# The R Journal

Volume 18/3, September 2026

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


## Contents


Editorial . . . . . 4

**Contributed Research Articles**

hdtg: An R Package for High-Dimensional Truncated Normal Simulation . . . . . 5
ODRF: An R Package for Oblique Decision Tree and Its Random Forest . . . . . 15
SelectionBias: An R Package for Bounding Selection Bias in Causal Estimands . . . . 37
psvmSDR: An R Package for a Unified Algorithm for Sufficient Dimension Reduction via Principal Machines . . . . . 55
DRIP: An R Package for Jump Regression and Image Analysis . . . . . 72
rSRD: An R Package for the Sum of Ranking Differences Statistical Procedure . . . . 97
echos: An R Package for Automatic Time Series Forecasting using Echo State Networks 120
ldmppr: Location Dependent Marked Point Processes in R . . . . . 144
mtarm: Bayesian Analysis of Multivariate Threshold Autoregressive Models in R . . 167
goSorensen: A Bioconductor R-Package to Detect Equivalence Between Gene Lists . . 201
MSTest: An R Package for Testing Markov Switching Models . . . . . 219
admix: An R Package for Estimation, Test and Clustering in Admixture Models . . . 242
VARshrink: An R Package for Shrinkage Estimation of High-Dimensional Vector Autoregressive Models . . . . . 269
sugarglider: Create Glyph-Maps of Spatiotemporal Data . . . . . 286
iAR: An R Package for Autoregressive Modeling Irregularly Observed Time Series . . 305
R Package dbrobust: Robust Distance-Based Visualization and Analysis of Mixed-Type Data . . . . . 327
xplainfi: Feature Importance and Statistical Inference for Machine Learning in R . . . 360
LBBNN: An R Package for Sparse and Explainable Bayesian Deep Learning with Latent Binary Bayesian Neural Networks . . . . . 381
demofit: A Unified Framework for Mortality Modelling Integrating Parametric Laws and Stochastic Models . . . . . 402
metafrontier: Unified Metafrontier Analysis for Efficiency and Productivity in R . . . 418

**News and Notes**

Bioconductor Notes, September 2026 . . . . . 441

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:**
Emi Tanaka, Australian National University, Australia

**Executive editors:**
Emily Zabor, Cleveland Clinic, United States
Vincent Arel-Bundock, University of Montreal, Canada
Rob J Hyndman, Monash University, Australia

**Technical editors:**
Abhishek Ulayil, , India

**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 Emi Tanaka*

## In this issue

On behalf of the Editorial Board, I am pleased to present Volume 18, Issue 3 of the R Journal. This issue features twenty research articles spanning a wide range of developments in statistical methodology, machine learning, time series, spatial analysis, causal inference, visualisation, and scientific computing.

Several articles introduce new tools for computationally challenging statistical problems. **hdtg** provides methods for simulating from high-dimensional truncated multivariate normal distributions, while **ODRF** develops an implementation of oblique decision trees and random forests. **psvmSDR** provides tools for sufficient dimension reduction using principal machines, and **DRIP** brings jump regression methods to image analysis. **SelectionBias** implements a range of methods for bounding selection bias in causal estimands. **LBBNN** provides a scalable approach to sparse and interpretable Bayesian neural networks, making use of variational inference and GPU acceleration. The issue also includes **rSRD** supporting the sum of ranking differences procedure, and **xplainfi** brings together a broad range of feature importance methods with tools for statistical inference.

Time series modelling is also well represented in this issue. **mtarm** provides Bayesian estimation and forecasting for multivariate threshold autoregressive models, while **echos** introduces automated time series forecasting using echo state networks. **MSTest** provides a collection of procedures for testing the number of regimes in Markov-switching models, **VARshrink** brings together shrinkage methods for high-dimensional vector autoregressive models, and **iAR** provides tools for modelling irregularly observed time series. **demofit** brings parametric and stochastic approaches to mortality modelling and forecasting into a unified framework.

There are also several contributions in spatial, spatiotemporal, and multivariate analysis. **ldmppr** provides tools for modelling marked point processes where the marks may depend on location, while **sugarglider** extends glyph-map visualisation for exploring spatiotemporal data. **goSorensen** provides methods for comparing gene lists using Gene Ontology information, **admix** supports estimation, testing and clustering for admixture models, and **dbrobust** provides tools for robust distance-based analysis and visualisation of mixed-type data.

Finally, **metafrontier** provides a comprehensive set of tools for metafrontier analysis of efficiency and productivity.

All packages discussed are available on CRAN or Bioconductor. Supplementary material with fully reproducible code is available for download from the R Journal website.

*Emi Tanaka*
*Australian National University*

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

# hdtg: An R Package for High-Dimensional Truncated Normal Simulation

*by Zhenyu Zhang, Andrew Chin, Akihiko Nishimura and Marc A. Suchard*

**Abstract** Simulating from the multivariate truncated normal distribution (MTN) is required in various statistical applications yet remains challenging in high dimensions. Currently available algorithms and their implementations often fail when the number of parameters exceeds a few hundred. To provide a general computational tool to efficiently sample from high-dimensional MTNs, we introduce the **hdtg** package that implements two state-of-the-art simulation algorithms: harmonic Hamiltonian Monte Carlo (Harmonic-HMC) and zigzag Hamiltonian Monte Carlo (Zigzag-HMC). Both algorithms exploit analytical solutions of the Hamiltonian dynamics under a quadratic potential energy with hard boundary constraints, leading to rejection-free methods. We compare their efficiencies against another state-of-the-art algorithm for MTN simulation, the minimax tilting accept-reject sampler (MET). The run-time of these three approaches heavily depends on the underlying multivariate normal correlation structure. Zigzag-HMC and Harmonic-HMC both achieve 100 effective samples within 3,600 seconds across all tests with dimension ranging from 100 to 1,600, while MET has difficulty in several high-dimensional examples. We provide guidance on how to choose an appropriate method for a given situation and illustrate the usage of **hdtg**.

## 1 Introduction

Sampling from a multivariate truncated normal (MTN) distribution is a recurring problem in many statistical applications. The MTN distribution of a $d$-dimensional random vector $\boldsymbol{x} \in \mathbb{R}^d$ has the form

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma) \text{ with } (\mathbf{F}\boldsymbol{x} + \mathbf{g})_i \geq 0 \text{ for } i = 1, \ldots, m, \tag{1}$$

where $\boldsymbol{\mu}$ and $\Sigma$ are the mean vector and covariance matrix, the $m \times d$ matrix $\mathbf{F}$ and $m$-dimensional vector $\mathbf{g}$ specify the $m$ linear constraints and $(\cdot)_i$ denotes the $i$th vector element (Pakman and Paninski, 2014). MTNs arise in various contexts including probit and tobit models (Albert and Chib, 1993; Tobin, 1958), latent Gaussian models (Bolin and Lindgren, 2015), copula regression (Pitt et al., 2006), spatial models (Tsionas and Michaelides, 2016; Baltagi et al., 2018; Zareifard and Khaledi, 2021), Bayesian metabolic flux analysis (Heinonen et al., 2019), and many others. When the dimension $d$ is small, a standard rejection sampler (Geweke, 1991; Kotecha and Djuric, 1999) works well and is a common choice. However, simulation from a larger MTN with hundreds or thousands of correlated dimensions remains a computational challenge. Work towards this goal includes harmonic Hamiltonian Monte Carlo (Pakman and Paninski, 2014, Harmonic-HMC), rejection sampling based on minimax (saddle point) exponential tilting (Botev, 2017, MET), and the most recent Zigzag Hamiltonian Monte Carlo (Nishimura et al., 2020, 2025, Zigzag-HMC) methods.

The MET method provides independent samples but can suffer from low acceptance rates and becomes impractical with $d > 100$, except in special cases like when the MTN has a strongly positive correlation structure (Botev, 2017). Both Harmonic-HMC and Zigzag-HMC are Markov chain Monte Carlo (MCMC) approaches that generate correlated samples, but can nonetheless be highly efficient and scale to thousands or more dimensions. To our knowledge, however, there is no general-purpose implementation of either method; the **tmg** package provided by Pakman and Paninski (2014) is no longer available on CRAN, and Zhang et al. (2023) implement Zigzag-HMC for their phylogenetics applications in the specialized software BEAST (Suchard et al., 2018). Therefore, we have developed the **hdtg** R package for efficient MTN simulation. The package implements tuning-free Zigzag-HMC and Harmonic-HMC. We provide performance comparisons among these two methods and a MET implementation from the **TruncatedNormal** package (Botev and Belzile, 2021). In most of the test cases with $d > 100$, Harmonic-HMC and Zigzag-HMC outperform MET. We then conclude with some empirical guidance on which method to use in different scenarios.

## 2 Algorithm

We begin by briefly introducing Harmonic-HMC and Zigzag-HMC, both of which are variants of HMC, an effective proposal generation mechanism exploiting the properties of Hamiltonian dynamics (Neal, 2011). Harmonic-HMC and Zigzag-HMC follow the same general framework. To sample $\boldsymbol{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$ from the target distribution $\pi_X(\boldsymbol{x})$, the HMC variants introduce an auxiliary *momentum* variable $\boldsymbol{p}$ and define an augmented target distribution $\pi(\boldsymbol{x}, \boldsymbol{p}) = \pi_X(\boldsymbol{x})\, \pi_P(\boldsymbol{p})$ in the joint

space. They then propose the next state by first re-sampling the momentum variable from its marginal and then simulating the solution of Hamiltonian dynamics governed by the differential equations

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \nabla K(\boldsymbol{p}), \quad \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = -\nabla U(\boldsymbol{x}), \tag{2}$$

where $U(\boldsymbol{x}) = -\log \pi_X(\boldsymbol{x})$ and $K(\boldsymbol{p}) = -\log \pi_P(\boldsymbol{p})$ are referred to as *potential* and *kinetic* energies. The dynamics are simulated for a pre-set time duration $T$ and the end state constitutes a valid Metropolis proposal to be accepted or rejected according to the standard formula (Metropolis et al., 1953; Hastings, 1970).

The most common versions of HMC use the momentum distribution $\pi_P(\boldsymbol{p}) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and rely on the leapfrog integrator to numerically solve (2), as its solutions are analytically intractable in general settings. Harmonic-HMC takes advantage of the fact that (2) admits analytical solutions when the target $\pi_X(\boldsymbol{x})$ is an MTN. The solution follows independent harmonic oscillations along the principal components of the covariance/precision matrix (Pakman and Paninski, 2014); we thus refer to the algorithm as Harmonic-HMC. Truncation boundaries are handled via elastic "bounces" against hard potential energy walls (Neal, 2011). Algorithm 1 provides pseudo-code for Harmonic-HMC following Pakman and Paninski (2014).

Zigzag-HMC differs from the common HMC versions in that it deploys a Laplace momentum (Nishimura et al., 2020, 2025)

$$\pi_P(\boldsymbol{p}) \propto \prod_{i=1}^{d} \exp(-|p_i|). \tag{3}$$

The Hamiltonian dynamics then become

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \mathrm{sign}(\boldsymbol{p}), \quad \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = -\nabla U(\boldsymbol{x}), \tag{4}$$

where $\mathrm{sign}(p_i)$ returns 1 if $p_i$ is positive and -1 otherwise. Because the velocity $\mathrm{d}\boldsymbol{x}/\mathrm{d}t \in \{\pm 1\}^d$ remains constant until one of the $p_i$ flips its sign, the trajectory of these Hamiltonian dynamics has a zigzag pattern, hence the name Zigzag-HMC. The zigzag dynamics also admit analytical solutions under an MTN target and can handle the truncation in the same manner as in Harmonic-HMC. Algorithm 2 provides pseudo-code for Zigzag-HMC. Our current Zigzag-HMC implementation is limited to the coordinate-wise bounds $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$. We refer interested readers to Nishimura et al. (2025) and Zhang et al. (2023) for more details.

The simulation duration $T$, i.e. how long Hamiltonian dynamics is simulated for each proposal generation, critically affects efficiencies of both Harmonic and Zigzag-HMC. For Harmonic-HMC, Pakman and Paninski (2014) suggest setting $T = \pi/2$; when using this fixed $T$, however, we observe inefficiencies in some of our examples in Section 2.4 due to Hamiltonian dynamics' periodic behaviors (Neal, 2011). We therefore randomize the duration $T$, as recommended by Neal (2011), and draw it from a uniform distribution on $[\pi/8, \pi/2]$. For Zigzag-HMC, we adopt the choice $T = \sqrt{2}\lambda_{\min}^{-1/2}$ based on the heuristics of Nishimura et al. (2025), where $\lambda_{\min}$ is the minimal eigenvalue of the precision matrix $\Omega = \Sigma^{-1}$. We compute $\lambda_{\min}$ using the Lanczos algorithm (Demmel, 1997) as in the **mgcv** package (Wood, 2017). We further implement the no-U-turn sampler (NUTS) of Hoffman and Gelman (2014) to automatically determine the integration time. With NUTS, we only need to pick a base integration time $\Delta T$ which we set to $0.1\lambda_{\min}^{-1/2}$ as recommended by Nishimura et al. (2025).

**Algorithm 1** Harmonic-HMC for MTNs

**Require:** $\boldsymbol{x}, T, \boldsymbol{\mu}, \Omega, \mathbf{F}, \mathbf{g}$,

**(1) Whitening:**

$R \leftarrow \text{Cholesky}(\Omega)$

$\boldsymbol{z} \leftarrow R(\boldsymbol{x} - \boldsymbol{\mu})$

**(2) Momentum draw:**

$\boldsymbol{p} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

**(3) Dynamics:**

$\tau \leftarrow T$

**while** $\tau > 0$ **do**

  % Find next boundary event

  $(t_b, j_b) \leftarrow \text{NEXTBOUNCE}(\boldsymbol{z}, \boldsymbol{p}, \mathbf{F}, \mathbf{g})$

  % The event time is bounded by $\tau$

  $t \leftarrow \min\{t_b, \tau\}$

  % Advance time $t$:

  $\boldsymbol{z} \leftarrow \boldsymbol{p}\sin(t) + \boldsymbol{z}\cos(t)$

  $\boldsymbol{p} \leftarrow \boldsymbol{p}\cos(t) - \boldsymbol{z}\sin(t)$

  % Update $\boldsymbol{p}$ for a boundary event

  % $\mathbf{F}_{j_b}$ is the $j_b$th row vector of $\mathbf{F}$

  **if** $t < \tau$ **then**

    $\alpha \leftarrow \dfrac{\mathbf{F}_{j_b} \cdot \boldsymbol{p}}{\mathbf{F}_{j_b} \cdot \mathbf{F}_{j_b}}$

    $\boldsymbol{p} \leftarrow \boldsymbol{p} - 2\alpha \mathbf{F}_{j_b}^{\top}$

  **end if**

  $\tau \leftarrow \tau - t$

**end while**

**(4) Unwhiten:**

$\boldsymbol{x} \leftarrow R^{-1}\boldsymbol{z} + \boldsymbol{\mu}$

**return** $\boldsymbol{x}$

**Function** NEXTBOUNCE($\boldsymbol{z}, \boldsymbol{p}, \mathbf{F}, \mathbf{g}$):

$t_{\text{b}} \leftarrow \infty$, $j_{\text{b}} \leftarrow -1$

**for** $j = 1$ **to** $m$ **do**

  % $(\mathbf{F}\boldsymbol{p})_j$ is the $j$th element of $\mathbf{F}\boldsymbol{p}$

  $f_a \leftarrow (\mathbf{F}\boldsymbol{p})_j$, $f_b \leftarrow (\mathbf{F}\boldsymbol{z})_j$

  $u_j \leftarrow \sqrt{f_a^2 + f_b^2}$

  **if** $u_j > |g_j|$ **then**

    $\varphi_j \leftarrow -\text{atan2}(f_a, f_b)$

    $A \leftarrow \arccos(-g_j/u_j)$

    $t_1 \leftarrow -\varphi_j + A$

    $t_2 \leftarrow -\varphi_j - A + 2\pi$

    **if** $t_1 < 0$ **then**

      $t_{\text{c}} \leftarrow t_2$

    **else**

      $t_{\text{c}} \leftarrow \min(t_1, t_2)$

    **end if**

    **if** $t_{\text{c}} < t_{\text{b}}$ **then**

      $t_{\text{b}} \leftarrow t_{\text{c}}$, $j_{\text{b}} \leftarrow j$

    **end if**

  **end if**

**end for**

**return** $(t_{\text{b}}, j_{\text{b}})$

**Algorithm 2** Zigzag-HMC for MTNs

**Require:** $\boldsymbol{x}$, T, $\boldsymbol{\mu}$, $\Omega$

**(1) Momentum draw:**

$p_i \sim \text{Laplace}(u = 0, b = 1)$

$\boldsymbol{v} \leftarrow \text{sign}(\boldsymbol{p})$

**(2) Pre-compute:**

$\boldsymbol{\varphi}_{\boldsymbol{x}} \leftarrow \Omega(\boldsymbol{x} - \boldsymbol{\mu})$

$\boldsymbol{\varphi}_{\boldsymbol{v}} \leftarrow \Omega\boldsymbol{v}$

**(3) Dynamics:**

$\tau \leftarrow T$

**while** $\tau > 0$ **do**

  % Find gradient event time $t_g$

  $\boldsymbol{a} \leftarrow \boldsymbol{\varphi}_{\boldsymbol{v}}/2$, $\boldsymbol{b} \leftarrow \boldsymbol{\varphi}_{\boldsymbol{x}}$, $\boldsymbol{c} \leftarrow -\boldsymbol{p}$

  $t_g \leftarrow \min_i\{\text{minPositiveRoot}(a_i, b_i, c_i)\}^*$

  %Find boundary event time $t_b$

  **for** $i = 1$ **to** $d$ **do**

    **if** $v_i > 0$ **then**

      $t_i \leftarrow U_i - x_i$

    **else**

      $t_i \leftarrow x_i - L_i$

    **end if**

  **end for**

  $t_b \leftarrow \min_i t_i$

  %The actual event happens at time $t$

  $t \leftarrow \min\{t_g, t_b, \tau\}$

  % Advance time $t$

  $\boldsymbol{x} \leftarrow \boldsymbol{x} + t\boldsymbol{v}$

  $\boldsymbol{p} \leftarrow \boldsymbol{p} - t\boldsymbol{\varphi}_{\boldsymbol{x}} - t^2\boldsymbol{\varphi}_{\boldsymbol{v}}/2$

  $\boldsymbol{\varphi}_{\boldsymbol{x}} \leftarrow \boldsymbol{\varphi}_{\boldsymbol{x}} + t\boldsymbol{\varphi}_{\boldsymbol{v}}$

  % Update momentum

  **if** a gradient event happens at $i_g$ **then**

    $v_{i_g} \leftarrow -v_{i_g}$

  **else if** a boundary event happens at $i_b$ **then**

    $v_{i_{\text{b}}} \leftarrow -v_{i_{\text{b}}}$, $p_{i_{\text{b}}} \leftarrow -p_{i_{\text{b}}}$

  **end if**

  $\boldsymbol{\varphi}_{\boldsymbol{v}} \leftarrow \boldsymbol{\varphi}_{\boldsymbol{v}} + 2v_i\Omega\boldsymbol{e}_i$

  % $\boldsymbol{e}_i$ has 1 in position $i$ and 0 elsewhere

  $\tau \leftarrow \tau - t$

**end while**

**return** $\boldsymbol{x}$

*minPositiveRoot$(a_i, b_i, c_i)$ returns the minimal positive root of the equation $a_i t^2 + b_i t + c_i = 0$, or returns $+\infty$ if no positive root exists.

The algorithmic divergence between the two methods stems from their momentum distributions. Harmonic-HMC employs Gaussian momentum, generating harmonic oscillatory dynamics via trigonometric functions. In contrast, Zigzag-HMC uses Laplace momentum, producing piecewise-linear trajectories with constant velocity updates. This distinction leads to differences in event detection: Harmonic-HMC computes boundary bounce times using trigonometric relationships, while Zigzag-HMC solves quadratic equations for gradient events and checks coordinate distances for boundary events (Algorithm 1 and 2 ). These algorithmic differences lead to varying opportunities for implementation optimization, as summarized in Table 1. The piecewise-linear dynamics of Zigzag-HMC, consisting primarily of vector additions and multiplications, map well to single instruction multiple data (SIMD) instructions, which can process multiple values simultaneously within a single CPU core. In contrast, Harmonic-HMC's trigonometric operations (sin, cos, atan2, arccos) are inherently less SIMD-friendly due to their transcendental nature. While both samplers are implemented in optimized C++ with Eigen for linear algebra, Zigzag-HMC's algorithmic structure is inherently more amenable to the low-level hardware optimizations (SIMD and aligned memory) that yield performance gains on modern processors. When examining the computational performance benchmarks in Section 2.4, it is important to recognize that the observed speed differences stem from both algorithmic efficiencies and the hardware-specific optimizations detailed here.

**Table 1:** Comparison of implementation optimizations between Harmonic-HMC and Zigzag-HMC

| Optimization | Harmonic-HMC | Zigzag-HMC |
|---|---|---|
| **Language** | C++ via Rcpp | C++ via Rcpp |
| **Linear algebra** | Eigen library | Eigen library |
| **RNG**[1] | `std::mt19937` | `std::mt19937` |
| **SIMD**[2] | No | Yes (SSE/AVX) |
| **Memory optimization** | Standard Eigen | Aligned allocation |

[1]RNG: random number generator. [2]SIMD: single instruction multiple data.

In addition to Harmonic-HMC and Zigzag-HMC, the **hdtg** package also implements the Markovian-zigzag sampler (Bierkens et al., 2019) for MTNs. The Markovian-zigzag belongs to another emerging class of MCMC algorithms that are based on piecewise deterministic Markov processes (PDMP) (Fearnhead et al., 2018). As established by Nishimura et al. (2025), Markovian-zigzag is a close cousin of the Hamiltonian-based Zigzag-HMC, with the key difference being the amount of retained momentum information during sampling. Markovian-zigzag exhibited lower sampling efficiency compared to Harmonic-HMC and Zigzag-HMC, particularly on targets with correlated parameters (Nishimura et al., 2025). Therefore, we do not include Markovian-zigzag in our performance comparisons of Section 2.4. The Markovian-zigzag function is nevertheless available in the package for users who wish to experiment with it.

## 3 Using hdtg

The **hdtg** package allows users to draw MCMC samples from an MTN. As an example, one may use the following code to generate 1,000 samples from a 10-dimensional MTN with zero mean and an identity covariance matrix truncated to the positive orthant:

```
# set the random seed
set.seed(1)
# draw MTN samples using Zigzag-HMC
samplesZHMC <- zigzagHMC(nSample = 1000, mean = rep(0, 10), prec = diag(10),
                         init = rep(0.1, 10), lowerBounds = rep(0, 10),
                         upperBounds = rep(Inf, 10))
# draw MTN samples using Harmonic-HMC
samplesHHMC <- harmonicHMC(nSample = 1000, mean = rep(0, 10), choleskyFactor = diag(10),
                           precFlg = TRUE, init = rep(0.1, 10),
                           constrainDirec = diag(10), constrainBound = rep(0, 10))
```

The arguments are:

- `nSample`: number of samples.
- `mean`: a $d$-dimensional mean vector.
- `prec`: the precision matrix.

- `init`: a vector of the initial value that must satisfy all constraints.
- `lowerBounds`: a $d$-dimensional vector specifying the lower bounds.
- `upperBounds`: a $d$-dimensional vector specifying the upper bounds.
- `choleskyFactor`: upper triangular matrix $\mathbf{U}$ from Cholesky decomposition of precision or covariance matrix into $\mathbf{U}^T\mathbf{U}$.
- `precFlg`: whether `choleskyFactor` is from precision (`TRUE`) or covariance matrix (`FALSE`).
- `constrainDirec`: the $\mathbf{F}$ matrix.
- `constrainBound`: the $\mathbf{g}$ vector.

In addition to applications with fixed $\boldsymbol{\mu}$ and $\Omega$, hierarchical modeling may require sampling these parameters from their respective conditional distributions, such as the phylogenetics example (Zhang et al., 2023) where their posterior distributions are of scientific interest. In such "random $\boldsymbol{\mu}$ or $\Omega$" settings, one may call `zigzagHMC` or `harmonicHMC` inside an MCMC loop, with $\boldsymbol{\mu}$ and $\Omega$ updated at each iteration. For Zigzag-HMC, a more efficient approach is to reuse the existing C++ engine object which already stores the truncation boundaries and SIMD configuration and simply pass the updated $\boldsymbol{\mu}$ and $\Omega$ to the sampler. The example below illustrates the 10-dimensional MTN case with a random mean and precision. Note that the stationary distribution of this toy example is the joint distribution of both the MTN parameters (`m`, `prec`) and the truncated MTN variable itself. While the conditional distributions are standard (normal for `m`, Wishart for `prec`, MTN for the variable), their joint distribution is not a standard named distribution and does not admit a closed-form expression.

```
set.seed(1)
n <- 1000
d <- 10
samples <- array(0, c(n, d))
# initialize MTN mean and precision
m <- rnorm(d, 0, 1)
prec <- rWishart(n = 1, df = d, Sigma = diag(d))[,,1]

# call createEngine once
engine <- createEngine(dimension = d, lowerBounds = rep(0, d),
upperBounds = rep(Inf, d), seed = 1, mean = m, precision = prec)

HZZtime <- sqrt(2) / sqrt(min(mgcv::slanczos(A = prec, k = 1,
                          kl = 1)[['values']]))

currentSample <- rep(0.1, d)
for (i in 1:n) {
   m <- rnorm(d, 0, 1)
   prec <- rWishart(n = 1, df = d, Sigma = diag(d))[,,1]
   setMean(engine = engine, mean = m)
   setPrecision(engine = engine, precision = prec)
   currentSample <- getZigzagSample(position = currentSample, nutsFlg = F,
                                    engine = engine, stepSize = HZZtime)
   samples[i, ] <- currentSample
}
```

## 4 Efficiency comparison and method choice

To assess the performance of Harmonic-HMC, Zigzag-HMC and MET, we compare them on MTNs with a variety of correlation structures. The three examples are: 1) MTNs with its covariance matrix $\Sigma$ drawn from the uniform LKJ distribution (Lewandowski et al., 2009) as implemented in the `rlkjcorr` function from package **trialr** (Brock, 2020); 2) MTNs with a compound symmetric covariance matrix such that $\Sigma_{i,i} = 1$ and $\Sigma_{i,j} = 0.9$ for $i \neq j$; and 3) a real-world MTN that arises as a posterior conditional distribution in a statistical phylogenetics model of HIV evolution (Zhang et al., 2021, 2023). For simplicity, we assume the truncation $x_i > 0$ for $i = 1, \ldots, d$ in the first two examples. For the HIV example, the truncation is determined by the signs of observed binary biological features.

We now specify our comparison criteria and the rationale behind them. A more efficient MCMC algorithm takes shorter time to achieve a certain effective sample size (ESS). For all three samplers considered, we compare their run-time to obtain the first 1, 100, or 1000 effectively independent samples ($t_1, t_{100}, t_{1000}$). We include all three metrics because $t_{100}$ and $t_{1000}$ reflect a practical run-time

for simulation from a fixed MTN and $t_1$ better captures the pre-processing overhead that remains relevant in cases where $\Sigma$ is random. Recall that the main pre-processing costs are the Cholesky decomposition of $\Sigma$ or $\Omega$ (Harmonic-HMC), calculating the minimal precision matrix eigenvalue $\lambda_{\min}$ (Zigzag-HMC), and solving the minimax optimization problem (MET). Therefore we have

$$\begin{aligned} t_1 &= t_0 + c, \\ t_{100} &= t_0 + 100c, \\ \text{and } t_{1000} &= t_0 + 1000c, \end{aligned} \tag{5}$$

where $t_0$ and $c$ are the pre-processing time required for each $\Sigma$ update and the average run-time per one effective sample. For simulation from a fixed MTN, $t_0$ is a one-time cost and so $t_{100}$ or $t_{1000}$ serve as better efficiency criteria. When $\Sigma$ is random (e.g. the second example in Section 2.3), if $\Sigma$ changes its value $k$ times, the total run-time to obtain one effective sample for each $\Sigma$ is $kt_1$ and so the $t_1$ criterion would be more informative.

For Harmonic-HMC and Zigzag-HMC, we estimate the ESS using the **coda** package (Plummer et al., 2006) and define $n_1$ as the average number of MCMC iterations required for one effectively independent sample. We approximate $n_1$ by $L/\text{ESS}_{\min}$, where $\text{ESS}_{\min}$ is the minimal ESS across all dimensions and $L$ is the chain length. We fix $n_1 = 1$ for MET as it generates independent samples. Therefore $c$ in Equation (5) equals the average time to complete $n_1$ iterations after pre-processing. Table 2 reports our efficiency comparison in terms of $t_1$, $t_{100}$, and $t_{1000}$. We run each test on an Apple M1 Pro equipped machine with 16GB of memory. Code and data to reproduce the results are available on Zenodo at https://doi.org/10.5281/zenodo.18618209.

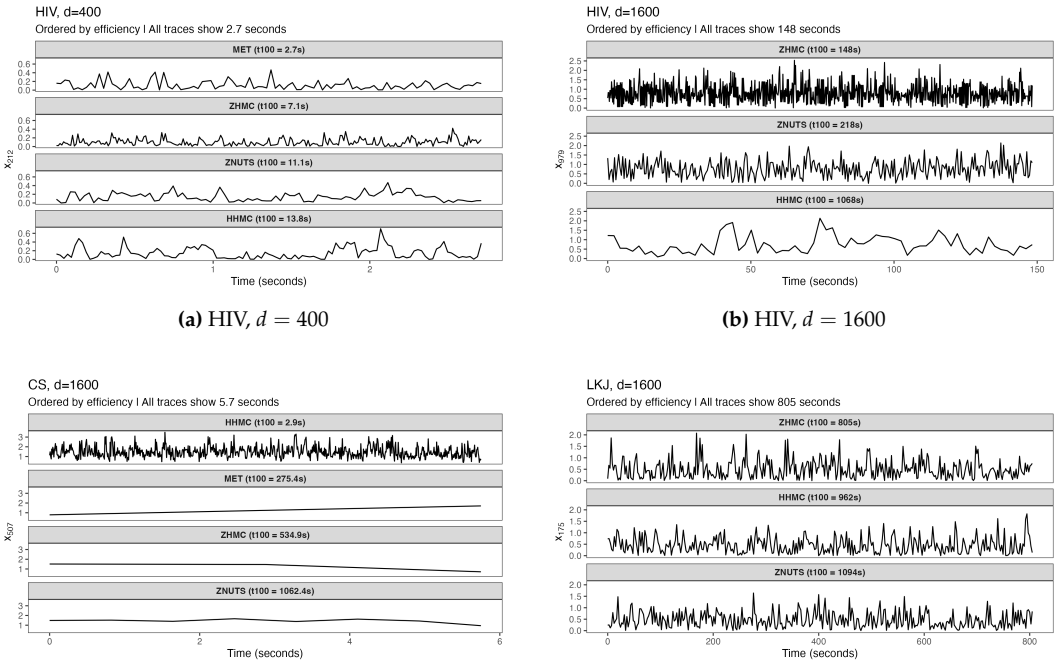
**Table 2:** Efficiency comparison of Harmonic-HMC, Zigzag-HMC, Zigzag-HMC with NUTS (Zigzag-NUTS), and MET sampling approaches across three example correlation structures. We report $t_1$, $t_{100}$, and $t_{1000}$ (in seconds) — the run-times to obtain 1, 100, and 1000 effective samples, respectively. LKJ and HIV cases where MET takes more than two hours to generate 100 samples are omitted. We benchmark each test for three replications using **rbenchmark** (Kusnierczyk, 2012) and report the average run-time. Bold numbers are column minimums in each test.

| | | $d = 100$ | | | 400 | | | 1600 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $t_1$ | $t_{100}$ | $t_{1000}$ | $t_1$ | $t_{100}$ | $t_{1000}$ | $t_1$ | $t_{100}$ | $t_{1000}$ |
| LKJ | Harmonic-HMC | **0.004** | **0.21** | **2.1** | **0.11** | 11 | 107 | **8.0** | 962 | 9680 |
| | Zigzag-HMC | 0.017 | 0.66 | 6.6 | 0.27 | **9.3** | **92** | 9.9 | **805** | **8015** |
| | Zigzag-NUTS | 0.014 | 0.60 | 6.1 | 0.29 | 10 | 102 | 13 | 1094 | 10891 |
| | MET | 1.9 | 19 | 101 | – | – | – | – | – | – |
| $CS_{0.9}$ | Harmonic-HMC | $<$ **0.001** | **0.014** | **0.15** | **0.007** | **0.086** | **0.83** | **0.42** | **2.9** | **25** |
| | Zigzag-HMC | 0.006 | 0.39 | 3.9 | 0.21 | 18 | 180 | 6.3 | 535 | 5317 |
| | Zigzag-NUTS | 0.013 | 1.1 | 12 | 1.0 | 102 | 1014 | 13 | 1062 | 10610 |
| | MET | 0.087 | 0.12 | 0.33 | 4.4 | 4.9 | 7.2 | 272 | 275 | 313 |
| HIV | Harmonic-HMC | **0.003** | 0.32 | 3.4 | 0.17 | 14 | 138 | 10.7 | 1068 | 10968 |
| | Zigzag-HMC | 0.009 | 0.35 | 3.4 | **0.14** | 7.1 | 71 | **2.8** | **148** | **1480** |
| | Zigzag-NUTS | 0.013 | 0.66 | 6.5 | 0.17 | 11 | 109 | 4.3 | 218 | 2261 |
| | MET | 0.040 | **0.056** | **0.19** | 2.3 | **2.7** | **5.1** | – | – | – |

Dashes (–) indicate the method required $>$ 2 hours for 100 samples.

