATSM** rely on the principal component weights provided by BR (2017). Such a matrix is simply a scaled-up version of the one provided by the function `pca_weights_one_country()` of the current package. Accordingly, despite the numerical differences on the weight matrices, both methods generate time series of spanned factors which are perfectly correlated. Another difference between the two approaches relates to the construction of the log-likelihood function: while in the BR (2017) code this is expressed in terms of a portfolio of yields, the **MultiATSM** package generates this same input directly as a function of observed yields (i.e. both procedures lead to equivalent log-likelihood values up to the Jacobian term).

Additionally, it is worth highlighting that the standard deviations for the portfolios of yields observed with errors are nearly identical, matching to seven decimal places: 0.0000546 for **MultiATSM** and 0.0000550 for BR (2017).

Candelon and Moura (2024) The multicountry framework introduced in [Candelon and Moura \(2024\)](#) enhances the tractability of large-scale ATSMs and deepens our understanding of the global economic mechanisms driving domestic yield curve fluctuations. This framework also generates more precise model estimates and enhances the forecasting capabilities of these models. This novel setup, embodied by the GVAR `multi` model class, is benchmarked against the findings of [Jotikasthira et al. \(2015\)](#), which are captured by the JLL original model class. The paper showcases an empirical illustration involving China, Brazil, Mexico, and Uruguay.

```
# 1) INPUTS
# A) Load database data
LoadData("CM_2024")

# B) GENERAL model inputs
ModelType <- "GVAR multi"
Economies <- c("China", "Brazil", "Mexico", "Uruguay")
GlobalVar <- c("Gl_Eco_Act", "Gl_Inflation")
DomVar <- c("Eco_Act", "Inflation")
```

```

N <- 3
t0_sample <- "01-06-2004"
tF_sample <- "01-01-2020"
OutputLabel <- "CM_jfec"
DataFreq <-"Monthly"
StatQ <- FALSE

# B.1) SPECIFIC model inputs
# GVAR-based models
GVARlist <- list( VARXtype = "unconstrained", W_type = "Sample Mean", t_First_Wgvar = "2004",
                  t_Last_Wgvar = "2019", DataConnectedness = TradeFlows )

# JLL-based models
JLLlist <- list(DomUnit = "China")

# BRW inputs
WishBC <- TRUE
BRWlist <- within(list(Cent_Measure = "Mean", gamma = 0.001, N_iter = 200, B = 50, checkBRW = TRUE,
                        B_check = 1000, Eigen_rest = 1), N_burn <- round(N_iter * 0.15))

# C) Decide on Settings for numerical outputs
WishFPremia <- TRUE
FPmatLim <- c(24,36)

Horiz <- 25
DesiredGraphs <- c("GIRF", "GFEVD", "TermPremia")
WishGraphRiskFac <- FALSE
WishGraphYields <- TRUE
WishOrthoJLLgraphs <- TRUE

# D) Bootstrap settings
WishBootstrap <- FALSE
BootList <- list(methodBS = 'bs', BlockLength = 4, ndraws = 1000, pctg = 95)

# E) Out-of-sample forecast
WishForecast <- TRUE
ForecastList <- list(ForHoriz = 12, t0Sample = 1, t0Forecast = 100, ForType = "Rolling")

# 2) Minor preliminary work: get the sets of factor labels and a vector of common maturities
FactorLabels <- LabFac(N, DomVar, GlobalVar, Economies, ModelType)

# 3) Prepare the inputs of the likelihood function
ATSMInputs <- InputsForOpt(t0_sample, tF_sample, ModelType, Yields, GlobalMacro,
                             DomMacro, FactorLabels, Economies, DataFreq,
                             GVARlist, JLLlist, WishBC, BRWlist)

# 4) Optimization of the ATSM (Point Estimates)
ModelParaList <- Optimization(ATSMInputs, StatQ, DataFreq, FactorLabels, Economies, ModelType)

# 5) Numerical and graphical outputs
# a) Prepare list of inputs for graphs and numerical outputs
InputsForOutputs <- InputsForOutputs(ModelType, Horiz, DesiredGraphs, OutputLabel, StatQ,
                                       DataFreq, WishGraphYields, WishGraphRiskFac,
                                       WishOrthoJLLgraphs, WishFPremia, FPMatLim,
                                       WishBootstrap, BootList, WishForecast,
                                       ForecastList)

```

```

# b) Fit, IRF, FEVD, GIRF, GFEVD, and Term Premia
NumericalOutputs <- NumOutputs(ModelType, ModelParaList, InputsForOutputs,
                                FactorLabels, Economies)

# c) Confidence intervals (bootstrap analysis)
BootstrapAnalysis <- Bootstrap(ModelType, ModelParaList, NumericalOutputs, Economies,
                                 InputsForOutputs, FactorLabels, JLLlist, GVARlist,
                                 WishBC, BRWlist)

# 6) Out-of-sample forecasting
Forecasts <- ForecastYields(ModelType, ModelParaList, InputsForOutputs, FactorLabels,
                             Economies, JLLlist, GVARlist, WishBC, BRWlist)

```

Candelon and Moura (2023) In this paper, [Candelon and Moura \(2023\)](#) investigate the underlying factors that shape the sovereign yield curves of Brazil, India, Mexico, and Russia during the COVID–19 pandemic crisis. The study adopts a GVAR multi approach to capture the complex global macrofinancial, and especially health-related interdependencies during the latest pandemic.

```

# 1) INPUTS
# A) Load database data
LoadData("CM_2023")

# B) GENERAL model inputs
ModelType <- "GVAR multi"
Economies <- c("Brazil", "India", "Russia", "Mexico")
GlobalVar <- c("US_Output_growth", "China_Output_growth", "SP500")
DomVar <- c("Inflation", "Output_growth", "CDS", "COVID")
N <- 2
t0_sample <- "22-03-2020"
tF_sample <- "26-09-2021"
OutputLabel <- "CM_EM"
DataFreq <-"Weekly"
StatQ <- FALSE

# B.1) SPECIFIC model inputs
# GVAR-based models
GVARlist <- list(VARXtype = "constrained: COVID", W_type = "Sample Mean",
                  t_First_Wgvar = "2015", t_Last_Wgvar = "2020",
                  DataConnectedness = TradeFlows_covid)

# BRW inputs
WishBC <- FALSE

# C) Decide on Settings for numerical outputs
WishFPremia <- TRUE
FPmatLim <- c(47,48)

Horiz <- 12
DesiredGraphs <- c("GIRF", "GFEVD", "TermPremia")
WishGraphRiskFac <- FALSE
WishGraphYields <- TRUE
WishOrthoJLLgraphs <- FALSE

# D) Bootstrap settings

```

```

WishBootstrap <- TRUE
BootList <- list(methodBS = 'bs', BlockLength = 4, ndraws = 100, pctg = 95)

# 2) Minor preliminary work: get the sets of factor labels and a vector of common maturities
FactorLabels <- LabFac(N, DomVar, GlobalVar, Economies, ModelType)

# 3) Prepare the inputs of the likelihood function
ATSMInputs <- InputsForOpt(t0_sample, tF_sample, ModelType, Yields_covid, GlobalMacro_covid,
                           DomMacro_covid, FactorLabels, Economies, DataFreq, GVARlist)

# 4) Optimization of the ATSM (Point Estimates)
ModelParaList <- Optimization(ATSMInputs, StatQ, DataFreq, FactorLabels, Economies, ModelType)

# 5) Numerical and graphical outputs
# a) Prepare list of inputs for graphs and numerical outputs
InputsForOutputs <- InputsForOutputs(ModelType, Horiz, DesiredGraphs, OutputLabel, StatQ,
                                      DataFreq, WishGraphYields, WishGraphRiskFac,
                                      WishOrthoJLLgraphs, WishFPremia, FPMatLim,
                                      WishBootstrap, BootList)

# b) Fit, IRF, FEVD, GIRF, GFEVD, and Term Premia
NumericalOutputs <- NumOutputs(ModelType, ModelParaList, InputsForOutputs, FactorLabels,
                                  Economies)

# c) Confidence intervals (bootstrap analysis)
BootstrapAnalysis <- Bootstrap(ModelType, ModelParaList, NumericalOutputs, Economies,
                                 InputsForOutputs, FactorLabels,
                                 JLLlist = NULL, GVARlist)

```

References

- M. Abbritti, S. Dell’Erba, A. Moreno, and S. Sola. Global factors in the term structure of interest rates. *International Journal of Central Banking*, 14(2):301–339, March 2018. URL <https://www.ijcb.org/journal/ijcb18q1a7.htm>. [p282]
- T. Adrian, R. K. Crump, and E. Moench. Pricing the term structure with linear regressions. *Journal of Financial Economics*, 110(1):110–138, 2013. URL <https://doi.org/10.1016/j.jfineco.2013.04.009>. [p283]
- A. Ang and M. Piazzesi. A no-arbitrage vector autoregression of term structure dynamics with macroeconomic and latent variables. *Journal of Monetary Economics*, 50(4):745–787, 2003. URL [https://doi.org/10.1016/S0304-3932\(03\)00032-1](https://doi.org/10.1016/S0304-3932(03)00032-1). [p275, 277, 282, 297]
- M. D. Bauer and G. D. Rudebusch. Resolving the spanning puzzle in macro-finance term structure models. *Review of Finance*, 21(2):511–553, 2017. URL <https://doi.org/10.1093/rof/rfw044>. [p297]
- M. D. Bauer, G. D. Rudebusch, and J. C. Wu. Correcting estimation bias in dynamic term structure models. *Journal of Business & Economic Statistics*, 30(3):454–467, 2012. URL <https://doi.org/10.1080/07350015.2012.693855>. [p283, 284]
- D. Beijers. *statespacer: State Space Modelling in R*, 2023. URL <https://CRAN.R-project.org/package=statespacer>. R package version 0.5.0. [p276]
- M. Boeck, M. Feldkircher, F. Huber, and D. Hosszejni. *BGVAR: Bayesian Global Vector Autoregressions*, 2024. URL <https://CRAN.R-project.org/package=BGVAR>. R package version 2.5.8. [p276]