The efficiency of all three methods strongly depends on the correlation structure. MET fails to generate 100 effectively independent samples within two hours in a few higher dimensional tests, while Harmonic-HMC and Zigzag-HMC/NUTS enjoy a $t_{100} < 3600$ seconds across all tests. In the LKJ example, Zigzag-HMC/NUTS achieves comparable performance to Harmonic-HMC when $d$ reaches 400. For high-dimensional LKJ and HIV ($d = 1600$) tests, Zigzag-HMC outperforms competing methods. Zigzag-NUTS is no more efficient than Zigzag-HMC across the tested examples. On the other hand, when $\Sigma$ is compound symmetric with a high correlation of 0.9, Harmonic-HMC consistently outperforms the other methods. For MET, since solving the initial minimax optimization dominates its run-time, the cost scales sub-linearly with sample count ($t_{100} < 100t_1$ and $t_{1000} < 1000t_1$). This makes MET the preferred method when many effective samples are desired (as in HIV $d = 100, 400$ examples).

Figure 1 shows traceplots for four target MTNs from Table 2. Different methods excel on different targets, highlighting the importance of target-specific algorithm selection. In practice, we recommend running a quick efficiency comparison to decide which method to use. Nevertheless we provide some general guidance on method choice for high-dimensional MTN simulation:




**(a)** HIV, $d = 400$ **(b)** HIV, $d = 1600$

**(c)** CS, $d = 1600$ **(d)** LKJ, $d = 1600$

**Figure 1:** Time-normalized traceplots for four target MTNs using **ggplot2** (Wickham, 2016). The y-axis shows a randomly selected dimension.

- If $d \leq 100$ or the correlation structure is strongly positive, use MET or Harmonic-HMC. Harmonic-HMC may run faster but MET has the advantage of generating independent samples.
- For $d > 1000$, Zigzag-HMC/NUTS is presumably more efficient, though Harmonic-HMC may still outperform under strongly positive correlations.
- It is always worth trying MET which is free of MCMC convergence concerns. Since our simulation only examines a few correlation structures, it is possible that MET can handle other large MTNs.

A final point that needs consideration is that Zigzag-HMC/NUTS requires $\Omega$ and if only $\Sigma$ is available, the method first inverts $\Sigma$. This is a one-time operation and likely negligible cost when $\Sigma$ is constant. The approach does become expensive if $\Sigma$ is random, as the $\mathcal{O}\left(d^3\right)$ inversion is necessary for each value of $\Sigma$. In practice, statistical models may be parameterized in terms of $\Sigma$ (Lachaab et al., 2006; Molstad et al., 2021) or $\Omega$ (Baltagi et al., 2018; Lehnert et al., 2019; Li et al., 2020). Harmonic-HMC carries a similar limitation since it requires a $\mathcal{O}\left(d^3\right)$ Cholesky decomposition of $\Sigma$ or $\Omega$, whichever is provided. Therefore, when $d$ is large and the target MTN has a random correlation structure, one may favor Zigzag-HMC/NUTS over Harmonic-HMC especially if a closed-form $\Omega$ is at hand.

## 5 Conclusion

This article introduces the **hdtg** package oriented for efficient MTN simulation. In most of our high-dimensional tests the implemented Harmonic-HMC and Zigzag-HMC algorithms outperform the current best approach available in the **TruncatedNormal** package. To our best knowledge, **hdtg** is the first tool that can generate samples from an arbitrary MTN with thousands of dimensions. We discuss the usage of functions and provide practical suggestions on method choice. We expect to see future large-scale statistical applications utilizing the efficiency of **hdtg**.

## 6 Acknowledgment

The **hdtg** package itself leverages several R infrastructure packages including **Rcpp** (Eddelbuettel and Balamuta, 2018), **RcppEigen** (Bates and Eddelbuettel, 2013), **RcppParallel**(Allaire et al., 2016), and **mgcv** (Wood, 2017) for efficient computation. It also incorporates external header-only libraries: `sse2neon.h` (DLTcollab and contributors, 2020) for ARM support and `span.h` (Brindle, 2018) for memory-safe array views. Research reported in this publication was supported by the National

Institute of General Medical Sciences and the National Institute of Allergy and Infectious Diseases of the National Institutes of Health under award numbers R35GM160458 (A.N.) and R01AI153044 (M.S.). The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

## References

J. H. Albert and S. Chib. Bayesian analysis of binary and polychotomous response data. *Journal of the American statistical Association*, 88(422):669–679, 1993. [p5]

J. Allaire, R. Francois, K. Ushey, G. Vandenbrouck, M. Geelnard, and H. Badr. Rcppparallel: Parallel programming tools for "rcpp". *R package version*, 4:20, 2016. [p11]

B. H. Baltagi, P. H. Egger, and M. Kesina. Generalized spatial autocorrelation in a panel-probit model with an application to exporting in China. *Empirical Economics*, 55(1):193–211, 2018. [p5, 11]

D. Bates and D. Eddelbuettel. Fast and elegant numerical linear algebra using the RcppEigen package. *Journal of Statistical Software*, 52(5):1–24, 2013. doi: 10.18637/jss.v052.i05. [p11]

J. Bierkens, G. O. Roberts, and P.-A. Zitt. Ergodicity of the zigzag process. *The Annals of Applied Probability*, 29(4):2266–2301, 2019. [p8]

D. Bolin and F. Lindgren. Excursion and contour uncertainty regions for latent Gaussian models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 77(1):85–106, 2015. [p5]

Z. Botev and L. Belzile. *TruncatedNormal: Truncated Multivariate Normal and Student Distributions*, 2021. URL https://CRAN.R-project.org/package=TruncatedNormal. R package version 2.2.2. [p5]

Z. I. Botev. The normal law under linear restrictions: simulation and estimation via minimax tilting. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79(1):125–148, 2017. [p5]

T. Brindle. An implementation of c++20's std::span, 2018. URL https://github.com/tcbrindle/span. Boost Software License 1.0. Implementation of C++20's std::span (P0122R7) providing bounds-safe array views for C++14/17 compilers. [p11]

K. Brock. *trialr: Clinical Trial Designs in 'rstan'*, 2020. URL https://CRAN.R-project.org/package=trialr. R package version 0.1.5. [p9]

J. W. Demmel. *Applied numerical linear algebra*. SIAM, 1997. [p6]

DLTcollab and contributors. sse2neon: Sse to neon intrinsics translator, 2020. URL https://github.com/DLTcollab/sse2neon. Used version: 05-Jan-2021. MIT License. [p11]

D. Eddelbuettel and J. J. Balamuta. Extending R with C++: A Brief Introduction to Rcpp. *The American Statistician*, 72(1):28–36, 2018. doi: 10.1080/00031305.2017.1375990. [p11]

P. Fearnhead, J. Bierkens, M. Pollock, and G. O. Roberts. Piecewise deterministic markov processes for continuous-time Monte Carlo. *Statistical Science*, 33(3):386–412, 2018. [p8]

J. Geweke. Efficient simulation from the multivariate normal and student-t distributions subject to linear constraints and the evaluation of constraint probabilities. In *Computing science and statistics: Proceedings of the 23rd symposium on the interface*, pages 571–578. Citeseer, 1991. [p5]

W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 1970. [p6]

M. Heinonen, M. Osmala, H. Mannerström, J. Wallenius, S. Kaski, J. Rousu, and H. Lähdesmäki. Bayesian metabolic flux analysis reveals intracellular flux couplings. *Bioinformatics*, 35(14):i548–i557, 2019. [p5]

M. D. Hoffman and A. Gelman. The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014. [p6]

J. H. Kotecha and P. M. Djuric. Gibbs sampling approach for generation of truncated multivariate Gaussian random variables. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258)*, volume 3, pages 1757–1760. IEEE, 1999. [p5]

W. Kusnierczyk. *rbenchmark: Benchmarking routine for R*, 2012. URL https://CRAN.R-project.org/package=rbenchmark. R package version 1.0.0. [p10]

M. Lachaab, A. Ansari, K. Jedidi, and A. Trabelsi. Modeling preference evolution in discrete choice models: A Bayesian state-space approach. *Quantitative Marketing and Economics*, 4(1):57–81, 2006. [p11]

J. Lehnert, C. Kolbitsch, G. Wübbeler, A. Chiribiri, T. Schaeffter, and C. Elster. Large-scale Bayesian spatial-temporal regression with application to cardiac MR-perfusion imaging. *SIAM Journal on Imaging Sciences*, 12(4):2035–2062, 2019. [p11]

D. Lewandowski, D. Kurowicka, and H. Joe. Generating random correlation matrices based on vines and extended onion method. *Journal of multivariate analysis*, 100(9):1989–2001, 2009. [p9]

Z. R. Li, T. H. McComick, and S. J. Clark. Using Bayesian latent Gaussian graphical models to infer symptom associations in verbal autopsies. *Bayesian analysis*, 15(3):781, 2020. [p11]

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953. [p6]

A. J. Molstad, L. Hsu, and W. Sun. Gaussian process regression for survival time prediction with genome-wide gene expression. *Biostatistics*, 22(1):164–180, 2021. [p11]

R. M. Neal. MCMC using Hamiltonian dynamics. In S. Brooks, A. Gelman, G. L. Jones, and X.-L. Meng, editors, *Handbook of Markov Chain Monte Carlo*, volume 2. CRC Press New York, NY, 2011. [p5, 6]

A. Nishimura, D. B. Dunson, and J. Lu. Discontinuous Hamiltonian Monte Carlo for discrete parameters and discontinuous likelihoods. *Biometrika*, 107(2):365–380, 2020. [p5, 6]

A. Nishimura, Z. Zhang, and M. A. Suchard. Zigzag path connects two Monte Carlo samplers: Hamiltonian counterpart to a piecewise deterministic markov process. *Journal of the American Statistical Association*, 120(550):1077–1089, 2025. [p5, 6, 8]

A. Pakman and L. Paninski. Exact Hamiltonian Monte Carlo for truncated multivariate Gaussians. *Journal of Computational and Graphical Statistics*, 23(2):518–542, 2014. [p5, 6]

M. Pitt, D. Chan, and R. Kohn. Efficient Bayesian inference for Gaussian copula regression models. *Biometrika*, 93(3):537–554, 2006. [p5]

M. Plummer, N. Best, K. Cowles, and K. Vines. Coda: Convergence diagnosis and output analysis for mcmc. *R News*, 6(1):7–11, 2006. URL https://journal.r-project.org/archive/. [p10]

M. A. Suchard, P. Lemey, G. Baele, D. L. Ayres, A. J. Drummond, and A. Rambaut. Bayesian phylogenetic and phylodynamic data integration using BEAST 1.10. *Virus Evolution*, 4(1):vey016, 2018. doi: 10.1093/ve/vey016. URL http://dx.doi.org/10.1093/ve/vey016. [p5]

J. Tobin. Estimation of relationships for limited dependent variables. *Econometrica: journal of the Econometric Society*, pages 24–36, 1958. [p5]

E. G. Tsionas and P. G. Michaelides. A spatial stochastic frontier model with spillovers: Evidence for italian regions. *Scottish Journal of Political Economy*, 63(3):243–257, 2016. [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. [p11]

S. Wood. *Generalized Additive Models: An Introduction with R*. Chapman and Hall/CRC, 2 edition, 2017. [p6, 11]

H. Zareifard and M. J. Khaledi. A heterogeneous Bayesian regression model for skewed spatial data. *Spatial Statistics*, 46:100545, 2021. [p5]

Z. Zhang, A. Nishimura, P. Bastide, X. Ji, R. P. Payne, P. Goulder, P. Lemey, and M. A. Suchard. Large-scale inference of correlation among mixed-type biological traits with phylogenetic multivariate probit models. *The Annals of Applied Statistics*, 15(1):230–251, 2021. [p9]

Z. Zhang, A. Nishimura, N. S. Trovão, J. L. Cherry, A. J. Holbrook, X. Ji, P. Lemey, and M. A. Suchard. Accelerating bayesian inference of dependency between mixed-type biological traits. *PLoS computational biology*, 19(8):e1011419, 2023. [p5, 6, 9]

*Zhenyu Zhang*
*Department of Biostatistics, Fielding School of Public Health,*
*University of California, Los Angeles,*
*650 Charles E. Young Dr. South, Los Angeles, CA*
*USA*
*ORCiD: 0000-0002-2176-2253*
zyz606@ucla.edu

*Andrew Chin*
*Department of Biostatistics, Bloomberg School of Public Health,*
*Johns Hopkins University,*
*615 North Wolfe Street, Baltimore, MD 21205*
*USA*
*ORCiD: 0009-0005-3832-841X*
achin23@jhu.edu

*Akihiko Nishimura*
*Department of Biostatistics, Bloomberg School of Public Health,*
*Johns Hopkins University,*
*615 North Wolfe Street, Baltimore, MD 21205*
*USA*
*ORCiD: 0000-0002-6932-2513*
aki.nishimura@jhu.edu

*Marc A. Suchard*
*Department of Human Genetics, David Geffen School of Medicine,*
*Department of Biomathematics, David Geffen School of Medicine,*
*Department of Biostatistics, Fielding School of Public Health,*
*University of California, Los Angeles,*
*695 Charles E. Young Drive, South, Los Angeles, CA*
*USA*
*ORCiD: 0000-0001-9818-479X*
msuchard@ucla.edu

# ODRF: An R Package for Oblique Decision Tree and Its Random Forest

*by Yu Liu and Yingcun Xia*

**Abstract** The classification and regression tree (CART) and Random Forest (RF) are popular machine learning methods that involve selecting one predictor at a time as the splitting variable for each node. The use of linear combinations of predictors as splitting variables is one of the popular extensions of CART and RF, known as Oblique Decision Trees (ODT), and hence ODT-based Random Forests (ODRF). Recent studies have also shown the theoretical advantages of ODT and ODRF over CART and RF. However, there is still no integrated and efficient software package that can demonstrate the numerical advantages of ODT and ODRF. To fill this gap, we make several modifications to the existing algorithms and develop an R package **ODRF** for both `ODT()` and `ODRF()`. In addition, the R package includes a series of new functions, such as ODT-based boosting trees (BODT), the ensemble version of BODT (ODBT), ODT visualization (plot.ODT), and an online update function for ODT (online.ODT). The main part of ODRF is executed using the **Rcpp** package. This paper presents the basic idea of our modifications and illustrates the use of the package. Through numerical experiments, ODRF was compared with other packages of decision trees and forests, showing a clear overall improvement in classification and regression tasks.

## 1 Introduction

The Classification and Regression Tree (CART) proposed by Professor Leo Breiman (Breiman et al., 1984) has attracted a great deal of attention from statisticians and data analysts of other disciplines. The method is widely used because it is easy to train and the resulting tree makes the analysis results easy to visualize and interpret (Johnson and Tong, 2014). On the other hand, much attention has been paid to the algorithm and many improvements have been proposed. Classification and regression trees (Quinlan, 1987, CART) and C4.5 (Quinlan, 1993) are perhaps the most commonly used decision trees. There is a long list of other decision trees, including the Evolutionary Learning of Globally Optimal Classification and Regression Trees (Grubinger et al., 2014, EVT), Conditional Inference Trees (Hothorn et al., 2006, CT), Extremely randomized trees (Geurts et al., 2006, ERT), Model-Based Recursive Partitioning (Zeileis and Hothorn, 2015, MOB), Bayesian additive regression trees (Maia et al., 2022, (BART) ) and generalized linear mixed-model trees (Fokkema et al., 2020, glmertree). The Random Forests (Breiman, 2001, RF), which is an ensemble of trees by either feature bagging or boosting, is arguably one of the most efficient machine learning methods, especially for the tabular data (Grinsztajn et al., 2022). Again, there are many ensemble methods that are based on different decision trees, for example, Conditional Random Forests (Hothorn et al., 2006, cforest), Learning Nonlinear Functions Using Regularized Greedy Forest (Johnson and Tong, 2014, RGF) and Generalized Random Forest (Athey et al., 2019, GRF).

One of the most appealing extensions to CART is the use of linear combinations of the predictors as splitting variables that is known as the Oblique Decision Tree (Heath et al., 1993, ODT). Recently, Zhan et al. (2025) proved the consistency of ODT and its random forest (ODRF) for very general regression functions as long as they are continuous, while CART or RF are consistent mainly for regressions with special structures such as additive models. Again, ensemble can be made based on ODT, resulting in the Oblique-type Random Forests, including Random Rotation Random Forest (RotRF) of Blaser and Fryzlewicz (2016), Random Projection Forests (RPFs) of Lee et al. (2015) and Sparse Projection Oblique Randomer Forests (SPORF) of Tomita et al. (2020). Another type of random forests is the model-based oblique decision forest, including mainly Canonical Correlation Forests (CCF) with classic correlation analysis (Rainforth and Wood, 2015), projection pursuit forest (PPF) with linear discriminant analysis (da Silva et al., 2021), oblique random forests (ORF) with ridge regression (Menze et al., 2011), oblique random survival forests (Jaeger et al., 2022, ORSF) and Heterogeneous oblique random forest (Katuwal et al., 2020, HORF). However, a primary limitation of these oblique methods lies in their computational intensity. To address this issue, Wickramarachchi et al. (2016) introduced a novel algorithm named "HHCART", which utilizes Householder matrices to reflect training data at non-terminal nodes. Subsequently, Wickramarachchi et al. (2019) developed an accompanying R package **hhcartr** for both classification and regression tasks.

In parallel with these developments, Gradient Boosting Trees (GBT) have emerged as another powerful machine learning paradigm. The GBT algorithm was first introduced by Friedman (2001), establishing a powerful framework for supervised learning by sequentially fitting weak learners (typically decision trees) to the residuals of previous models. To address overfitting, Friedman (2002)

later proposed stochastic gradient boosting, which incorporates regularization by subsampling both observations and features. Further enhancements tailored to tree-based models were introduced through modern implementations such as XGBoost (Chen and Guestrin, 2016), LightGBM (Ke et al., 2017), and CatBoost (Prokhorenkova et al., 2018). These frameworks optimize tree growth strategies, efficiently handle categorical variables, and significantly improve computational scalability. Traditionally, boosting algorithms employed axis-aligned splits (as in CART-style trees). However, recent research has explored Oblique Decision Trees (ODT) within boosting frameworks, resulting in Boosting Oblique Decision Trees (BODT). Integrating BODT with randomization techniques, such as random projections (Lee et al., 2015), further enhances robustness and generalizability. Notably, our recent work demonstrated that ensembles based on BODT achieve consistency for general continuous regression functions, in contrast to traditional CART-based ensembles that require additive or structured assumptions.

Although the theoretical advantages of oblique decision trees and their random forests have been well understood, the existing packages implementing those extensions only show their better numerical performance in some special cases, and thus have not commonly received and have not got much popularity as the methods deserve. As a consequence, the conventional packages **rpart** (Therneau and Atkinson, 2022) of Therneau and Atkinson (2000) for CART and **randomForest** (Liaw and Wiener, 2002a) of Liaw and Wiener (2002b) for RF are still the most commonly used packages. The main difficulty in implementing ODT or ODRF is the estimation of the coefficient, $\theta$, for the linear combinations, which is also one of the main differences amongst all the existing packages. The estimation methods of $\theta$ include random projection, logistic regression, dimension reduction and many others. For example, package **rerf** (Browne and Tomita, 2019) of Tomita et al. (2020) uses random projections; package **PPforest** (da Silva et al., 2025) of da Silva et al. (2021) uses linear discriminant and penalized discriminant analysis; package **obliqueRF** (Menze and Splitthoff, 2012) of Menze et al. (2011) uses the partial least squares regression, logistic regression, and random projection to find the projections. On the other hand, most of the existing R packages for such types of random forests can only be used for classification. Some of the R packages have been removed from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/`. For comprehensive details, see Table 1, which presents a comparative analysis of nine R packages implementing oblique tree methods. The evaluation assesses their support for six specific functionalities and determines whether they remain available on CRAN.

Our package, called **ODRF**, is written based on the above-mentioned packages and the recent work of Zhan et al. (2025). In **ODRF**, the projection pursuit regression (Friedman and Stuetzle, 1981) is used to find $\theta$; the details will be stated in the following section. Other options for the estimations of projections are also provided in the package. Compared with the existing random forests, the advantages of **ODRF** are as follows.

- **ODRF** can be used for both classification and regression, while most existing packages of oblique-type trees or forests can only be used for classification.
- Both the tree (`ODT`) and random forests (`ODRF`) of **ODRF** have better overall accuracy than existing trees and forests, including traditional CART and RF and other oblique trees or forests.
- **ODRF** allows users to define their own functions to find the projections at each node, which is essential to the performance of the forests.
- **ODRF** provides an `online` class of functions that facilitate the application to streaming data, enabling continuous updates and improvements to existing trees or random forests.
- **ODRF** develops a new ensemble method (ODBT), which builds random forests of ODT-based boosting trees and outperforms traditional boosting trees and random forests in both classification and regression tasks.
- **ODRF** proposes an innovative approach for constructing linear model trees by incorporating LASSO regularization, while distinctly separating the sets of variables used for node splitting and model fitting.

The remainder of this paper is organized as follows: Section 2 describes the model or methods of the main functions in **ODRF**. Section 3 provides the usage of the main functions in **ODRF** package. Section 4 showcases the specific application of **ODRF** to analyze two data sets with continuous and categorical responses respectively, and compares the predictive accuracy in classification and regression with other R packages using 43 datasets. A conclusion is made in Section 5.

## 2 Statistical methods

Suppose $Y = (y_1, ..., y_K)$ is the response vector of interest and $X = (x_1, ..., x_p)^\top : p \times 1$ is the vector of predictors. We allow $Y$ to be multiple to accommodate the categorical response. That is, if $Y$ has $K$

| | classification | regression | forests | boosting | custom splits | online updating | CRAN |
|---|---|---|---|---|---|---|---|
| oblique.tree | ✔ | ✘ | ✘ | ✘ | ✘ | ✘ | ✘ |
| pptreeviz | ✓ | ✘ | ✘ | ✘ | ✘ | ✘ | ✔ |
| PPtreereg | ✘ | ✔ | ✘ | ✘ | ✘ | ✘ | ✘ |
| rotationForest | ✔ | ✘ | ✔ | ✘ | ✘ | ✘ | ✔ |
| rerf | ✔ | ✘ | ✔ | ✘ | ✘ | ✘ | ✘ |
| PPforest | ✔ | ✘ | ✔ | ✘ | ✘ | ✘ | ✔ |
| obliqueRF | ✔ | ✘ | ✔ | ✘ | ✘ | ✘ | ✘ |
| aorsf | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ | ✔ |
| ODRF | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |

**Table 1:** A comprehensive overview of the R package for oblique tree methods

classes, then it is represented by $K$ dummy variables, with each taking values 0 and 1. Generally, we need to estimate the regression functions:

$$m(x) = (m_1(x), ..., m_K(x)) = (E(y_1|X = x), ..., E(y_K|X = x)).$$

### 2.1 Create an ODT

With observations $\mathbb{A}_0^0 = \{(X_i, Y_i), i = 1, ...., n\}$, where $Y_i = (y_{i1}, ..., y_{iK})$ and $X_i = (x_{i1}, ..., x_{ip})$, an ODT is illustrated by the following diagram in Figure 1. For ease of exposition, even if a node is not split further, we still rewrite it in the next layer (by a dashed line in the diagram). For any node $\mathbb{A}_\ell^\tau$, where the subscript $\ell$ represents the layer and superscript the number of nodes in the layer, the splitting is as follows. Given any $p$-dimensional vector $\theta$ and a splitting value $c$, define the daughter nodes by

$$\mathbb{A}_{\ell+1}^{\tau'} = \{X_i : X_i \in \mathbb{A}_\ell^\tau,\ \theta^\top X_i \le c\}, \quad \mathbb{A}_{\ell+1}^{\tau''} = \{X_i : X_i \in \mathbb{A}_\ell^\tau,\ \theta^\top X_i > c\},$$

and loss function

$$\Delta(c|\mathbb{A}_\ell^\tau, \theta) = \sum_{k=1}^{K} \sum_{X_i \in \mathbb{A}_{\ell+1}^{\tau'}} (y_{ik} - \bar{y}_k(\mathbb{A}_{\ell+1}^{\tau'}))^2 + \sum_{k=1}^{K} \sum_{X_i \in \mathbb{A}_{\ell+1}^{\tau''}} (y_{ik} - \bar{y}_k(\mathbb{A}_{\ell+1}^{\tau''}))^2,$$

where $\bar{y}_k(\mathbb{A}) = \sum_{X_i \in \mathbb{A}} y_{ik}/\#\mathbb{A}$ and $\#\mathbb{A}$ denotes the cardinality of set $\mathbb{A}$. When $\theta$ is given, the splitting value $c$ should minimize $\Delta(c|\theta, \mathbb{A}_\ell^\tau)$. If $Y$ is a univariate quantitative response, then $\Delta(c|\theta, \mathbb{A}_\ell^\tau)$ is the residual sum of squares for regression that is used as the criterion of CART. If $Y$ is categorical, then $\bar{y}_k(\mathbb{A}) = \hat{p}_k(\mathbb{A})$, the ratio of category $k$ in $\mathbb{A}_{\ell+1}^{\tau'}$, and

$$\sum_{X_i \in \mathbb{A}} (y_{ik} - \bar{y}_k(\mathbb{A}))^2 = \#(\mathbb{A}) \times (1 - \hat{p}_k(\mathbb{A}))\hat{p}_k(\mathbb{A}).$$

Thus, $\Delta(c|\theta, \mathbb{A}_\ell^\tau)$ is also the Gini impurity but multiplied by the number of observations in the node, which thus is also the criterion used by CART for classification.

To determine whether a split is necessary, the cross-validation (CV) method is used as follows. For any set $\mathbb{A}$, the CV value is defined as a scaled residual sum of squares:

$$CV(\mathbb{A}) = \Big(\frac{\#\mathbb{A}}{\#\mathbb{A} - \lambda}\Big)^2 \sum_{k=1}^{K} \sum_{X_i \in \mathbb{A}} (y_{ik} - \bar{y}_k(\mathbb{A}))^2,$$

Note that if $\lambda = 1$, then $CV(\mathbb{A})$ is the leave-one-out CV. We can show that $\lambda = \log(\#\mathbb{A})$ also gives a consistent stopping time. In our package, if

$$CV(\mathbb{A}_\ell^\tau) \le CV(\mathbb{A}_{\ell+1}^{\tau'}) + CV(\mathbb{A}_{\ell+1}^{\tau''}),$$

then $\mathbb{A}_\ell^\tau$ is a leaf and no longer needs to be split; otherwise, it will be split to daughter nodes $\mathbb{A}_{\ell+1}^{\tau'}$ and $\mathbb{A}_{\ell+1}^{\tau''}$; see the first equation of this section.

Let $\{\mathbb{A}_L^j\}_{j=1}^{t_n}$ be all the leaves (i.e. the nodes in the last layer) of the generated tree. Then, $m(x)$ is

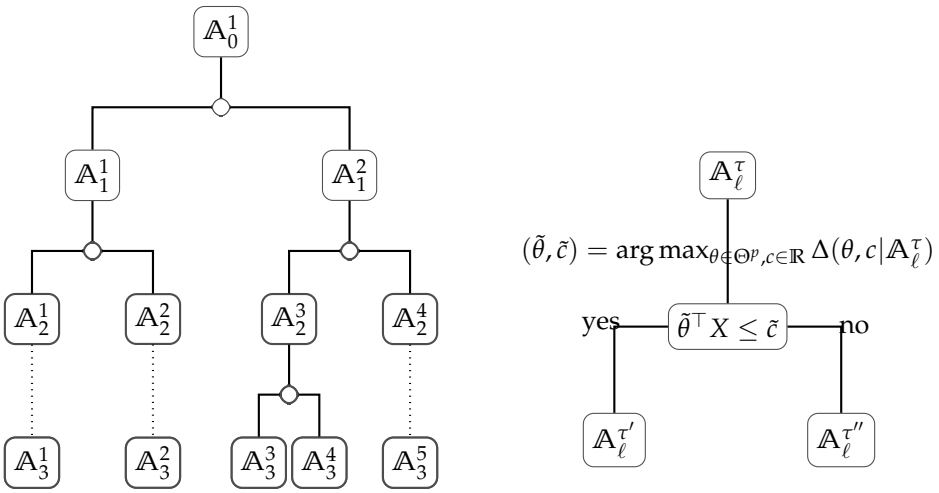



**Figure 1:** A diagram of the oblique decision tree

estimated by

$$m_n(x) = \sum_{j=1}^{t_n} \mathbb{I}(x \in \mathbb{A}_L^j) \times (\bar{y}_1(\mathbb{A}_L^j), ..., \bar{y}_K(\mathbb{A}_L^j)).$$

Note that if $Y$ is categorical, $m_n(x)$ is the vector of probabilities of each class.

It can be seen that the main step in ODT is the estimation of the projection $\theta$, i.e. the coefficients of the linear combinations. Although many methods have been proposed as mentioned in the Introduction section, we find the projection pursuit regression (Friedman and Stuetzle, 1981) is still the most efficient and is used in **ODRF**. The estimation is as follows. In any node $\mathbb{A}_\ell^\tau$, define a loss function

$$\Delta(\theta) = \sum_{k=1}^{K} \sum_{X_i \in \mathbb{A}_\ell^\tau} \left\{ y_{ik} - m_k(\theta^\top X_i) \right\}^2,$$

where $m_k$ is a nonparametric smoothing of the regression function that minimizes $\sum_{X_i \in \mathbb{A}_\ell^\tau} \left\{ y_{ik} - m_k(\theta^\top X_i) \right\}^2$ with $\theta$ given. The nonparametric smoothing can be either a spline representation, kernel smoothing, or a super smoother of Friedman (1984). Projection $\theta$ is estimated as

$$\theta = \arg\min_\theta \Delta(\theta).$$

Our package also allows users to define their own methods of estimating $\theta$; see Section 3.5.

### 2.2 Build an ODRF

**ODRF** builds the random forest slightly differently from the existing forests, but its basic idea is still the feature bagging (Ho, 1998). The detail is as follows. Denote by $X_{[q]} = (x'_1, ..., x'_q)$ a random subset of predictors $X = (x_1, ..., x_p)$, where $q < p$ and $\{x'_1, ..., x'_q\} \subset \{x_1, ..., x_p\}$, and thus by $X_{[q],r}$, $r = 1, 2, ...$, we mean a sequence of such subsets that may differ from one another. In other words, with the same $q$ and $r$, set $X_{[q],r}$ changes from place to place but has the same cardinality. The ODRF is implemented as follows.

- Create a random ODT tree, labeled as $b$, using the idea of feature bagging as follows.
  - For each node $\mathbb{A}_k^\tau$, randomly select $q$, for example $q = \lfloor p/3 \rfloor$ or $q = \lfloor \sqrt{p} \rfloor$, variables $X_{[q],r} = (x'_1, ..., x'_q) \subset X$, where $r = 1, ..., R$ denotes $R$ random sets of variables. In **ODRF**, $R \geq \lfloor p/q \rfloor$ is used as default.
  - For each $X_{[q],r}$, find

$$\tilde{\theta}_r = \arg\min_\theta \sum_{k=1}^{K} \sum_{X_i \in \mathbb{A}} \left\{ y_{ik} - m_k(\theta^\top X_{[q],r,i}) \right\}^2,$$

    where $X_{[q],r,i} = (x'_{i1}, ..., x'_{iq})^\top$.

- Define

$$\mathbb{A}_{k,r}^{\tau'} = \{X_i : X_i \in \mathbb{A}_k^{\tau},\ \theta_{(q)}^{\top} X_{[q],r,i} \le c_{(q)}\}, \quad \mathbb{A}_{k,r}^{\tau''} = \{X_i : X_i \in \mathbb{A}_k^{\tau},\ \theta_{(q)}^{\top} X_{[q],r,i} > c_{(q)}\}.$$

- Calculate

$$(\tilde{c}_{(q)}, \tilde{r}) = \arg \min_{c,r=1,\ldots,R} \Big\{ \sum_{k=1}^{K} \sum_{X_i \in \mathbb{A}_{\ell,r}^{\tau'}} (y_{ik} - \bar{y}_{.k})^2 + \sum_{k=1}^{K} \sum_{X_i \in \mathbb{A}_{\ell,r}^{\tau''}} (y_{ik} - \bar{y}_{.k})^2 \Big\}.$$

- Split the node with $(\tilde{\theta}_{\tilde{r}}, \tilde{c}_{\tilde{r}}, \tilde{r})$ to produce daughter nodes

$$\mathbb{A}_{\ell+1}^{\tau'} = \mathbb{A}_{\ell+1,\tilde{r}}^{\tau'} \text{ and } \mathbb{A}_{\ell+1}^{\tau''} = \mathbb{A}_{\ell+1,\tilde{r}}^{\tau''}.$$

If $CV(\mathbb{A}_k^{\tau}) < CV(\mathbb{A}_{\ell+1}^{\tau'}) + CV(\mathbb{A}_{\ell+1}^{\tau''})$, we discard the two daughter nodes, and make $\mathbb{A}_k^{\tau}$ as a leaf; otherwise, we continue to split the nodes.

- Each tree produces one estimator $\hat{m}^b(x) = (\hat{m}_1^b(x), ..., \hat{m}_K^b(x))$.

- An Oblique Decision Random forest (ODRF) estimator is then

$$\hat{m}_{ODRF}(x) = B^{-1} \sum_{b=1}^{B} \hat{m}_n^b(x).$$

### 2.3 Build an ODBT

Building upon the foundational gradient boosting framework established by Friedman (2001), the boosting process constructs trees sequentially. Specifically, the $k$-th tree is trained using the predictors and residuals derived from the previous $k-1$ trees. Consequently, the estimator $m^{(k)}(x)$ at step $k$ is expressed as a linear combination of the first $k$ trees. In this work, we propose using Oblique Decision Trees (ODT) as base learners within the boosting framework. The resulting ODT-based boosting algorithm (BODT) proceeds according to the following key steps:

- Step 1. Initialize $k = 1$, training set $\mathcal{D}_n^1 = \mathbb{A}_0^0$, residuals $r_{0,i} = Y_i$ for $i = 1, \ldots, n$, and estimator $m_{t,0}^r = 0$.
- Step 2. Train the $k$-th ODT using data $\mathcal{D}_n^k = \{(X_i, r_{k-1,i})\}_{i=1}^n$. Denote the resulting tree as $m_{t,k}^r(x)$. (Note that for $k = 1$, $m_{t,1}^r$ is simply the ODT trained on $\mathcal{D}_n$.)
- Step 3. Update the estimator of $m(x)$ at step $k$ by:

$$m_{k,\text{boost}}(x) := \sum_{j=1}^{k} a_{k,j}^* m_{t,j}^r(x), \tag{2.1}$$

where the coefficients $(a_{k,1}^*, a_{k,2}^*, \ldots, a_{k,k}^*)$ are obtained by solving:

$$(a_{k,1}^*, \ldots, a_{k,k}^*) := \underset{(a_{k,1},\ldots,a_{k,k}) \in \mathbb{R}^k}{\arg\min} \sum_{i=1}^{n} \left( Y_i - \sum_{j=1}^{k} a_{k,j} m_{t,j}^r(X_i) \right)^2. \tag{2.2}$$

- Step 4. Update the residuals as $r_{k,i} = Y_i - m_{k,\text{boost}}(X_i)$ and the training data as $\mathcal{D}_n^{k+1} = \{(X_i, r_{k,i})\}_{i=1}^n$.
- Step 5. Set $k \leftarrow k + 1$ and return to Step 2.

To further improve the stability and predictive accuracy of the boosting trees, we incorporate a bagging step by replacing each deterministic ODT with a randomly constructed ODT. For simplicity, we set $a_n = n$, meaning the full dataset $\mathcal{D}_n$ is used at every boosting iteration. During the construction of each random ODT, independent random seeds (as defined at the beginning of Section 2.2) are employed. By a slight abuse of notation, we continue to denote the resulting estimator as $m_{k,\text{boost}}(x)$. Running the BODT procedure independently $B$ times, each with its own set of random ODTs, produces an ensemble estimator, referred to as ODBT, defined by

$$m_{k,\text{boost}}^{\text{ens}}(x) := \frac{1}{B} \sum_{b=1}^{B} m_{k,\text{boost}}^b(x),$$

where $m_{k,\text{boost}}^b(x)$ corresponds to the boosting estimator from the $b$-th independent run.

For a comprehensive discussion of the technical details and theoretical properties of ODBT, we refer the reader to our parallel work.

### 2.4 Online training with batches of data

The package allows for easy model updating (for ODT or ODRF) when new data is available. Essentially, we achieve this by splitting a leaf node upon the arrival of new data. The specifics of this process vary based on the amount of available data. To keep things simple, we focus on classification trees that use Gini impurity as the splitting criterion.

Suppose the trained ODT has leaves $A_L^j, j = 1, ..., J$. When the new data comes, we can simply fit the new data into the leaves of the trained tree. Suppose $D_L^j = \{(X_i', y_i'), i = 1, ..., n_j'\}$ fall into $A_L^j$, where $y_i' = (y_{i,1}', ..., y_{i,K}')$, thus the data in the leaf becomes $A_L^j \cup D_L^j$. Next, we discuss whether we need to split $A_L^j \cup D_L^j$ in different scenarios of data availability.

- If the full data for the original trained model is known, that is, the data in $A_L^j \cup D_L^j$ is fully known, we can use the method of ODT to split the data.
- If only the impurity is known, i.e. $n_j = \#A_L^j$ and $(\hat{p}_{L,1}^j, ..., \hat{p}_{L,K}^j)$ are kept for the leaf, which is indeed the case for many existing packages of trees. Here, $\hat{p}_{L,k}^j$ is the probability of class $k$ in the leaf $A_L^j$. The impurity of $A_L^j$ and $A_L^j \cup D_L^j$ can be calculated respectively as

$$G_L^j = \sum_{k=1}^{K} \hat{p}_k(1 - \hat{p}_k)$$

and

$$\tilde{G}_L^j = \sum_{k=1}^{K} \tilde{p}_k(1 - \tilde{p}_k),$$

where

$$\tilde{p}_k = (n_j \hat{p}_{L,k}^j + n_j' \check{p}_{L,k}^j)/(n_j + n_j')$$

and $n_j' = \#D_L^j$ and $\check{p}_{L,k}^j = \sum_{X_i' \in D_L^j} y_{i,k}'/n_j'$. If $\hat{G}_L^j \leq \tilde{G}_L^j$, we don't need to split. Otherwise, we first split $D_L^j$ in the same way as ODT does, and denote the daughter nodes by $A_L^{j,l}$ and $A_L^{j,r}$. We assume that the original data in the leaf are also proportionally split into the two daughter nodes. Thus, their impurities are

$$G(A_\ell^{j,l}) = \sum_{k=1}^{K} \hat{p}_k^l(1 - \hat{p}_k^l), \quad G(A_\ell^{j,r}) = \sum_{k=1}^{K} \hat{p}_k^r(1 - \hat{p}_k^r),$$

where $n_j^l = \tau n_j + \#A_L^{j,l}$ and $n_j^r = (1 - \tau)n_j + \#A_L^{j,r}$, $\hat{p}_k^l = (\tau n_j \hat{p}_{L,k}^j + \sum_{X_i' \in A_L^{j,l}} y_{i,k}')/n_j^l)$ and $\hat{p}_k^r = ((1 - \tau)n_j \hat{p}_{L,k}^j + \sum_{X_i' \in A_L^{j,r}} y_{i,k}')/n_j^r$, and $\tau = \#A_\ell^{j,l}/n_j'$. Following the same rule as ODT in the splitting, if

$$\left(\frac{n_j + n_j'}{n_j + n_j' - \lambda}\right)^2 (n_j + n_j')\tilde{G}_j \leq \left(\frac{n_j^l}{n_j^l - \lambda}\right)^2 n_j^l G(A_\ell^{j,l}) + \left(\frac{n_j^r}{n_j^r - \lambda}\right)^2 n_j^r G(A_\ell^{j,r}),$$

then we split the leaf, and continue to split its daughter nodes if necessary; otherwise, $A_L^j$ is not split and remains as a leaf.

The above procedure is a combination of the batches of data in the leaves. In **ODRF**, the second scenario is used as we believe most trees have the data required.

## 3 Overview of ODRF functions

ODRF is written in **Rcpp** (Eddelbuettel and François, 2011) package and functions of R's S3, including the base R functions `print()`, `predict()`, and `plot()` in the `base` package, the conversion function `as.party()` in package **partykit** (Hothorn and Zeileis, 2015). Next, we introduce the main functions of **ODRF** package, including:

- `ODT()` constructs a single oblique decision tree for classification and regression tasks, where each node split is determined by a linear combination of predictors. Returns an ODT S3 object compatible with `plot()`, `predict()`, and `print()` methods.
- `ODRF()` implements oblique random forest ensembles for classification and regression, extending the random forest framework through ODT-based construction while encompassing standard random forest as a special case. Returns an ODRF S3 object and supports the
- `ODBT()` implements boosted oblique trees by applying feature bagging during ODT-based boosting training to ensemble multiple models. It accepts training and test data sets as input and directly outputs fitted values and predictions.
- `online()` supports incremental learning scenarios and is applicable to ODT and ODRF S3 objects.
- `VarImp()` computes variable importance measures and is compatible with ODT and ODRF S3 objects.

Finally, we demonstrate the implementation of user-defined projection estimation functions through customizable template components, enabling researchers to tailor hyperplane optimization strategies to specific problem domains.

### 3.1 Print the tree structure of `ODT` and `ODRF`

`ODT()`, `ODRF()`, and `ODBT()` are the three main functions of **ODRF** package. They can be used for classification and regression and are similar in usage to packages **rpart** and **randomForest** respectively.

We provide two ways of data input following the `S3` methods as follows.

```
## S3 method for class 'formula'
ODT(formula, data = NULL, split = "auto", NodeRotateFun = "RotMatPPO", ...)
ODRF(formula, data = NULL, split = "auto", NodeRotateFun = "RotMatPPO", ...)
ODBT(formula, data = NULL, Xnew = NULL, split = "auto", model = "ODT", ...)

## Default S3 method:
ODT(X, y, split = "auto", NodeRotateFun = "RotMatPPO", ...)
ODRF(X, y, split = "auto", NodeRotateFun = "RotMatPPO", ...)
ODBT(X, y, Xnew, split = "auto", model = "ODT", ...)
```

The `formula` and `data` are standard formats in R, so do the remaining arguments such as `subset`, `na.action`, and `weights`. For additional information on setting the values of arguments that are not introduced here, please refer to the documentation for the functions `ODT` and `ODRF` using commands `?ODT`, `?ODRF`, and `?ODBT`. However, in most cases, the default values work well.

`print()` can be used to display the trained tree and structure of each node of `ODT` in detail. Here is one example.

```
set.seed(38)
data(iris, package = "datasets")
tree <- ODT(Species ~ ., data = iris)
print(tree)

============================================================
Oblique Classification Tree structure
============================================================

1) root
node2)# proj1*X < 0.29 -(leaf1 = setosa)
node3)  proj1*X >= 0.29
node4)# proj2*X < 0.88 -(leaf2 = versicolor)
node5)# proj2*X >= 0.88 -(leaf3 = virginica)

party.tree <- as.party(tree, data = iris)
print(party.tree)

Model formula:
Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width

Fitted party:
[1] root
```

```
|   [2] proj1*X >= 0.29167
|   |   [3] proj2*X >= 0.88395: virginica (n = 53, err = 5.7%)
|   |   [4] proj2*X < 0.88395: versicolor (n = 47, err = 0.0%)
|   [5] proj1*X < 0.29167: setosa (n = 50, err = 0.0%)

Number of inner nodes:    2
Number of terminal nodes: 3
```

In addition, we can also use function `print()` to print the model fitted error for `ODRF`.

```
set.seed(38)
forest <- ODRF(Species ~ ., data = iris, parallel = FALSE)
print(forest)
```

```
Call:
ODRF.formula(formula = Species ~ ., data = data, parallel = FALSE)
Type of oblique decision random forest: classification
Number of trees: 100
OOB estimate of error rate: 5.33\%
Confusion matrix:
           setosa versicolor virginica class_error
setosa         50          0         0    0.0000000
versicolor      0         47         5    0.0961537
virginica       0          3        45    0.0625000
```

Beyond its core functionality, the `ODT` implementation can also perform linear model trees (LMT) with LASSO regularization as proposed by Craig et al. (2024). This approach differs fundamentally from standard ODT in its data partitioning: the splitting variables and linear modeling variables are trained on distinct datasets. We demonstrate this functionality through the following illustrative case, where we use "Z" as the splitting variable to build a linear model tree for "X" and "y".

```
set.seed(10); cutpoint <- 50; mu <- rep(0, 100)
X <- matrix(rnorm(100 * 10), 100, 10)
age <- sample(seq(20, 80), 100, replace = TRUE)
height <- sample(seq(50, 200), 100, replace = TRUE)
weight <- sample(seq(5, 150), 100, replace = TRUE)
Z <- cbind(age = age, height = height, weight = weight)
mu[age <= cutpoint] <- X[age <= cutpoint, 1] + X[age <= cutpoint, 2]
mu[age > cutpoint] <- X[age > cutpoint, 1] + X[age > cutpoint, 3]
y <- mu + rnorm(100)
my.tree <- ODT(X = X, y = y, Xsplit = Z, split = "linear", lambda = 0, NodeRotateFun =
"RotMatRF", glmnetParList = list(lambda = 0, family = "gaussian"))
pred <- predict(my.tree, X, Xsplit = Z)
mean((pred - y)^2)
[1] 0.9035932
```

### 3.2 Classification and regression using functions `ODT()`, `ODRF()`, and `ODBT()`

`predict()` is the standard `S3` method to predict new data for various objects of classes. We defined the functions `predict.ODT()` and `predict.ODRF()` to predict `Xnew` for classes `ODT()` and `ODRF()` respectively. The default output of `predict()` is `response` which is the predicted values of the new data. Use `?predict.ODT` and `?predict.ODRF` to see the detail of the prediction.

The standard usage is as follows.

```
## S3 method for class 'ODT'
predict(object, Xnew,  ...)
## S3 method for class 'ODRF'
predict(object, Xnew, type = "response", ...)
```

Examples of classification and regression using `ODT()`, `ODRF()`, and `ODBT()` are as follows.

```
data(body_fat, package = "ODRF")
train <- sample(1:252, 200)
bodyfat_train <- data.frame(body_fat[train, ])
bodyfat_test <- data.frame(body_fat[-train, ])
tree <- ODT(Density ~ ., bodyfat_train, split = "mse")
```

```
pred <- predict(tree, bodyfat_test[, -1])
(e.tree <- mean((pred - bodyfat_test[, 1])^2))
[1] 4.775053e-05

set.seed(12)
data(seeds, package = "ODRF")
train <- sample(1:209, 150)
seeds_train <- data.frame(seeds[train, ])
seeds_test <- data.frame(seeds[-train, ])
forest <- ODRF(varieties_of_wheat ~ ., seeds_train, split = "gini", parallel = FALSE)
pred <- predict(forest, seeds_test[, -8])
(e.forest <- mean(pred != seeds_test[, 8]))
[1] 0.01694915

forest <- ODBT(varieties_of_wheat ~ ., seeds_train, seeds_test[, -8], model = "rpart",
 type = "class", max.terms = 10, parallel = FALSE, NodeRotateFun = "RotMatRF")
pred <- forest$results$prediction
(mean(pred != seeds_test[, 8]))
[1] 0.01694915
```

### 3.3 Online updating

**ODRF** provides online training for sequential data with function `online()` and update existing `ODT` and `ODRF` using batches of data.

The usage is as follows.

```
## S3 method for class 'ODT' and 'ODRF'
online(obj, X = NULL, y = NULL, ...)
```

In the following example, the training data are available in two batches. The first batch is used to train `ODT` and `ODRF`, and the second batch is used to update the trained model.

```
set.seed(17)
index <- sample(nrow(seeds_train), floor(nrow(seeds_train) / 2))
forest1 <- ODRF(varieties_of_wheat ~ ., seeds_train[index, ], split = "gini",
  parallel = FALSE)
pred <- predict(forest1, seeds_test[, -8])
(e.forest.1 <- mean(pred != seeds_test[, 8]))

[1] 0.03389831

forest2 <- online(forest1, seeds_train[-index, -8], seeds_train[-index, 8])
pred <- predict(forest2, seeds_test[, -8])
(e.forest.online <- mean(pred != seeds_test[, 8]))

[1] 0.01694915

index <- seq(floor(nrow(bodyfat_train) / 2))
tree1 <- ODT(Density ~ ., bodyfat_train[index, ], split = "mse")
pred <- predict(tree1, bodyfat_test[, -1])
(e.tree.1 <- mean((pred - bodyfat_test[, 1])^2))

[1] 6.37745e-05

tree2 <- online(tree1, bodyfat_train[-index, -1], bodyfat_train[-index, 1])
pred <- predict(tree2, bodyfat_test[, -1])
(e.tree.online <- mean((pred - bodyfat_test[, 1])^2))

[1] 5.659303e-05
```

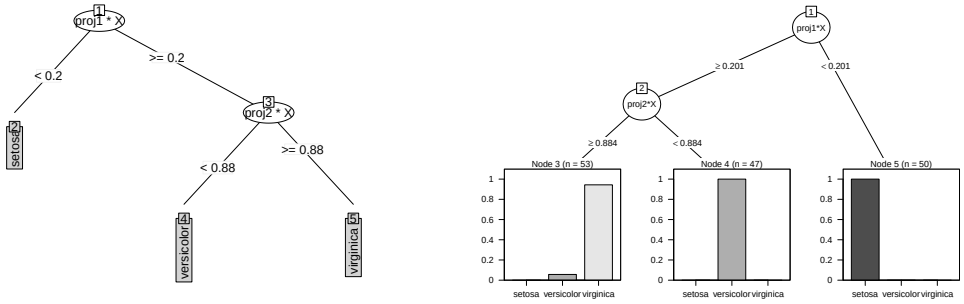
It can be seen that the errors after updating are notably smaller than those resulting from using a single batch of data alone.

### 3.4 Visualization of ODT and ODRF and importance of variables

**ODRF** provides several plot functions to visualize ODT and ODRF. `plot.ODT()` plots the oblique decision tree structure based on package **PPtreeViz**. We can convert the class `ODT` to class `party` by `as.party()` and then use `plot()` to draw the tree structure.

Suppose x or obj is an object of class ODT. The standard usage is as follows.

```
## S3 method for class 'ODT'
plot(x, font.size = 17, width.size = 1, ...)
## S3 method for class 'ODT'
as.party(obj, data, ...)
```




**(a)** The tree structure of class of ODT.

**(b)** The tree structure of class party of ODT

**Figure 2:** Two types of the tree structure.

Below is one example, while the two types of tree plots are shown in Figure 2.

```
set.seed(0308)
tree <- ODT(Species ~ ., data = iris, split = "gini")
plot(tree, main = "")
party.tree <- as.party(tree, data = iris)
plot(party.tree)
```

For `ODRF()`, we provide functions `Accuracy()` to calculate the fitting accuracy and `plot.Accuracy()` to plot the fitting errors. Hereafter, we refer to errors as misclassification rate (MR) for classification or mean squared error (MSE) for regression. Functions `VarImp()` and `plot.VarImp()` measure the variable importance and plot the dotchart of variable importance, respectively, where the variable importance is similarly calculated as `importance()` in **randomForest** package. Our implementation uses impurity-based and permutation-based methods to measure the variable importance. It should be emphasized that our impurity-based method inherently incorporates these projection coefficients into the importance ranking. Specifically, for each splitting node, we randomly select $q$ projection variables from the $p$ predictor variables and define the variable importance measure as:

$$V_k = D \cdot |Q_k|, \quad k = j_1, j_2, \cdots, j_q$$

where $D$ represents the total decrease in node impurities from splitting on the variable, and $Q_k$ denotes the coefficient of the $k$-th projection. Consequently, $V_k$ constitutes a comprehensive metric for variable importance that integrates both impurity reduction and projection coefficient information.

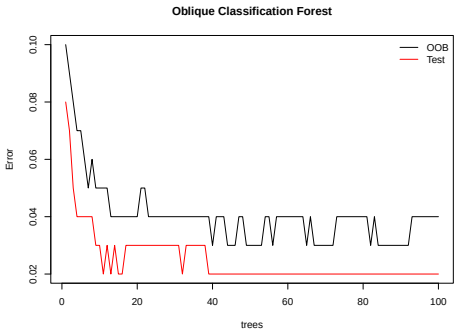
```
## S3 method for class 'VarImp'
varimp = VarImp(obj, X, y)
plot(varimp, nvar, digits = NULL, ...)

## S3 method for class 'Accuracy'
accuracy = Accuracy(obj, data, newdata = NULL)
plot(accuracy, lty = 1, digits = NULL, main = NULL, ...)
```

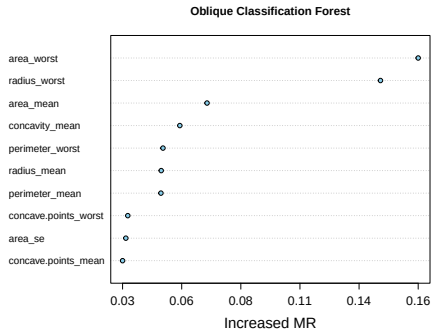
Below is an example.

```
set.seed(3)
data(breast_cancer, package = "ODRF")
train <- sample(1:569, 300)
train_data <- breast_cancer[train, -1]
```

```
test_data <- breast_cancer[-train, -1]
forest <- ODRF(diagnosis ~ ., train_data,split = "gini", parallel = FALSE)
error <- Accuracy(forest, train_data, test_data)
plot(error)
varimp <- VarImp(forest, train_data[, -1], train_data[, 1])
plot(varimp, nvar = 10)
```




**(a)** Fitting errors of ODRF.




**(b)** Dotchart of variable's importance.

**Figure 3:** The error of ODRF for classification and variable's importance.

The plots are shown in Figure 3. The left panel is the plot of errors of ODRF against the number of trees. It can be seen that the errors of OOB and test data decrease with the number of trees and that the error of test data is bigger than the training data. The right panel is the dotchart of variable importance, the horizontal axis is the error of ODRF after removing one variable; the larger the error increases, the more important the variable is.

### 3.5 Create a rotation matrix with `RotMat*` and custom functions

**ODRF** provides many ways to select the rotation directions $\theta$ in each nodes, including `RotMatPPO()`, `RotMatRand()` and `RotMatRF()`. The generated rotation matrix has three columns, the first column (`Variable`) is variables to be projected, the second column (`Number`) is the index of a projection, and the third column (`Coefficient`) is the coefficient of the projection for the variable. Details can be found by `?ODT` and `?ODRF`.

The default method for the projection of `RotMatPPO()` is `model = "PPR"`, projection pursuit regression from `ppr()` in `stats` package (Friedman and Stuetzle, 1981). Standard usage is as follows.

```
RotMatPPO(X, y, model = "PPR", dimProj, numProj, ...)
RotMatMake(X = NULL,y = NULL,RotMatFun = "RotMatPPO",PPFun = "PPO", ...)
```

Here is one example.

```
set.seed(14)
X <- matrix(rnorm(1000), 200, 5)
y <- (X[,1]+X[,2])^2 + X[,4]-X[,5] +  runif(200)
tree <- ODT(X, y, split = "mse", NodeRotateFun = "RotMatPPO",
  paramList = list(model = "PPR", dimProj = 5, numProj = 1))
round(tree[["projections"]],2)
```

```
         X1    X2    X3   X4    X5
proj1  0.20  0.29 -0.09 0.63 -0.69
proj2 -0.69 -0.68  0.02 0.19 -0.16
proj3 -0.06 -0.09  0.16 0.72 -0.67
proj4 -0.57 -0.70  0.03 0.31 -0.29
```

It can be seen that the projections are roughly parallel to those in the model, i.e. (1, 1, 0, 0, 0) and (0, 0, 0, 1, -1).

It is worth mentioning that the package allows users to define their rotation matrix functions and link them with the function `RotMatMake()` in the package. The first function, named, for example, `makeRotMat()`, is to select the variables to be projected, with a specified format of output, including the projection dimensions and the number of projections (the first two columns of the rotation matrix):

```
makeRotMat <- function(dimX, dimProj, numProj, ...) {
  RotMat <- matrix(1, dimProj * numProj, 3)
  for (np in seq(numProj)) {
    RotMat[(dimProj * (np - 1) + 1):(dimProj * np), 1] <-
    sample(1:dimX, dimProj, replace = FALSE)
    RotMat[(dimProj * (np - 1) + 1):(dimProj * np), 2] <- np
    RotMat[(dimProj * (np - 1) + 1):(dimProj * np), 3] <-
    sample(c(1L, -1L), dimProj, replace = TRUE, prob = c(0.5, 0.5))
  }
  return(RotMat)
}
set.seed(35)
RotMat1 <- makeRotMat(dimX = 5, dimProj = 3, numProj = 2)
RotMat1
```

```
     [,1] [,2] [,3]
[1,]    2    1   -1
[2,]    5    1   -1
[3,]    1    1   -1
[4,]    5    2    1
[5,]    2    2    1
[6,]    4    2   -1
```

The second function, named as, for example, `makePP()`, is defined to estimate the projection coefficients (the third column of the rotation matrix):

```
makePP <- function(X, y, ...) {
  LM <- lm(y ~ ., data = data.frame(X,y=y))
  theta <- as.matrix(LM[["coefficients"]])[-1, , drop = FALSE]
  theta <- theta / sqrt(sum(theta^2))
  return(theta)
}
set.seed(35)
RotMat3 <- RotMatMake(X = X, y = y, RotMatFun = "makeRotMat", PPFun = "makePP",
  paramList = list(dimX = 5, dimProj = 3, numProj = 2))
RotMat3
```

```
     Variable Number Coefficient
[1,]        2      1   0.4297059
[2,]        5      1  -0.9007557
[3,]        1      1   0.0631821
[4,]        5      2  -0.7227304
[5,]        2      2   0.2141932
[6,]        4      2   0.6571012
```

Then, we put these functions as augments of `NodeRotateFun` and `PPFun` respectively as follows.

```
set.seed(23)
tree <- ODT(X, y, split = "mse", NodeRotateFun = "RotMatMake", paramList =
  list(RotMatFun = "makeRotMat", PPFun = "makePP", dimX = 5, dimProj = 5, numProj = 1))
round(tree[["projections"]], 2)
```

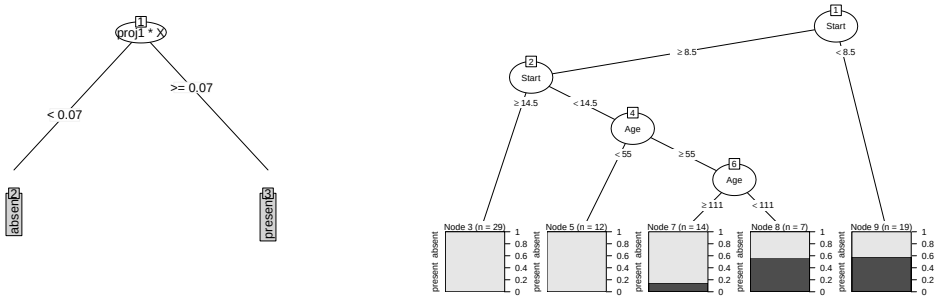
```
         X1    X2    X3   X4    X5
proj1  0.08  0.21 -0.11 0.69 -0.68
proj2  0.22  0.39 -0.16 0.69 -0.55
proj3 -0.27 -0.03  0.05 0.68 -0.68
```

## 4 Real examples

In this section, we first use `ODT()` to analyze two data sets and see how it improves the analysis of CART, in terms of model complexity, model fitting and variable importance. In the second example, we compare the prediction performance of ODT and ODRF in **ODRF** with the other popular methods using 43 real data sets, including 20 continuous responses, 8 multinomial responses and 15 binary responses. All real data analyses were conducted in R version 4.2.2 on a Windows 11 Operating System with AMD Ryzen 7 5800H with Radeon Graphics (16 CPUs), 3.2 GHz with 16 GB of memory.

### 4.1 Two data sets for complexity analysis

The first dataset is the kyphosis data, which has 81 observations. The data (`kyphosis`) are available at **rpart**. The response is *kyphosis* with values 'absent' or 'present', indicating whether kyphosis (a type of deformity) was present after surgery. The three predictors are (1) *age*: the age of the child in months; (2) *number*: the number of vertebrae involved; (3) *start*: the number of the first (uppermost) vertebra operated on. Below is the code and analysis. We first fit the data to ODT and CART, and calculate the




**(a)** The ODT tree structure of kyphosis.

**(b)** The **rpart** tree structure of kyphosis.

**Figure 4:** The tree structure of kyphosis of ODT and the conventional CART respectively.

fitted errors.

```
data(kyphosis, package = "rpart")
odt <- ODT(Kyphosis ~ Age + Number + Start, data = kyphosis, split = "gini",
  paramList = list(model = "PPR", numProj = 1))
tree <- rpart(Kyphosis ~ Age + Number + Start, data = kyphosis, method = "class")
pred <- predict(odt, kyphosis[, -1])
e.odt <- mean(pred != kyphosis[, 1])
pred <- predict(tree, kyphosis, type = "class")
e.tree <- mean(pred != kyphosis[, 1])
print(c(e.odt = e.odt, e.tree = e.tree))

    e.ODT    e.CART
0.1604938 0.1604938

print(round(odt[["projections"]], 3))

       Age Number  Start
proj1 0.31  0.596 -0.741
```

The trees of ODT and CART are shown in Figure 4. We can see from the trees that ODT has a much smaller complexity, i.e. 2 leaves, than CART which has 5 leaves, but has the same fitted error as CART. There is only one projection for ODT with splitting variable $0.310 \times \text{Age} + 0.596 \times \text{Number} - 0.74 \times \text{Start}$, which determines whether kyphosis will present or not after an operation.

The second data set is about the relative performance and characteristics of 209 CPUs. This dataset contains the following variables: *syct* (cycle time in nanoseconds), *mmin* (minimum main memory in kilobytes), *mmax* (maximum main memory in kilobytes), *catch* (cache size in kilobytes), *chmin* ( minimum number of channels), *chmax* (maximum number of channels), *perf* (published performance on a benchmark mix relative to an IBM 370/158-3), and *estperf* (estimated performance by Ein-Dor & Feldmesser). This data (cpus) is available in **MASS**. The interest is to predict the *perf* using *syct*, *mmin*, *mmax*, *cach*, *chmin* and *chmax*. The logarithm transformation is made to *perf*.

```
data(cpus, package = "MASS")
odt <- ODT(log10(perf) ~ syct + mmin + mmax + cach + chmin + chmax, data = cpus,
  lambda = log(nrow(cpus)), split = "mse", paramList = list(model = "PPR", numProj = 1))
pred <- predict(odt, cpus[, 2:7])
e.odt <- mean((pred - log10(cpus[, 8]))^2)
tree <- rpart(log10(perf) ~ syct + mmin + mmax + cach + chmin + chmax,   data = cpus)
pred <- predict(tree, cpus)
e.tree <- mean((pred - log10(cpus[, 8]))^2)
print(c(e.ODT = e.odt, e.CART = e.tree))
```

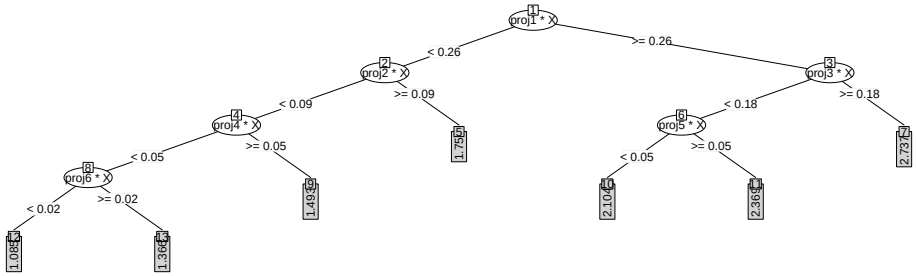



**Figure 5:** ODT tree structure of cpus

```
    e.ODT     e.CART
0.02297886 0.03034244

print(round(odt[["projections"]], 3))

        syct   mmin  mmax   cach chmin  chmax
proj1 -0.047  0.546 0.586  0.553 0.206  0.094
proj2 -0.021  0.298 0.505  0.744 0.319  0.014
proj3 -0.963  0.125 0.181  0.121 0.010  0.101
proj4 -0.034  0.751 0.419  0.407 0.302  0.051
proj5 -0.982 -0.002 0.126  0.070 0.000  0.125
proj6 -0.012 -0.351 0.920 -0.156 0.061 -0.049
```

The ODT and CART trees are depicted in Figure 5 and Figure 6, respectively. Once again, it is evident that ODT has fewer leaves and a lower fitting error than CART. Interestingly, the first, second, and fourth projections primarily relate to a computer's memory and the time required to fetch data from main memory or disk storage (represented by *mmin*, *mmax*, *catch*, *chmin*), as their coefficients have larger absolute values. Another group of projections, specifically the third and fifth, are associated with *syct*, the time it takes a computer to complete one clock cycle, which can contribute to faster processing speeds and improved performance. In some special cases, *mmax* plays a role, i.e. projection 6, possibly when the data set is large. These projections correspond to the common understanding of the performance of a computer and its hardware settings. These examples show that ODT has better interpretability than CART.

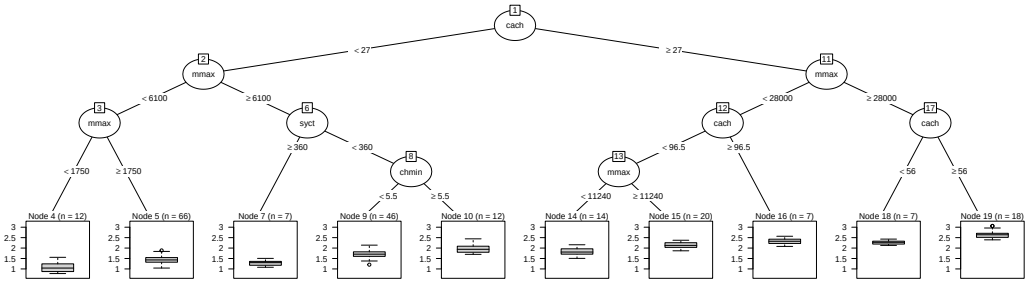
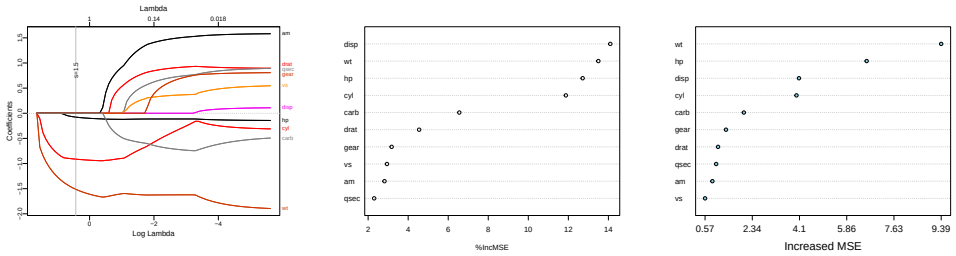



**Figure 6:** The conventional tree structure of cpus using **rpart**.

### 4.2 Analysis of the importance of variables

The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design, including mpg: Miles/(US) gallon, cyl: Number of cylinders, disp: Displacement (cu.in.), hp: Gross horsepower, drat: Rear axle ratio, wt: Weight (1000 lbs), qsec: 1/4 mile time, vs: Engine (0 = V-shaped, 1 = straight), am: Transmission (0 = automatic, 1 = manual), gear: Number of forward gears, and carb: Number of carburetors, and performance, which is measured by the fuel consumption (mpg). This data (`mtcars`) is available in `datasets` (Henderson and Velleman, 1981). Our interest is to explore which aspects of automotive design affect the performance (mpg).