- B. Candelon and R. Moura. Sovereign yield curves and the COVID-19 in emerging markets. *Economic Modelling*, 127:106453, 2023. URL <https://doi.org/10.1016/j.econmod.2023.106453>. [p281, 300]
- B. Candelon and R. Moura. A multicountry model of the term structures of interest rates with a GVAR. *Journal of Financial Econometrics*, 22(5):1558–1587, 2024. URL <https://doi.org/10.1093/jjfinec/nbae008>. [p275, 281, 282, 295, 298]
- A. Chudik and M. H. Pesaran. Theory and practice of GVAR modelling. *Journal of Economic Surveys*, 30(1):165–197, 2016. URL <https://doi.org/10.1111/joes.12095>. [p278]
- Q. Dai and K. J. Singleton. Specification analysis of affine term structure models. *Journal of Finance*, 55(5):1943–1978, 2000. URL <https://doi.org/10.1111/0022-1082.00278>. [p277]
- Q. Dai and K. J. Singleton. Expectation puzzles, time-varying risk premia, and affine models of the term structure. *Journal of Financial Economics*, 63(3):415–441, 2002. URL [https://doi.org/10.1016/S0304-405X\(02\)00067-3](https://doi.org/10.1016/S0304-405X(02)00067-3). [p275]
- D. Duffie and R. Kan. A yield-factor model of interest rates. *Mathematical Finance*, 6(4):379–406, 1996. URL <https://doi.org/10.1111/j.1467-9965.1996.tb00123.x>. [p275]
- S. S. Guirreri. *YieldCurve: Modelling and Estimation of the Yield Curve*, 2015. URL <https://CRAN.R-project.org/package=YieldCurve>. R package version 4.1. [p276]
- R. S. Gürkaynak and J. H. Wright. Macroeconomics and the term structure. *Journal of Economic Literature*, 50(2):331–67, 2012. URL <https://www.aeaweb.org/articles?id=10.1257/jel.50.2.331>. [p275]
- J. L. Horowitz. Bootstrap methods in econometrics. *Annual Review of Economics*, 11(1):193–224, 2019. URL <https://doi.org/10.1146/annurev-economics-080218-025651>. [p286]
- R. J. Hyndman and R. Killick. CRAN task view: Time series analysis. 2025. URL <https://cran.r-project.org/web/views/TimeSeries.html>. [p276]
- S. Joslin, K. J. Singleton, and H. Zhu. A new perspective on Gaussian dynamic term structure models. *Review of Financial Studies*, 24(3):926–970, 2011. URL <https://doi.org/10.1093/rfs/hhq128>. [p277, 290]
- S. Joslin, M. Priebsch, and K. J. Singleton. Risk premiums in dynamic term structure models with unspanned macro risks. *Journal of Finance*, 69(3):1197–1233, 2014. URL <https://doi.org/10.1111/jofi.12131>. [p275, 282, 283, 295]
- C. Jotikasthira, A. Le, and C. Lundblad. Why do term structures in different currencies co-move? *Journal of Financial Economics*, 115:58–83, 2015. URL <https://doi.org/10.1016/j.jfineco.2014.09.004>. [p275, 281, 282, 295, 298]
- L. Kilian and H. Lütkepohl. *Structural Vector Autoregressive Analysis*. Cambridge University Press, 2017. URL <https://doi.org/10.1017/9781108164818>. [p286]
- A. Lange, B. Dalheimer, H. Herwartz, S. Maxand, and H. Riebl. *svars: Data-Driven Identification of SVAR Models*, 2023. URL <https://CRAN.R-project.org/package=svars>. R package version 1.3.11. [p276]
- R. Litteman and J. Scheinkman. Common factors affecting bond returns. *Journal of Fixed Income*, 1:54–61, 1991. doi: 10.3905/jfi.1991.692347. [p283, 287]
- R. Moura. *MultiATSM: Multicountry Term Structure of Interest Rates Models*, 2025. URL <https://CRAN.R-project.org/package=MultiATSM>. R package version 1.5.1. [p275]
- R. G. T. d. Moura. Modelling the term structure of interest rates in a multicountry setting. 2022. URL <https://dial.uclouvain.be/pr/boreal/object/boreal:262850>. [p295]

- C. R. Nelson and A. F. Siegel. Parsimonious modeling of yield curves. *Journal of Business*, pages 473–489, 1987. URL <https://www.jstor.org/stable/2352957>. [p276]
- W. Nicholson, D. Matteson, and J. Bien. *BigVAR: Dimension Reduction Methods for Multivariate Time Series*, 2025. URL <https://CRAN.R-project.org/package=BigVAR>. R package version 1.1.3. [p276]
- H. H. Pesaran and Y. Shin. Generalized impulse response analysis in linear multivariate models. *Economics Letters*, 58(1):17–29, 1998. URL [https://doi.org/10.1016/S0165-1765\(97\)00214-0](https://doi.org/10.1016/S0165-1765(97)00214-0). [p285]
- I. J. A. Pesigan. *simStateSpace: Simulate Data from State Space Models*, 2025. URL <https://CRAN.R-project.org/package=simStateSpace>. R package version 1.2.10. [p276]
- B. Pfaff and M. Stigler. *vars: VAR Modelling*, 2024. URL <https://CRAN.R-project.org/package=vars>. R package version 1.6-1. [p276]
- M. Piazzesi. Affine term structure models. In *Handbook of Financial Econometrics: Tools and Techniques*, pages 691–766. Elsevier, 2010. URL <https://doi.org/10.1016/B978-0-444-50897-3.50015-8>. [p275]
- G. D. Rudebusch and T. Wu. A macro-finance model of the term structure, monetary policy and the economy. *The Economic Journal*, 118(530):906–926, 2008. URL <https://doi.org/10.1111/j.1468-0297.2008.02155.x>. [p275]
- T. Setz. *fBonds: Rmetrics - Pricing and Evaluating Bonds*, 2017. URL <https://CRAN.R-project.org/package=fBonds>. R package version 3042.78. [p276]
- L. E. Svensson. Estimating and interpreting forward interest rates: Sweden 1992-1994, 1994. URL <https://www.nber.org/papers/w4871>. [p276]
- R. S. Tsay, D. Wood, and J. Lachmann. *MTS: All-Purpose Toolkit for Analyzing Multivariate Time Series (MTS) and Estimating Multivariate Volatility Models*, 2022. URL <https://CRAN.R-project.org/package=MTS>. R package version 1.2.1. [p276]
- J. Urbina. *Spillover: Spillover/Connectedness Index Based on VAR Modelling*, 2024. URL <https://CRAN.R-project.org/package=Spillover>. R package version 0.1.1. [p276]
- O. Vasicek. An equilibrium characterization of the term structure. *Journal of Financial Economics*, 5(2):177–188, 1977. URL [https://doi.org/10.1016/0304-405X\(77\)90016-2](https://doi.org/10.1016/0304-405X(77)90016-2). [p275]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p287]
- I. Wilms, D. S. Matteson, J. Bien, S. Basu, and W. N. E. Wegner. *bigtime: Sparse Estimation of Large Time Series Models*, 2023. URL <https://CRAN.R-project.org/package=bigtime>. R package version 0.2.3. [p276]

Rubens Moura
Banco de Mexico
Avenida 5 de Mayo, 2
Mexico City, Mexico
ORCID: 0000-0001-8105-4729
rubens.guimaraes@banxico.org.mx

memshare: Memory Sharing for Multicore Computation in R with an Application to Feature Selection by Mutual Information using PDE

by Michael C. Thrun and Julian Märte

Abstract We present memshare, a package that enables shared-memory multicore computation in R by allocating buffers in C++ shared memory and exposing them to R through ALTREP views. We compare memshare to SharedObject (Bioconductor), discuss semantics and safety, and report a 2x speedup over SharedObject with no additional resident memory in a column-wise apply benchmark. Finally, we illustrate a downstream analytics use case: feature selection by mutual information, in which densities are estimated per feature via Pareto Density Estimation (PDE). The analytical use case is an RNA-seq dataset consisting of $N = 10,446$ cases and $d = 19,637$ gene expression measurements, requiring roughly $n_threads * 10\text{GB}$ of memory in the case of using parallel R sessions. Such and larger use cases are common in big data analytics and make R feel limiting, which is mitigated by the library presented in this work.

0.1 Introduction

Parallel computing in R is usually realized through PSOCK or FORK clusters, where multiple R processes work in parallel ([R Core Team, 2025](#)), ([Corporation and Weston, 2025](#)). A practical issue arises immediately: each worker process receives its own private copy of the data. If a matrix consumes several gigabytes in memory, then spawning ten workers results in ten redundant copies, potentially exhausting system memory and severely reducing performance due to paging. This overhead becomes especially prohibitive in genomics or imaging, where matrices of tens of gigabytes are commonplace. Copying also incurs serialization and deserialization costs when transmitting objects to workers, delaying the onset of actual computation.

Shared memory frameworks address this issue by allowing workers to view the same physical memory without duplication. Instead of copying the whole object, only small handles or identifiers are communicated, while the underlying data is stored once in RAM. This enables efficient multicore computation on large datasets that would otherwise be infeasible.

ALTREP (short for ALTernative REPresentations) is a framework in R that allows vectors or matrices to be backed by alternative storage while still behaving like ordinary R objects. Method hooks determine how to access length, data pointers, and duplication, so that package developers can integrate external memory management, including shared memory segments, without changing downstream R code.

A common alternative to in-RAM shared memory is file-backed memory mapping, where binary files on disk are mapped into R and the operating system pages data into memory on demand ([Kane et al., 2013](#)). Packages such as `bigmemory` and `bigstatsr` create matrices stored on disk but accessible through R as if they were in memory ([Prive et al., 2018](#)). This enables analyses on datasets larger than RAM by working column-wise with a small memory footprint. However, because file-backed matrices rely on disk I/O, they are slower than in-RAM shared memory when a single copy of the data would fit into physical memory, but multiple per-worker copies would not. Therefore, this work focuses on ALTREP-based, in-RAM techniques for data that fit into memory once but not many times. Our contributions are:

1. A fully independent and user-friendly implementation based on the ALTREP framework.
2. A comparison of `memshare` vs `SharedObject`: data model, safety, copy-on-write, and developer surface showing a runtime up to twice as fast for `memshare` on parallel column operations without extra RSS.
3. A practical template for MI-PDE feature selection on RNA-seq.

0.2 Background

A more detailed description of ALTREP internals and our shared-memory use case is provided in Appendix~B; here we summarize only the concepts needed for the comparison of `memshare` to existing approaches.

SharedObject baseline

Having outlined the general ALTREP and shared-memory mechanism, we now briefly review the `SharedObject` package, which serves as our main existing ALTREP-based shared-memory baseline.

`SharedObject` allocates shared segments and wraps them as ALTREP (Wang and Morgan, 2025). It exposes properties like `copyOnWrite`, `sharedSubset`, and `sharedCopy`; it supports atomic types and (with caveats) character vectors. Developers can map or unmap shared regions and query whether an object is shared. `SharedObject` was among the first implementations that showed how ALTREP can enable multicore parallelism by avoiding data duplication. `SharedObject` provides `share()` to wrap an R object as a shared ALTREP, with tunables:

- `copyOnWrite` (default TRUE): duplicates on write; setting FALSE enables in-place edits but is not fully supported and can lead to surprising behavior (e.g., unary minus mutating the source).
- `sharedSubset`: whether slices are also shared; can incur extra duplication in some IDEs; often left FALSE.
- `sharedCopy`: whether duplication of a shared object remains shared. It supports raw, logical, integer, double, or complex and, with restrictions, character (recommended read-only; new, previously unseen strings cannot be assigned). Developers can also directly allocate, map, unmap, and free shared regions and query `is.shared` or `is.altrep`.

R's threading model

R's C API is single-threaded; package code must not call R from secondary threads. Process-level parallelism (clusters) remains the primary avenue. Consequently, shared-memory frameworks must ensure that mutation is either controlled in the main thread or performed at the raw buffer level without touching R internals.