ODRF measures variable importance using a similar permutation method as RF. We will compare the rank of importance of ODRF with RF. In addition, LASSO is a popular variable selection method, so we use it as a benchmark. Figure 7 shows that ODRF ranks Weight (wt) as the most important




**(a)** LASSO **(b)** RF **(c)** ODRF

**Figure 7:** Plots of the importance of variables for data "mtcars" using three methods.

factor affecting fuel consumption, which is consistent with the LASSO results. RF ranks Displacement (disp) as an important influencing factor, while the LASSO results show Displacement as the least influencing factor. Further comparison of the other variable rankings shows that our ODRF rankings of variable importance are generally consistent with LASSO, while RF differs more.

| | | | Axis-aligned | | | | Oblique | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | n | p | CART | ERT | EVT | CT | RotT | SPOT | PPT | ODT |
| Servo (C) | 166 | 4 | 0.298 | 0.812 | **0.257** | 0.300 | 0.770 | 0.563 | 0.387 | 0.306 |
| AutoMpg (C) | 391 | 7 | 0.213 | 0.314 | 0.217 | 0.192 | 0.281 | 0.245 | 0.184 | **0.175** |
| Concrete Compressive (A) | 1030 | 8 | 0.314 | 0.513 | 0.308 | 0.248 | 0.539 | 0.446 | 0.502 | **0.202** |
| Boston house price (A) | 506 | 13 | 0.275 | 0.458 | 0.294 | 0.275 | 0.439 | 0.379 | 0.262 | **0.261** |
| Wild blueberry yield (B) | 777 | 13 | 0.228 | 0.347 | 0.224 | 0.189 | 0.319 | 0.265 | **0.099** | 0.113 |
| Body fat (B) | 252 | 14 | 0.073 | 0.418 | 0.085 | **0.061** | 0.446 | 0.402 | 0.096 | 0.072 |
| Paris housing price (B) | 10000 | 16 | 0.021 | 0.370 | 0.005 | **0.000** | 0.761 | 0.662 | 0.006 | **0.000** |
| House sales in King County (B) | 21613 | 18 | 0.327 | 0.482 | 0.324 | 0.287 | 0.442 | 0.360 | 0.314 | **0.256** |
| Bar crawl (B) | 7590 | 21 | 0.323 | 0.498 | 0.326 | 0.297 | 0.499 | 0.429 | 0.326 | **0.280** |
| Auto 93 (C) | 81 | 22 | 0.620 | 0.927 | 0.706 | 0.602 | 0.753 | 0.683 | 0.707 | **0.599** |
| Auto horsepower (C) | 159 | 24 | 0.236 | 0.340 | 0.255 | 0.238 | 0.487 | 0.384 | 0.230 | **0.189** |
| Facebook comment volume (A) | 18370 | 52 | **0.700** | 1.210 | 0.756 | 0.717 | 0.971 | 0.846 | 0.841 | 0.743 |
| Gold price prediction (B) | 1718 | 74 | 0.043 | 0.023 | 0.023 | 0.016 | 0.136 | 0.029 | 0.027 | **0.010** |
| Baseball player statistics (B) | 4535 | 74 | 0.039 | 0.225 | 0.017 | 0.010 | 0.646 | 0.470 | 0.054 | **0.007** |
| CNNpred (A) | 1441 | 76 | 0.032 | 0.016 | 0.010 | 0.003 | 0.124 | 0.068 | 0.019 | **0.002** |
| Superconductivity (A) | 21263 | 81 | 0.296 | 0.387 | 0.293 | 0.287 | 0.332 | 0.309 | 0.323 | **0.271** |
| Warsaw flat rent price (B) | 3472 | 78 | 0.447 | 0.750 | 0.462 | **0.425** | 0.965 | 0.624 | 0.518 | 0.469 |
| Buzz in social media (A) | 28179 | 96 | 0.206 | 0.347 | 0.234 | **0.201** | 0.489 | 0.349 | 0.378 | 0.344 |
| Communities and Crime (A) | 1994 | 101 | 0.475 | 0.746 | 0.480 | **0.445** | 0.568 | 0.526 | 0.617 | 0.545 |
| Residential building-Sales (A) | 372 | 103 | 0.060 | 0.231 | 0.060 | **0.040** | 0.356 | 0.285 | 0.176 | 0.046 |
| Average RPE across all 20 datasets | | | 0.261 | 0.471 | 0.267 | 0.242 | 0.516 | 0.416 | 0.303 | 0.245 |
| No. of bests in 20 datasets | | | 1 | 0 | 1 | 6 | 0 | 0 | 1 | 11 |

**Table 2:** Regression: average RPE based on 100 random partitions of each data set into training and test sets. The alphabetic notation in the brackets of each dataset represents the data source.

### 4.3 Prediction performance

Next, we show the performance of `ODT` and `ODRF` as well. We use 20 real data sets with continuous responses and 23 data sets with categorical responses for regression and classification respectively. Our data are mainly obtained from the UCI machine learning database (A) `https://archive.ics.uci.edu/ml/datasets`, the Kaggle database (B) `https://www.kaggle.com`, and the Collection of Datasets of Rainforth and Wood (2015) (C) `https://github.com/twgr/ccfs/`. All data can also be obtained directly from `https://github.com/liuyu-star/RJ_ODRF_Codes`. If there are any missing values in the data, the corresponding observations are removed from the data. We scale the predictors individually before performing the calculation. Although this makes no theoretical difference, it sometimes enhances computational stability and facilitates variable importance ranking.

The estimation methods of the coefficients of these projections include random projection, logistic regression, dimension reduction and many others. However, our experiments suggest that these estimations actually make little difference in the results. In our calculation, instead of using one single projection or linear combination, we provide a number of projections, each of which is for the projection of a set of randomly selected predictors, and then use Gini impurity or residual sum of squares to choose one combination as splitting variable and splitting point. The logistic regression function is used to find projections for each combination of predictors, but other alternatives are also provided in the package.

| | | | Axis-aligned | | | | Oblique | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | n | p | CART | ERT | EVT | CT | RotT | SPOT | PPT | OT | ODT |
| Iris (A, 3) | 149 | 4 | 6.96 | 21.58 | 7.08 | 5.98 | 14.74 | 15.20 | **2.66** | 4.24 | 5.70 |
| Seeds (C, 3) | 209 | 7 | 9.66 | 16.24 | 10.94 | 12.39 | 13.29 | 14.26 | **4.27** | 5.30 | 6.93 |
| Milk Quality (B, 3) | 1059 | 7 | 2.05 | 37.92 | 6.29 | 1.31 | 4.87 | 8.64 | 29.56 | **0.80** | 2.22 |
| Breast tissue (C, 6) | 105 | 9 | 36.37 | 47.23 | 38.74 | 42.66 | 57.69 | 59.23 | **33.63** | 39.23 | 55.26 |
| Penguin (B, 3) | 333 | 9 | 6.61 | 12.47 | 3.96 | 6.50 | 7.25 | 14.12 | **0.63** | 0.97 | 2.14 |
| Wine (A, 3) | 177 | 13 | 12.53 | 18.49 | 10.10 | 12.00 | 18.86 | 18.56 | **2.27** | 4.42 | 7.02 |
| Mobile Price (B, 4) | 2000 | 20 | 20.60 | 46.89 | 21.91 | 17.34 | 64.98 | 62.51 | 16.14 | 4.16 | **3.89** |
| Waveform (C, 3) | 4999 | 21 | 26.52 | 31.89 | 26.62 | 26.27 | 28.68 | 27.96 | 22.28 | 21.06 | **18.22** |
| Gamma telescope (A,2) | 579 | 10 | 32.01 | 33.90 | 30.41 | **29.64** | 30.33 | 30.65 | 37.30 | 33.50 | 32.24 |
| Indian liver patient (A,2) | 19020 | 10 | **18.19** | 26.44 | 19.50 | 19.21 | 22.00 | 21.45 | 20.92 | 21.02 | 18.83 |
| Heart disease (A,2) | 270 | 13 | 21.91 | 28.38 | 23.04 | 25.28 | 25.39 | 25.22 | **16.20** | 23.34 | 20.61 |
| EEG eye state (A,2) | 14980 | 14 | 29.98 | 36.15 | 30.90 | 36.27 | 34.36 | 33.29 | 38.19 | 27.44 | **25.76** |
| Seismic-bumps (A,2) | 2584 | 15 | 7.61 | 9.60 | 6.64 | **6.62** | 6.93 | 6.88 | 18.35 | 10.43 | 8.34 |
| Retinopathy debrecen (A,2) | 1151 | 19 | 35.89 | 40.55 | 35.43 | 37.03 | 38.25 | 35.40 | **29.37** | 32.71 | 30.20 |
| Parkinson multiple sound (B,2) | 1208 | 26 | **34.39** | 38.69 | 34.81 | 37.10 | 36.49 | 35.74 | 35.23 | 36.84 | 34.88 |
| Pistachio (B,2) | 2148 | 28 | 13.73 | 18.10 | 14.20 | 14.38 | 16.64 | 15.38 | 11.75 | 12.16 | **10.55** |
| Breast cancer (B,2) | 569 | 30 | 7.07 | 9.28 | 6.62 | 6.34 | 8.55 | 8.23 | 4.41 | 6.16 | **3.91** |
| Ionosphere (A,2) | 351 | 33 | 12.40 | 18.38 | 11.14 | **10.22** | 17.01 | 16.27 | 13.92 | 16.26 | 12.21 |
| QSAR biodegradation (A,2) | 1055 | 41 | 17.88 | 20.61 | 18.10 | 19.65 | 20.41 | 19.41 | **15.52** | 19.64 | 17.14 |
| Mice protein expression (A,2) | 1047 | 70 | 13.60 | 18.31 | 16.80 | 13.93 | 20.68 | 19.03 | **3.58** | 4.62 | 6.03 |
| Ozone level detection (A,2) | 1847 | 72 | 8.16 | 9.94 | **7.00** | 7.51 | 8.15 | 8.28 | 19.40 | 11.85 | 8.43 |
| Hill valley (C,2) | 1212 | 100 | 48.22 | 46.07 | 48.43 | 51.41 | 41.56 | 22.53 | 29.90 | 13.84 | **0.04** |
| Musk (A,2) | 6598 | 166 | 8.61 | 12.02 | 11.48 | 9.39 | 13.18 | 11.49 | **7.73** | 10.96 | 9.34 |
| Average MR (%) across all 23 datasets | | | 18.74 | 26.05 | 19.14 | 19.50 | 23.93 | 23.03 | 17.97 | 15.69 | 14.78 |
| No. of bests in 23 datasets | | | 2 | 0 | 1 | 3 | 0 | 0 | 10 | 1 | 6 |

**Table 3:** Classification: average MR (%) based on 100 random partitions of each data set into training and test sets. The alphabetic notation in the brackets of each dataset represents the data source; and the numbers are the number of classes.

We compare two different random rotation oblique tree methods: the Random Rotation Random Forest (RotRF) proposed by Blaser and Fryzlewicz (2016), and Sparse Projection Oblique Randomer Forests (SPORF) proposed by Tomita et al. (2020). Note that the trees in SPORF and RotRF are all random, but we still use one of the trees, denoted by RotT and SPOT respectively, for the comparison of different trees. In addition, there are two oblique decision tree methods used for classification: projection pursuit classification and regression trees (PPT) of Lee (2018); Cho and Lee (2021) with **PPtreeViz** and **PPtreereg** packages; and Oblique Trees for Classification Data (OT) of Truong (2009) with function `oblique.tree()` in **oblique.tree** package. We also compare four axis-aligned tree methods, including CART with function `rpart()` in **rpart** package, ERT with function `RLT()` in **RLT** package, EVT with function `evtree()` in **evtree** package, and CT with function `ctree()` in **partykit** package. For all the R functions, their default values of tuning parameters are used. Note that we only report the classification results because OT cannot be used for regression. For the forests, the competitors include RotRF with function `rotationForest()` in **rotationForest** package, SPORF with function `RerF()` in **rerf** package, RF with function `randomForest()` in **randomForest** package, GRF with functions `regression_forest()` and `Classification_forest()` in package **grf**, extreme gradient boosting (XGB) of Chen and Guestrin (2016) with function `xgboost()` in **xgboost** package, ORF with function `obliqueRF()` in **obliqueRF** package, ORSF with function `orsf()` in **aorsf** package, and PPF with function `PPforest()` in **PPforest** package. We used the default tuning parameter values for all packages; however, we used 100 trees for the ensemble methods. All datasets and codes used in the paper are publicly accessible at `https://github.com/liuyu-star/RJ_ODRF_Codes`. The results presented in the paper can be reproduced by running R script `"ODRF.R"`.

For each data set, we randomly partition it into a training set and a test set. The training set consists of $n = \min(\lfloor 2N/3 \rfloor, 1000)$ randomly selected observations, where $N$ is the number of observations in the original data sets, and the remaining observations form the test set. For regression, the relative prediction error defined as

$$RPE = \sum_{i \in \text{test set}} (\hat{y}_i - y_i)^2 / \sum_{i \in \text{test set}} (\bar{y}_{\text{train}} - y_i)^2,$$

where $\bar{y}_{\text{train}}$ is a naive prediction based on the average of $y$ in the training sets, is used to measure the performance of a method. For classification, the misclassification rate, defined as

$$MR = \sum_{i \in \text{test set}} 1(\hat{y}_i \neq y_i)/(N - n),$$

is used to measure the performance. For each data set, the random partition is repeated 100 times, and averages of the RPEs or MRs are calculated using different methods. The calculation results are listed in Table 2 and Table 3. The smallest RPE or MR for each data set is highlighted in **bold** font.

By comparing prediction errors across tasks, our Oblique Decision Tree (ODT) method shows strong performance in regression, with generally lower Relative Prediction Error (RPE) than competing methods. For classification tasks, ODT achieves competitive misclassification rates (MR) on certain datasets, though its performance varies relative to other approaches such as PPT. ODT consistently demonstrates stability across all data sets, as shown in Table 2 and Table 3. In contrast, other oblique trees may fail; for instance, RotT and SPOT fail in the body fat data and Paris housing price data, while all other oblique trees fail in the Hill Valley data. The efficacy of ODT is further supported by having the lowest average RPEs (or MRs) across all data sets among all methods. Here, the term "no. of bests" refers to the number of data sets in which a particular method outperforms all competitors.

| Method | | Classification | | | Regression | | |
|---|---|---|---|---|---|---|---|
| | | MR (%) | Time | Complexity | RPE | Time | Complexity |
| Axis-aligned tree | CART | 21.32 (0) | 0.11 (0) | 11.40 (0) | 0.241 (0) | 0.08 (1) | 8.20 (9) |
| | ERT | 23.05 (0) | 0.00 (13) | 91.67 (0) | 0.514 (0) | 0.00 (18) | 185.90 (0) |
| | EVT | 20.24 (2) | 0.57 (0) | 6.33 (1) | 0.262 (0) | 0.78 (0) | 10.35 (10) |
| | CT | 21.75 (2) | 0.16 (0) | 9.00 (1) | 0.224 (7) | 0.42 (0) | 24.45 (1) |
| Oblique tree | RotT | 15.49 (4) | 0.13 (1) | 9.53 (0) | - | - | - |
| | SPOT | 20.70 (0) | 0.05 (1) | 63.20 (0) | - | - | - |
| | OT | 18.41 (2) | 11.19 (0) | 25.60 (2) | - | - | - |
| | PPT | 21.39 (1) | 0.16 (0) | 2.00 (11) | 0.274 (1) | 1.580(0) | 14.45(1) |
| | ODT | 16.36 (4) | 0.14 (0) | 14.13 (0) | 0.210 (13) | 0.32 (1) | 40.30 (0) |
| Axis-aligned forest | RF | 16.60 (1) | 0.14 (1) | - | 0.156 (2) | 0.59 (1) | - |
| | GRF | 17.97 (2) | 0.06 (12) | - | 0.196 (0) | 0.03 (17) | - |
| | ERT | 16.62 (0) | 0.17 (2) | - | 0.191 (1) | 0.24 (2) | - |
| | XGB | 18.70 (0) | 0.17 (0) | - | 0.142 (9) | 0.35 (0) | - |
| Oblique forest | RotRF | 13.43 (1) | 10.24 (0) | - | - | - | - |
| | SPORF | 13.29 (3) | 2.71 (0) | - | - | - | - |
| | PPF | 21.27 (2) | 1.16 (0) | - | - | - | - |
| | ORF | 12.60 (3) | 11.84 (0) | - | - | - | - |
| | ORSF | 13.02 (0) | 2.51 (0) | - | 0.178 (3) | 0.14 (0) | - |
| | ODRF | 12.90 (3) | 9.90 (0) | - | 0.142 (8) | 38.86 (0) | - |

**Table 4:** Performance comparison of different methods for classification and regression. The numbers in the brackets are the number of datasets in which a method performs best among all competitors, and "–" means the method is not applicable.

We also provide a comprehensive summary of three performance aspects for all competing methods: prediction error (MR for classification tasks or RPE for regression tasks), calculation time (Time), and the number of terminal nodes (Complexity). For $n$ training samples and $p$ predictors, the ODT algorithm demonstrates higher computational complexity in the training phase compared to CART (which typically has $O(p \cdot n \log n)$ complexity) due to the overhead of linear optimizations, such as covariance matrix estimation and eigenvalue decomposition, though heuristic approaches like depth constraints can mitigate this cost. In the prediction phase, ODT requires $O(p \cdot \log n)$ time per sample, as each node involves a linear combination computation, whereas CART achieves $O(\log n)$

time by relying on single-variable splits without linear operations.

Due to rotationForest and obliqueRF packages only supporting binary classification problems, Table 4 presents the average values for these three aspects based on 15 binary classification datasets and 20 regression datasets. As shown in Table 4, after removing the five multinomial datasets, our ODT and ODRF generally exhibit smaller errors compared to their competitors. For ODT, its complexity and calculation time are approximately average among the other methods, yet it demonstrates reduced errors. ODRF significantly reduces prediction error compared to its competitors, as well as other oblique forests including ODT. However, the calculation time for ODRF is considerably longer than that of traditional random forests, while being comparable to other oblique forests.

Finally, we evaluated the performance of ODRF and other methods under three noise conditions using six classification datasets, followed by a comparative analysis without noise conditions. Specifically: irrelevant features noise was applied to the Iris and Penguin datasets; label noise was applied to the Patient and Retinopathy datasets; correlated features noise was applied to the QSAR and Musk datasets. The three noise conditions are defined as follows:

- **Add label noise** Let $y \in \{0,1\}^n$ be a binary label vector, and $\epsilon \in [0,1]$ be the noise rate (In the computation, it is set to 0.1). Randomly select a subset of indices $S \subseteq \{1,2,\ldots,n\}$ of size $m = \lfloor n\epsilon \rfloor$. Then, the noisy label vector $y'$ is defined as:

$$y'_i = \begin{cases} 1 - y_i & \text{if } i \in S \\ y_i & \text{otherwise} \end{cases}$$

  where $S$ is uniformly randomly sampled from $\{1,\ldots,n\}$ and $|S| = m$.

- **Add irrelevant features noise.** Let $X \in \mathbb{R}^{n\times p}$ be the original feature matrix. Add $q \in \mathbb{N}$ (In the computation, it is set to 10) irrelevant features by generating a random matrix $Z \in \mathbb{R}^{n\times q}$, where each element $Z_{ij} \sim \mathcal{N}(0,1)$ is independently and identically distributed (i.i.d.). The augmented feature matrix is:

$$X' = \begin{bmatrix} X & Z \end{bmatrix}$$

- **Add correlated features noise.** Let $X \in \mathbb{R}^{n\times p}$ be the original feature matrix, and $x^{(k)} \in \mathbb{R}^n$ be its $k$-th column vector. Given a target correlation coefficient $\rho \in [-1,1]$ (In the computation, $k = 1$ and $\rho = 0.9$), add a new feature $z \in \mathbb{R}^n$ such that:

$$z = \rho x^{(k)} + \delta, \quad \delta \sim \mathcal{N}(0, \sigma^2), \quad \sigma^2 = 1 - \rho^2$$

  where $\delta$ is independent of $x^{(k)}$. The augmented feature matrix is:

$$X' = \begin{bmatrix} X & z \end{bmatrix}$$

  In our computational experiments, noise was introduced to only one correlated feature. Alternatively, the introduction of noise to multiple correlated features could be considered for further analysis.

We have computed the corresponding results as detailed in Table 5. It can be observed that our ODRF is less sensitive to noisy data compared to other methods and even demonstrates enhanced performance in the presence of noise, as illustrated by the Patient and QSAR datasets.

| | | | Without noise | | | | | With noise | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | n | p | CART | SPOT | PPT | OT | ODT | CART | SPOT | PPT | OT | ODT |
| Iris (A, 3) | 149 | 4 | 6.96 | 14.74 | 2.66 | 4.24 | 5.76 | 6.96 | 31.42 | 3.90 | 7.24 | 6.18 |
| Penguin (B, 3) | 333 | 9 | 6.61 | 13.18 | 0.63 | 0.97 | 2.11 | 6.61 | 15.29 | 0.70 | 1.39 | 2.59 |
| Patient (A,2) | 19020 | 10 | 18.19 | 21.44 | 20.92 | 21.05 | 25.46 | 28.15 | 27.11 | 32.03 | 27.76 | 18.94 |
| Retinopathy (A,2) | 1151 | 19 | 35.89 | 35.35 | 29.37 | 32.76 | 29.88 | 39.51 | 39.61 | 33.90 | 39.59 | 36.57 |
| QSAR (A,2) | 1055 | 41 | 17.89 | 19.52 | 15.52 | 19.69 | 17.11 | 18.00 | 19.64 | 15.62 | 19.92 | 16.99 |
| Musk (A,2) | 6598 | 166 | 8.61 | 11.71 | 7.73 | 10.96 | 9.53 | 8.60 | 11.53 | 7.72 | 11.07 | 9.67 |

**Table 5:** Classification: average MR (%) based on 100 random partitions of each data set into training and test sets. The alphabetic notation in the brackets of each dataset represents the data source; and the numbers are the number of classes.

## 5 Conclusion

The Oblique Decision Tree (ODT) has been touted as superior to the conventional CART, but it has lacked theoretical justification, and its numerical benefits have not been supported by existing packages. Our recent work (Zhan et al., 2025) provides the first theoretical proof of ODT's superiority over CART for a wider range of regression functions, as long as they are continuous, ensuring consistency; similarly, ODRF demonstrates advantages over conventional Random Forests. Building on this theoretical insight, we developed the **ODRF** package and found that, as demonstrated in this paper, ODT and ODRF indeed deliver more accurate predictions with fewer leaves, resulting in lower model complexity and enhanced interpretability for data analysis. Furthermore, we propose an enhanced ODT-based boosting ensemble (ODBT) whose performance parallels ODRF, with detailed computational proofs available in our recent work.

Despite these advantages, ODRF and ODT may not perform optimally in certain scenarios. For instance, they can struggle with small datasets due to insufficient data for effective oblique splits, exhibit reduced efficiency with highly redundant features that dilute the impact of linear combinations, and face challenges with special data structures (e.g., highly imbalanced classes or non-linear relationships not captured by oblique projections). Additionally, we acknowledge limitations in the current implementation. The package experiences performance bottlenecks with large-scale datasets due to the computational overhead of oblique splits, and dependency management requires refinement to ensure robust installation across different environments. The current work also lacks certain preprocessing capabilities and needs expansion to handle diverse data types, such as automated feature engineering or support for complex data formats. Future work will address these limitations by optimizing computational efficiency, enhancing preprocessing features, and extending support for challenging data conditions.

## 6 Acknowledgments

We are grateful to the Editor-in-Chief, Executive Editor and three referees for their meticulous review and valuable comments. Yu Liu is supported by the National Natural Science Foundation of China (12501377) and Sichuan Science and Technology Program (2026NSFSC0788). Yingcun Xia is supported by the National Natural Science Foundation of China (72033002 and 12271081).

## References

S. Athey, J. Tibshirani, and S. Wager. Generalized random forests. *The Annals of Statistics*, 47(2): 1148–1178, 2019. [p15]

R. Blaser and P. Fryzlewicz. Random rotation ensembles. *The Journal of Machine Learning Research*, 17 (1):126–151, 2016. [p15, 30]

L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. [p15]

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Taylor & Francis, Boca Raton, 1984. [p15]

J. Browne and T. Tomita. *rerf: Randomer Forest*, 2019. URL https://CRAN.R-project.org/package=rerf. R package version 2.0.4. [p16]

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016. [p16, 30]

H. Cho and E.-K. Lee. Tree-structured regression model using a projection pursuit approach. *Applied Sciences*, 11(21), 2021. [p30]

E. Craig, M. Pilanci, T. L. Menestrel, B. Narasimhan, M. Rivas, S.-E. Gullaksen, R. Dehghannasiri, J. Salzman, J. Taylor, and R. Tibshirani. Pretraining and the lasso. *arXiv preprint arXiv:2401.12911*, 2024. [p22]

N. da Silva, D. Cook, and E.-K. Lee. A projection pursuit forest algorithm for supervised classification. *Journal of Computational and Graphical Statistics*, 30(4):1168–1180, 2021. [p15, 16]

N. da Silva, D. Cook, and E.-K. Lee. *PPforest: Projection Pursuit Classification Forest*, 2025. URL https://CRAN.R-project.org/package=PPforest. R package version 0.2.0. [p16]

D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. [p20]

M. Fokkema, J. Edbrooke-Childs, and M. Wolpert. Generalized linear mixed-model (glmm) trees: A flexible decision-tree method for multilevel and longitudinal data. *Psychotherapy Research*, 1(22): 1–13, 2020. [p15]

J. H. Friedman. A variable span smoother. Technical report, Stanford Univ CA lab for computational statistics, 1984. [p18]

J. H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001. [p15, 19]

J. H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002. [p15]

J. H. Friedman and W. Stuetzle. Projection pursuit regression. *Journal of the American statistical Association*, 76(376):817–823, 1981. [p16, 18, 25]

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006. [p15]

L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on tabular data?, 2022. [p15]

Grubinger, Zeileis, Pfeiffer, and KP. evtree: Evolutionary learning of globally optimal classification and regression trees in R. *Journal of Statistical Software*, 2014. [p15]

D. Heath, S. Kasif, and S. Salzberg. Induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2(2):1–32, 1993. [p15]

H. V. Henderson and P. F. Velleman. Building multiple regression models interactively. *Biometrics*, pages 391–411, 1981. [p28]

T. K. Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998. [p18]

T. Hothorn and A. Zeileis. Partykit: a modular toolkit for recursive partytioning in R. *The Journal of Machine Learning Research*, 2015. [p20]

T. Hothorn, K. Hornik, and A. Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical statistics*, 15(3):651–674, 2006. [p15]

B. C. Jaeger, S. Welden, K. Lenoir, J. L. Speiser, M. W. Segar, A. Pandey, and N. M. Pajewski. Accelerated and interpretable oblique random survival forests. *arXiv preprint arXiv:2208.01129*, 2022. [p15]

R. Johnson and Z. Tong. Learning nonlinear functions using regularized greedy forest. *IEEE Trans Pattern Anal Mach Intell*, 36(5):942–954, 2014. [p15]

R. Katuwal, P. N. Suganthan, and L. Zhang. Heterogeneous oblique random forest. *Pattern Recognition*, 99:107078, 2020. [p15]

G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: a highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3149–3157. Curran Associates Inc., 2017. [p16]

D. Lee, M.-H. Yang, and S. Oh. Fast and accurate head pose estimation via random projection forests. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1958–1966, 2015. [p15, 16]

E.-K. Lee. PPtreeViz: An r package for visualizing projection pursuit classification trees. *Journal of Statistical Software*, 83:1–30, 2018. [p30]

A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002a. URL https://CRAN.R-project.org/doc/Rnews/. [p16]

A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002b. [p16]

M. Maia, K. Murphy, and A. C. Parnell. GP-BART: a novel bayesian additive regression trees approach using gaussian processes. *arXiv preprint arXiv:2204.02112*, 2022. [p15]

B. Menze and N. Splitthoff. *obliqueRF: Oblique Random Forests from Recursive Linear Model Splits*, 2012. URL https://CRAN.R-project.org/package=obliqueRF. R package version 0.3. [p16]

B. H. Menze, B. M. Kelm, D. N. Splitthoff, U. Koethe, and F. A. Hamprecht. On oblique random forests. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 453–469. Springer, 2011. [p15, 16]

L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. Catboost: unbiased boosting with categorical features. *Advances in neural information processing systems*, 31, 2018. [p16]

J. R. Quinlan. Decision trees as probabilistic classifiers. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 31–37. Elsevier, 1987. [p15]

J. R. Quinlan. Program for machine learning. C4. 5, 1993. [p15]

T. Rainforth and F. Wood. Canonical correlation forests. *arXiv preprint arXiv:1507.05444*, 2015. [p15, 29]

T. Therneau and B. Atkinson. *rpart: Recursive Partitioning and Regression Trees*, 2022. URL https://CRAN.R-project.org/package=rpart. R package version 4.1.19. [p16]

T. M. Therneau and E. J. Atkinson. An introduction to recursive partitioning using the rpart routines. *Rochester Mayo Foundation*, 2000. [p16]

T. M. Tomita, J. Browne, C. Shen, J. Chung, J. L. Patsolic, B. Falk, C. E. Priebe, J. Yim, R. Burns, M. Maggioni, et al. Sparse projection oblique randomer forests. *Journal of machine learning research*, 21(104), 2020. [p15, 16, 30]

A. K. Y. Truong. *Fast growing and interpretable oblique trees via logistic regression models*. PhD thesis, Oxford University, UK, 2009. [p30]

D. Wickramarachchi, B. Robertson, M. Reale, C. Price, and J. Brown. Hhcart: An oblique decision tree. *Computational Statistics & Data Analysis*, 96:12–23, 2016. [p15]

D. C. Wickramarachchi, B. L. Robertson, M. Reale, C. J. Price, and J. A. Brown. A reflected feature space for cart. *Australian & New Zealand Journal of Statistics*, 61(3):380–391, 2019. [p15]

A. Zeileis and T. Hothorn. Parties, models, mobsters: A new implementation of model-based recursive partitioning in R, 2015. [p15]

H. Zhan, Y. Liu, and Y. Xia. Consistency of oblique decision tree and its boosting and random forest. *Bernoulli*, 2025. [p15, 16, 33]

*Yu Liu*
*Sichuan Normal University*
*School of Mathematical Sciences*
*China*
liuy8stat@sicnu.edu.cn

*Yingcun Xia*
*University of Electronic Science and Technology of China*
*School of Mathematical Sciences*
*China*
yingcun.xia@uestc.edu.cn

# SelectionBias: An R Package for Bounding Selection Bias in Causal Estimands

*by Stina Zetterstrom and Ingeborg Waernbaum*

**Abstract** Selection bias may arise when there are dropouts or missing data in the analysis, or when subjects are included or excluded in the analysis based upon some selection criteria for the study population. Selection bias can jeopardize the validity of the study and a sensitivity analysis for assessing the effect of the selection is desired. Recently, there has been a surge of results for selection bias in causal inference, with several suggestions for sensitivity analyses. One method is to construct bounds for the bias. Here, we present the R package SelectionBias that can be used to calculate previously proposed bounds for selection bias for the causal risk ratio and causal risk difference for both the total and the selected populations. The first bound, derived by Smith and VanderWeele (SV), is based on values of sensitivity parameters that describe parts of the joint distribution of the outcome, treatment, selection indicator and unobserved variables. The second bound is an improved sharp bound that uses the same sensitivity parameters as the SV bound. The third bound is based solely on the observed data, and is therefore referred to as the assumption-free (AF) bound. The fourth and fifth bounds, the generalized assumption-free (GAF) and counterfactual assumption-free (CAF) bounds, utilize both the data and sensitivity parameters. The R package is illustrated with a simulated dataset that emulates a study where the effect of the zika virus on microcephaly in Brazil is investigated. Lastly, its performance and features are compared to the already existing R package EValue, highlighting situations where the two packages provide distinct advantages over each other.

## 1 Introduction

Selecting a study population from a larger source population is a common procedure, for example in an observational study with data from a population register. Based on the research question, the study population is commonly constructed using inclusion/exclusion criteria. The construction of the study population from these selection criteria might introduce bias when estimating the causal effect of the treatment on the outcome of interest (Hernán et al., 2004). This systematic error is commonly referred to as selection bias and can occur both when interest lies in the total population and in the selected subpopulation. Selection bias can also arise if, for example, there are dropouts or other missing values for some individuals in the study (Lu et al., 2022). In an empirical study, it is often of interest to assess the magnitude of potential biases using a sensitivity analysis. One type of sensitivity analysis is calculating bounds for the bias, see e.g. Huang and Lee (2015), Greenland (2003), Flanders and Ye (2019), Smith and VanderWeele (2019), Zetterstrom and Waernbaum (2022), Zetterstrom (2024), Zetterstrom et al. (2025), and Duarte et al. (2024) for selection bias.