PDE-based Mutual Information

For feature selection with a discrete response Y and a continuous feature X , mutual information can be expressed as:

$$I(X; Y) = \sum_y p(y) KL(p(x|y) || p(x)) \quad (1)$$

This formulation requires only univariate densities $p(x)$ and $p(x|y)$ per class. This lends itself to Pareto Density Estimation (PDE), a density estimator based on hyperspheres with the Pareto radius chosen by an information-optimal criterion. In PDE, the idea is to select a subset S of the data with relative size $p = |S|/|D|$. The information content is $I(p) = -p \ln(p)$. (Ultsch, 2005) showed that the optimal set size corresponds to about 20.1%, retrieving roughly 88% of the maximum possible information. The unrealized potential (URP) quantifies deviation from the optimal set size and is minimized when $p \approx 20\%$. For univariate density estimation this yields the Pareto radius R , which can be approximated by the 18% quantile distance in one dimension. PDE thus adapts the neighborhood size following the Pareto rule (80–20 rule) to maximize information content. Empirical studies report that PDE can outperform standard density estimators under default settings (Thrun et al., 2020).

With respect to the categorical variable, no density estimation is needed, as the most accurate density estimate in this case is simply the relative label count, $p(y) = \frac{\#\{\omega \in \Omega \mid Y(\omega) = y\}}{\#\Omega}$. Here Ω is the set of cases, Y is the categorical random variable, and y runs over the range of $Y, Y(\Omega)$.

0.3 Methods

In idiomatic use, `memshare` coordinates PSOCK clusters (separate processes) that attach to one shared segment. Within workers the relevant shared segments are retrieved and the task is executed on them. The key win is replacing per-worker duplication of the large matrix with a far cheaper retrieval of a handle to the shared memory segment and subsequent wrapping in an ALTREP instead.

The `memshare` API

Shared memory pages in `memshare` are handled by unique string identifiers on the OS side. These identifiers can be requested and retrieved via C/C++. To prevent two master R sessions from accidentally accessing each other's memory space because duplicate allocations can lead to undefined behavior at the OS level, users may define one or more **namespaces** in which the current session operates. The `memshare` API closely mirrors C's memory ownership model but applies it to R sessions. A master (primary) session owns the memory, while worker (secondary) sessions can access it.

Shared memory semantics

A crucial aspect of `memshare`'s design is how shared memory is managed and exposed through R. Three definitions clarify the terminology:

A **namespace** refers to a character string that defines the identifier of the shared memory context. It allows the initialization, retrieval, and release of shared variables under a common label, ensuring that multiple sessions or clusters can coordinate access to the same objects. While this does not provide absolute protection, it makes it the user's responsibility to avoid assigning the same namespace to multiple master sessions.

Pages are variables owned by the current compilation unit of the code, such as the R session or terminal that loaded the DLL. Pages are realized as shared memory objects: on Windows via `MapViewOfFile`, and on Unix systems via `shm` in combination with `mmap`.

Views are references to variables owned by another or the same compilation unit. Views are always ALTREP wrappers, providing pointers to the shared memory chunk so that R can interact with them as if they were ordinary vectors or matrices.

Together, these concepts enforce a lifecycle: pages represent ownership of memory segments, views represent references to them, and namespaces serve as the coordination mechanism. The combination guarantees both memory efficiency and safety when performing multicore computations across R processes.

If the user detaches the `memshare` package, all handles are destroyed. This means that all variables of all namespaces are cleared, provided there is no other R thread still using them. In other words, unloading the package cleans up shared memory regions and ensures that no dangling references remain. Other threads still holding a handle to the memory will prevent this cleanup, as it would invalidate the working memory of those threads. The shared memory is then cleared whenever all handles are released.

Master session A master session takes ownership of a memory page using:

- `registerVariables(namespace, variableList)`

where `variableList` is a named list of supported types. These are double matrices, double vectors, or lists of these. The names define the memory pages ID through which they can be accessed, while the values of the actual variables define the size and content of the memory page.

To deallocate memory pages, the master session can call:

- `releaseVariables(namespace, variableNames)`

where `variableNames` is a character vector containing the names of previously shared variables.

`memshare` also allows for releasing all the memory allocated for a given namespace by a memory context, i.e., a parallel cluster with a master session, via the

- `memshare_gc(namespace, cluster)`

function. This first removes every view handle in the context and then releases all pages.

Note. Memory pages are not permanently deallocated if another session still holds a view of them. This ensures stability: allowing workers to continue with valid but outdated memory is safer than letting them access invalidated memory. However, releasing variables still in use is always a user error and must be avoided.

Worker session Once the master session has shared variables, worker sessions can retrieve them via:

- `retrieveViews(namespace, variableNames)`

This returns a named list of R objects. These objects are raw ALTREP objects indistinguishable from the originals (`is.matrix`, `is.numeric`, etc.), and all behave the same.

When operating on these objects, workers interact directly with the underlying C buffer, backed by `mmap` (Unix) or `MapViewOfFile` (Windows). Changes to such objects modify the shared memory for all sessions. In this framework, however, modification is secondary—the main goal is to transfer data from the master to worker sessions.

For metadata access without retrieving views, workers can call:

- `retrieveMetadata(namespace, variableName)`

which provides information for a single variable.

After processing, workers must return their views to allow memory release by calling:

- `releaseViews(namespace, variableNames)`

The overall high-level concept is summarized in Figure 1.

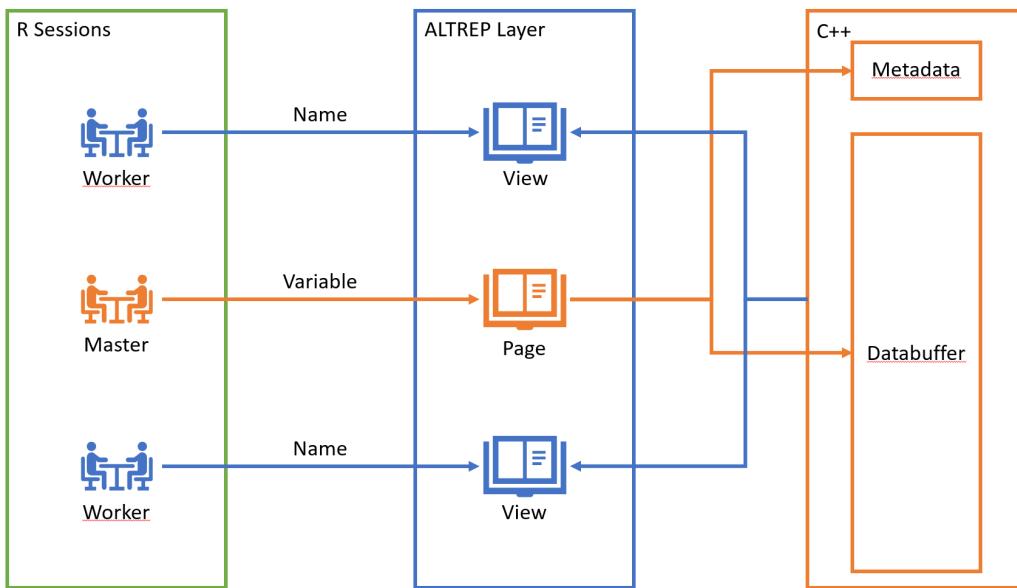


Figure 1: A schematic about where the memory is located and how different sessions access it.

Diagnostic tools To verify correct memory management, two diagnostic functions are available:

- `pageList()`: lists all variables owned by the current session.
- `viewList()`: lists all views (handles) currently held by the session.

The former is stricter, since it identifies ownership, whereas the latter only tracks held views.

User-friendly wrapper functions for `apply` and `lapply` Since memory ownership and address sharing are low-level concepts, `memshare` provides wrapper functions that mimic `parallel::parApply` and `parallel::parLapply`.

- `memApply(X, MARGIN, FUN, NAMESPACE, CLUSTER, VARS, MAX.CORES)`
 - Mimics `parallel::parApply`.
 - `X`: a double matrix.
 - `MARGIN`: direction (1 = row-wise, 2 = column-wise).
 - `FUN`: function applied to rows/columns.
 - `CLUSTER`: a prepared parallel cluster (variables exported via `parallel::clusterExport`).
 - `VARS`: additional shared variables (names must match `FUN` arguments).
 - `MAX.CORES`: only relevant if `CLUSTER` is uninitialized.

`memApply` automatically manages sharing and cleanup of `X` and `VARS`, ensuring no residual C/C++ buffers remain. Both `X` and `VARS` can also refer to previously allocated shared variables, though in that case the user must manage their lifetime.

- `memLapply(X, FUN, NAMESPACE, CLUSTER, VARS, MAX.CORES)`
 - Equivalent to `parallel::parLapply`, but within the `memshare` framework.

- X: a list of double matrices or vectors.
- Other arguments behave the same way as in `memApply`.

Examples of Use

We provide two top-level examples for the use of the `memshare` package: one with `memLapply` and one with `memApply`.

The first example computes the correlation between each column of a matrix and a reference vector using shared memory and `memApply`. The matrix can be provided directly and will be registered automatically, or by name if already registered.

```
library(memshare)
set.seed(1)
n <- 10000
p <- 2000
# Numeric double matrix (required): n rows (cases) x d columns (features)
X <- matrix(rnorm(n * p), n, p)
# Reference vector to correlate with each column
y <- rnorm(n)
f <- function(v, y) cor(v, y)

ns <- "my_namespace"
res <- memshare::memApply(
  X = X, MARGIN = 2,
  FUN = f,
  NAMESPACE = ns,
  VARS = list(y = y),
  MAX.CORES = NULL # defaults to detectCores() - 1
)
```

`memApply` parallelizes a row- or column-wise map over a matrix that lives once in shared memory. If X is passed as an ordinary R matrix, it is registered under a generated name in the namespace ns. Additional variables (here y) can be provided as a named list; these are registered and retrieved as ALTREP views on the workers. A cluster is created automatically if none is provided. Each worker obtains a cached view of the matrix (and any shared variables), extracts the i-th row or column as a vector v according to MARGIN, calls `FUN(v, ...)`, and returns the result. Views are released after the computation, and any objects registered by this call are freed. Because workers operate on shared views rather than copies, the total resident memory remains close to a single in-RAM copy of X, while runtime scales with the available cores.

As a second example, consider a case where a list of 1000 random matrices is multiplied by a random vector. This task is parallelizable at the element level and demonstrates the use of `memshare::memLapply`, which applies a function across list elements in a shared memory context:

```
library(memshare)
list_length <- 1000
matrix_dim <- 100

# Create the list of random matrices
l <- lapply(
  1:list_length,
  function(i) matrix(rnorm(matrix_dim * matrix_dim),
    nrow = matrix_dim, ncol = matrix_dim)
)
```

```

y <- rnorm(matrix_dim)
namespace <- "my_namespace"

res <- memLapply(1, function(el, y) {
  el %*% y
}, NAMESPACE = namespace, VARS = list(y=y),
MAX.CORES = 1) #MAX.CORES=1 for simplicity

```

`memLapply()` provides a parallel version of `lapply()` where the list elements and optional auxiliary variables are stored in shared memory. If the input X is an ordinary R list, it is first registered in a shared memory namespace. Additional variables can be supplied either as names of existing shared objects or as a named list to be registered. A parallel cluster is created automatically if none is provided, and each worker is initialized with the `memshare` environment.

For each index of the list, the worker retrieves an ALTREP view of the corresponding element (and of any shared variables), applies the user-defined function `FUN` to these objects, and then releases the views to avoid memory leaks. The function enforces that the first argument of `FUN` corresponds to the list element and that the names of shared variables match exactly between the namespace and the function signature. Results are collected with `parLapply`, yielding an ordinary R list of the same length as the input.

Because only lightweight references to the shared objects are passed to the workers, no duplication of large data occurs, making the approach memory-efficient. Finally, `memLapply()` includes cleanup routines to release temporary registrations, stop the cluster if it was created internally, and free shared memory, ensuring safe reuse in subsequent computations.

Benchmark design

We compare `memshare` and `SharedObject` on a column-wise apply task across square matrices of sizes $10^i \times 10^i$ for $i = 1, \dots, 5$. We use a PSOCK cluster with 32 cores on an iMac Pro, 256 GB DDR4, 2.3 GHz 18-core Intel Xeon W on macOS Sequoia 16.6.1 (24G90) with R 4.5.1 x86_64-apple-darwin20. For each size, we run 100 repetitions and recorded wall-clock times and resident set size (RSS) across all worker PIDs plus the master. The RSS is summed via `ps()` and our helper `total_rss_mb()`. We define the memory overhead as the difference in total RSS before and after the call, i.e., we measure the additional memory required by the computation beyond the base process footprint.

For `SharedObject` we create `A2`, `share(A1)`, and `parApply()`; for `memshare` we call `memApply` directly on `A1` with a namespace, so that only ALTREP views are created on the workers. A serial baseline uses `apply()`, and an additional baseline uses `parApply()` without shared memory. A minimally edited version of the full script (setup, PID collection, loops, and data saving) is provided in Appendix~A to ensure reproducibility.

As part of our safety and lifecycle checks, we ensure that views, which keep shared segments alive, are always released in the workers before returning control. Once all work is complete, the corresponding variables are then released in the master. To maintain fairness, we avoid creating incidental copies, such as those introduced by coercions, remove variables with `rm()`, and use R's garbage collection `gc()` after each call.

RNA-seq dataset via FireBrowse

FireBrowse ([Broad Institute of MIT and Harvard, 2025](#)) delivers raw counts of gene expression indexed by NCBI identifiers. For each gene identifier i (from Ensembl or NCBI), we obtain a raw count r_i that quantifies the observed read abundance. These raw counts represent the number of reads mapped to each gene, without length normalization. To convert raw counts into TPM (transcripts per million) ([Li and Dewey, 2011](#)), we require gene or transcript lengths l_i . For each gene i , we compute:

$$\hat{r}_i = \frac{r_i}{l_i} \quad (2)$$

The total sum $R = \sum_i \hat{r}_i$ across all genes is then used to scale values as:

$$TPM_i = \frac{\hat{r}_i}{R} \times 10^6 \quad (3)$$

This transformation allows comparison of expression levels across genes and samples by correcting for gene length and sequencing depth (Li and Dewey, 2011). After transformation, our dataset consists of $d = 19,637$ gene expressions across $N = 10,446$ cases spanning 32 diagnoses. It can be found under (Thrun and Märte, 2025).

0.4 Results

In the first subsection, the efficiency of `memshare` is compared to `SharedObject`, and in the second subsection the application is presented.

Performance and Memory

In Figure 2, the results for square matrices of increasing magnitudes are shown as summary plots with variability bars. In these error-bar-style plots the bars indicate median \pm AMAD. The bottom subfigures (C–D) provide a zoomed view of the first four matrix sizes, while the top subfigures (A–B) display all five magnitudes from 10^1 to 10^5 . The x-axis represents the magnitude of the matrix size, and the y-axis shows either runtime in seconds (subfigures A and C) or memory overhead in megabytes (subfigures B and D). Memory overhead is measured as the difference in total RSS. For each magnitude, 100 trials were conducted. The magenta line indicates the performance of the single-threaded R baseline.

Table~1 reports the median runtime and memory overhead, while variability is visualized in Figure~4 via the scatter of 100 runs and summarized numerically by the robust AMAD dispersion statistic in Table~2.

`memshare` (orange diamonds) consistently outperforms `SharedObject` (blue circles) in Figure 2. In the scatter plot of the 100 trials in Figure~4, it is evident that for the first three magnitudes both packages use less memory than the baseline. At 10^4 , average memory usage is comparable to the baseline, while at 10^5 , `memshare` slightly exceeds it. `SharedObject`, however, could only be executed at this magnitude from the terminal, but not within RStudio (see Appendix~B).

Considering relative differences (Ultsch, 2008), `memshare` achieves computation times that are 90–170% faster than `SharedObject`. For matrices of size 10^2 and larger, memory consumption is reduced by 132–153% compared to `SharedObject`.

Table 1: The benchmark compares four types: `memshare`, `SharedObject`, a single-threaded baseline, and a parallel baseline. For `memshare` and `SharedObject`, the reported values are the medians over 100 iterations, while the baselines are the result from either a single-threaded R or a simple `parApply` run using one iteration. Magnitude refers to the matrix size. Entries are given as *Time Consumed (Memory Overhead)*, where time is measured in seconds and memory in megabytes (MB); the memory after call is mentioned in Appendix C.

Type / Magnitude	Baseline	Baseline <code>parApply</code>	<code>SharedObject</code>	<code>memshare</code>
1	0.0003 (0.0234)	0.0049 (0.9023)	0.2492 (0.0801)	0.0416 (1.1426)
2	0.0008 (0.1461)	0.0034 (0.1461)	0.2531 (0.5117)	0.0419 (0.3594)
3	0.0231 (15.2656)	0.0356 (7.6406)	0.3238 (11.6387)	0.0481 (1.4688)
4	2.2322 (1664.9727)	3.5015 (2627.1133)	1.5526 (1570.0566)	0.6655 (895.4473)

Type / Magnitude	Baseline	Baseline parApply	SharedObject	memshare
4.2	9.2883 (4490.4648)	12.9872 (6040.4023)	3.1147 (3901.5137)	1.6223 (3881.7441)
4.5	36.3783 (17206.4688)	53.6183 (24983.7852)	10.3513 (15391.0020)	6.6583 (15285.4258)
4.7	92.0136 (39355.5703)	130.6937 (62936.4766)	32.3116 (38533.7266)	16.3157 (38389.7305)
5	217.4490 (157812.9492)	—	128.0942 (152967.0273)	67.0000 (76311.7402)

Table 2: AMAD for the benchmark of SharedObject vs memshare.

Magnitude	Type	Time Con- sumed (seconds)	Memory Overhead (MB)	Memory after Call
1	SharedObject	0.01241	0.06400	7.47613
2	SharedObject	0.00549	0.05270	12.85171
3	SharedObject	0.00743	0.23716	109.60844
4	SharedObject	0.05179	9.98323	364.98253
4.2	SharedObject	0.13739	9.28681	533.72239
4.5	SharedObject	0.63438	14.73392	1619.81984
4.7	SharedObject	2.09502	15.81807	823.25853
6	SharedObject	4.01802	22.88011	555.74045
1	memshare	0.00198	1.41166	51.33531
2	memshare	0.00251	0.27480	17.12808
3	memshare	0.00228	1.91232	78.65743
4	memshare	0.02038	110.18064	5956.56036
4.2	memshare	0.03206	39.21014	2346.06943
4.5	memshare	0.24571	36.97784	1086.58683
4.7	memshare	0.53134	89.44624	4081.05399
5	memshare	2.48280	31.98623	2413.87405

Application to Feature Selection by Mutual Information using Pareto Density Estimation

The computation of mutual information produced values ranging from 0 to 0.54 (Figure 3). The QQ-plot shows clear deviation from a straight line, indicating that the distribution is not Gaussian. Both the histogram and the PDE plot provide consistent estimates of the probability density, revealing a bimodal structure. The boxplot further highlights the presence of outliers with values above 0.4.

The analysis required about two hours of computation time and approximately 47 GB of RAM, feasible only through memory sharing. In practice, mutual information values can guide feature selection, either by applying a hard threshold or by using a soft approach via a mixture model, depending on the requirements of the subsequent machine learning task.

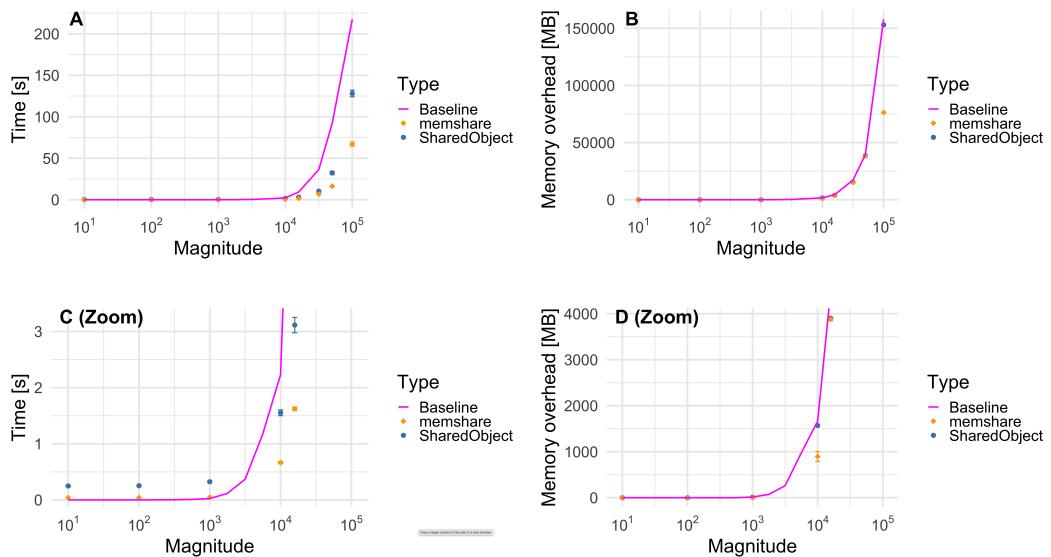


Figure 2: Matrix size depicted as magnitude vs median runtime (left) and vs memory overhead (MB) during the run relative to idle (right) for ‘memshare’, ‘SharedObject’ as error-bar style plots with intervals given by the median \pm AMAD across 100 runs. In addition, the serial baseline is shown as a line in magenta. The top subfigures present the full range of matrix sizes, and the bottom subfigures zoom in.