This paper presents the R package **SelectionBias** which can be used to calculate different bounds for selection bias. The ability to calculate different bounds is attractive as these bounds are in many cases based on assumptions that may or may not be fulfilled. One of the bounds included in **SelectionBias** is the bound presented in Smith and VanderWeele (2019), hereafter referred to as the SV bound. This bound is already included in the R package **EValue** and can also be calculated using an online calculator (Smith and VanderWeele, 2019; Smith et al., 2021). For these software tools, the user inputs assumed values of sensitivity parameters, which are summary measures of a joint distribution. Thus, these sensitivity parameters can be difficult to specify, especially in the presence of many selection variables. In **SelectionBias**, we also allow the user to input the model, which might be easier to specify than the sensitivity parameters. **SelectionBias** also allows the user to assess whether the SV bound is sharp. The sharpness of a bound is a desirable property, and recently proposed sharp bounds based on the same sensitivity parameters as the SV bound (Zetterstrom et al., 2025) are also included in **SelectionBias**. These bounds are referred to as the sharp bounds. **SelectionBias** also includes functions for calculating the AF, GAF, and CAF bounds (Zetterstrom and Waernbaum, 2022; Zetterstrom, 2024). The sharp, AF, GAF, and CAF bounds cannot be calculated using other packages available online. Notably, all the bounds in the **SelectionBias** package are calculated within strata of the pre-treatment covariates. The contents of the **SelectionBias** package is:

- `zika_learner`: a simulated dataset inspired by the study in de Araújo et al. (2018) and the zika example in Smith and VanderWeele (2019). The dataset includes seven variables: zika virus, microcephaly, the selection variables birth and public hospital, the selection indicator variable and the two "unmeasured" variables living area and socioeconomic status.
- `sensitivityparametersM()`: a function that calculates the sensitivity parameters for the SV, sharp, and GAF bounds for a model following the generalized M-structure (Figure 1) defined

by the user.

- `SVbound()`: a function that calculates the SV lower and upper bound for the risk ratio or risk difference in either the total or subpopulation for sensitivity parameters given by the user, or calculated from `sensitivityparametersM()`.
- `sharpbound()`: a function that calculates the sharp lower and upper bound for the risk ratio or risk difference in either the total or subpopulation for sensitivity parameters given by the user, or calculated from `sensitivityparametersM()`.
- `AFbound()`: a function that calculates the AF lower and upper bounds for the risk ratio or risk difference in either the total or subpopulation. The input can either be a dataset or probabilities calculated from the data.
- `GAFbound()`: a function that calculates the GAF lower and upper bounds for the risk ratio or risk difference in either the total or subpopulation for a dataset or probabilities from the data, together with sensitivity parameters either given by the user, or calculated from `sensitivityparametersM()`.
- `CAFbound()`: a function that calculates the CAF lower and upper bounds for the risk ratio or risk difference in either the total or subpopulation for a dataset or probabilities from the data, together with sensitivity parameters given by the user.
- `checksharpSVbound()`: a function that evaluates if the SV bound is sharp.

The paper is outlined as follows. In Section 2, we present the causal framework, estimands and corresponding selection biases together with an introduction of the bounds. In Section 3, the simulated example dataset `zika_learner` is described. In Section 4, the R package **SelectionBias** is demonstrated using the `zika_learner` data. The performance and features of **SelectionBias** are compared to other R packages for sensitivity analyses, especially **EValue**, in Section 5. Finally, the paper is summarized in Section 6.

## 2 Causal framework and selection bias

In this section, we provide a theoretical background for the bounds calculated in the R package **SelectionBias**. First, we introduce the notation and causal framework, and second, we describe the SV, sharp, AF, GAF, and CAF bounds.

We assume $K$ binary selection variables, $S_1, \ldots, S_k, \ldots, S_K$, where $S_k$ indicates if the subject passes the $k$th selection criterion, from which we define a selection indicator variable $I_S$ such that

$$I_S = \begin{cases} 1 & \text{if } \prod_{k=1}^{K} S_k = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

For a subject to be included in the study, the corresponding $I_S$ must be equal to 1. We consider an i.i.d. sample of size $i = 1, \ldots, n$ units from the subpopulation such that $I_S = 1$, but henceforth suppress the index $i$ representing units in the sample. We also assume a binary treatment, $T = 1$ if the unit is treated and $T = 0$ if the unit is not treated, and two corresponding binary potential outcomes, $Y(1)$ and $Y(0)$ as well as an observed outcome such that $Y = TY(1) + (1-T)Y(0)$, i.e. consistency is assumed (Rubin, 1974). Additionally, we assume a vector of observed pre-treatment covariates, denoted by $X$, such that conditional exchangeability holds in the total population: $Y(t) \perp\!\!\!\perp T \mid X$, $t = 0, 1$. However, similar to the previous literature, we suppress $X$ throughout to make the notation simpler and assume that all calculations are performed within strata of the pre-treatment covariates. Lastly, we define a vector of unobserved pre-treatment covariates, $U$, which is part of the assumptions in the sensitivity analysis with the SV, sharp, and GAF bounds. It is illustrated in the generalized M-structure (Figure 1), where $U$ is a predictor of the outcome. In contrast, the unobserved covariates $U$ are not needed in the AF and CAF bounds, which impose fewer assumptions since dependencies with $U$ are not needed.

The causal estimands of interest are the causal risk ratio, $\beta_R = P(Y(1) = 1)/P(Y(0) = 1)$, and causal risk difference, $\beta_D = P(Y(1) = 1) - P(Y(0) = 1)$, in the total population, and the causal risk ratio, $\beta_{R_s} = P(Y(1) = 1|I_S = 1)/P(Y(0) = 1|I_S = 1)$, and the causal risk difference, $\beta_{D_s} = P(Y(1) = 1|I_S = 1) - P(Y(0) = 1|I_S = 1)$, in the subpopulation. We also define the observed risk ratio and risk difference under selection, $I_S = 1$, as $\beta_R^{obs} = P(Y = 1|T = 1, I_S = 1)/P(Y = 1|T = 0, I_S = 1)$ and $\beta_D^{obs} = P(Y = 1|T = 1, I_S = 1) - P(Y = 1|T = 0, I_S = 1)$. These are estimated from data, but we assume that there is no sampling variability and that $\beta_R^{obs}$ and $\beta_D^{obs}$ are population quantities. The selection bias for the risk ratio estimands, $\beta_R$ and $\beta_{R_s}$, are defined as the ratio between the risk ratio and the causal estimand,

$$Bias(\beta_R) = \beta_R^{obs}/\beta_R, \tag{2}$$

$$Bias(\beta_{R_s}) = \beta_R^{obs}/\beta_{R_s}, \tag{3}$$

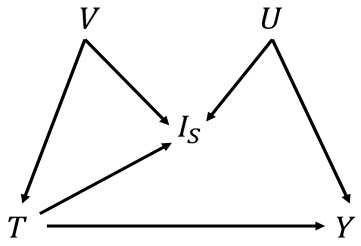



**Figure 1:** The generalized M-structure relating treatment, outcome, and selection to two unmeasured variables. Nodes represent the treatment ($T$), outcome ($Y$), selection indicator ($I_S$), and two unmeasured pre-treatment variables ($V$ and $U$); arrows represent possible direct effects between the connected variables, with $V$ pointing to both $T$ and $I_S$, and $U$ pointing to both $I_S$ and $Y$. This structure satisfies the conditional independence assumptions required for the SV, sharp, and GAF bounds, since $Y$ does not have a direct path to $I_S$ that bypasses the other variables.

and the selection bias for the risk difference estimands, $\beta_D$ and $\beta_{D_s}$, are defined as the difference between the observed risk difference and the causal estimand,

$$Bias(\beta_D) = \beta_D^{obs} - \beta_D, \tag{4}$$

$$Bias(\beta_{D_s}) = \beta_D^{obs} - \beta_{D_s}. \tag{5}$$

When the observed risk ratio or risk difference do not coincide with the causal estimands, i.e. when selection bias is present, bounds for the causal estimands can be calculated and compared to the observed quantity in order to assess the magnitude of the selection bias. We present a package that can be used to perform sensitivity analysis for the selection bias, under various assumptions. Bounds from the following works are considered; Smith and VanderWeele (2019), Zetterstrom and Waernbaum (2022), Zetterstrom (2024), and Zetterstrom et al. (2025). The ideas behind the bounds are shortly described here with a summary of their properties in Table 1. For completeness, all bounds are presented in the Appendix.

The SV bounds rely on two versions of conditional independence assumptions involving the outcome, selection indicator, treatment, and a vector of unmeasured variables (see Appendix). The assumptions differ depending on the estimand of interest. If total population estimands $\beta_R$ and $\beta_D$ are of interest, conditioning on $U$ and $T$ must be sufficient for conditional independence of the observed outcome $Y$ and the selection indicator, $I_S$. For the case when subpopulation estimands $\beta_{R_S}$ and $\beta_{D_S}$ are considered, an assumption requiring exchangeability conditional on $U$ and $I_S$ is instead made. An example where both assumptions are fulfilled is the M-structure, Figure 1, although other structures are possible. The purpose of the different assumptions is to describe a setting where unbiased estimates could be obtained if the unmeasured variables were observed. The SV bounds are based on sensitivity parameters that are constructed as risk ratios formed by the joint distribution of the outcome, treatment, selection indicator variable and unmeasured variables, $(Y, T, I_S, U)$. The sensitivity parameters are

$$\mathrm{RR}_{SU|ts} = \max_u \frac{p(U = u \mid I_S = s, T = t)}{p(U = u \mid I_S = 1 - s, T = t)},$$

$$\mathrm{RR}_{UY|t} = \frac{\max_u p(Y = 1 \mid U = u, T = t)}{\min_u p(Y = 1 \mid U = u, T = t)},$$

and

$$\mathrm{BF}_{ts} = \frac{\mathrm{RR}_{SU|ts} \times \mathrm{RR}_{UY|t}}{\mathrm{RR}_{SU|ts} + \mathrm{RR}_{UY|t} - 1},$$

from which bounds for the causal risk ratio and causal risk difference can be calculated. For the subpopulation, SV defined the sensitivity parameters

$$\mathrm{RR}_{TU|t} = \max_u \frac{p(U = u \mid I_S = 1, T = t)}{p(U = u \mid I_S = 1, T = 1 - t)},$$

$$\mathrm{RR}_{UY|S=1} = \max_t \frac{\max_u p(Y = 1 \mid U = u, T = t, I_S = 1)}{\min_u p(Y = 1 \mid U = u, T = t, I_S = 1)},$$

and

$$\text{BF}_t = \frac{\text{RR}_{TU|t} \times \text{RR}_{UY|S=1}}{\text{RR}_{TU|t} + \text{RR}_{UY|S=1} - 1}.$$

More details of the sensitivity parameters and bounds can be found in the original work. It may be difficult to interpret and specify the sensitivity parameters, especially in the presence of multiple selection variables and multiple unmeasured variables. Instead, it may be easier to break down the bound into smaller pieces, and calculate the sensitivity parameters for a specified data generating process (DGP). We facilitate such calculations in the R package **SelectionBias**.

A bound is sharp if the bias can equal the value of the bound, given the necessary assumptions, an observed distribution, and correctly specified sensitivity parameters. Since different bounds require different assumptions and information, two bounds for the same causal estimand can be simultaneously sharp if each is valid under its own requirements. The SV bounds are only sharp under certain conditions, i.e. sometimes the SV bounds are too conservative and are larger than the bias can actually be. As an alternative, the sensitivity parameters have been used to construct improved bounds that are generally sharp (Zetterstrom et al., 2025), referred to as the sharp bounds. However, it should be noted that they do in some cases require some additional knowledge of the selection probability.

The AF bounds are constructed solely from the data and have the advantage that no additional assumptions regarding the causal model are needed, only that $P(I_S = 1)$ is known. The bounds are constructed as the minimum and maximum possible values of the causal estimand, $\beta^{min}$ and $\beta^{max}$. These values are then used as lower and upper bounds for the causal estimand. As the bounds also demonstrate the maximum selection bias, they can be used to assess whether other bounds are informative: if another bound is greater than the AF bound, it is not informative. However, if the treatment or outcome is rare, the AF bounds for the risk ratios can be very wide, and practically non-informative. When this is the case, and additional knowledge is available, it is instead advisable to use other bounds that take that knowledge into account.

The GAF bounds are similar to the AF bounds in so far as they also utilize the observed data, but they also rely on the same conditional independence assumptions as the SV bound (see Appendix), which results in tighter bounds compared to the AF bounds. The sensitivity parameters are probabilities instead of risk ratios, which can in some instances be easier to specify. The sensitivity parameters are defined as

$$m_T = \min_{t,u} P(Y = 1|T = t, U = u)$$

and

$$M_T = \max_{t,u} P(Y = 1|T = t, U = u)$$

for the total population, and

$$m_S = \min_{t,u} P(Y = 1|T = t, U = u, S = 1)$$

and

$$M_S = \max_{t,u} P(Y = 1|T = t, U = u, S = 1)$$

for the subpopulation.

The CAF bounds are an alternative when the conditional independence assumptions are not fulfilled, but they require the user to specify counterfactual probabilities, $P(Y(t) = 1|T = 1 - t, I_S = 1)$. The sensitivity parameters are defined as

$$m'_T = \min_{t} P(Y = 1|T = t, I_S = 0)$$

and

$$M'_T = \max_{t} P(Y = 1|T = t, I_S = 0),$$

for the total population, and

$$m'_S = \min_{t} P(Y(t) = 1|T = 1 - t, I_S = 1)$$

and

$$M'_S = \max_{t} P(Y(t) = 1|T = 1 - t, I_S = 1),$$

for the subpopulation. The AF bounds can of course be used if the conditional independence assumption is not fulfilled, but the CAF bounds are less conservative. The GAF and CAF bounds are equal to the AF bound when the most extreme values on sensitivity parameters are used.

**Table 1:** Summary of the different bounds.

| Bound | Unknown sensitivity parameters | Uses data | Additional assumptions | Specify counterfactual probabilities |
|---|---|---|---|---|
| Smith and VanderWeele (SV) | ✓ | | ✓ | |
| Sharp | ✓ | ✓ | ✓ | |
| Assumption-free (AF) | | ✓ | | |
| Generalized assumption-free (GAF) | ✓ | ✓ | ✓ | |
| Counterfactual assumption-free (CAF) | ✓ | ✓ | | ✓ |

## 3 The simulated data set

For the purpose of illustration of the bounds, we construct a simulated dataset, `zika_learner`. This is inspired by a case-control study that investigates the effect of zika virus on microcephaly (de Araújo et al., 2018) and the numerical zika example introduced in Smith and VanderWeele (2019). The data can be loaded by

```
data("zika_learner", package = "SelectionBias")
```

The zika example in Smith and VanderWeele (2019) covers the case of a single selection, and we expand the example with a second selection variable to demonstrate multiple inclusion criteria which are common in empirical applications. The two selections are the binary variables `birth` and `public hospital`, and the selection process is described in Figure 2 (left). In the original study, pre-treatment covariates for both the mother and the infant were included in order to control for confounding (de Araújo et al., 2018), however, we only consider one stratum of the covariates and thus exclude all observed pre-treatment covariates in the simulated dataset. The causal model of the dataset is given in Figure 2 (right). All variables are binary and generated from the binomial distribution such that the prevalences of the variables, and strengths of dependencies between them, mimic the real empirical data and the assumed values for the sensitivity parameters in Smith and VanderWeele (2019). The causal dependencies are generated by the logistic models described in Table 2. Even though the dataset is inspired by a case-control study, the simulated dataset emulates a register with 5000 observations, similar to an observational study of zika virus (Lebov et al., 2019). The variables included are:

- `urban` ($V$). A binary, unobserved variable indicating whether the subject lives in an urban area ($V = 1$) or not ($V = 0$). The probability of living in an urban area is set to 0.85 following numbers from the World Bank (World Bank Group).
- `ses` ($U$). A binary variable indicating socioeconomic status (high vs low), with probability set to 0.5.
- `zika` ($T$). In 2016 there were approximately 34000 possible cases of zika virus during pregnancy (de Oliveira et al., 2017), and there are approximately 2.9 million births per year in Brazil (Diniz et al., 2019). The risk of getting zika was higher in urban areas (Ali et al., 2017). Thus, the prevalence of zika in the simulated dataset is around 1% with a positive impact of $V$, see Table 2 for details.
- `mic_ceph` ($Y$). The estimated prevalence of microcephaly was 74 cases per 10000 births and the controlled odds ratio of zika virus on microcephaly was estimated at 73.1 (de Araújo et al., 2018). We mimic these values with an overall prevalence of 0.8% and a risk ratio of 74.5 among the subpopulation after two selections, see Table 2 for details.
- `birth` ($S_1$). Pregnancies that ended in a live or still birth were included in the study, and pregnancies that ended in a miscarriage or an abortion were excluded. The probability of a pregnancy ending in birth is assumed to be affected both by zika virus infection and socioeconomic status. The number of births per year in Brazil is approximately 2.9 million and the number of unregulated abortions per year is estimated at 500000 (Malta et al., 2019). The prevalence of birth in the simulated dataset is 0.86, with a strong negative impact of zika virus, and a positive impact of socioeconomic status, see Table 2 for details.
- `hospital` ($S_2$). Births that occurred in public hospitals were included in the study, and births that occurred in private hospitals were excluded. A majority of the population in Brazil visits public hospitals, and here it is assumed that it is strongly affected by socioeconomic status and weakly affected by living area, see Table 2 for details.

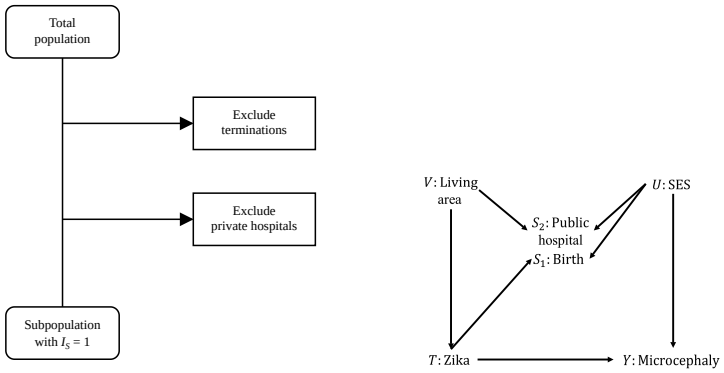



**Figure 2:** Selection process and causal structure for the simulated dataset. Left: flow chart of the two-step selection from the total population. Pregnancies ending in termination are excluded first, followed by births occurring at private hospitals, leaving the subpopulation with $I_S = 1$. Right: the corresponding causal DAG, where living area ($V$) and socioeconomic status ($U$) are unmeasured variables affecting the two selection steps, birth ($S_1$) and public hospital ($S_2$), while zika infection ($T$) affects microcephaly ($Y$) directly. The DAG shows that both selection steps depend on the same unmeasured variables that also relate to the outcome, which is why selection bias can arise even though $T$ and $Y$ are not confounded in the total population.

**Table 2:** Data generating process for the dataset `zika_learner`. Models generating causal dependencies are logistic, $g(X'\theta)$, for predictor variables $X$ (including a constant 1 for the intercept) and model parameter $\theta$ (including the intercept).

| Model | Coefficients ($\theta$)/Proportions | Function argument |
|---|---|---|
| $P(V = 1)$ | 0.85 | `Vval` |
| $P(U = 1)$ | 0.50 | `Uval` |
| $P(T = 1\|V) = g(V'\theta_T)$ | $(-6.20, 1.75)$ | `Tcoef` |
| $P(Y = 1\|T, U) = g[(T, U)'\theta_Y]$ | $(-5.20, 5.00, -1.00)$ | `Ycoef` |
| $P(S_1 = 1\|V, U, T) = g[(V, U, T)'\theta_{S1}]$ | $(1.20, 0.00, 2.00, -4.00)$ | `Scoef` |
| $P(S_2 = 1\|V, U, T) = g[(V, U, T)'\theta_{S2}]$ | $(2.20, 0.50, -2.75, 0.00)$ | `Scoef` |

The unobserved variables, `urban` ($V$) and `ses` ($U$) are constructed such that the true sensitivity parameters in our DGP approximately match the assumed values in Smith and VanderWeele (2019). In the simulated dataset, the proportions of the variables presented in Tables 3-5 reflect this. The proportions are calculated both by treatment status (zika virus infection) and overall, for the total dataset, the subset with $S_1 = 1$ and the subset with $S_1 = 1$ and $S_2 = 1$.

**Table 3:** Proportions and number of subjects for the simulated dataset, by treatment (zika) status and overall.

| | Not zika infected | Zika infected | Overall |
|---|---|---|---|
| Number of subjects | 4939 | 61 | 5000 |
| Microcephaly | 0.003 | 0.361 | 0.008 |
| Living area | 0.849 | 0.951 | 0.850 |
| SES | 0.499 | 0.426 | 0.498 |

**Table 4:** Proportions and number of subjects for the simulated dataset, by treatment (zika) status and overall, after the first selection.

| | Not zika infected | Zika infected | Overall |
|---|---|---|---|
| Number of subjects | 4268 | 11 | 4279 |
| Microcephaly | 0.003 | 0.273 | 0.004 |
| Living area | 0.845 | 1.000 | 0.846 |
| SES | 0.556 | 0.818 | 0.557 |

For the DGP, the causal estimand is $\beta_R = 90.7$ in the total population, $\beta_{R_S} = 92.3$ in the subset of $S_1 = 1$, and $\beta_{R_S} = 88.1$ in the subset of $S_1 = 1$ and $S_2 = 1$. The observed risk ratios for the dataset are $\beta_R^{obs} = 89.5$ (after one selection), and $\beta_R^{obs} = 74.5$ (after two selections). The dataset and DGP can be used to apply the functions in **SelectionBias**.

**Table 5:** Proportions and number of subjects for the simulated dataset, by treatment (zika) status and overall, after both selections.

| | Not zika infected | Zika infected | Overall |
|---|---|---|---|
| Number of subjects | 2869 | 7 | 2876 |
| Microcephaly | 0.004 | 0.286 | 0.005 |
| Living area | 0.858 | 1.000 | 0.858 |
| SES | 0.382 | 0.714 | 0.382 |

## 4 R package SelectionBias

In this section, we describe the package **SelectionBias**. The different functions are illustrated using either the simulated dataset `zika_learner`, or the model structure that the dataset is simulated from (see Table 2). The package **SelectionBias** is available under a GPL-compatible license from the Comprehensive R Archive Network (CRAN) at https://cran.r-project.org/package=SelectionBias, and can be installed and loaded using:

```
install.packages("SelectionBias")

library(SelectionBias)
```

### 4.1 `sensitivityparametersM()`

The sensitivity parameters for the SV, sharp, and GAF bounds are calculated for the generalized M-structure illustrated in Figure 1 in the function `sensitivityparametersM()`. The input in the function is the model structure, and the observed probabilities of the outcome, $P(Y = 1|T = t, I_S = 1), t = 0, 1$. The code and the output are:

```
# SV bound
sensparSV = sensitivityparametersM(whichEst = "RR_tot",
                   whichBound = "SV",
                   Vval = matrix(c(1, 0, 0.85, 0.15), ncol = 2),
                   Uval = matrix(c(1, 0, 0.5, 0.5), ncol = 2),
                   Tcoef = c(-6.2, 1.75),
                   Ycoef = c(-5.2, 5.0, -1.0),
                   Scoef = matrix(c(1.2, 2.2, 0.0, 0.5,
                                    2.0, -2.75, -4.0, 0.0),
                                  ncol = 4),
                   Mmodel = "L",
                   pY1_T1_S1 = 0.286,
                   pY1_T0_S1 = 0.004)
print(sensparSV)

#> $parameter
#>  [1] "BF_11"     "BF_00"     "BF_10"     "BF_01"     "RR_UY|T=1" "RR_UY|T=0"
#>  [7] "RR_SU|11"  "RR_SU|00"  "RR_SU|10"  "RR_SU|01"
#>
#> $value
#>  [1] 1.2102 1.3532 1.3208 1.3895 1.9448 2.7089 1.5566 1.7058 1.9998 1.7998
#>
#> attr(,"class")
#> [1] "sensparams"
#> attr(,"row.names")
#>  [1]  1  2  3  4  5  6  7  8  9 10
```

```
# Sharp bound
sensparSharp = sensitivityparametersM(whichEst = "RR_tot",
                   whichBound = "sharp",
                   Vval = matrix(c(1, 0, 0.85, 0.15), ncol = 2),
                   Uval = matrix(c(1, 0, 0.5, 0.5), ncol = 2),
                   Tcoef = c(-6.2, 1.75),
                   Ycoef = c(-5.2, 5.0, -1.0),
                   Scoef = matrix(c(1.2, 2.2, 0.0, 0.5,
                                    2.0, -2.75, -4.0, 0.0),
                                  ncol = 4),
                   Mmodel = "L",
                   pY1_T1_S1 = 0.286,
                   pY1_T0_S1 = 0.004)

print(sensparSharp)

#> $parameter
#>  [1] "BF_11"     "BF_00"     "BF_10"     "BF_01"     "RR_UY|T=1" "RR_UY|T=0"
#>  [7] "RR_SU|11"  "RR_SU|00"  "RR_SU|10"  "RR_SU|01"
#>
#> $value
#>  [1] 1.2102 1.3532 1.3208 1.3895 1.9448 2.7089 1.5566 1.7058 1.9998 1.7998
#>
#> attr(,"class")
#> [1] "sensparams"
#> attr(,"row.names")
#>  [1]  1  2  3  4  5  6  7  8  9 10

# GAF bound
sensparGAF = sensitivityparametersM(whichEst = "RR_tot",
                   whichBound = "GAF",
                   Vval = matrix(c(1, 0, 0.85, 0.15), ncol = 2),
                   Uval = matrix(c(1, 0, 0.5, 0.5), ncol = 2),
                   Tcoef = c(-6.2, 1.75),
                   Ycoef = c(-5.2, 5.0, -1.0),
                   Scoef = matrix(c(1.2, 2.2, 0.0, 0.5,
                                    2.0, -2.75, -4.0, 0.0),
                                  ncol = 4),
                   Mmodel = "L",
                   pY1_T1_S1 = 0.286,
                   pY1_T0_S1 = 0.004)

print(sensparGAF)

#> $parameter
#> [1] "m_T" "M_T"
#>
#> $value
#> [1] 0.0020 0.4502
#>
#> attr(,"class")
#> [1] "sensparams"
#> attr(,"row.names")
#> [1] 1 2
```

The first argument is `whichEst`, where the user inputs the causal estimand of interest. It must be one of the four `"RR_tot"`, `"RD_tot"`, `"RR_sub"` or `"RD_sub"`. The second argument is `whichBound`, where the user inputs the bound they want to use, either `"SV"`, `"sharp"`, or `"GAF"`. Third, the argument `Vval` takes the matrix for $V$ as input. The first column contains the values that $V$ can take, and the second column contains the corresponding probabilities. In this example, $V$ is binary, so the first two elements in the matrix are 1 and 0. However, any discrete $V$ can be used. An approximation of a continuous $V$ can be used, if it is discretized. The fourth argument is `Uval`, which takes the matrix for $U$ as input. The matrix $U$ has a similar structure as $V$. The fifth argument is `Tcoef`, containing the coefficients used in the model for $T$. The first entry in `Tcoef` is the intercept of the model, and the second the slope for $V$.

The sixth argument is `Ycoef`, containing the coefficient vector for the outcome model, where the first entry is the intercept, the second the slope coefficient for $T$ and third is the slope coefficient for $U$. The seventh argument is `Scoef`, the coefficient matrix for the selection variables. The number of rows is equal to the number of selection variables, and the number of columns is equal to four. The columns represent the intercept, and slope coefficients for $V$, $U$ and $T$, respectively. A summary of the code notation is seen in the last column of Table 2. The eighth argument is `Mmodel`, which indicates whether the models in the M-structure are probit (`Mmodel = "P"`) or logit (`Mmodel = "L"`). The ninth and tenth arguments are `pY1_T1_S1` and `pY1_T0_S1`. They are the observed probabilities $P(Y = 1|T = 1, I_S = 1)$ and $P(Y = 1|T = 0, I_S = 1)$. The output is the sensitivity parameters for the chosen bound.

In the zika example, the estimand of interest is the risk ratio in the total population, `whichEst = "RR_tot"`, the DGP is found in Table 2, logistic models are used in the DGP, and the probabilities are found in Table 5. For the SV and sharp bounds, the output is $RR_{UY|T=1} = 1.9448$, $RR_{UY|T=0} = 2.7089$, $RR_{SU|11} = 1.5566$, $RR_{SU|00} = 1.7058$, $RR_{SU|10} = 1.9998$, and $RR_{SU|01} = 1.7998$, which gives $BF_{11} = 1.2102$ and $BF_{00} = 1.3532$, $BF_{10} = 1.3208$, and $BF_{01} = 1.3895$. For the GAF bound, the output is $M_T = 0.4502$ and $m_T = 0.002$.

### 4.2 SVbound()

The SV bound can be calculated using the function `SVbound()`. The first argument is `whichEst`, indicating the causal estimand of interest (`"RR_tot"`, `"RD_tot"`, `"RR_sub"` or `"RD_sub"`). The second argument is `sens`, which is either output from `sensitivityparametersM()`, a named list, or a data frame with column names parameter and value, which includes the sensitivity parameters. This input is optional as the sensitivity parameters can also be specified directly in the function with the below arguments. The third and fourth arguments are the observed conditional probabilities $P(Y = 1|T = 1, I_S = 1)$ and $P(Y = 1|T = 0, I_S = 1)$, which are needed to calculate bounds for the causal estimands and not the selection bias itself. The optional fifth and sixth arguments are the treatment probabilities $P(T = 1|I_S = 1)$ and $P(T = 0|I_S = 1)$. These are only needed for the alternative SV bound for the causal risk difference in the subpopulation (Zetterstrom et al., 2025). If they are not included, the original SV bound for the risk difference is calculated. The subsequent arguments are the sensitivity parameters provided by the user. The default value for all sensitivity parameters is `NULL`, and the user must then specify numeric values on the sensitivity parameters that are necessary for the bound for the chosen estimand. These arguments are also optional in case the argument `sens` is used. The sensitivity parameters can either be calculated using the function `sensitivityparametersM()`, or found elsewhere. For sensitivity parameters found elsewhere, `SVbound()` is not restricted to the generalized M-structure. However, the necessary assumptions for the SV bound must still be fulfilled (Smith and VanderWeele, 2019). The output is the SV bound. The code and output are:

```
SVbound(whichEst = "RR_tot",
        pY1_T1_S1 = 0.286,
        pY1_T0_S1 = 0.004,
        RR_UY_T1 = 1.9448,
        RR_UY_T0 = 2.7089,
        RR_SU_11 = 1.5566,
        RR_SU_00 = 1.7058,
        RR_SU_10 = 1.9998,
        RR_SU_01 = 1.7998)
```

```
#>      [,1]             [,2]
#> [1,] "SV lower bound" 43.66
#> [2,] "SV upper bound" 131.22
```

As before in the zika example, the causal estimand is the risk ratio in the total population, `whichEst = "RR_tot"`. The sensitivity parameters are $RR_{UY|T=1} = 1.9448$, $RR_{UY|T=0} = 2.7089$, $RR_{SU|11} = 1.5566$, $RR_{SU|00} = 1.7058$, $RR_{SU|10} = 1.9998$, and $RR_{SU|01} = 1.7998$, calculated above in `sensitivityparametersM()`, which gives a lower SV bound equal to 43.66 and a higher SV bound equal to 131.22. Alternatively, the output from the function `sensitivityparametersM()` can be used directly as input which gives the code:

```
SVbound(whichEst = "RR_tot",
        sens = sensparSV,
        pY1_T1_S1 = 0.286,
        pY1_T0_S1 = 0.004)
```

```
#>      [,1]             [,2]
#> [1,] "SV lower bound" 43.66
#> [2,] "SV upper bound" 131.22
```

This approach gives the same bounds as specifying the sensitivity parameters manually.

### 4.3 sharpbound()

The sharp bound can be calculated using the function `sharpbound()`. The input arguments are very similar to that of `SVbound()`. The first argument is `whichEst`, indicating the causal estimand of interest (`"RR_tot"`, `"RD_tot"`, `"RR_sub"` or `"RD_sub"`). The second argument is `sens`, which is either output from `sensitivityparametersM()`, a named list, or a data frame with column names parameter and value, which includes the sensitivity parameters. This input is optional as the sensitivity parameters can also be specified directly in the function with the below arguments. The third and fourth arguments are the observed conditional probabilities $P(Y = 1|T = 1, I_S = 1)$ and $P(Y = 1|T = 0, I_S = 1)$, which are needed to calculate bounds for the causal estimands and not the selection bias itself. The fifth and sixth arguments are the treatment probabilities $P(T = 1|I_S = 1)$ and $P(T = 0|I_S = 1)$. These are only needed for the bounds for the causal estimands in the subpopulation. The seventh and eighth arguments are the probabilities $P(I_S = 1|T = 1)$ and $P(I_S = 1|T = 0)$ which are only needed for the bounds for the causal estimands in the total population. If they are unknown, they can be set to 0 for more conservative bounds. The subsequent arguments are the sensitivity parameters provided by the user. The default value for all sensitivity parameters is `NULL`, and the user must then specify numeric values on the sensitivity parameters that are necessary for the bound for the chosen estimand. These arguments are also optional in case the argument `sens` is used. The sensitivity parameters can either be calculated using the function `sensitivityparametersM()`, or found elsewhere. For sensitivity parameters found elsewhere, `sharpbound()` is not restricted to the generalized M-structure. However, the necessary assumptions for the sharp bound must still be fulfilled (Zetterstrom et al., 2025). The output is the sharp bound. The code and output with the two options to specify the sensitivity parameters are:

```
sharpbound(whichEst = "RR_tot",
        pY1_T1_S1 = 0.286,
        pY1_T0_S1 = 0.004,
        pS1_T1 = 0.11,
        pS1_T0 = 0.58,
        RR_UY_T1 = 1.9448,
        RR_UY_T0 = 2.7089,
        RR_SU_11 = 1.5566,
        RR_SU_00 = 1.7058,
        RR_SU_10 = 1.9998,
        RR_SU_01 = 1.7998)
```

```
#>      [,1]               [,2]
#> [1,] "Sharp lower bound" 73.91
#> [2,] "Sharp upper bound" 78.99
```

```
sharpbound(whichEst = "RR_tot",
        sens = sensparSharp,
        pY1_T1_S1 = 0.286,
        pY1_T0_S1 = 0.004,
        pS1_T1 = 0.11,
        pS1_T0 = 0.58)
```

```
#>      [,1]               [,2]
#> [1,] "Sharp lower bound" 73.91
#> [2,] "Sharp upper bound" 78.99
```

As before in the zika example, the causal estimand is the risk ratio in the total population, `whichEst = "RR_tot"`. The observed probabilities come from the dataset, and the sensitivity parameters are $RR_{UY|T=1} = 1.9448$, $RR_{UY|T=0} = 2.7089$, $RR_{SU|11} = 1.5566$, $RR_{SU|00} = 1.7058$, $RR_{SU|10} = 1.9998$, and $RR_{SU|01} = 1.7998$, calculated above in `sensitivityparametersM()`, which gives a lower sharp bound equal to 73.91 and a higher sharp bound equal to 78.99.