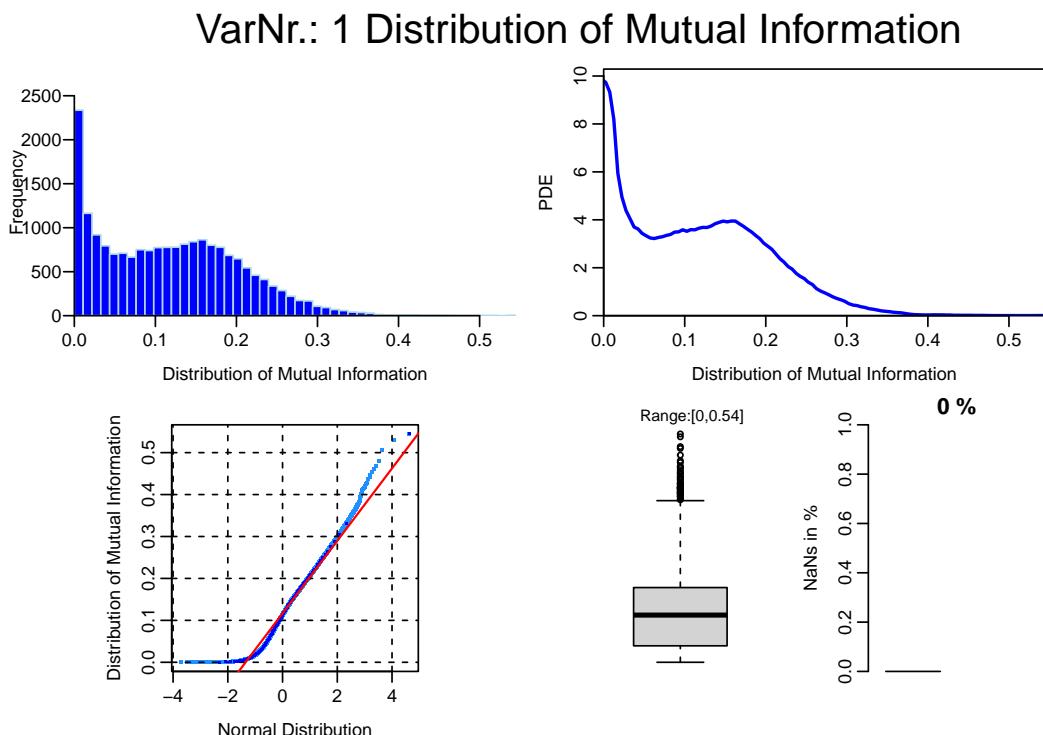


Figure 3: The distribution of mutual information for 19637 gene expressions as a histogram, Pareto Density Estimation (PDE), QQ-plot against normal distribution and boxplot. There are no missing values (NaN).

0.5 Discussion

On all major platforms, PSOCK clusters execute each worker in a separate R process. Consequently, a call to `parApply(c1, X, MARGIN, FUN, . . .)` requires R to serialize the matrix to transmit it to each worker, and deserialize it into a full in-memory copy in every process. As shown in Table 1, this replication leads to out-of-memory (OOM) failures once the matrix reaches a size of $10^5 \times 10^5$, despite the machine providing 256 GB of RAM. Even for smaller magnitudes, substantial redundant memory allocation occurs: multiple workers may begin materializing private copies of the matrix faster than they can complete their portion of the computation, resulting in transient but significant memory amplification. This behavior, inherent to PSOCK-based parallelization, explains the observed OOM conditions under `parApply`.

Consequently, shared memory becomes a foundational requirement for scalable high-performance computing in R, because it avoids redundant data replication and stabilizes memory usage as problem sizes grow. In our experiments, this advantage is reflected directly in performance: across matrix sizes, `memshare` achieved a two-fold reduction in median computation time compared to `SharedObject` on the column-wise task. For large matrix sizes, both `memshare` and `SharedObject` show a lower total memory overhead than the single-threaded baseline, because R’s serial `apply()` implementation creates substantial temporary objects. Each column extraction allocates a full-length numeric vector, and additional intermediates are produced by `FUN`. These private allocations inflate the baseline’s RSS, and memory is not promptly returned to the OS due to R’s garbage-collection strategy. In contrast, `memshare` and `SharedObject` provide ALTREP-backed views into a shared memory segment, eliminating the need to materialize full column vectors.

For matrices of size 10^2 and larger, memory overhead was between half and a third of that of `SharedObject`. At the smallest size (10^1), `memshare` consumed more memory than `SharedObject` because, in R, the metadata must also be shared; this requires a second shared-memory segment whose fixed overhead dominates at small sizes.

The experiments show that `SharedObject` exhibited overhead consistent with copy-on-write materializations and temporary object creation up to size 10^4 . Its memory usage was approximately an order of magnitude higher on macOS than that of `memshare` or the single thread baseline, as illustrated by the triangles aligning with the single-thread baseline of a higher magnitude in Figure 4. For matrices of size 10^5 , `SharedObject` caused RStudio to crash (see appendix~D), although results were computable in R via the terminal.

Beyond these synthetic benchmarks, the RNA-seq case study illustrates how this computational behavior translates into a practical high-dimensional analysis pipeline. Biologically, the bimodal structure in Figure~3 is consistent with a separation between largely uninformative background genes and a smaller subset of diagnosis, or subtype-specific markers. Genes in the lower mode, i.e., with MI values close to zero, show little association with the 32 diagnostic labels and plausibly correspond to broadly expressed housekeeping or pathway genes whose expression varies only weakly across cancer types. In contrast, genes in the upper mode, i.e., MI values in the right-hand peak, exhibit strong dependence on the diagnostic label and are therefore candidates for disease- or tissue-specific markers, including lineage markers and immune-related genes that differ systematically between tumor entities. Although a detailed pathway or gene-set enrichment analysis is beyond the scope of this work, the presence of a distinct high-MI mode indicates that the PDE-based mutual information filter successfully highlights genes whose expression patterns are highly structured with respect to the underlying diagnostic classes and likely reflect underlying molecular subtypes.

At the same time, the fact that this analysis required approximately two hours of computation time and about 47~GB of RAM underscores the central role of memory sharing: without `memshare`, running such a MI-PDE filter on 19,637 genes and 10,446 cases in parallel would be prohibitively memory-intensive on typical multicore hardware. In sum, `memshare` provides a more stable memory-sharing interface, scales more effectively to large matrix sizes, and achieves greater computational efficiency than `SharedObject`.

0.6 Summary

Regardless of the package, R’s single-threaded API implies that multi-threaded computation should touch only raw memory and synchronize results at the main thread boundary. Shared mutation requires external synchronization if multiple workers write to overlapping regions. In practice, read-mostly patterns are ideal.

Here, `memshare`’s namespace + view model and `memApply` wrapper simplify cross-process sharing compared to manual `share()` + cluster wiring. Its explicit release-Views/Variables lifecycle makes retention and cleanup auditable. `SharedObject`’s fine-grained properties are powerful, but the interaction of copy-on-write and duplication semantics increases cognitive load.

`memshare` combines ALTREP-backed shared memory with a pragmatic parallel API to deliver strong speed and memory efficiency on multicore systems. In analytic pipelines like MI-based feature selection for RNA-seq, this enables simple, scalable patterns—one in-RAM copy of the data, many cores, and no serialization overhead.

0.7 Appendix A: code listing of benchmark

The full code is accessible via ([Thrun, 2025](#)). To avoid the crash message for `SharedObject` in Appendix~B, we tried manual garbage collection and performed saves for each iteration without being able to change the outcome. Only by restricting usage to the ‘R’ terminal console without RStudio were we able to compute all results.

0.8 Appendix B: ALTREP and shared memory in R

In R, ALTREP (short for ALTerate REPresentations) is a framework introduced in version 3.5.0 that allows vectors and other objects to be stored and accessed in non-standard ways while maintaining their usual R interface. Built-in type checks cannot tell the difference between an ALTREP object and its ordinary counterpart, which ensures compatibility.

Instead of relying solely on R’s default contiguous in-memory arrays, ALTREP permits objects such as integers, doubles, or strings to be backed by alternative storage mechanisms. Developers can override fundamental methods that govern vector behavior—such as length queries, element access (`DATAPTR`, `DATAPTR_OR_NULL`, etc.), duplication, coercion, and even printing, so that objects can behave normally while drawing data from different sources.

Because these overrides are transparent to R’s higher-level functions, ALTREP objects can be passed, transformed, and manipulated like regular vectors, regardless of whether their contents reside in memory, on disk, or are computed lazily.

For package authors, this framework makes it possible to expose objects that look identical to standard R vectors but internally retrieve their data from sources like memory-mapped files, shared memory, compressed formats, or custom C++ buffers. In practice, this enables efficient handling of large datasets and unconventional data representations while keeping existing R code unchanged.

0.9 Appendix C: benchmarking in detail

In Figure 4, the results are presented as scatter plots for square matrices of increasing magnitudes. The left panel shows detailed scatter plots for the first three matrix sizes, while the right panel summarizes all five magnitudes from 10^1 to 10^5 . The x-axis represents computation time (log seconds), and the y-axis represents memory overhead (log megabytes). It is measured as the difference in total RSS. Each point corresponds to one of 100 trials per magnitude. The magenta baseline indicates the performance of a single-threaded R computation.

To compute the results for Figure 5, we used a PSOCK cluster with 15 workers on a different iMac, namely, 128 GB DDR4, 3.8 GHz 8-Core Intel Xeon W with Windows 10

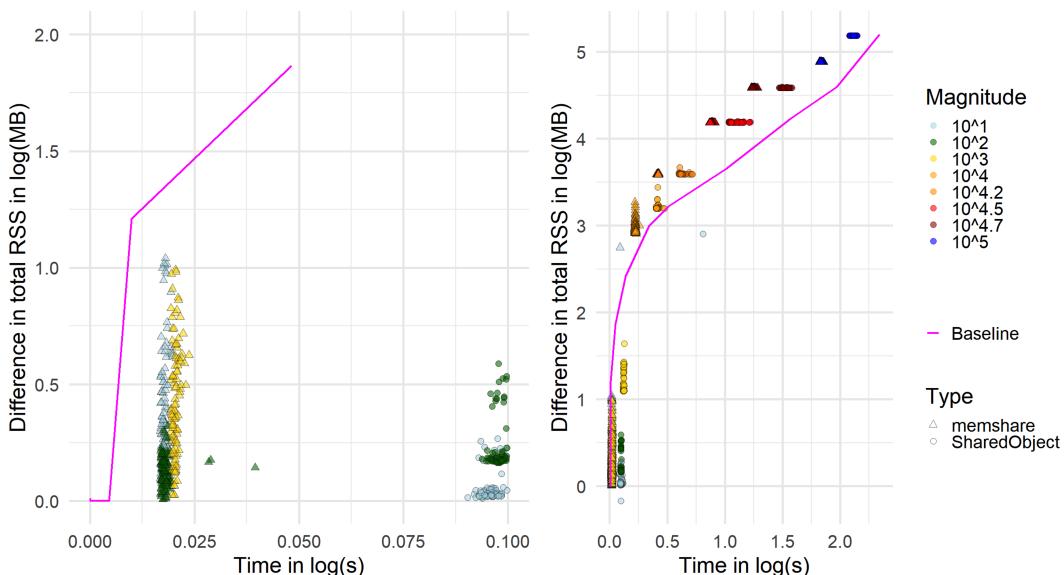


Figure 4: Median runtime (log-scale) vs matrix size for ‘memshare’, ‘SharedObject’, and serial baseline; ribbons show IQR across 100 runs. Insets show difference in total RSS in log(MB), i.e., the memory overhead, during the run relative to idle for Mac presenting the details of Figure effig:figure1, echo=FALSE.

on Boot Camp, and R 4.5.1. For each size, we run 100 repetitions and record wall-clock times and resident set size (RSS) across all worker PIDs plus the master. The results of the benchmark on windows are presented in Table 3 and 4, and in Figure 5.

Within macOs Tahoe, for SharedObject, the memory after a call increases from about 3062 MB at 10^1 to 173,083 MB at exponent 10^5 , i.e. from roughly 3 GB to 169 GB. For memshare, the memory after a call grows from about 3490 MB at 10^1 to 128,393 MB at 10^5 , i.e. from roughly 3.5 GB to 125 GB. Based on these numbers, memshare slightly uses more memory after the call than SharedObject for small and medium problem sizes (exponents 10^1 to $10^{4.7}$, but at the largest matrix size 10^5 memshare requires substantially less overall memory than SharedObject. However, within Windows~10 the situation between SharedObject and memshare changes as depicted in Table 3.