### 4.4 checksharpSVbound()

The sharpness of the SV bound can be evaluated using `checksharpSVbound()` (Zetterstrom et al., 2025). The first argument is `whichEst`, indicating the causal estimand of interest (`"RR_tot"`, `"RR_sub"` or `"RD_sub"`). Note that the SV bound for the risk difference in the total population is not sharp. The second argument is `sens`, which is either output from `sensitivityparametersM()`, a named list or a data frame with column names parameter and value, which includes the bounding factors ($BF$). This input is optional as the bounding factors can also be specified directly in the function with the argument below. The third argument is `BF`, the bounding factors $BF_{00}$ and $BF_{10}$ for the total population, and $BF_0$ and $BF_1$ for the subpopulation. The fourth argument is `pY1`, the probabilities $P(Y = 1|T = 1, I_S = 1)$ and $P(Y = 1|T = 0, I_S = 1)$. Note that the order of the bounding factors and probabilities matters. The output is two strings stating whether the lower and upper SV bound are sharp or not. Note that the SV bound for the risk ratio in the total population and risk difference in the subpopulation can only be arbitrarily sharp, see (Zetterstrom et al., 2025) for more details. The code is:

```
checksharpSVbound(whichEst = "RR_tot",
        sens = sensparSV,
        pY1 = c(0.286, 0.004))

#> The lower SV bound for the risk ratio in the total population is arbitrarily sharp.
#> See vignette for details.

#> The upper SV bound for the risk ratio in the total population is arbitrarily sharp.
#> See vignette for details.
```

The output is a string that states whether the SV bounds are sharp or not. For these settings, both the lower and upper SV bounds are arbitrarily sharp.

### 4.5 AFbound()

The AF bound is calculated using the function `AFbound()`. The first argument is the causal estimand of interest (`"RR_tot"`, `"RD_tot"`, `"RR_sub"` or `"RD_sub"`). The second argument is `outcome`, where the user inputs either the numeric vector with the observed outcome variable or a vector with the conditional outcome probabilities, $P(Y = 1|T = 1, I_S = 1)$ and $P(Y = 1|T = 0, I_S = 1)$. The third argument is `treatment`, where the user inputs either the numeric vector with the observed treatment variable or a vector with the conditional treatment probabilities, $P(T = 1|I_S = 1)$ and $P(T = 0|I_S = 1)$. The fourth argument is `selection` where the user inputs either the numeric vector with the observed selection variable, or the selection probability, $P(I_S = 1)$. Its default value is `NULL` since it is only required when the causal estimands in the total population are of interest. If the subpopulation is of interest and `selection = NULL`, the outcome and treatment vectors must only include the selected subjects. The output is the lower and upper AF bounds. The code and output are:

```
attach(zika_learner)

AFbound(whichEst = "RR_tot",
        outcome = mic_ceph[sel_ind == 1],
        treatment = zika[sel_ind == 1],
        selection = mean(sel_ind))

#>      [,1]             [,2]
#> [1,] "AF lower bound" 0
#> [2,] "AF upper bound" 454.09
```

Similar to before, `whichEst = "RR_tot"`. Furthermore, the outcome and treatment variables are microcephaly and zika. The selection probability is specified since the other variables are restricted to those subjects with $I_S = 1$. The output is the lower and upper AF bounds, which are 0 and 454.09 in the zika example.

If the raw data is not available, one can input the conditional probabilities instead. In this example, these probabilities are:

```
AFbound(whichEst = "RR_tot",
        outcome = c(0.286, 0.004),
        treatment = c(0.002, 0.998),
        selection = mean(sel_ind))
```

```
#>      [,1]             [,2]
#> [1,] "AF lower bound" 0
#> [2,] "AF upper bound" 435.14
```

The difference in these two examples comes from rounding errors in the input probabilities.

### 4.6 GAFbound()

The GAF bound is calculated using the function `GAFbound()`. The first argument is the causal estimand of interest (`"RR_tot"`, `"RD_tot"`, `"RR_sub"` or `"RD_sub"`). The second argument is `sens`, which is either output from `sensitivityparametersM()`, a named list, or a data frame with column names "parameter" and "value", which includes the sensitivity parameters. This input is optional as the sensitivity parameters can also be specified directly in the function with the below arguments. The third and fourth arguments are `M` and `m` which are the two sensitivity parameters for the GAF bound. The sensitivity parameters can either be calculated using `sensitivityparametersM()`, or found elsewhere. For sensitivity parameters found elsewhere, `GAFbound()` is not restricted to the generalized M-structure. However, the necessary assumptions for the GAF bound must still be fulfilled (Zetterstrom, 2024). The fifth argument is `outcome`, where the user inputs either the observed numeric vector with the outcome variable or a vector with the conditional outcome probabilities, $P(Y = 1|T = 1, I_S = 1)$ and $P(Y = 1|T = 0, I_S = 1)$. The sixth argument is `treatment`, where the user inputs either the observed numeric vector with the treatment variable or a vector with the conditional treatment probabilities, $P(T = 1|I_S = 1)$ and $P(T = 0|I_S = 1)$. The seventh argument is `selection` where the user can either input the observed selection vector or selection probability. Its default value is `NULL` since it is only required when the causal estimands in the total population are of interest. If the subpopulation is of interest and `selection = NULL`, the outcome and treatment vectors must only include the selected subjects. The output is the lower and upper GAF bounds. The code and output are:

```
GAFbound(whichEst = "RR_tot",
         M = 0.4502,
         m = 0.002,
         outcome = mic_ceph[sel_ind == 1],
         treatment = zika[sel_ind == 1],
         selection = mean(sel_ind))

#>      [,1]             [,2]
#> [1,] "GAF lower bound" 0.01
#> [2,] "GAF upper bound" 147.42

GAFbound(whichEst = "RR_tot",
         sens = sensparGAF,
         outcome = mic_ceph[sel_ind == 1],
         treatment = zika[sel_ind == 1],
         selection = mean(sel_ind))

#>      [,1]             [,2]
#> [1,] "GAF lower bound" 0.01
#> [2,] "GAF upper bound" 147.42
```

Similar to before, the estimand of interest is the risk ratio in the total population, `whichEst = "RR_tot"`. The sensitivity parameters are the output from `sensitivityparametersM()`. Furthermore, the outcome and treatment variables are microcephaly and zika. The selection probability is specified since the other variables are restricted to those subjects with $I_S = 1$. The output is the lower and upper GAF bounds, which are 0.01 and 147.42 in the zika example. If the raw data is not available, one can input the conditional probabilities instead, similar to the AF bound.

### 4.7 CAFbound()

The CAF bound is calculated using the function `CAFbound()`. The first argument is the causal estimand of interest (`"RR_tot"`, `"RD_tot"`, `"RR_sub"` or `"RD_sub"`). The second and third arguments are `M` and `m` which are the two sensitivity parameters for the CAF bound. The fourth argument is `outcome`, where the user inputs either the numeric vector with the observed outcome variable or a vector with the conditional outcome probabilities, $P(Y = 1|T = 1, I_S = 1)$ and $P(Y = 1|T = 0, I_S = 1)$. The fifth

argument is `treatment`, where the user inputs either the numeric vector with the observed treatment variable or a vector with the conditional treatment probabilities, $P(T = 1|I_S = 1)$ and $P(T = 0|I_S = 1)$. The sixth argument is `selection` where the user inputs either the numeric vector with the observed selection variable, or the selection probability, $P(I_S = 1)$. Its default value is `NULL` since it is only required when the causal estimands in the total population are of interest. If the subpopulation is of interest and `selection = NULL`, the outcome and treatment vectors must only include the selected subjects. The output is the lower and upper CAF bounds. The code and output are:

```
CAFbound(whichEst = "RR_tot",
         M = 0.3,
         m = 0.005,
         outcome = c(0.286, 0.004),
         treatment = c(0.002, 0.998),
         selection = mean(sel_ind))
```

```
#>      [,1]              [,2]
#> [1,] "CAF lower bound" 0.04
#> [2,] "CAF upper bound" 67.78
```

Similar to before, `whichEst = "RR_tot"`. The sensitivity parameters are chosen by the user. Furthermore, the outcome and treatment variables are microcephaly and zika. The selection probability is specified since the other variables are restricted to those subjects with $I_S = 1$. The output is the lower and upper CAF bounds, which are 0.04 and 67.78 in the zika example. If the raw data is not available, one can input the conditional probabilities instead, similar to the AF and GAF bounds.

## 5 Comparison to other software for sensitivity analysis

Sensitivity analysis is an important tool when the statistical analysis relies on untestable assumptions, and there are many R packages that can be used to perform sensitivity analyses for different types of bias and using different methods. Here, we address some of them and compare them to **SelectionBias**.

The R packages **tipr** (McGowan, 2022) and **sensemakr** (Cinelli et al., 2024) solely focus on bias that arises from unmeasured confounding between a treatment and outcome variable (although **sensemakr** calls this "omitted-variable bias"), and they are therefore not concerning the same situation as **SelectionBias**. The numerical bounds for multiple types of biases presented in Duarte et al. (2024) are accompanied by a Python package. These bounds make fewer assumptions than the SV, sharp, and GAF bounds, and can therefore be more conservative, and are not available in R. The R package with most overlap is the **EValue** package, and below follows a more detailed comparison of them.

The R package **EValue** provides sensitivity analysis methods for unmeasured confounding, measurement error, selection bias, and combinations of them (VanderWeele and Ding, 2017; Mathur and VanderWeele, 2020). However, **EValue** only provides one method of sensitivity analysis for selection bias: the SV bound (and other versions of the SV bound that require more information and assumptions) and the accompanying selection E-value. The selection E-value is a summary measure that calculates how large the sensitivity parameters must be in order to explain away an effect under the assumption that they all are equal, i.e. a type of Cornfield inequality (Cornfield et al., 1959).

**SelectionBias** on the other hand includes four other bounds (in addition to the SV bound): the sharp, GAF, CAF, and AF bounds. The user therefore has more options for their sensitivity analyses for selection bias. This is important since the different bounds are valid under different assumptions, meaning that they can be used in various cases. As a result, the package **SelectionBias** can cover more cases than the **EValue** package. The sensitivity parameters required in the SV bound may in some cases also be difficult to specify, and **SelectionBias** can calculate the sensitivity parameters for a user-specified DGP whereas **EValue** requires the user to specify them. This DGP may be easier to specify than the sensitivity parameters in some cases. **SelectionBias** can also check sharpness of the SV bound, i.e. check if it is possible that the bias is as large as the bound or if it is too conservative.

To summarize, if the assumptions for the SV bound are fulfilled, and the user is confident in specifying the sensitivity parameters for the SV bounds, or wants to calculate bounds for other types of biases as well, the package **EValue** provides the desired functionality. On the other hand, if the assumptions for the SV bound are not valid, or the user is more confident in specifying a DGP, or wishes to compare different bounds for the selection bias, **SelectionBias** provides the required functionality. Including several bounds in one framework, together with more comprehensive numerical and data examples, allows **SelectionBias** to facilitate comparisons between bounds. The different features of these packages are summarized in Table 6.

**Table 6:** Summary of the features of the different packages.

| Package | Calculate sensitivity parameters from DGP | Multiple bounds | Additional SV bounds | Summary measure | Evaluate sharpness | Other biases |
|---|---|---|---|---|---|---|
| EValue | No | No | Yes | Yes | No | Yes |
| SelectionBias | Yes | Yes: sharp GAF CAF AF | No | No | Yes | No |

## 6 Conclusion

To perform sensitivity analyses for selection bias, an option is to calculate bounds. For applied researchers, we present the R package **SelectionBias**. The **SelectionBias** package provides functions for calculating sensitivity parameters and bounds previously proposed in the literature. In detail, the package includes a function that calculates the sensitivity parameters for the SV, sharp, and GAF bounds for the generalized M-structure specified by the user, functions for calculating the SV, sharp, AF, GAF, and CAF bounds, and a function for checking whether the calculated SV bound is sharp (Smith and VanderWeele, 2019; Zetterstrom et al., 2025; Zetterstrom, 2024; Zetterstrom and Waernbaum, 2022). For users to become familiar with the functions, the R package also includes a simulated dataset, `zika_learner`, and the DGP the dataset is simulated from. The simulated data emulates a register-based cohort study, inspired by a case-control study investigating the effect of zika virus on microcephaly in Brazil (de Araújo et al., 2018). Hopefully, **SelectionBias** can be a complement to the already existing package **EValue**. **EValue** is a good option when the user has knowledge of the sensitivity parameters, or when the user wants to address several types of biases. **SelectionBias** on the other hand can be an option if the user wants to compare several different bounds or has knowledge about the DGP. Furthermore, with **SelectionBias**, the user can perform the calculations for the sensitivity analysis for many possible designs and different bounds, and thereby gain a broader knowledge of the possible magnitude of the selection bias. The package is continuously updated with new sensitivity analyses and additional functions. For instance, bootstrap is expected to be included to account for sampling variability.

## Appendix

### Underlying assumptions for the SV, sharp, and GAF bounds

The SV, sharp, and GAF bounds are valid under additional assumptions. The bounds are derived under two different conditional independence assumptions for the causal model, depending on whether the total or subpopulation estimands are of interest:

1. *(Total population estimands $\beta_R$ and $\beta_D$) For some unmeasured variable(s) U:* $Y \perp I_S | (T = t, U = u)$, *for* $t = 0, 1$.
2. *(Subpopulation estimands $\beta_{R_S}$ and $\beta_{D_S}$) For some unmeasured variable(s) U:* $Y(t) \perp T | (I_S = 1, U = u)$, *for* $t = 0, 1$.

The assumptions differ since their purpose is to provide conditions for unbiased estimation of the estimands of interest under selection if the unmeasured variable was observed. The difference arises from either i) a lack of generalization of a causal effect from a subpopulation to the total population, or ii) a violation of conditional exchangeability in the subpopulation, i.e., $Y(t) \perp T | X, I_S = 1$ does not hold. There are however structures in which both assumptions are fulfilled, for example the generalized M-structure in Figure 1.

### The bounds

All bounds have the structure

$$LB < \beta < UB.$$

The SV bounds for the risk ratio in the total population are

$$\frac{\beta_R^{obs}}{BF_{11}BF_{00}} \leq \beta_R \leq \beta_R^{obs} \cdot BF_{10}BF_{01}.$$

The SV bounds for the risk difference in the total population are

$$\beta_D^{obs} - (BF_{11} - \frac{P(Y=1|T=1,I_S=1)}{BF_{11}} + P(Y=1|T=0,I_S=1) \cdot BF_{00}) \leq \beta_D \leq$$

$$\beta_D^{obs} + (BF_{01} - \frac{P(Y=1|T=0,I_S=1)}{BF_{01}} + P(Y=1|T=1,I_S=1) \cdot BF_{10}).$$

The SV bounds for the risk ratio in the subpopulation are

$$\frac{\beta_R^{obs}}{BF_1} \leq \beta_{R_S} \leq \beta_R^{obs} \cdot BF_0.$$

The SV bounds for the risk difference in the subpopulation are

$$\beta_D^{obs} - \max(P(Y=1|T=0,I_S=1) \cdot (BF_1-1), P(Y=1|T=1,I_S=1) \cdot (1-1/BF_1)) \leq \beta_{D_S} \leq$$

$$\beta_D^{obs} + \max(P(Y=1|T=1,I_S=1) \cdot (BF_0-1), P(Y=1|T=0,I_S=1) \cdot (1-1/BF_0)).$$

The alternative SV bounds for the risk difference in the subpopulation are

$$\beta_D^{obs} - [P(Y=1|T=0,I_S=1) \cdot P(T=1|I_S=1) \cdot (BF_1-1) + P(Y=1|T=1,I_S=1) \cdot P(T=0|I_S=1) \cdot (1-1/BF_1)]$$

$$\leq \beta_{D_S} \leq$$

$$\beta_D^{obs} + [P(Y=1|T=1,I_S=1) \cdot P(T=0|I_S=1) \cdot (BF_0-1) + P(Y=1|T=0,I_S=1) \cdot P(T=1|I_S=1) \cdot (1-1/BF_0)].$$

The sharp bounds are presented as bounds for the potential outcome probabilities which can then be used to construct bounds for the causal estimands. The bounds for the total population are

$$l_t = p(Y=1 \mid T=t, I_S=1)\{p(I_S=1 \mid T=t) + p(I_S=0 \mid T=t)/BF_{t1}\}$$

and

$$u_t = p(Y=1 \mid T=t, I_S=1) \times [p(I_S=1 \mid T=t) + p(I_S=0 \mid T=t) \times \min(BF_{t0}, 1/p(Y=1 \mid T=t, I_S=1))],$$

which results in the following bounds for $p(Y(t)=1)$:

$$l_t \leq p(Y(t)=1) \leq u_t.$$

Upper (lower) bounds for the causal risk ratio and causal risk difference are found by combining $l_0$ ($l_1$) and $u_1$ ($u_0$). The sharp bounds for the subpopulation potential outcome probabilities are

$$l'_t = p(Y=1 \mid T=t, I_S=1)[p(T=t \mid I_S=1) + p(T=1-t \mid I_S=1)/BF_t]$$

and

$$u'_t = p(Y=1 \mid T=t, I_S=1) \times \left[p(T=t \mid I_S=1) + p(T=1-t \mid I_S=1) \times \min\{BF_{(1-t)}, 1/p(Y=1 \mid T=t, I_S=1)\}\right],$$

which results in the following bounds for $p(Y(t)=1 \mid I_S=1)$:

$$l'_t \leq p(Y(t)=1 \mid I_S=1) \leq u'_t$$

which can then be used to construct bounds for the causal risk ratio and causal risk difference.

For the GAF, CAF, and AF bounds the potential outcome probabilities in the total population can be bounded as

$$P(Y=1,T=t,S=1) + [1-P(T=t,S=1)] \cdot m_T < P(Y(t)=1) <$$
$$P(Y=1,T=t,S=1) + [1-P(T=t,S=1)] \cdot M_T.$$

The GAF bounds are calculated by using

$$m_T = \min_{t,u} P(Y=1|T=t,U=u)$$

and

$$M_T = \max_{t,u} P(Y = 1|T = t, U = u),$$

the CAF bounds are calculated by using

$$m_T = \min_t P(Y = 1|T = t, I_S = 0)$$

and

$$M_T = \max_t P(Y = 1|T = t, I_S = 0),$$

and the AF bounds are calculated by using $m_T = 0$ and $M_T = 1$. These bounds for the potential outcome probabilities can then be used to construct bounds for the causal risk ratio and causal risk difference.

For the GAF, CAF, and AF bounds the potential outcome probabilities in the subpopulation can be bounded as

$$P(Y = 1, T = t|I_S = 1) + P(T = 1 - t|I_S = 1) \cdot m_S < P(Y(t) = 1|S = 1) <$$

$$P(Y = 1, T = t|S = 1) + P(T = 1 - t|I_S = 1) \cdot M_S.$$

For the GAF bounds,

$$m_S = \min_{t,u} P(Y = 1|T = t, U = u, S = 1)$$

and

$$M_S = \max_{t,u} P(Y = 1|T = t, U = u, S = 1),$$

for the CAF bounds

$$m_S = \min_t P(Y(t) = 1|T = 1 - t, I_S = 1)$$

and

$$M_S = \max_t P(Y(t) = 1|T = 1 - t, I_S = 1),$$

and for the AF bounds $m_S = 0$ and $M_S = 1$. These bounds for the potential outcome probabilities can then be used to construct bounds for the causal risk ratio and causal risk difference.

## References

S. Ali, O. Gugliemini, S. Harber, A. Harrison, L. Houle, J. Ivory, S. Kersten, R. Khan, J. Kim, C. LeBoa, E. Nez-Whitfield, J. O'Marr, E. Rothenberg, R. M. Segnitz, S. Sila, A. Verwillow, M. Vogt, A. Yang, and E. A. Mordecai. Environmental and social change drive the explosive emergence of Zika virus in the Americas. *PLOS Neglected Tropical Diseases*, 11(2):e0005135, 2017. URL https://doi.org/10.1371/journal.pntd.0005135. [p41]

C. Cinelli, J. Ferwerda, and C. Hazlett. *sensemakr: Sensitivity analysis tools for regression models*, 2024. URL https://CRAN.R-project.org/package=sensemakr. R package version 0.1.6. [p49]

J. Cornfield, W. Haenszel, E. C. Hammond, A. M. Lilienfeld, M. B. Shimkin, and E. L. Wynder. Smoking and lung cancer: Recent evidence and a discussion of some questions. *JNCI: Journal of the National Cancer Institute*, 22(1):173–203, 01 1959. ISSN 0027-8874. doi: 10.1093/jnci/22.1.173. [p49]

T. V. B. de Araújo, R. A. de Alencar Ximenes, D. de Barros Miranda-Filho, W. V. Souza, U. R. Montarroyos, A. P. L. de Melo, S. Valongueiro, M. de Fátima Pessoa Militão de Albuquerque, C. Braga, S. P. B. Filho, M. T. Cordeiro, E. Vazquez, D. di Cavalcanti Souza Cruz, C. M. P. Henriques, L. C. A. Bezerra, P. M. da Silva Castanha, R. Dhalia, E. T. A. Marques-Júnior, C. M. T. Martelli, L. C. Rodrigues, C. Dhalia, M. Santos, F. Cortes, W. Kleber de Oliveira, G. Evelim Coelho, J. J. Cortez-Escalante, C. F. Campelo de Albuquerque de Melo, P. Ramon-Pardo, S. Aldighieri, J. Mendez-Rico, M. Espinal, L. Torres, A. Nassri Hazin, A. Van der Linden, M. Coentro, G. Santiago Dimech, R. Siqueira de Assuncao, P. Ismael de Carvalho, and V. Felix Oliveira. Association between microcephaly, Zika virus infection, and other risk factors in Brazil: Final report of a case-control study. *The Lancet Infectious Diseases*, 18(3):328–336, 2018. URL https://doi.org/10.1016/S1473-3099(17)30727-2. [p37, 41, 50]

W. K. de Oliveira, G. V. A. de França, E. H. Carmo, B. B. Duncan, R. de Souza Kuchenbecker, and M. I. Schmidt. Infection-related microcephaly after the 2015 and 2016 Zika virus outbreaks in Brazil: A surveillance-based analysis. *The Lancet*, 390(10097):861–870, 2017. URL https://doi.org/10.1016/S0140-6736(17)31368-5. [p41]

D. Diniz, L. Brito, I. Ambrogi, A. B. Tavares, and M. Ali. Understanding the sexual and reproductive health needs in Brazil's Zika-affected region: Placing women at the center of the discussion. *International Journal of Gynecology & Obstetrics*, 147(2):268–270, 2019. URL https://doi.org/10.1002/ijgo.12924. [p41]

G. Duarte, N. Finkelstein, D. Knox, J. Mummolo, and I. Shpitser. An automated approach to causal inference in discrete settings. *Journal of the American Statistical Association*, 119(547):1778–1793, 2024. URL https://doi.org/10.1080/01621459.2023.2216909. [p37, 49]

W. D. Flanders and D. Ye. Limits for the magnitude of M-bias and certain other types of structural selection bias. *Epidemiology*, 30(4):501–508, 2019. URL https://doi.org/10.1097/EDE.0000000000001031. [p37]

S. Greenland. Quantifying biases in causal models: Classical confounding vs collider-stratification bias. *Epidemiology*, 14(3):300–306, 2003. URL https://doi.org/10.1097/01.EDE.0000042804.12056.6C. [p37]

M. A. Hernán, S. Hernández-Díaz, and J. M. Robins. A structural approach to selection bias. *Epidemiology*, 15(5):615–625, 2004. URL http://www.jstor.org/stable/20485961. [p37]

T.-H. Huang and W.-C. Lee. Bounding formulas for selection bias. *American Journal of Epidemiology*, 182(10):868–872, 2015. URL https://doi.org/10.1093/aje/kwv130. [p37]

J. F. Lebov, J. F. Arias, A. Balmaseda, W. Britt, J. F. Cordero, L. A. Galvão, A. L. Garces, K. M. Hambidge, E. Harris, A. Ko, N. Krebs, E. T. A. Marques, A. M. Martinez, E. McClure, D. B. Miranda-Filho, M. E. L. Moreira, M. M. Mussi-Pinhata, T. J. Ochoa, J. E. Osorio, D. M. F. Scalabrin, S. Schultz-Cherry, G. R. Seage, K. Stolka, C. A. Ugarte-Gil, C. M. V. Vega, M. Welton, R. Ximenes, and C. Zorrilla. International prospective observational cohort study of Zika in infants and pregnancy (ZIP Study): Study protocol. *BMC Pregnancy and Childbirth*, 19(1):1–10, 2019. URL https://doi.org/10.1186/s12884-019-2430-4. [p41]

H. Lu, S. R. Cole, C. J. Howe, and D. Westreich. Toward a clearer definition of selection bias when estimating causal effects. *Epidemiology*, 33(5):699–706, 2022. URL https://doi.org/10.1097/EDE.0000000000001516. [p37]

M. Malta, S. Wells, S. LeGrand, M. Seixas, A. Baptista, C. M. F. P. da Silva, C. Kalume, and K. Whetten. Abortion in Brazil: The case for women's rights, lives, and choices. *The Lancet Public Health*, 4(11): e552, 2019. URL https://doi.org/10.1016/S2468-2667(19)30204-X. [p41]

M. B. Mathur and T. J. VanderWeele. Sensitivity analysis for unmeasured confounding in meta-analyses. *Journal of the American Statistical Association*, 115(529):163–172, 2020. URL https://doi.org/10.1080/01621459.2018.1529598. PMID: 32981992. [p49]

L. D. McGowan. tipr: An R package for sensitivity analyses for unmeasured confounders. *Journal of Open Source Software*, 7(77):4495, 2022. doi: 10.21105/joss.04495. [p49]

D. B. Rubin. Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of Educational Psychology*, 66(5):688–701, 1974. URL https://doi.org/10.1037/h0037350. [p38]

L. H. Smith and T. J. VanderWeele. Bounding bias due to selection. *Epidemiology*, 30(4):509–516, 2019. URL https://doi.org/10.1097/EDE.0000000000001032. [p37, 39, 41, 42, 45, 50]

L. H. Smith, M. B. Mathur, and T. J. VanderWeele. Multiple-bias sensitivity analysis using bounds. *Epidemiology*, 32(5):625–634, 2021. URL https://doi.org/10.1097/EDE.0000000000001380. [p37]

T. J. VanderWeele and P. Ding. Sensitivity analysis in observational research: Introducing the E-value. *Annals of Internal Medicine*, 167(4):268–274, 2017. URL https://www.acpjournals.org/doi/abs/10.7326/M16-2607. PMID: 28693043. [p49]

World Bank Group. Urban population - Brazil. URL https://data.worldbank.org/indicator/SP.URB.TOTL.IN.ZS?locations=BR. Accessed: 2024-10-05. [p41]

S. Zetterstrom. Bounds for selection bias using outcome probabilities. *Epidemiologic Methods*, 13(1): 20230033, 2024. URL https://doi.org/10.1515/em-2023-0033. [p37, 39, 48, 50]

S. Zetterstrom and I. Waernbaum. Selection bias and multiple inclusion criteria in observational studies. *Epidemiologic Methods*, 11(1):1–21, 2022. URL https://doi.org/10.1515/em-2022-0108. [p37, 39, 50]

S. Zetterstrom, A. Sjölander, and I. Waernbaum. Investigations of sharp bounds for causal effects under selection bias. *Statistical Methods in Medical Research*, 34(12):09622802251374168, 2025. doi: 10.1177/09622802251374168. URL https://doi.org/10.1177/09622802251374168. [p37, 39, 40, 45, 46, 47, 50]

*Stina Zetterstrom*
*Uppsala University*
*Department of Statistics*
*Uppsala, Sweden*
https://www.statistik.uu.se/?languageId=1
*ORCID: 0000-0003-0730-9425*
stina.zetterstrom@gmail.com

*Ingeborg Waernbaum*
*Uppsala University*
*Department of Statistics*
*Uppsala, Sweden*
https://www.statistik.uu.se/?languageId=1
*ORCID: 0000-0002-4457-5311*
ingeborg.waernbaum@statistics.uu.se

# psvmSDR: An R Package for a Unified Algorithm for Sufficient Dimension Reduction via Principal Machines

*by Jungmin Shin, Seung Jun Shin and Andreas Artemiou*

**Abstract** Sufficient dimension reduction (SDR), which seeks a lower-dimensional subspace of the predictors containing regression or classification information, has been popular in a machine learning community. In this work, we present a new R software package **psvmSDR** that implements a new class of SDR estimators, which we call the principal machine (PM) generalized from the principal support vector machine (PSVM). The package covers both linear and nonlinear SDR and provides a function applicable to realtime update scenarios. The package implements the descent algorithm for the PMs to efficiently compute the SDR estimators in various situations. This easy-to-use package will be an attractive alternative to the **dr** R package that implements classical SDR methods.

## 1 Introduction

Dimension reduction is essential in modern applications in statistics and machine learning as the data size grows. In this regard, the **sufficient dimension reduction** (SDR, Li, 1991; Cook, 1998) has gained great popularity in supervised learning problems. For a given pair of univariate response $Y$ and $p$-dimensional predictor $\mathbf{X}$, SDR assumes

$$Y \perp \mathbf{X} \mid \mathbf{B}^\top \mathbf{X}, \tag{1}$$

where $\perp$ denotes statistical independence. Under (1), SDR seeks the minimal space spanned by $\mathbf{B} \in \mathbb{R}^{p\times d}$, called the *central subspace*, and denoted by $\mathcal{S}_{Y|\mathbf{X}}$ that contains complete regression information of $Y$ on $\mathbf{X}$. We assume $\mathcal{S}_{Y|\mathbf{X}} = \text{span}\{\mathbf{B}\}$ in (1). The dimension of $\mathcal{S}_{Y|\mathbf{X}}$, $d$ is also an important quantity to be estimated from the data.

There are numerous SDR estimators to estimate $\mathcal{S}_{Y|\mathbf{X}}$. Earliest proposals are based on an inverse moment, such as sliced inverse regression (SIR, Li, 1991) and sliced average variance estimates (SAVE, Cook and Weisberg, 1991). Other methods are based on forward regression models, including principal Hessian directions (pHd, Li, 1992) and minimum average variance estimation (MAVE, Xia et al., 2002). Several R packages are available for the practical implementation of these classical SDR methods. The **dr** package (Weisberg, 2015), for instance, concisely implements many of these techniques (e.g., SIR, SAVE, pHd). Other packages such as **itdr** (De Alwis et al., 2024) and **orthoDr** (Zhu et al., 2019) also provide functions for various SDR approaches.

The assumption (1) is often called a linear SDR. Cook (2007) generalized (1), and proposed a nonlinear SDR that pursues a function $\phi : \mathbb{R}^p \to \mathbb{R}^d$ satisfying

$$Y \perp \mathbf{X} \mid \phi(\mathbf{X}). \tag{2}$$

Nonlinear SDR methods have been developed by directly extending the classical inverse methods to Reproducing Kernel Hilbert Space (RKHS, Aronszajn, 1950). However, there is no hands-on R package that implements them.

Meanwhile, Li et al. (2011) proposed the principal support vector machine (PSVM), a unified framework for both linear and nonlinear SDR, by connecting the SDR problem to the support vector machine (SVM, Vapnik, 1999). More importantly, PSVM can be easily generalized to other supervised machine learning approaches other than the SVM with the hinge loss. We call the class of SDR methods that generalizes the PSVM the **principal machine** (PM, Shin and Shin, 2024). The PM methods, such as PSVM as well as principal quantile regression (PQR, Wang et al., 2018), principal least squares SVM (PLSSVM, Artemiou et al., 2021), and principal asymmetric $L_2$ regression (PALS, Soale and Dong, 2022), have reportedly outperformed classical SDR approaches.

In this article, we develop an R package **psvmSDR** that renders a simple gradient descent algorithm to compute a wide variety of the PMs in a unified framework. The package offers two main functions `psdr()` and `npsdr()` that compute a wide variety of PMs for linear and nonlinear SDR, respectively. In addition, it also has a function `rtpsdr()` that executes the realtime SDR estimation for streamed data.

We highlight the main advantages of the package **psvmSDR** in the following.

- It can solve both linear and nonlinear SDR problems in a unified framework using a simple

and efficient gradient descent algorithm.

- It can be applied to a binary classification, where most inverse-moment-based approaches suffer.
- It can compute a PM estimator with a user-specified arbitrary convex loss, adding flexibility to the methods.
- It can handle streamed data by directly updating the SDR estimator without storing the entire data.

The package is now available from CRAN at https://cran.r-project.org/web/packages/psvmSDR and can be installed from R with the following command:

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

The rest of the article is organized as follows. In Section 2.2, we introduce the principal machine (PM) under a linear, nonlinear, and realtime SDR framework along with the corresponding functions `psdr()`, `npsdr()`, and `rtpsdr()`. In Section 2.3, we describe computational details about the package, and its efficiencies are investigated. In Section 2.4, we overview the package structure and explain how to implement the functions with examples. We conclude with a summary and discussion in Section 2.6.