It is important to emphasize that our benchmark was conducted on a specific and somewhat idiosyncratic hardware, i.e., a 2021 iMac running Windows~10 via Boot Camp. This configuration combines Apple hardware, Apple’s firmware and drivers, and Microsoft’s operating system and memory manager in a way that is not representative of typical server or workstation deployments. As a consequence, low-level aspects that are relevant for shared-memory performance, such as page allocation strategy, memory-mapped file handling, and the interaction between R, BLAS, and the operating system scheduler, may differ substantially on other platforms. We therefore refrain from drawing broader conclusions about absolute performance or cross-platform behavior from this benchmark and instead interpret the results as a comparative case study of memshare versus SharedObject on this concrete, well-specified environment.

Table 3: Median runtime and memory overhead for the benchmark on Windows 10 via Boot Camp.

Type	Magnitude	Time Consumed (seconds)	Memory Overhead (MB)	Memory after Call
SharedObject	1	0.0047	0.0391	2870.6816
SharedObject	2	0.0124	0.4648	2873.7266
SharedObject	3	0.0707	13.3691	2929.9121
SharedObject	4	1.0768	2594.9395	5799.9766
SharedObject	4.2	2.0788	7029.2070	10131.3750

Type	Magnitude	Time Consumed (seconds)	Memory Overhead (MB)	Memory after Call
SharedObject	4.5	6.1894	23832.9727	26923.1855
SharedObject	4.7	16.2314	38973.7051	42078.9980
memshare	1	0.0417	1.3848	1609.0801
memshare	2	0.0412	1.3145	1619.4062
memshare	3	0.0487	3.3945	1675.7246
memshare	4	0.6164	764.8184	2823.6172
memshare	4.2	1.4858	3841.7480	5881.9570
memshare	4.5	5.1790	15275.3418	17298.4473
memshare	4.7	13.7764	38336.5898	40407.6250
Baseline	1	0.0002	0.0000	796.5848
Baseline	2	0.0008	0.0021	796.8240
Baseline	3	0.0138	0.1461	816.7769
Baseline	4	1.6573	1686.0985	2542.8888
Baseline	4.2	9.1679	4356.6723	5234.8967
Baseline	4.5	34.7380	17936.2663	18959.0366
Baseline	4.7	86.3598	45320.6208	46571.1692
Baseline	1	0.0041	0.9023	78916.7539
Parallel				
Baseline	2	0.0043	0.0000	739.4570
Parallel				
Baseline	3	0.0389	7.1484	765.4844
Parallel				
Baseline	4	1.9595	1053.2031	3416.6445
Parallel				
Baseline	4.2	10.6068	2621.7109	6546.7422
Parallel				
Baseline	4.5	44.0077	23568.1680	24471.5742
Parallel				
Baseline	4.7	108.1856	58851.6836	59758.3945
Parallel				

Table 4: AMAD for the benchmark grid for SharedObject and memshare on Windows 10 via Boot Camp.

Type	Magnitude	Time Consumed (seconds)	Memory Overhead (MB)	Memory after Call
SharedObject	1	0.00065	0.00000	0.38021
SharedObject	2	0.00126	0.00000	0.26351
SharedObject	3	0.00855	2.11560	1.31002
SharedObject	4	0.02707	5.14219	2.51839
SharedObject	4.2	0.02521	18.96512	2.62003
SharedObject	4.5	0.06582	43.49404	29.24573
SharedObject	4.7	0.22797	186.75258	181.13231
memshare	1	0.00558	0.99004	9.55032
memshare	2	0.00411	1.33637	6.46726
memshare	3	0.00373	6.80983	8.52263
memshare	4	0.01511	304.55616	114.84474
memshare	4.2	0.02341	158.45925	99.27513
memshare	4.5	0.04168	173.53949	125.11030
memshare	4.7	0.09361	148.23133	148.96163

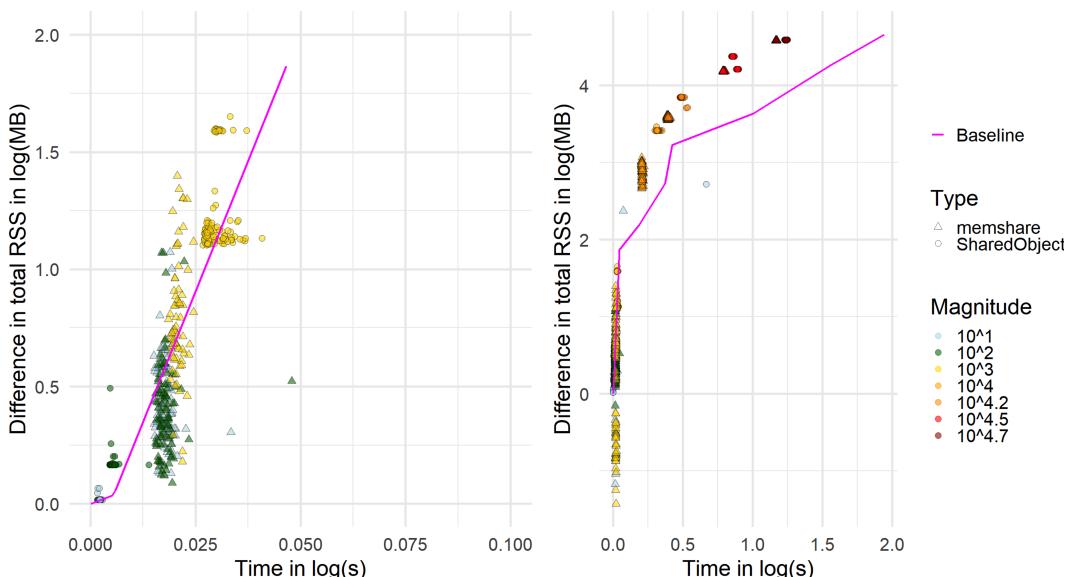


Figure 5: Median runtime (log-scale) vs matrix size for ‘memshare’, ‘SharedObject’, and serial baseline; ribbons show IQR across 100 runs. Insets show the difference in total RSS in log(MB) (i.e., memory overhead) during the run relative to idle for Windows 10 via Boot Camp.

0.10 Appendix D: screenshot

Report as screenshots in Figure 6 and subsequent after forcing to close RStudio in 7 of the crash of RStudio if SharedObject is called with a matrix of size 10^5 .

References

- Broad Institute of MIT and Harvard. Firebrowse (rrid:scr_026320). <http://firebrowse.org/>, 2025. Accessed via <https://gdac.broadinstitute.org>. [p310]
- M. Corporation and S. Weston. *doParallel: Foreach Parallel Adaptor for the ‘parallel’ Package*, 2025. URL <https://github.com/revolutionanalytics/doparallel>. R package version 1.0.17. [p304]
- M. Kane, J. W. Emerson, and S. Weston. Scalable strategies for computing with massive data. *Journal of Statistical Software*, 55(14):1–19, 2013. doi: 10.18637/jss.v055.i14. URL <https://doi.org/10.18637/jss.v055.i14>. [p304]
- B. Li and C. N. Dewey. Rsem: accurate transcript quantification from rna-seq data with or without a reference genome. *BMC Bioinformatics*, 12(1):323, 2011. doi: 10.1186/1471-2105-12-323. URL <https://doi.org/10.1186/1471-2105-12-323>. [p310, 311]
- F. Prive, H. Aschard, A. Ziyatdinov, and M. G. B. Blum. Efficient analysis of large-scale genome-wide data with two R packages: bigstatsr and bigsnpr. *Bioinformatics*, 34(16):2781–2787, 2018. doi: 10.1093/bioinformatics/bty185. URL <https://doi.org/10.1093/bioinformatics/bty185>. [p304]
- R Core Team. *Support for Parallel Computation in R*, 2025. URL <https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf>. R package ‘parallel’ version included in R. [p304]
- M. Thrun. Appendixa: Memshare: Memory sharing for multicore computation in R with an application to feature selection by mutual information using PDE, 2025. URL <https://zenodo.org/records/17762666>. [p315]

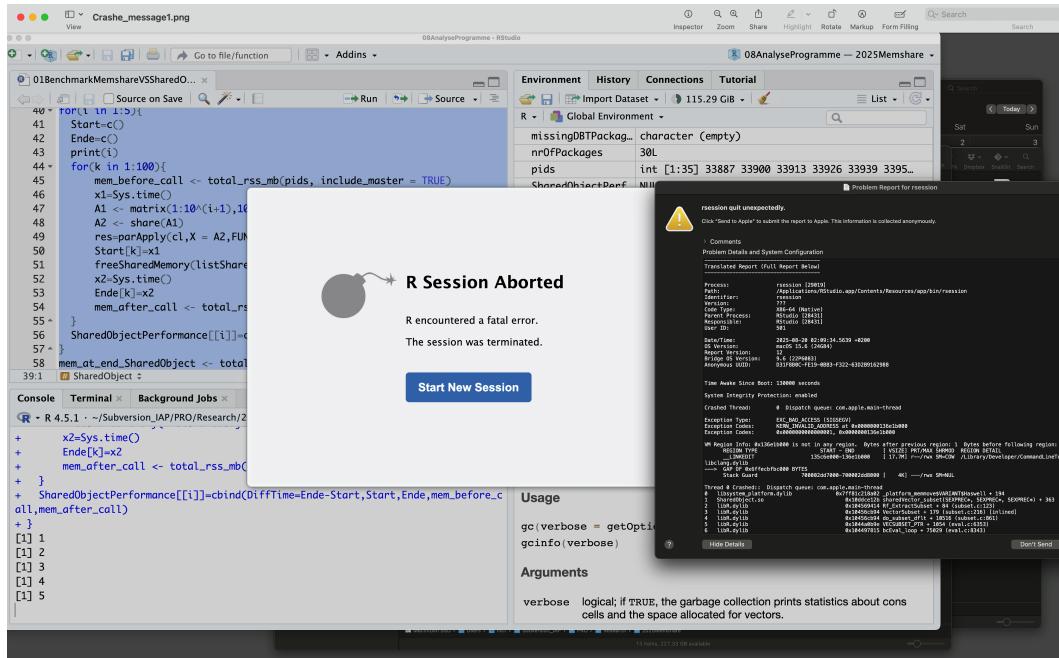


Figure 6: First Screenshot of ShareObjects Computation.

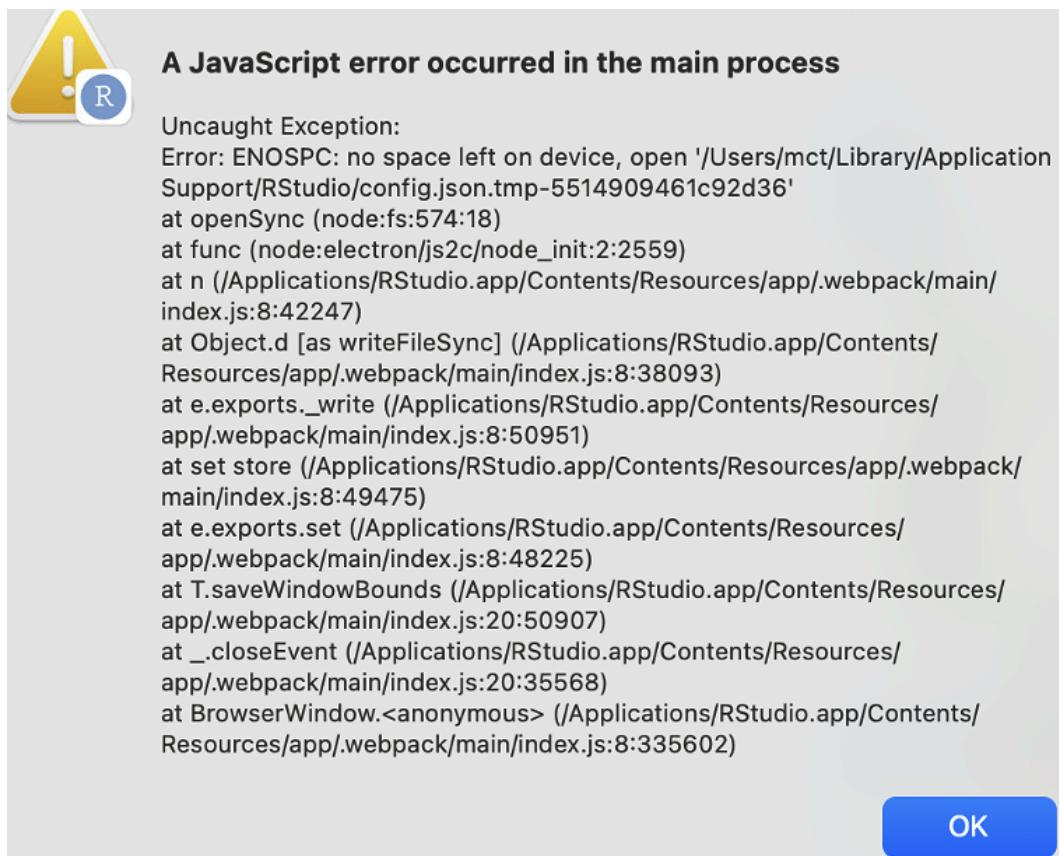


Figure 7: Second Screenshot of ShareObjects Computation.

- M. Thrun and J. Märte. Genexpressions dataset derived from FireBrowse, 2025. URL <https://zenodo.org/records/16937028>. [p311]
- M. C. Thrun, T. Gehlert, and A. Ultsch. Analyzing the fine structure of distributions. *PLOS ONE*, 15(10):e0238835, 2020. doi: 10.1371/journal.pone.0238835. URL <https://doi.org/10.1371/journal.pone.0238835>. [p306]
- A. Ultsch. Pareto density estimation: A density estimation for knowledge discovery. In *Proceedings of the 28th Annual Conference of the German Classification Society (GfKI), Studies in Classification, Data Analysis, and Knowledge Organization*, pages 91–98. Springer, 2005. [p306]
- A. Ultsch. Is log ratio a good value for measuring return in stock investments? In *Advances in Data Analysis, Data Handling and Business Intelligence, Studies in Classification, Data Analysis, and Knowledge Organization*, pages 505–511. Springer, 2008. [p311]
- J. Wang and M. Morgan. *SharedObject: Sharing R objects across multiple R processes without memory duplication*, 2025. URL <https://bioconductor.org/packages/SharedObject>. R package version (Bioconductor Release). [p305]

Michael C. Thrun
University of Marburg IAP-GmbH Intelligent Analytics Projects
Mathematics and Computer Science, D-35032 Marburg
In den Birken 10A, 29352 Adelheidsdorf
<https://www.iap-gmbh.de>
ORCID: 0000-0001-9542-5543
mthrun@informatik.uni-marburg.de

Julian Märte
IAP-GmbH Intelligent Analytics Projects
In den Birken 10A, 29352 Adelheidsdorf
<https://www.iap-gmbh.de>
ORCID: 0000-0001-5451-1023
j.maerte@iap-gmbh.de

Bioconductor Notes, December 2025

by Maria Doyle and Bioconductor Core Developer Team

1 Introduction

Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data. The project has entered its twenty-first year, with funding for core development and infrastructure maintenance secured through 2025 (NIH NHGRI 2U24HG004059). Additional support is provided by NIH NCI, Chan-Zuckerberg Initiative, National Science Foundation, Microsoft, and Amazon. In this news report, we give some updates on core team and project activities.

2 Software

Bioconductor 3.22, released in October 2025, is now available. It is compatible with R 4.5 and consists of 2361 software packages, 435 experiment data packages, 928 up-to-date annotation packages, 29 workflows, and 8 books. Books are built regularly from source, ensuring full reproducibility; an example is the community-developed [Orchestrating Single-Cell Analysis with Bioconductor](#).

3 Core Team and Infrastructure Updates

Bioconductor has introduced new GPU infrastructure, funded by CZI EOSS 6, to support developers building GPU-accelerated packages, including Nvidia GPU build nodes, GPU-aware containers, and a new biocViews term. Maintainers can now opt into GPU software builds and mark packages as GPU-optional or GPU-required. See [the blog post](#) for more details.

NEWS summaries for three contributed packages chosen at random from the 59 new software contributions are:

- **dmGsea**: The R package dmGsea provides efficient gene set enrichment analysis specifically for DNA methylation data. It addresses key biases, including probe dependency and varying probe numbers per gene. The package supports Illumina 450K, EPIC, and mouse methylation arrays. Users can also apply it to other omics data by supplying custom probe-to-gene mapping annotations. dmGsea is flexible, fast, and well-suited for large-scale epigenomic studies.
- **goatea**: Geneset Ordinal Association Test Enrichment Analysis (GOATEA) provides a ‘Shiny’ interface with interactive visualizations and utility functions for performing and exploring automated gene set enrichment analysis using the ‘GOAT’ package. ‘GOATEA’ is designed to support large-scale and user-friendly enrichment workflows across multiple gene lists and comparisons, with flexible plotting and output options. Visualizations pre-enrichment include interactive ‘Volcano’ and ‘UpSet’ (overlap) plots. Visualizations post-enrichment include interactive geneset dotplot, geneset treeplot, gene-effectsize heatmap, gene-geneset heatmap and ‘STRING’ database of protein-protein-interactions network graph. ‘GOAT’ reference: Frank Koopmans (2024) [doi:10.1038/s42003-024-06454-5](https://doi.org/10.1038/s42003-024-06454-5).
- **igblastr**: The igblastr package provides functions to conveniently install and use a local IgBLAST installation from within R. IgBLAST is described at <https://pubmed.ncbi.nlm.nih.gov/23671333/>. Online IgBLAST: <https://www.ncbi.nlm.nih.gov/igblast/>.

See the NEWS section in the [release announcement](#) for a complete account of changes throughout the ecosystem.

4 Community and Impact

4.1 Community Team Updates

Nicholas Cooley has joined the Bioconductor Community Team as Developer Engagement Lead, based at the University of Limerick in Ireland. Working with the Community Manager, his role focuses on supporting package developers, improving onboarding resources, and strengthening connections across the developer community. Nick is funded through [CZI EOSS 6](#).

Laurah Ondari has also joined the team part-time, based at the International Institute of Tropical Agriculture (IITA) in Kenya. She works with the Community Manager on communications, social media, and community engagement, including supporting Africa-focused capacity-building efforts. Laurah's position is also supported by [CZI EOSS 6](#) funding.

4.2 Outreachy Internships

The June–August 2025 [Outreachy](#) program concluded successfully, with interns contributing to Bioconductor and sharing their reflections in a [blog post](#).

4.3 Community Updates

The Bioconductor Seminar Series is a new quarterly online event showcasing recent advances in computational biology and their relevance to Bioconductor methods, workflows, and community practice. Conceived by Bioconductor founder Robert Gentleman and organised by Erica Feick, the series began in December 2025 and brings together expert speakers, moderated discussions, and open Q&A. The first session was on *Deep-learning-based Gene Perturbation Effect Prediction Does Not Yet Outperform Simple Linear Baselines* (*Nature Methods*, 2025) with speakers Constantin Ahlmann-Eltze, Wolfgang Huber, Simon Anders and discussant Davide Risso. It was well attended with engaging discussion. More details can be found on the [Bioconductor website](#).

4.4 Publications and Preprints

In November 2025, Crowell et al. released a [preprint](#) describing their online book *Orchestrating Spatial Transcriptomics Analysis with Bioconductor (OSTA)*, a collaborative effort supported by leaders across the community and built on two decades of Bioconductor's foundational work. The project invites feedback, suggestions, and contributions from Bioconductor users and developers.

5 Conferences and Workshops

5.1 Recaps

- **EuroBioC 2025:** The European Bioconductor Conference (GBCC 2025) was held in September 2025 in Barcelona. A blog post recap of the conference can be found in the [Bioconductor blog](#). Recordings of talks are available in a [YouTube playlist](#).
- **BioCAAsia 2025:** BioCAAsia 2025 was held as part of the ABACBS conference in Adelaide, November 27-28. The focus was on hands-on workshops and saw >100 participants. For more information, visit the [conference website](#).
- **Global Training:** We have had a busy second half of the year of training events, with two in-person courses in Africa, in [Ethiopia](#) and [Benin](#).

5.2 Announcements

- **EuroBioC 2026:** The European Bioconductor conference is taking place in Turku, Finland from June 3-5, with pre-conference activities June 1-2. For more information, visit the [conference website](#).
- **BioC 2026:** The North American Bioconductor conference is taking place in Seattle from August 10-12, with post-conference activities August 13-14. For more information, visit the [conference website](#).

6 Boards and Working Groups Updates

6.1 New Board Members

- The [Community Advisory Board](#) (CAB) welcomes new members Fabricio Almeida-Silva, Tuomas Borman, Laurent Gatto, Zuguang Gu, Eliana Ibrahim, Martha Luka and Izabela Mamede. We extend our gratitude to outgoing members Jasmine Daly, Leo Lahti, Nicole Ortogero, Janani Ravi, Luyi Tian, Hedia Tnani and Jiefei Wang for their service.
- The [Technical Advisory Board](#) (TAB) welcomes new members Robert Castelo and Hugo Gruson and Gabriele Sales. Outgoing members Stephanie Hicks, Davide Risso and Charlotte Soneson are also warmly thanked for their contributions.

7 Using Bioconductor

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 Zulip workspace ([sign up](#)) for extended technical discussion.
- The [F1000Research Bioconductor gateway](#) for peer-reviewed Bioconductor workflows as well as conference contributions.
- The [Bioconductor YouTube](#) channel includes recordings of keynote and talks from recent conferences, in addition to video recordings of training courses.
- Our [package submission](#) repository for open technical review of new packages.

Upcoming and recently completed events are browsable at our [events page](#).

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!

We welcome your feedback on these updates and invite you to connect with us through the [Bioconductor Zulip](#) workspace or by emailing community@bioconductor.org.