## 2 Principal machines: generalization of PSVM

We start by introducing the PSVM (Li et al., 2011). For a given sequence of $c_1 < c_2 < \cdots < c_h$, the PSVM minimizes the following objective function in the population level:

$$(\alpha_k, \boldsymbol{\beta}_k) = \underset{\alpha,\boldsymbol{\beta}}{\operatorname{argmin}}\, \boldsymbol{\beta}^\top \boldsymbol{\Sigma} \boldsymbol{\beta} + \lambda\, \mathbb{E}\left[\tilde{Y}_k\{\alpha + \boldsymbol{\beta}^\top(\mathbf{X} - \boldsymbol{\mu})\}\right]_+, \quad k = 1, 2, \cdots, h, \tag{3}$$

where $\boldsymbol{\mu} = \mathbb{E}(\mathbf{X})$, $\boldsymbol{\Sigma} = \operatorname{cov}(\mathbf{X})$, and $[u]_+ = \max\{0, u\}$. Here $\tilde{Y}_k$ is a pseudo-binary response, taking 1 if $Y < c_k$ and $-1$ otherwise, and a positive constant $\lambda$ is a cost parameter whose choice is not overly sensitive for estimating $\mathcal{S}_{Y|\mathbf{X}}$. Li et al. (2011) showed the unbiasedness of $\boldsymbol{\beta}_k$, i.e., $\boldsymbol{\beta}_k \in \mathcal{S}_{Y|\mathbf{X}}, \forall k = 1, 2, \cdots, h$.

It is important to note that convexity of the objective function in (3) is the only requirement for the PSVM to be unbiased, and this naturally leads to a generalized version of the PSVM, which we call the PM. We can categorize the PM into two types: one is response-based PM (RPM), and the other is loss-based PM (LPM). The RPM obtains multiple solutions by perturbing the pseudo-response $\tilde{Y}_k$, while the loss function remains unchanged. The PSVM belongs to this category. On the other hand, the LPM calculates multiple solutions by changing the loss functions, while the pseudo-response $Y_k$ is fixed as $Y$ for all $k$. The principal weighted support vector machine (PWSVM, Shin et al., 2017) and the principal quantile regression (PQR, Wang et al., 2018) are the earliest proposals of the LPM.

### 2.1 Linear principal machines

The linear PM solves

$$(\alpha_k, \boldsymbol{\beta}_k) = \underset{\alpha,\boldsymbol{\beta}}{\operatorname{argmin}}\, \boldsymbol{\beta}^\top \boldsymbol{\Sigma} \boldsymbol{\beta} + \lambda\, \mathbb{E}\left[L_k\left\{\tilde{Y}_k, f(\mathbf{X})\right\}\right], \qquad k = 1, 2, \cdots h, \tag{4}$$

where $f(\mathbf{X}) = \alpha + \boldsymbol{\beta}^\top(\mathbf{X} - \boldsymbol{\mu})$ and $L_k(y, f)$ denotes a convex loss function of a margin, $yf$ for a binary $y \in \{-1, 1\}$ or residual $y - f$ for a continuous $y \in \mathbb{R}$. The RPM solves (4) for different values of $\tilde{Y}_k$ over $k$ to obtain multiple solutions while keeping $L_k$ fixed as, say, $L$. On the other hand, the LPM solves it with different loss functions $L_k$ while a pseudo-response $\tilde{Y}_k$ remains fixed as the original response $Y$. For instance, PWSVM, the first proposal under the LPM framework, solves (4) using the loss function $L_k(y, f) = \pi_k(y)[1 - yf]_+$, where $\pi_k(y) = c_k$ if $y = 1$, and $\pi_k(y) = 1 - c_k$ otherwise, for a given $c_k \in (0, 1)$ while $\tilde{Y}$ is fixed as $Y \in \{-1, 1\}$. The PWSVM is proposed for SDR in binary classification problems, addressing the limitations of PSVM when the dimensionality $d > 1$.

Given $(y_i, \mathbf{x}_i), i = 1, 2, \cdots, n$, the sample counter part of (4) is

$$(\hat{\alpha}_k, \hat{\boldsymbol{\beta}}_k) = \underset{\alpha,\boldsymbol{\beta}}{\operatorname{argmin}}\, \boldsymbol{\beta}^\top \hat{\boldsymbol{\Sigma}} \boldsymbol{\beta} + \frac{\lambda}{n} \sum_{i=1}^{n} L_k\left(\tilde{y}_{k,i}, \alpha + \boldsymbol{\beta}^\top \mathbf{z}_i\right), \qquad k = 1, 2, \cdots h, \tag{5}$$

| Type | Method | Response | Margin (Residual) | Loss | `loss` |
|---|---|---|---|---|---|
| RPM | SVM | $Y \in \mathbb{R}$, $\tilde{Y}_k = \mathbb{1}\{Y \geq c_k\} - \mathbb{1}\{Y < c_k\}$ | $m_k = \tilde{Y}_k f$ | $[1 - m_k]_+$ | `'svm'` |
| | Logistic | | | $\log(1 + e^{-m_k})^{-1}$ | `'logit'` |
| | $L_2$-SVM | | | $\{[1 - m_k]_+\}^2$ | `'l2svm'` |
| | LS-SVM | | | $[1 - m_k]^2$ | `'lssvm'` |
| LPM | WSVM | $Y = \tilde{Y}_k \in \{-1, 1\}$ | $m = Yf$ | $\pi_k(Y)[1 - m]_+$ | `'wsvm'` |
| | Wlogistic | | | $\pi_k(Y) \log(1 + e^{-m})^{-1}$ | `'wlogit'` |
| | W$L_2$-SVM | | | $\pi_k(Y)\{[1 - m]_+\}^2$ | `'wl2svm'` |
| | WLS-SVM | | | $\pi_k(Y)[1 - m]^2$ | `'wlssvm'` |
| | Quantile | $Y = \tilde{Y}_k \in \mathbb{R}$ | $r = Y - f$ | $r\{c_k - I(r < 0)\}$ | `'qr'` |
| | Asym. LS | | | $r^2\{c_k - I(r < 0)\}$ | `'asls'` |

**Table 1:** Different types of the convex loss functions available in the package **psvmSDR**. The column `loss` indicates a specific syntax of the argument that is passed to `psdr()` and `npsdr()` to implement the corresponding methods.

where $\mathbf{z}_i = \mathbf{x}_i - \sum_{i=1}^n \mathbf{x}_i/n$ and $\hat{\mathbf{\Sigma}} = n^{-1} \sum_{i=1}^n \mathbf{z}_i \mathbf{z}_i^\top$ denotes a sample covariance matrix. The PM working matrix is then

$$\hat{\mathbf{M}} = \sum_{k=1}^{h} \hat{\boldsymbol{\beta}}_k \hat{\boldsymbol{\beta}}_k^\top, \tag{6}$$

and its first $d$ eigenvectors of $\hat{\mathbf{M}}$ estimate $\mathbf{B}$ under (1), where $d$ denotes the dimension of $\mathcal{S}_{Y|\mathbf{X}}$, often called the *structural dimension*. The **psvmSDR** package offers `psdr()` function to compute the linear PM estimates with a given loss function.

In the linear SDR, it is also crucial to estimate the structural dimension $d$. The linear PM employs the following BIC-type criterion proposed by Li et al. (2011) to estimate $d$:

$$\hat{d} = \underset{d \in \{1, \cdots, p\}}{\operatorname{argmax}} \sum_{j=1}^{d} v_j - \rho \frac{d \log n}{\sqrt{n}} v_1, \tag{7}$$

where $v_1 \geq \cdots \geq v_p (p > d)$ are eigenvalues of $\hat{\mathbf{M}}$ in (6), with $\rho$ being a hyperparameter that can be chosen in a data-adaptive manner. The **psvmSDR** package contains `psdr_bic()` function for estimating $d$ based on (7).

There are numerous PMs by employing a variety of convex loss functions in supervised learning problems. For RPMs, the principal logistic regression with negative log-likelihood loss (PLR, Shin and Artemiou, 2017), the principal $L_q$-SVM with $L_q$ hinge loss ($L_q$-PSVM, Artemiou and Dong, 2016), and the principal least square SVM with the squared loss (PLSSVM, Artemiou et al., 2021) are proposed. For LPMs, Wang et al. (2018) suggested the principal quantile regression (PQR) with the check loss, Kim and Shin (2019) proposed the principal weighted logistic regression (PWLR), Soale and Dong (2022) introduced the principal asymmetric least square regression (PALSR) with the asymmetric least square loss, and Jang et al. (2023) proposed the principal weighted least square SVM (PWLSSVM).

Table 1 provides a complete list of PMs implemented in **psvmSDR** with the corresponding loss functions.

### 2.2 Kernel principal machines for nonlinear SDR

A nonlinear generalization of (4) under (2) is

$$(\alpha_{0,k}, g_{0,k}) = \underset{\alpha \in \mathbb{R}, g \in \mathcal{H}}{\operatorname{argmin}} \operatorname{var}\{\psi(\mathbf{X})\} + \lambda \mathbb{E}\left[L_k\left\{\tilde{Y}_k, f(\mathbf{X})\right\}\right], \qquad k = 1, 2, \cdots, h, \tag{8}$$

where $f(\mathbf{X}) = \alpha + g(\mathbf{X}) - \mathbb{E}\{g(\mathbf{X})\}$ with $g$ being a function in a Hilbert space $\mathcal{H}$ of functions of $\mathbf{X}$. Let $(\alpha_{0,k}, g_{0,k})$ be the minimizer of (8), then $g_{0,k}(\mathbf{X})$ is necessarily a function of the sufficient predictor $\phi(\mathbf{X})$ in (2) and its unbiasedness for the nonlinear SDR is established by Li et al. (2011).

Employing the reproducing kernel Hilbert space (RKHS) generated by a positive definite kernel

$K(\cdot,\cdot)$ for the space of $g$, we have the following finite-dimensional basis representation of $g$:

$$g(\mathbf{x}) = \sum_{j=1}^{b} \beta_j \left\{ \psi_j(\mathbf{x}) - \bar{\psi}_j \right\} \tag{9}$$

with

$$\psi_j(\mathbf{x}) = \{\mathbf{k}(\mathbf{x})\}^\top \mathbf{q}_j / \lambda_j, \quad j = 1, \cdots b, \qquad \text{and} \qquad \bar{\psi}_j = \sum_{i=1}^{n} \psi_j(\mathbf{x}_i)/n, \tag{10}$$

where $\mathbf{k}(\cdot) = \{K(\cdot, \mathbf{x}_i) : i = 1, \ldots, n\}^\top$, and $\mathbf{q}_j$ and $\lambda_j$ are the $j$th leading eigenvector and eigenvalue of the centered kernel matrix $\mathbf{K} = \{K_{ij}\} \in \mathbb{R}^{n\times n}$ with $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) - \sum_{k=1}^{n} K(\mathbf{x}_k, \mathbf{x}_j)/n$. We refer Section 6 of Li et al. (2011) for the theoretical justification for (9) and (10). In **psvmSDR**, we employ the radial/Gaussian kernel and set $b = n/3$ as recommended by Li et al. (2011).

By employing (9), the sample counter part of (8) is

$$(\hat{\alpha}_k, \hat{\boldsymbol{\beta}}_k) = \underset{\alpha,\boldsymbol{\beta}}{\operatorname{argmin}} \boldsymbol{\beta}^\top \hat{\boldsymbol{\Sigma}} \boldsymbol{\beta} + \frac{\lambda}{n} \sum_{i=1}^{n} L_k \left( \tilde{y}_{k,i}, \alpha + \boldsymbol{\beta}^\top \mathbf{z}_i \right), \qquad k = 1, 2, \cdots, h, \tag{11}$$

where $\mathbf{z}_i = \sum_{j=1}^{b} \psi_j(\mathbf{x}_i) - \bar{\psi}_j$ and $\hat{\boldsymbol{\Sigma}} = n^{-1} \sum_{i=1}^{n} \mathbf{z}_i \mathbf{z}_i^\top$. We note that this kernel PM in (11) is also linear in the parameter, exactly the same as the linear PM (5). For a given $\mathbf{x}$, one can compute the sufficient predictors, $\hat{\phi}(\mathbf{x}) = \hat{\mathbf{V}}^\top \{\psi_1(\mathbf{x}), \ldots, \psi_b(\mathbf{x})\}^\top$, where $\hat{\mathbf{V}}$ denotes the $d$-leading eigenvectors of $\sum_{k=1}^{h} \hat{\boldsymbol{\beta}}_k \hat{\boldsymbol{\beta}}_k^\top$. The **psvmSDR** package offers `npsdr()` function to compute the nonlinear PM estimates.

### 2.3 Principal least square machines for realtime SDR

When data is collected in a streamed fashion, it is prohibitive to use all data to compute the SDR estimator due to memory constraints. Therefore, it is important to develop real-time SDR algorithms that directly update the SDR estimators without storing the entire data. In this regard, Artemiou et al. (2021) proposed PLSSVM which solves (4) with the squared loss $L(\tilde{y}, f) = (1 - \tilde{y}f)^2$.

Let's consider the following scenario. In addition to the currently available (old) data $\mathbb{D}_{\mathtt{O}} = (y_i, \mathbf{x}_i), i = 1, 2, \cdots, n$, suppose $m$ additional (new) data $\mathbb{D}_{\mathtt{N}} = (y_i, \mathbf{x}_i), i = n+1, \cdots, n+m$ arrives. Let $\mathbb{D}_{\mathtt{W}} = \mathbb{D}_{\mathtt{O}} \cup \mathbb{D}_{\mathtt{N}}$ denote the whole data. The subscripts $\mathtt{O}, \mathtt{N}$, and $\mathtt{W}$ are used to denote the quantities related to old, new, and whole data, respectively. Artemiou et al. (2021) showed that the PLSSVM solution for the whole data, $\mathbf{r}_{\mathtt{W}} = (\hat{\alpha}_{\mathtt{W}}, \hat{\boldsymbol{\beta}}_{\mathtt{W}}^\top)^\top \in \mathbb{R}^{p+1}$, is given by

$$\mathbf{r}_{\mathtt{W}} = \{\mathbf{I} - \mathbf{A}_{\mathtt{O}}^{-1} \mathbf{B}_{\mathtt{N}} (\mathbf{I} + \mathbf{A}_{\mathtt{O}}^{-1} \mathbf{B}_{\mathtt{N}})^{-1}\} (\mathbf{r}_{\mathtt{O}} + \mathbf{A}_{\mathtt{O}}^{-1} \mathbf{c}_{\mathtt{N}}), \tag{12}$$

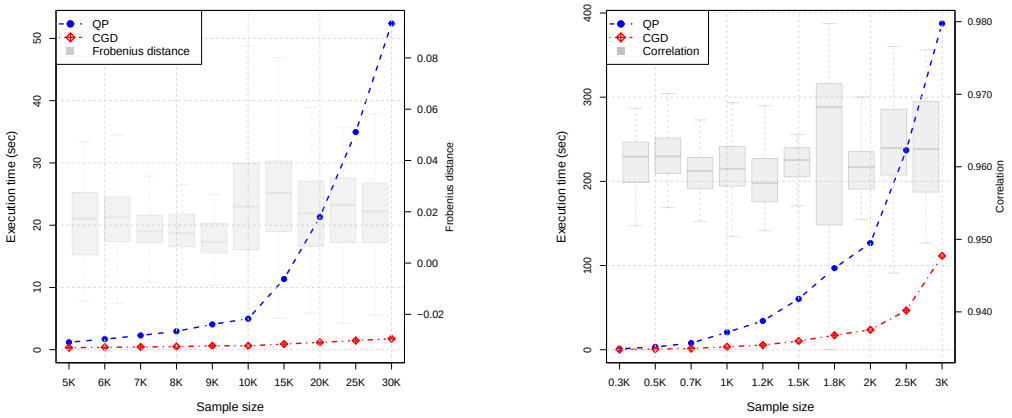
where $\mathbf{r}_{\mathtt{O}}$ is the solution for $\mathbb{D}_{\mathtt{O}}$, and $\mathbf{A}_{\mathtt{O}}^{-1} \in \mathbb{R}^{(p+1)\times(p+1)}$ is also computable from $\mathbb{D}_{\mathtt{O}}$, while $\mathbf{B}_{\mathtt{N}} \in \mathbb{R}^{(p+1)\times(p+1)}$ and $\mathbf{c}_{\mathtt{N}} \in \mathbb{R}^{p+1}$ are computable from $\mathbb{D}_{\mathtt{N}}$. That is, one can directly update $\mathbf{r}_{\mathtt{W}}$ from $\mathbf{r}_{\mathtt{O}}$ without storing the entire $\mathbb{D}_{\mathtt{O}}$, but $\mathbf{A}_{\mathtt{O}}$ only. We remark that $\mathbf{A}_{\mathtt{W}}$ can be directly updated from $\mathbf{A}_{\mathtt{O}}$ and the new data, and its computational complexity does not depend on the size of $\mathbb{D}_{\mathtt{O}}$. We refer Section 3 of Artemiou et al. (2021) for the exact forms of these quantities.

The **psvmSDR** package offers `rtpsdr()` that implements the realtime SDR using PLSSVM for a continuous response as well as PWLSSVM by Jang et al. (2023) for a binary response.

## 3 Computation

In this section, we describe computational details about `psdr()` and `npsdr()` in **psvmSDR**. Slightly abusing notation as $\boldsymbol{\beta}^\top = \left(\alpha, \boldsymbol{\beta}^\top\right)$ and $\mathbf{z}_i^\top = \left(1, \mathbf{z}_i^\top\right)$, the objective functions (5) and (11) are then rewritten as

$$L(\boldsymbol{\beta}) = \boldsymbol{\beta}^\top \operatorname{Diag}\left\{0, \hat{\boldsymbol{\Sigma}}\right\} \boldsymbol{\beta} + \frac{\lambda}{n} \sum_{i=1}^{n} L_k \left( \tilde{y}_{k,i}, \boldsymbol{\beta}^\top \mathbf{z}_i \right). \tag{13}$$




**(a)** Linear PSVM **(b)** Kernel PSVM

**Figure 1:** Average computing times of the QP and CGD algorithms over 100 repetitions are shown. The left and right panels correspond to the linear and kernel PSVM results, respectively. The blue and red dashed lines denote QP and CGD, respectively, illustrating the superior scalability of CGD. Overlaid gray boxplots depict estimation agreement measured by Frobenius distance for the linear case and prediction correlation for the nonlinear case.

We propose the coordinatewise gradient descent (CGD) algorithm to minimize $L(\boldsymbol{\beta})$ in (13), which updates $\boldsymbol{\beta}$ coordinatewisely as follows until converge:

$$\beta_j \leftarrow \beta_j - \eta \frac{\partial L(\boldsymbol{\beta})}{\partial \beta_j}, \quad j = 0, 1, 2, \cdots, p,$$

where $\eta > 0$ denotes a learning rate determined by the user. We remark that some loss functions, such as the hinge loss and check loss are not theoretically differentiable at most finite number of points while numerically differentiable. In practice, the CGD implementation ensures stable updates for non-smooth functions by employing analytical subgradients for built-in losses (e.g., hinge, quantile) and numerical approximations for user-defined losses. The two main functions `psdr()` and `npsdr()` in **psvmSDR** solve linear and nonlinear PMs via this CGD algorithm, respectively, and cover a wide variety of PM estimators by simply changing the loss functions (via `` `loss' `` argument) as listed in Table 1. The CGD algorithm is easily extended to any user-defined convex loss function, since the corresponding derivative can be readily evaluated numerically. Both functions can take the name of the user-defined-loss function object as the `loss` argument which brings further flexibility to the PM estimators.

Each CGD iteration costs $\mathcal{O}(np)$ for the linear PM (`psdr()`) and $\mathcal{O}(nb)$ for the nonlinear PM (`npsdr()`), where $p$ is the data dimension and $b$ is the number of kernel basis functions used for the RKHS expansion in (10) (Luo and Tseng, 1992; Friedman et al., 2007). Following the empirical recommendation of Li et al. (2011), we set $b$ between $n/3$ and $2n/3$ in practice. In contrast, quadratic programming (QP) approaches for SVM-like objectives (e.g., **kernlab**) have worst-case polynomial time $\mathcal{O}(n^3p^3)$ (or $\mathcal{O}(n^3b^3)$ for kernel PMs) (Platt, 1998; Cristianini and Shawe-Taylor, 2000), which scale less favorably as $n$ grows.

To evaluate the computational efficiency of the proposed CGD algorithm, we compare its computing times for both linear and nonlinear (kernel) PSVM estimators to `ipop()` function in **kernlab**, a standard QP solver for the SVM over 100 independent repetitions. A toy data is generated from the following regression model:

$$y_i = x_{i1} / \left\{0.5 + (x_{i2} + 1)^2\right\} + 0.2\epsilon_i, \tag{14}$$

where both $x_{ij}$ and $\epsilon_i$ are i.i.d. random samples from $N(0, 1)$ for $i = 1, \cdots, n$ and $j = 1, 2, \cdots, 5$. Different sample sizes $n$ are considered from $5,000$ to $30,000$ for linear PSVM while from 300 to $3,000$ for the kernel PSVM where the parameter dimension depends on $n$.

Figure 1 highlights the computational advantage of the CGD implementation, which achieves substantially lower runtimes than QP. In addition to runtime, we assessed estimation agreement. For the linear PSVM, we measured subspace similarity via the Frobenius distance between projection matrices, $|\mathbf{P}_{\text{CGD}} - \mathbf{P}_{\text{QP}}|_F$, where $\mathbf{P} = \hat{\mathbf{B}}(\hat{\mathbf{B}}^\top \hat{\mathbf{B}})^{-1}\hat{\mathbf{B}}^\top$. For the kernel PSVM, predictive agreement

| Function | Description | Class | Compatible methods |
|---|---|---|---|
| `psdr()` | Basic function for applying linear PMs with various loss functions | `'psdr'` | `psdr_bic()`<br>`plot.psdr()`<br>`print.psdr()`<br>`summary.psdr()` |
| `rtpsdr()` | Real-time sufficient dimension reduction through principal (weighted) least squares SVM | | |
| `npsdr()` | Basic function for applying kernel PMs with various loss functions | `'npsdr'` | `npsdr_x()`<br>`plot.npsdr()`<br>`print.npsdr()`<br>`summary.npsdr()` |

**Table 2:** Main functions and compatible methods in the **psvmSDR** package.

is quantified by the correlation between fitted test responses, $\text{cor}(\hat{y}_{\text{QP}}, \hat{y}_{\text{CGD}})$, where $\hat{y}_{\text{QP}}$ and $\hat{y}_{\text{CGD}}$ are obtained from regressions of $y$ on the estimated sufficient predictors $\hat{\phi}_{\text{QP}}(\mathbf{X})$ and $\hat{\phi}_{\text{CGD}}(\mathbf{X})$. Across all sample sizes, the Frobenius distance remained near zero and the correlation close to one, confirming that CGD achieves accuracy indistinguishable from QP while being substantially faster.

# 4 Package overview and implementation

## 4.1 Overview of the package psvmSDR

The **psvmSDR** package is organized to allow users to run different types of PM methods in a unified fashion by calling just a few interface functions. Several auxiliary codes are then internally called to perform computations according to the options specified in those interface functions. The overall design of **psvmSDR** adopts the functional object-oriented programming approach (Chambers, 2014) with `S3` classes and methods. Every function in the package is either a wrapper that creates a single instance of an object or a method that can be applied to a class object.

The package can be installed and loaded in an R session via:

```
> install.packages("psvmSDR")
> library("psvmSDR")
```

The main functions in the package are `psdr()` and `npsdr()`. These are designed to be analogous to linear and nonlinear principal sufficient dimension reduction methods, respectively. Each function returns an `S3` object, `` `psdr' `` and `` `npsdr' ``, respectively. Also, compatible `S3` methods `plot()` and `print()` let the users enjoy the result of dimension reduction through familiar generic functions. Moreover, some explicit methods of which are related to the main functions are also provided, such as `psdr_bic()` and `npsdr_x()`.

A function `rtpsdr()` implements the PLSSVM in a realtime fashion (Artemiou et al., 2021; Jang et al., 2023) discussed in Section 2.2.3. Table 2 summarizes the main functions along with their compatible methods. In the following subsections, we demonstrate the general usage of the main functions and methods of **psvmSDR** by the `S3` class of the function.

## 4.2 Functions for `S3` class 'psdr'

A function `psdr`() provides a unified interface for all linear PMs solved by the CGD algorithm. The following arguments are used (some are mandatory) when calling `psdr`() function and they must be provided in the order that follows:

```
> psdr <- function(x, y, loss = "svm", h = 10, lambda = 1, eps = 1e-5, max.iter = 100,
                   eta = 0.1, mtype = "m", plot = FALSE)
```

- `x`: input matrix; each row is an observation vector. `x` should have 2 or more columns.
- `y`: response vector, either can be continuous or $\{1, -1\}$-coded binary variable.
- `loss`: name of loss function objects listed in Table 1, or the name of a user-defined (convex) loss function object.

- `h`: unified control for slicing or weighting; either a positive integer (number of slices/weights) or a numeric vector of cutpoints in $(0, 1)$.
- `lambda`: cost parameter.
- `eps`: stopping criterion of the CGD algorithm.
- `max.iter`: maximum iteration number of the CGD algorithm.
- `eta`: learning rate for the CGD algorithm.
- `mtype`: a margin type, which is either margin (`"m"`) or residual (`"r"`) (See, Table 1). Only needed when a user-defined loss is used. The default is "m".
- `plot`: boolean. If `TRUE`, scatter plots of $Y$ and sufficient predictors will be presented.

The main function `psdr()` returns an object with `S3` class `'psdr'` including

- `evalues`: eigenvalues of the estimated working matrix (6).
- `evectors`: eigenvectors of the estimated working matrix (6).
- `fit`: metadata (n, p, hyperparameters, per-slice iteration/convergence info, etc.).

To further illustrate how to use function `psdr()`, we reuse the toy example which is formulated in (14) with $n = 200$.

```
> set.seed(100)
> n <- 200; p <- 5
> x <- matrix(rnorm(n*p, 0, 1), n, p)
> y <- x[,1]/(0.5 + (x[,2] + 1)^2) + 0.2 * rnorm(n)
```

We then can apply the PSVM (Li et al., 2011) using function `psdr()` with calling the argument `loss = "svm"` (default value) as below:

```
> obj <- psdr(x, y)
> summary(obj)
=== Summary of psdr Object ===
Loss function: svm
Sample size (n): 200  | Variables (p): 5  | Response type: continuous
Lambda: 1  | Eta: 0.1  | Eps: 1e-05  | Max.iter: 100

--- Eigen Decomposition of Working Matrix (M) ---
Top eigenvalues (up to 10):
[1] 0.7803 0.0451 0.0070 0.0013 0.0002

Estimated Eigenvectors (columns = central subspace basis):
        [,1]    [,2]    [,3]    [,4]    [,5]
[1,]  0.9963  0.0003 -0.0069  0.0792  0.0328
[2,]  0.0020 -0.9578  0.1250 -0.1082  0.2352
[3,] -0.0189 -0.2385  0.1258  0.5707 -0.7754
[4,] -0.0451  0.1507  0.7675  0.4600  0.4178
[5,]  0.0708  0.0553  0.6159 -0.6669 -0.4096

--- Per-slice diagnostics ---
  slice iter converged          obj
1     1   46      TRUE 6.468460e-06
2     2   46      TRUE 6.112350e-06
3     3   45      TRUE 5.661422e-06
4     4   46      TRUE 4.309816e-06
5     5   45      TRUE 4.690990e-06
6     6   45      TRUE 5.603063e-06
7     7   45      TRUE 6.321184e-06
8     8   46      TRUE 5.886324e-06
9     9   46      TRUE 6.368887e-06

Convergence summary:
  Total iterations: 410  | All slices converged: TRUE
```

The output eigenvectors correctly estimate the true basis vectors $(1, 0, 0, 0, 0)^\top$ and $(0, 1, 0, 0, 0)^\top$. The `summary()` function returns a summary of a `psdr` object. More detailed usage of the function and its arguments is provided in the package manual. The method `plot.psdr()` creates the scatter plots between response and the $j$-th sufficient predictors.

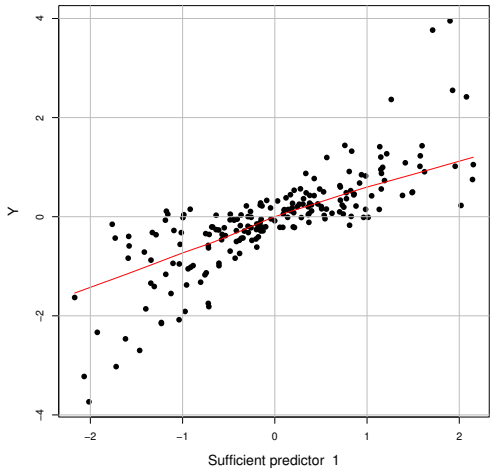



**Figure 2:** Scatter plots of $Y$ versus the first sufficient predictor $\hat{B}_1^\top \mathbf{X}$ from the generic method `plot.psdr()`.

```
> plot(obj, d=1, lowess=TRUE)
```

The argument `obj` is the object from the function `psdr()` which contains information on estimated $\mathcal{S}_{Y|\mathbf{X}}$. The number of sufficient predictors to be plotted is specified by the argument `d` whose default value is 1. By default, a locally weighted scatterplot smoothing (LOWESS) curve is plotted, unless `lowess=FALSE` is specified. The argument `"..."` allows for additional arguments to be passed to generic `plot()` function such as `main`, `cex`, `col`, `lty`, and etc.

Now, we generate binary response $\tilde{y}_i = \text{sign}(y_i)$ from (14) to illustrate the PWSVM (Shin et al., 2017), an example of the LPM. One can apply PWSVM by letting `loss = "wsvm"`.

```
> y.binary <- sign(y)
> obj_wsvm <- psdr(x, y.binary, loss = "wsvm")
> print(obj_wsvm)
> plot(obj_wsvm)

--- Principal Sufficient Dimension Reduction (linear) ---
Loss: wsvm  | n: 200  p: 5  | Response: binary
Lambda: 1  | Eta: 0.1  | Max.iter: 100  | Converged: TRUE

Eigenvalues (first 5):  0.3864, 0, 0, 0, 0

Eigenvectors (columns are SDR directions):
        [,1]    [,2]    [,3]    [,4]    [,5]
[1,]  0.9938 -0.0392  0.0572  0.0000  0.0869
[2,]  0.0590  0.8544 -0.0892 -0.4531 -0.2308
[3,] -0.0181 -0.4386 -0.2064 -0.8624  0.1449
[4,]  0.0201  0.1295 -0.8786  0.2124  0.4071
[5,]  0.0903 -0.2437 -0.4174  0.0762 -0.8674
```

Figure 3 visualizes the classification performance with only two sufficient predictors estimated by PWSVM. With only two sufficient predictors, the two classes are well separated.

One of many advantages of `psdr()` is that it can adopt any convex user-defined loss function in addition to existing choices listed in Table 1. A basic syntax of the arbitrary loss function follows:

```
loss_name <- function(u, ...){ body of a function }
```

Argument `u` is a variable of a function (any character is possible) and any additional parameters of the loss function can be specified via `...` argument. For example, one can define `"mylogistic"` and apply it to the function `psdr()` by specifying `loss="mylogistic"`.

```
> mylogistic <- function(u) log(1+exp(-u))
> obj_mylogistic <- psdr(x, y, loss="mylogistic")
> print(obj_mylogistic)
```

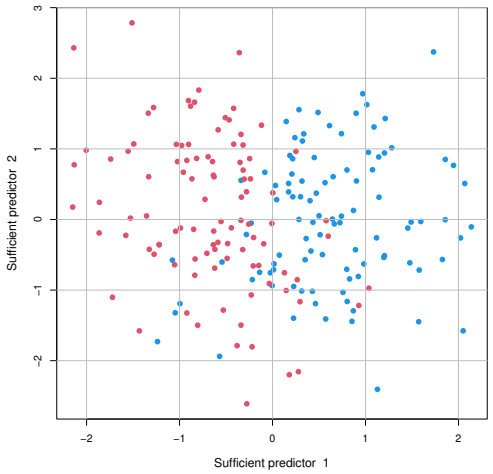



**Figure 3:** A scatter plot of the predicted projections on the first and second sufficient predictors, i.e., $\hat{b}_1^T\mathbf{X}$, $\hat{b}_2^T\mathbf{X}$, respectively, estimated by PWSVM with `loss="wsvm"`.

```
--- Principal Sufficient Dimension Reduction (linear) ---
Loss: mylogistic  | n: 200  p: 5  | Response: continuous
Lambda: 1  | Eta: 0.1  | Max.iter: 100  | Converged: TRUE

Eigenvalues (first 5):  0.0337, 0.0011, 3e-04, 1e-04, 0

Eigenvectors (columns are SDR directions):
        [,1]    [,2]    [,3]    [,4]    [,5]
[1,]  0.9948 -0.0845  0.0499  0.0214 -0.0143
[2,] -0.0910 -0.9651  0.1157  0.0039 -0.2165
[3,] -0.0001 -0.2001  0.0673 -0.3248  0.9219
[4,] -0.0444  0.1015  0.9337  0.3331  0.0712
[5,] -0.0070 -0.1054 -0.3285  0.8849  0.3128
```

As aforementioned, **psvmSDR** provides a function `psdr_bic()` that implements the BIC-type criterion in (7). Here, $\rho$ serves as a tuning parameter. To determine its optimal value, `psdr_bic` employs a $K$-fold cross-validation procedure that selects the $\rho$ yielding the most stable dimension estimates across folds.