*Maria Doyle
University of Limerick
Bioconductor Community Manager*

*Bioconductor Core Developer Team
Dana-Farber Cancer Institute
Roswell Park Comprehensive Cancer Center, City University of New York, Fred Hutchinson Cancer Research Center, Mass General Brigham*

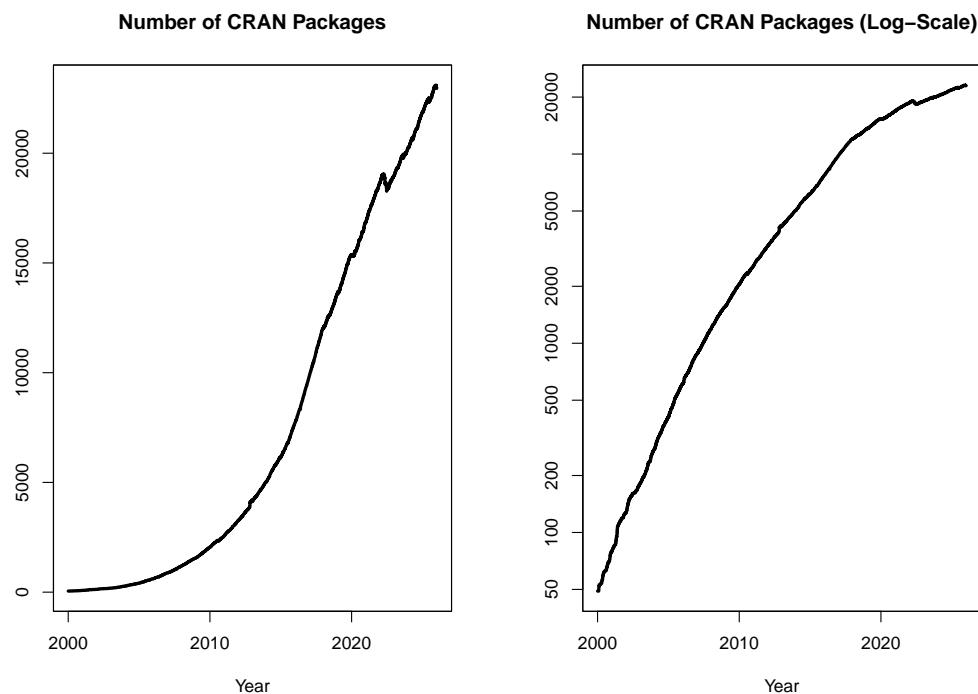
Changes on CRAN

2025-10-01 to 2025-12-31

by Kurt Hornik, Uwe Ligges, and Achim Zeileis

1 CRAN growth

In the past 3 months, 590 new packages were added to the CRAN package repository. 158 packages were unarchived, 594 were archived and 0 had to be removed. The following shows the growth of the number of active packages in the CRAN package repository:



On 2025-12-31, the number of active packages was around 22969.

2 CRAN package submissions

From October 2025 to December 2025 CRAN received 7066 package submissions. For these, 11187 actions took place of which 8337 (75%) were auto processed actions and 2850 (25%) manual actions.

Minus some special cases, a summary of the auto-processed and manually triggered actions follows:

	archive	inspect	newbies	pending	pretest	publish	recheck	waiting
auto	2258	634	1643	146	0	2306	770	308
manual	1072	1	7	7	87	1345	244	83

These include the final decisions for the submissions which were

	archive	publish
auto	2132 (31.2%)	2099 (30.7%)
manual	1066 (15.6%)	1545 (22.6%)

where we only count those as *auto* processed whose publication or rejection happened automatically in all steps.

3 CRAN mirror security

Currently, there are 93 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).

4 CRAN Task View Initiative

Currently, there are 49 task views (see <https://CRAN.R-project.org/web/views/>), with median and mean numbers of CRAN packages covered 110 and 123, respectively. Overall, these task views cover 5003 CRAN packages, which is about 22% of all active CRAN packages.

Kurt Hornik
WU Wirtschaftsuniversität Wien
Austria
[ORCID: 0000-0003-4198-9911](#)
Kurt.Hornik@R-project.org

Uwe Ligges
TU Dortmund
Germany
[ORCID: 0000-0001-5875-6167](#)
Uwe.Ligges@R-project.org

Achim Zeileis
Universität Innsbruck
Austria
[ORCID: 0000-0003-0918-3766](#)
Achim.Zeileis@R-project.org

News from the Forwards Taskforce

by Heather Turner

Abstract 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 2025.

1 Accessibility

Work continued on the [R Consortium funded project](#) to facilitate adding alternative text (*alt text*) to figures in R Journal articles. Di Cook, Jonathan Godfrey and Heather Turner advised Maliny Po in developing two tools:

- A [Shiny app](#) for generating alt text based on a plot and the code used to generate it.
- The [autoAlt](#) package for generating alt text for Quarto or R markdown files.

Both of these are works in progress. Di worked with Jacob Voo at [OceaniaR Hackathon 2025](#) to package up some of the scripts used by the Shiny app into the autoAlt package.

2 Community engagement

Ella Kaye was selected as a 2025 Software Sustainability Institute Fellow, supporting her work with the [rainbowR](#) community for LGBTQ+ folk who code in R. This has enabled rainbowR to become more established, with an expanded [leadership team](#) and [an improved Code of Conduct](#). The membership has grown to over 250 members and the community have had the capacity to expand their activities to include a book club (discussing [Queer Data](#) as the first book) and an [online conference](#) which will be held 25-26 February 2026. The committee has also worked towards securing the long-term sustainability of rainbowR, including adopting a constitution and electing the leadership committee. They will hold their first Annual General Meeting to vote on these developments in January 2026.

Heather Turner and Ella Kaye attended the “Data Science for Girls” open event hosted by [R-Girls](#) at Green Oak Academy, Birmingham, UK. They wrote a [post on the Forwards blog](#) to report back on this event. Mohammed A Mohammed, a co-founder of the R-Girls initiative, has joined Forwards to help maintain the connection with this group.

Ella was also Rotating Curator on the [We Are R-Ladies Bluesky account](#) for a week in December, where she was able to promote Forwards and rainbowR, as well as share tips and resources regarding contributing to base R.

3 R Contribution/on-ramps

Forwards was once again heavily involved in activities of the [R Contribution Working Group](#), aiming to foster a larger and more diverse community of contributors to R. 2025 saw the organization of the first bilingual French/English R Dev Day as a satellite to [RencontresR 2025](#), as well as the first R Dev Days in Oceania. The one in [Australia](#) facilitated collaboration between online participants and people at venues in Melbourne, Perth and Brisbane, while the one in [New Zealand](#) ran in two streams to facilitate collaboration between online participants and people attending in Auckland. These experiments were part of a commitment to run R Contributor events as hybrid in future, for greater inclusion. The R Dev Day at RSECon25 was another significant event, as funding was available from Heather Turner’s EPSRC Research Software Engineering Fellowship (grant number: EP/V052128/1) to provide full travel support for 9 participants from the Global South, enabling them to participate in person.

Heather presented a keynote talk at LatinR 2025 on *Lowering Barriers to Contributing to R* and highlighted the contributions of *R-Ladies at R Dev Days* in a talk to the R-Ladies Melbourne chapter.

Ella Kaye facilitated a second run of the C Study Group - a cohort for Oceania was organised by Nick Tierney and Fonti Kar.

4 Conferences

Dillon Sparks finalised the report on useR! 2024 survey and joined the organising committee for useR! 2025 to support the running of this survey that helps to track demographics of participants over time, as well as inform future events. He has been working with our new members Imani Oluwafumilayo Maliti and Lois Adler-Johnson on promoting the survey and analysing the results. Another new member, Jesica Formoso, is helping to maintain the useR! infoboard that summarises key data from useR! conferences over time.

Kevin O'Brien and gwynn gebeyhu were on the advisory committee for the GhanaR 2025 conference, the second instance of this conference.

5 Teaching

The materials for the Forwards Package Development Workshop were updated and added to a new section of the Forwards website: <https://forwards.github.io/package-dev/>. They are shared under the CC-BY-NC-SA 4.0 license.

The updated materials were used to teach the workshop online to two cohorts in June-July, the first led by Ella Kaye and Pao Corrales, the second led by Joyce Robbins, Emma Rand and Heather Turner. The workshops were promoted in collaboration with rainbowR and R-Ladies Remote, encouraging participation from underrepresented groups. Around 50 people participated in these events and we plan to re-run the workshops in 2026.

6 Social Media/Branding

The Forwards website has a fresh look, thanks to Ella Kaye. Ella also developed a Quarto revealjs extension for Forwards that is used in the updated teaching materials.

7 Membership changes

We were happy to welcome four new members in 2025: Lois Adler-Johnson, Jesica Formoso, Mohammed A Mohammed, and Imani Oluwafumilayo Maliti.

One member, Emma Rand, has stepped down from the taskforce and we thank her for her contributions over several years.

*Heather Turner
University of Warwick
Coventry, United Kingdom
<https://warwick.ac.uk/heatherturner>
ORCID: 0000-0002-1256-3375
heather.turner@r-project.org*

R Foundation News

by *Torsten Hothorn*

1 Donations and members

Membership fees and donations received between 2025-11-06 and 2026-01-05.

1.1 Donations

Andreas Büttner (Germany); Tobias Fellinger (Austria); Shalese Fitzgerald (United States); Roger Koenker (United Kingdom); Bruce Larson (United States); Joseph Luchman (United States); HMS Analytical Software GmbH (Germany); Kem Phillips (United States); Fergus Reig Gracia (Spain); Zane Troyer (United States); Roland Wedekind (France).

1.2 Supporting institutions

oikostat GmbH, Ettiswil (Switzerland).

1.3 Supporting members

Richard Abdill (United States); Justan Baker (United States); Gilberto Camara (Brazil); Michael Chirico (United States); Tom Clarke (United Kingdom); Robin Crockett (United Kingdom); Dan Dediu (Spain); Kevin DeMaio (United States); Ian Dinwoodie (United States); Anna Doizy (Réunion); Fraser Edwards (United Kingdom); Anthony Alan Egerton (Malaysia); Neil Frazer (United States); Sven Garbade (Germany); Brian Gramberg (Netherlands); Spencer Graves (United States); Krushi Gurudu (United States); Joe Harwood (United Kingdom); Kieran Healy (United States); Adam Hill (United States); Sebastian Jeworutzki (Germany); June Kee Kim (South Korea); Vishal Lama (United States); Thierry Lecerf (Switzerland); Thomas Levine (United States); David Luckett (Australia); Mehrad Mahmoudian (Finland); Keon-Woong Moon (South Korea); Yoshinobu Nakahashi (Japan); Dan Orsholits (Switzerland); Sermet Pekin (Türkiye); Elgin Perry (United States); Peter Ruckdeschel (Germany); Choonghyun Ryu (South Korea); Raoul Schorer (Switzerland); Dejan Schuster (Germany); Tobias Strapatsas (Germany); Kai Streicher (Switzerland); Robert Szabo (Sweden); Koki Takayama (Japan); Jan Tarabek (Czechia); Marcus Vollmer (Germany); Petr Waldauf (Czechia); Jaap Walhout (Netherlands); Sandra Ware (Australia); Fredrik Wartenberg (Sweden); Vaidotas Zemlys-Balevičius (Lithuania); Lim Zhong Hao (Singapore); 广宇曾 (China).

Torsten Hothorn

Universität Zürich

Switzerland

ORCID: 0000-0001-8301-0471

Torsten.Hothorn@R-project.org