Following arguments are mandatory when calling the `psdr_bic()`:

```
psdr_bic <- function(obj, rho_grid = seq(0.001, 0.05, length = 10), cv_folds = 5,
                     plot = TRUE, seed = 123, ...)
```

- `obj`: the output object from the main function `psdr()`
- `rho_grid`: numeric vector of candidate $\rho$ values; the default is `seq(0.001,0.05,length = 10)`.
- `cv_folds`: number of cross-validation folds used for stability evaluation (default = 5).
- `plot`: Boolean. If `TRUE`, the plot of BIC values is depicted.
- `seed`: random seed for reproducibility.
- $\cdots$: additional arguments to be passed to generic `plot()`.

The function `psdr_bic()` returns an object with S3 class `'psdr_bic'` containing

- `rho_star`: the selected tuning parameter $\rho$ that minimizes the cross-validated variation of dimension estimates.
- `d_hat`: the estimated structural dimension corresponding to `rho_star`.
- `G_values`: a matrix of BIC-type criterion values evaluated over candidate $\rho$ values.
- `cv_variation`: the variation (variance) of estimated dimensions across CV folds, used as a stability measure.

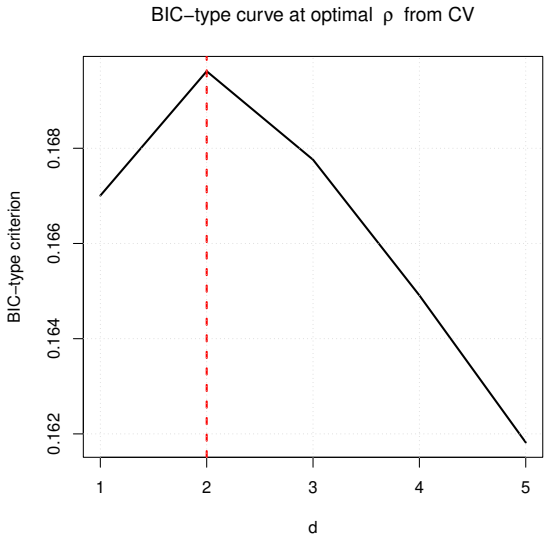



**Figure 4:** BIC-type criterion values computed from `psdr_bic()` with `plot = TRUE`.

- `fold_dhat`: the estimated dimensions obtained from each CV fold.

The following code demonstrates the use of `psdr_bic()`, which by default visualizes the BIC-type criterion curves as shown in Figure 4.

```
> d.hat <- psdr_bic(obj, rho_grid=seq(0.05, 0.1, length=5), cv_folds=5)
> print(d.hat)
$rho_star
[1] 0.05

$d_hat
[1] 2

$G_values
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.1670058 0.1662087 0.1654117 0.1646147 0.1638176
[2,] 0.1696176 0.1680236 0.1664295 0.1648354 0.1632414
[3,] 0.1677601 0.1653690 0.1629779 0.1605868 0.1581957
[4,] 0.1649097 0.1617216 0.1585334 0.1553453 0.1521572
[5,] 0.1618201 0.1578349 0.1538497 0.1498645 0.1458794

$cv_variation
rho=0.0500 rho=0.0625 rho=0.0750 rho=0.0875 rho=0.1000
       0.0        0.2        0.3        0.3        0.0

$fold_dhat
     rho=0.0500 rho=0.0625 rho=0.0750 rho=0.0875 rho=0.1000
[1,]          2          2          2          2          1
[2,]          2          2          2          2          1
[3,]          2          1          1          1          1
[4,]          2          2          1          1          1
[5,]          2          2          2          1          1

attr(,"class")
[1] "psdr_bic"
```

### 4.3 Functions for S3 class 'npsdr'

The function `npsdr()` is a nonlinear version of `psdr` that implements kernel PMs. A function `npsdr()` receives the following arguments:

```
> npsdr(x, y, loss="svm", h = 10, lambda = 1, b = floor(n/3), eps = 1.0e-5, mtype="m",
        max.iter = 100)
```

All input arguments are identical to those for `psdr()` except `b` which denotes the number of basis functions on RKHS in (10). The function `npsdr()` returns an object with `S3` class `"npsdr"` that contains the following.

- `evalues`: eigenvalues of the estimated working matrix in (6).
- `evectors`: eigenvectors of the estimated working matrix in (6).

The kernel information is stored internally and is not explicitly returned unless forced to print the object named `obj.psi`.

To further illustrate how to use function `npsdr()`, we use the following model used in Li et al. (2011):

$$y_i = 0.5\left(x_{i1}^2 + x_{i2}^2\right)^{1/2} \log\left(x_{i1}^2 + x_{i2}^2\right) + 0.2\epsilon_i, \quad i = 1, \ldots, 200. \tag{15}$$

Following code is for demonstrating kernel PSVM using function `npsdr()`.

```
> set.seed(100)
> n <- 200; p <- 5
> x <- matrix(rnorm(n*p, 0, 1), n, p)
> y <- 0.5*sqrt((x[,1]^2+x[,2]^2))*(log(x[,1]^2+x[,2]^2))+ 0.2*rnorm(n)
> obj_kernel <- npsdr(x, y, max.iter = 200, eta = 0.8, plot = FALSE)
> print(obj_kernel)

$evalues
[1] 1.724557e+02  2.108836e+01  7.665943e+00  3.202983e+00  2.171408e+00
[6] 1.608822e+00  1.029958e+00  ...
$evectors
        [,1]         [,2]          [,3]          [,4]       ...      ...
[1,] -0.0068922070 -0.0373206692 -0.180768482 -0.031654139   ...      ...
[2,] 0.1642382375 -0.0793322112 -0.084408595 -0.190419815   ...      ...
      ...            ...            ...           ...       ...      ...
 [ reached getOption("max.print") -- omitted 51 rows ]
```

Beside the main function, `npsdr_x()` computes the estimated sufficient predictors $\hat{\phi}(\mathbf{x}) = \hat{\mathbf{V}}_n^\top \{\psi_1(\mathbf{x}), \ldots, \psi_b(\mathbf{x})\}^\top$ in (2) for a given $\mathbf{x}$. The usage of a function `npsdr_x()` is given below.

```
> set.seed(200)
> new.x <- matrix(rnorm(n*p, 0, 1), n, p)
> new.y <- 0.5*sqrt((new.x[,1]^2+new.x[,2]^2))*(log(new.x[,1]^2+new.x[,2]^2))
+          + 0.2*rnorm(n)
> reduced_data <- npsdr_x(object = obj_kernel, newdata = new.x, d = 2)
```

The arguments `object` is the object from the result of `npsdr()`, and `newdata` is a new data matrix. The argument `d` is for the number of sufficient predictors and its default value is 2. In the final line of the code above, `npsdr_x()` computes $\hat{\phi}(\mathbf{x})$, the nonlinear dimension reduction of `new.x` under (2) using kernel PSVM estimates.

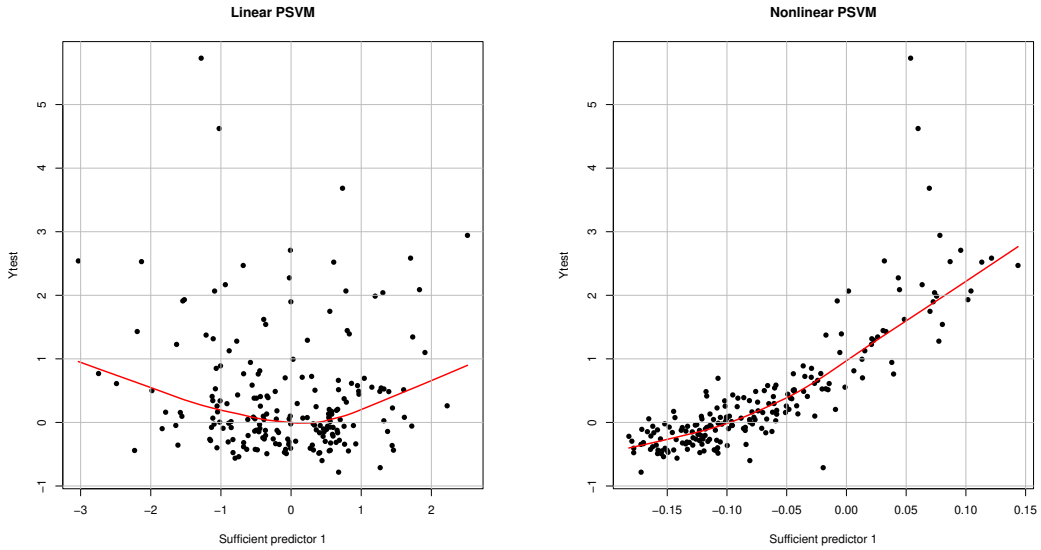
Note that the regression function (15) is symmetric about the origin. In such a scenario, linear PMs do not work and the kernel PM would be an attractive alternative. Figure 5 depicts the scatter plots of the response and the first sufficient predictor estimated by (a) the linear PSVM and (b) the kernel PSVM. As expected, the linear PSVM fails to find the central subspace when the regression function is symmetric about the origin, while the nonlinear PSVM still efficiently identifies sufficient predictors and captures the linear trend.

### 4.4 Functions for 'rtpsdr'

The package **psvmSDR** also provides `rtpsdr()` for the realtime SDR using squared loss (Artemiou et al., 2021; Jang et al., 2023), and its usage is given below.

```
> rtpsdr(x, y, obj = NULL, h = 10, lambda = 1)
```

- `x`: covariate matrix of a new data.
- `y`: response vector of a new data.




**(a)** Linear PSVM **(b)** Kernel PSVM

**Figure 5:** Scatter plots of the response and the first sufficient predictor estimated by (a) the linear PSVM and (b) the kernel PSVM.

- `obj`: the output object from `rtpsdr()` for the old data that contains the PM solutions and matrices for the old data. If it is not given, `psdr()` with the squared loss (PLSSVM) is to be applied to the given data, `y` and `x` to obtain the PM solutions and the matrices. The PWLSSVM is applied if `y` is binary.
- `h`: unified control for slicing or weighting; either a positive integer (number of slices/weights) or a numeric vector of cutpoints in $(0, 1)$.
- `lambda` hyperparameter for the loss function. The default is set to 0.1.

`rtpsdr()` returns an object `S3` class `'psdr'` that includes

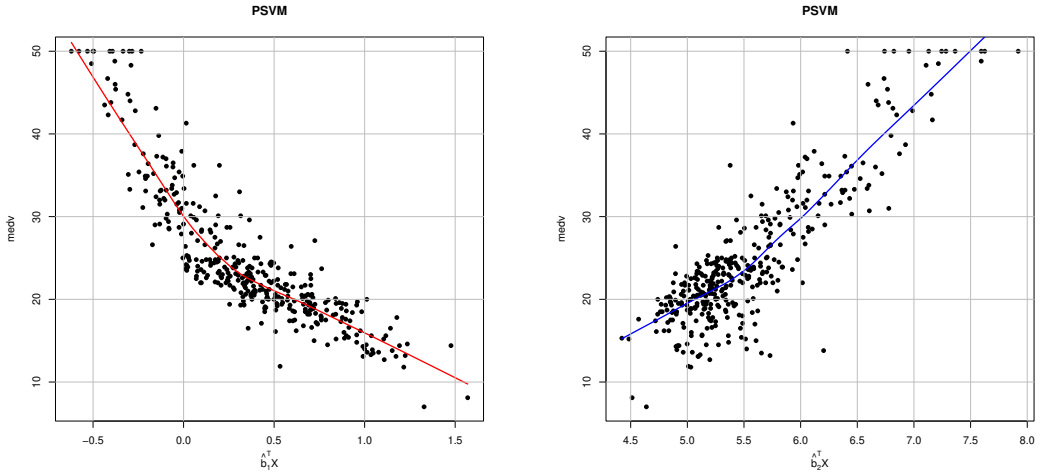
- `evalues`: eigenvalues of the estimated working matrix in (6).
- `evectors`: eigenvectors of the estimated working matrix in (6).
- `r`: the PLSSVM/PWLSSVM solutions $\mathbf{r}$ given in (12).
- `A`: updated matrix $\mathbf{A}$ for the realtime update in (12).

A returned object from `rtpsdr()` stores the state of the algorithm at each iteration $t$. This object is passed to the function as an argument and is returned at each iteration $t+1$ containing the state of the model parameters at that step.

The following command is an example of `rtpsdr()` under the streamed data scenario. The data are collected in a batch-wise manner with the batch size of $m = 500$.

```
> set.seed(1234)
> p <- 5
> m <- 500 # batch size
> N <- 10  # number of batches
> obj <- NULL
> for (iter in 1:N){
+         x <- matrix(rnorm(m*p), m, p)
+         y <-  x[,1]/(0.5 + (x[,2] + 1)^2) + 0.2 * rnorm(m)
+         obj <- rtpsdr(x = x, y = y, obj = obj)} # Real time Update
> round(obj$evectors, 3)
          [,1]     [,2]      [,3]      [,4]      [,5]
[1,]  1.000  0.005 -0.005 -0.007  0.007
[2,]  0.005 -0.999  0.037  0.011  0.013
[3,] -0.006 -0.003 -0.474  0.323  0.819
[4,] -0.006 -0.038 -0.768 -0.606 -0.206
[5,]  0.006 -0.014 -0.430  0.727 -0.535
```

For binary classification, `rtpsdr()` computes PWLSSVM solution (Jang et al., 2023). We note that `rtpsdr()` returns `psdr` object and thus `print.psdr()` and `plot.psdr()` are applicable.




**(a)** A scatter plot with lowess curve of medv versus estimated $\hat{b}_1^\top \mathbf{X}$.

**(b)** A scatter plot with lowess curve of medv versus estimated $\hat{b}_2^\top \mathbf{X}$.

**Figure 6:** Plots of the reduction of the predictors against the response of the Boston housing data using PSVM method via `psdr`() function.

## 5 Real data application

This section presents a more in-depth data analysis using the package **psvmSDR**. We demonstrate the application of the `psdr`() and `npsdr`() functions on two publicly available datasets: the Boston housing data and the Wisconsin diagnostic breast cancer data.

### 5.1 Boston housing data

We apply the proposed methods to Boston housing data (Harrison Jr and Rubinfeld, 1978), which is available on the R package **mlbench**. There are 13 predictors along with 506 observations, and the response variable $Y$ (medv) is the median of owner occupied homes in the Boston Standard Metropolitan Statistical Areas in $1,000. Some predictors are explained below: per capita crime rate by town (crim), the proportion of residential land zoned for lots over 25,000sq.ft (zn), nitricoxides concentration (nox), etc. Two categorical variables are removed, both chas and rad. Previous research on the data Chen and Li (1998) claimed to remove observations corresponding to a crime rate greater than 3.2 for the purpose of building a better model. In the end, we used 374 observations. We excluded $X_4$ in the following analysis, which represents houses with tract bounds the Charles River. The data dimension ends up to $(Y, \mathbf{X}) = \mathbb{R} \times \mathbb{R}^{12}$.

We can load data `BostonHousing` as follows:

```
> data("BostonHousing", package = "mlbench")
> attach(BostonHousing)
> BostonHousing <- BostonHousing[BostonHousing$crim < 3.2 , -c(4,9)]
> X <- as.matrix(BostonHousing[,-12])
> Y <- BostonHousing["medv"]
```

Then, we apply PSVM to the data as follows:

```
> set.seed(1)
> rslt <- psdr(X, Y)
```

After fitting the data, we visualize regression relationships between response and predictors projected onto the estimated $\mathcal{S}_{Y|\mathbf{X}}$.

```
> lsvm <- rslt$evectors
> x.lsvm <- X %*% lsvm
> plot(x.lsvm[,1], Y, type = "p", xlab = expression(hat(b)[1]^T*X), ylab="medv")
> lines(lowess( x.lsvm[,1], Y), col="red", lwd=2)
> plot(x.lsvm[,2], Y, type = "p", xlab = expression(hat(b)[2]^T*X), ylab="medv")
> lines(lowess(x.lsvm[,2], Y), col="blue", lwd=2)
```

Through PSVM, the original 12-dimensional data is reduced to two-dimensional data via linear transformation through $\hat{\mathbf{B}}$. In Figure (6), we are able to claim that the relationships between 1st and 2nd sufficient predictors and $Y$ look almost linear.

Then, we can apply `psdr_bic()` function to determine the structural dimension $d$.

```
> bic_boston <- psdr_bic(rslt, rho_grid=seq(0.005, 0.05, length=5), cv_folds=5)
> print(bic_boston$rho_star)
[1] 0.01625
> print(bic_boston$d_hat)
[1] 2
```

### 5.2 Wisconsin diagnostic breast cancer data

We use the Wisconsin Diagnostic Breast Cancer (WDBC) data which is available on UCI Machine Learning Repository (https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic). The dataset contains diagnoses of breast cancer for 569 patients with 32 predictors. We assume $d = 2$ for the purpose of visualization. We analyze both linear and nonlinear SDR cases via `psdr()` and `npsdr()`, respectively. We first load data from the repository as follows:

```
> wisc <- read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/
                      breast-cancer-wisconsin/wdbc.data", sep = ",")
> x.wisc <- matrix(unlist(wisc[,-c(1,2)]), ncol = 30)
> y.wisc <- 2*as.numeric(as.factor(unlist(wisc[,2]))) - 3
```

First, we apply principal weighted logistic regression (PWLR) for the binary response by setting `loss = "wlogit"`. Specifying `plot = TRUE` generates a scatterplot of sufficient predictors that are projected onto the estimated $\mathcal{S}_{Y|\mathbf{X}}$.

```
> psdr(x.wisc, y.wisc, loss = "wlogit", h = 20, lambda = 0.1, eta = 0.5, plot = TRUE)
```

Figure 7a displays that two classes are well separated in the estimated $\mathcal{S}_{Y|\mathbf{X}}$ which has only 2 dimensions.

Second, as a nonlinear approach, we apply the WDBC data to kernel PWLR (Shin et al., 2017) using `npsdr()` function with `loss=`wlogit'`. We employ a Gaussian kernel, with the bandwidth determined by the median of the pairwise Euclidean distances between predictors. The number of slices which is the same as `h` is set to 20. The number of basis functions for a kernel trick is set to `k=floor(length(y)/3)`.

```
> nonlinear.obj <- npsdr(x.wisc, y.wisc, loss = "wlogit", h = 20, lambda = 5, eta = 1,
                         max.iter = 30, plot = FALSE)
```

In the code below, `npsdr_x()` computes $\hat{\phi}(\mathbf{x})$ under (2) using `nonlinear.obj` estimates.

```
> x.nsvm <- npsdr_x(nonlinear.obj, newdata = x.wisc, d = 2)
```

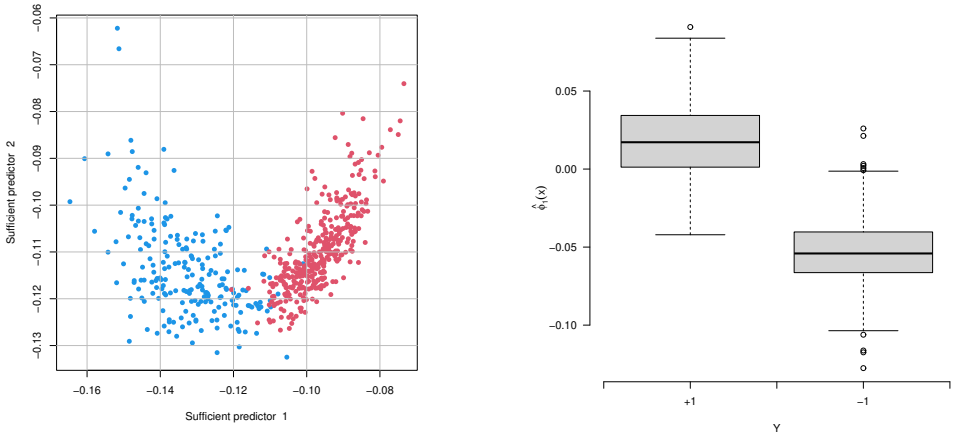
We illustrate how the two classes are well classified by the first sufficient predictor, $\hat{\phi}_1(\mathbf{x})$, using boxplots.

```
> boxplot(x.nsvm[y.wisc == 1,1], x.nsvm[y.wisc != 1,1], xlab = "Y", axes = F,
        ylab = expression(hat(phi)[1](x)))
> axis(1, seq(0.5, 2.5, by = 0.5), c(NA, "+1", NA, "-1", NA)); axis(2, las = 1)
```

The kernel PWLR performs well, as shown in Figure 7b, where the two classes are clearly separated by $\hat{\phi}_1(\mathbf{x})$.

## 6 Conclusion

In this article, we introduced and tutored the R software package **psvmSDR**, by unifying linear and nonlinear SDR under the principal machine (PM) framework. The package emphasizes its practical and convenient utilities through simple and user-friendly code syntax and access to various PMs. The use of gradient descent for computation ensures efficiency and scalability, particularly beneficial in large-scale datasets. Moreover, the package offers a realtime update scheme for batch-wise data collection scenarios. The **psvmSDR** invites researchers and practitioners in R communities to explore and use the powerful tools for the principal sufficient dimension reduction in their work.




**(a)** A scatter plot of the predicted projections, $\hat{b}_1^T\mathbf{X}$, $\hat{b}_2^T\mathbf{X}$ estimated by PWLR with the WDBC dataset. With only 2 sufficient predictors, the two classes are well separated.

**(b)** Displays boxplots of classification result using kernel PWLR. Two classes are well classified with a sole nonlinear mapping $\hat{\phi}(\mathbf{x})$.

**Figure 7:** WDBC dimension reduction result.

## Computational Details

The results in this paper were obtained using R 4.4.1 with the package **psvmSDR 3.0.1**. It is possible to have different simulation results depending on which linear algebra package is installed and which version of R is used.

## Acknowledgments

Seung Jun Shin is supported by the National Research Foundation of Korea grants (2022M3J6A1063595 and 2023R1A2C1006587) funded by the Korea government (MSIT), and by Korea University (K2521221). Seung Jun Shin is a corresponding author.

## References

N. Aronszajn. Theory of Reproducing Kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950. URL https://doi.org/10.1090/S0002-9947-1950-0051437-7. [p55]

A. Artemiou and Y. Dong. Sufficient dimension reduction via principal L $q$ support vector machine. *Electronic Journal of Statistics*, 10(1):783–805, 2016. URL https://doi.org/10.1214/16-EJS1122. [p57]

A. Artemiou, Y. Dong, and S. J. Shin. Real-time sufficient dimension reduction through principal least squares support vector machines. *Pattern Recognition*, 112:107768, 2021. URL https://doi.org/10.1016/j.patcog.2020.107768. [p55, 57, 58, 60, 65]

J. M. Chambers. Object-Oriented Programming, Functional Programming and R. *Statistical Science*, 29(2):167 – 180, 2014. URL https://doi.org/10.1214/13-STS452. [p60]

C.-H. Chen and K.-C. Li. Can SIR be as popular as multiple linear regression? *Statistica Sinica*, pages 289–316, 1998. URL https://www.jstor.org/stable/24306494. [p67]

R. D. Cook. *Regression Graphics: Ideas for Studying Regressions Through Graphics*. Wiley, New York, 1998. URL https://doi.org/10.1002/9780470316931. [p55]

R. D. Cook. Fisher Lecture: Dimension Reduction in Regression. *Statistical Science*, 22:1–26, 2007. URL https://doi.org/10.1214/088342306000000682. [p55]

R. D. Cook and S. Weisberg. Discussion of "Sliced Inverse Regression for Dimension Reduction". *Journal of the American Statistical Association*, 86:28–33, 1991. URL https://doi.org/10.2307/2290564. [p55]

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge university press, 2000. URL https://doi.org/10.1017/CBO9780511801389. [p59]

T. P. De Alwis, S. Y. Samadi, and J. Weng. *itdr: Integral Transformation Methods for SDR in Regression*, 2024. URL https://CRAN.R-project.org/package=itdr. R package version 2.0.1. [p55]

J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise Coordinate Optimization. *The Annals of Applied Statistics*, 1(2):302 – 332, 2007. URL https://doi.org/10.1214/07-AOAS131. [p59]

D. Harrison Jr and D. L. Rubinfeld. Hedonic Housing Prices and the Demand for Clean Air. *Journal of environmental economics and management*, 5(1):81–102, 1978. URL https://doi.org/10.1016/0095-0696(78)90006-2. [p67]

H. J. Jang, S. J. Shin, and A. Artemiou. Principal weighted least square support vector machine: An online dimension-reduction tool for binary classification. *Computational Statistics & Data Analysis*, 187:107818, 2023. URL https://doi.org/10.1016/j.csda.2023.107818. [p57, 58, 60, 65, 66]

B. Kim and S. J. Shin. Principal weighted logistic regression for sufficient dimension reduction in binary classification. *Journal of the Korean Statistical Society*, 48(2):194–206, 2019. URL https://doi.org/10.1016/j.jkss.2018.11.001. [p57]

B. Li, A. Artemiou, and L. Li. Principal Support Vector Machines for Linear and Nonlinear Sufficient Dimension Reduction. *Annals of Statistics*, 39(6):3182–3210, 2011. URL https://doi.org/10.1214/11-AOS932. [p55, 56, 57, 58, 59, 61, 65]

K.-C. Li. Sliced Inverse Regression for Dimension Reduction (with discussion). *Journal of the American Statistical Association*, 86:316–342, 1991. URL https://doi.org/10.2307/2290563. [p55]

K.-C. Li. On Principal Hessian Directions for Data Visualization and Dimension Reduction: Another Application of Stein's Lemma. *Journal of the American Statistical Association*, 87:1025–1039, 1992. URL https://doi.org/10.2307/2290640. [p55]

Z.-Q. Luo and P. Tseng. On the Convergence of the Coordinate Descent Method for Convex Differentiable Minimization. *Journal of Optimization Theory and Applications*, 72(1):7–35, 1992. URL https://doi.org/10.1007/BF00939948. [p59]

J. Platt. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. Technical Report MSR-TR-98-14, Microsoft, April 1998. URL https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=1f1c4b7041112940e21e19855771d12f005090b4. [p59]

J. Shin and S. J. Shin. A Concise Overview of Principal Support Vector Machines and Its Generalization. *Communications for Statistical Applications and Methods*, 31(2):235–246, 2024. URL https://doi.org/10.29220/CSAM.2024.31.2.235. [p55]

S. J. Shin and A. Artemiou. Penalized principal logistic regression for sparse sufficient dimension reduction. *Computational Statistics & Data Analysis*, 111:48–58, 2017. URL https://doi.org/10.1016/j.csda.2016.12.003. [p57]

S. J. Shin, Y. Wu, H. H. Zhang, and Y. Liu. Principal weighted support vector machines for sufficient dimension reduction in binary classification. *Biometrika*, 104(1):67–81, 2017. URL https://www.jstor.org/stable/26363643. [p56, 62, 68]

A.-N. Soale and Y. Dong. On sufficient dimension reduction via principal asymmetric least squares. *Journal of Nonparametric Statistics*, 34(1):77–94, 2022. URL https://doi.org/10.1080/10485252.2021.2025237. [p55, 57]

V. Vapnik. *The Nature of Statistical Learning Theory*. Springer science & business media, 1999. URL https://doi.org/10.1007/978-1-4757-3264-1. [p55]

C. Wang, S. J. Shin, and Y. Wu. Principal Quantile Regression for Sufficient Dimension Reduction with Heteroscedasticity. *Electronic Journal of Statistics*, 12(2):2114–2140, 2018. URL https://doi.org/10.1214/18-EJS1432. [p55, 56, 57]

S. Weisberg. Dimension Reduction Regression in R. *Journal of Statistical Software*, 7(1):1–22, 2015. URL https://cran.r-project.org/web/packages/dr. [p55]

Y. Xia, H. Tong, W. K. Li, and L.-X. Zhu. An adaptive estimation of dimension reduction space. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 64(3):363–410, 2002. [p55]

R. Zhu, J. Zhang, R. Zhao, P. Xu, W. Zhou, and X. Zhang. orthoDr: Semiparametric Dimension Reduction via Orthogonality Constrained Optimization. *The R Journal*, 11(2):24–37, 2019. doi: 10.32614/RJ-2019-006. [p55]

*Jungmin Shin*
*Department of Statistics*
*Korea University*
*02841 Anam-dong, Seongbuk-gu, Seoul*
*Republic of Korea*
*and*
*Department of Biomedical Informatics*
*The Ohio State University*
*2255 Kenny Rd, Columbus, OH 43210*
*USA*
*(ORCiD: 0009-0006-7471-0437)*
c16267@gmail.com
*https://sites.google.com/view/jungminshin/home*

*Seung Jun Shin*
*Department of Statistics*
*Korea University*
*02841 Anam-dong, Seongbuk-gu, Seoul*
*Republic of Korea*
sjshin@korea.ac.kr
*https://www.sjshin.net/*

*Andreas Artemiou*
*Department of Information Technologies*
*University of Limassol*
*21 Glafkou Clerides Avenue 2107 Aglandjia Nicosia*
*Cyprus*
artemiou@uol.ac.cy
*https://www.uol.ac.cy/team/andreas-artemiou/*

# DRIP: An R Package for Jump Regression and Image Analysis

*by Yicheng Kang*

**Abstract** Jump regression performs nonparametric regression analysis where the regression function can be discontinuous. Because of its ability to conduct jump detection and jump-preserving estimation, jump regression provides a natural statistical framework for image analysis. It enables statistical inference involved in many image processing problems including edge detection, image denoising and image deblurring. As the jump regression literature grows over the past two decades, the methodological development has far outpaced the availability of jump regression software. Although there are many image processing software programs, most of them implement either generic image operations or statistical analysis of images of specific types. In this article, we introduce DRIP, an R package with a number of state-of-the-art jump regression methods for jump detection and surface reconstruction. Its open-source nature and user-friendly interface make it convenient to develop and evaluate jump regression methods, encouraging wider adoption and future research. Although the current version of DRIP supports analysis of monochrome images only, as the literature on multivariate jump regression is still lacking, our methods can be applied separately to each color channel when handling color images.

## 1 Introduction

Rapid development in image acquisition technology has made images widely available in engineering, scientific research and everyday life. Because of their ability to capture spatial patterns and geometric information, images play increasingly important roles in areas such as astronomy, medical imaging, microscopy, motion tracking, optics, photography, and remote sensing, among others. Since image data are unstructured in nature and may contain measurement errors, analyzing images requires special techniques. In remote sensing, for instance, satellite images, which are often used in landscape monitoring, are affected by atmospheric noise due to light reflection from aerosols and clouds (Boonprong et al., 2018). In such cases, noise removal, i.e., image denoising (Buades et al., 2005; Polzehl and Spokoiny, 2000), may be necessary before any subsequent analysis. In medical diagnostics, magnetic resonance imaging (MRI) is used for measuring tumor growth in cancer patients. Images taken at different times may be misaligned because of the difference in the relative position between the patient and the imaging device. Meaningful comparison of the images for quantifying changes in tumor size cannot be done before geometrically matching up the images (Klein et al., 2009), a technique known as image registration (Zitova and Flusser, 2003). In photography, camera shake causes motion blur in images (Bai et al., 2018), making tasks such as recognizing objects in the image difficult. In such situations, a number of image deblurring techniques can be useful, e.g., Bayesian-based deblurring (Wang et al., 2011), diffusion-based deblurring (Yue et al., 2025) and deep learning-based deblurring (Quan et al., 2024).

There are a number of choices of software that implement various image processing techniques. MATLAB (The MathWorks Inc, 2023) includes an image processing toolbox with many useful tools for general-purpose image processing. A nice feature of MATLAB is that it has a built-in parallel computing toolbox to offer high-speed processing. In recent years, it has been expanded to support graphics processing unit (GPU) acceleration for some of its image processing functions (Georgantzoglou et al., 2014). A major limitation of MATLAB is that its commercial licensing can limit its large-scale adoption by practitioners. In particular, the licensing cost increases significantly for parallel computing, with per-worker pricing. Additionally, MATLAB's closed-source nature prevents users from modifying code for prototyping new methods, a common undertaking in scientific research. Among the open-source options, **OpenCV** (Bradski and Kaehler, 2008) is a comprehensive library for computer vision. It is traditionally written in C with most new algorithms developed in C++. As a result, **OpenCV** offers fast execution. Because of the same reason, it is more difficult for users to modify its code. More recently, **OpenCV** has added a Python interface to attract wider adoption. As an alternative to **OpenCV**, **scikit-image** (Van der Walt et al., 2014), written completely in Python, is an open-source library that provides users with easy access to algorithms for image processing. It offers a wide array of generic image operations such as image cropping, image resizing, and conversion between color and grayscale images. With the rising popularity of Python, **scikit-image** has seen significant growth in both adoption and contribution. Although favored by computer scientists, Python remains less popular than R (**R** Core Team, 2023) in the statistics community who are accustomed to R's rich ecosystem of complex statistical models. A number of R packages for image processing are available. Some of them are for general image processing. For example, the **EBImage** package

(Pau et al., 2010) provides general functionality for the reading, writing, processing and analysis of images. The **adimpro** package (Polzehl and Tabelow, 2007) and **aws** package (Polzehl and Tabelow, 2023; Polzehl et al., 2020) implement tools for manipulation of digital images and adaptive image denoising. The **OpenImageR** package (Mouselimis, 2023) offers functions for image filtering too. Its recent update allows the **Rcpp** functions of the **OpenImageR** package to be called in the C++ files of another package. Some other packages focus on images of certain types. For example, the **dti** package described in Tabelow et al. (2008) and Tabelow and Polzehl (2011) handles diffusion weighted imaging data with methods for structural adaptive smoothing and fiber tracking for tensor and tensor mixture models. The **landsat** package suggested in Goslee (2023) and Goslee (2011) provides radiometric and topographic correction routines for satellite imagery.

These existing image processing software either implement generic operations or carry out special-purpose analysis of images in specific categories. None of them has implemented methods in jump regression (Qiu, 2005), a general regression framework for image analysis. A major benefit with jump regression methods is that it enables statistical inference, such as surface estimation, standard error calculation and data-driven parameter selection, in many image processing problems including edge detection (Sun and Qiu, 2007), image denoising (Gijbels et al., 2006) and image deblurring (Hall and Qiu, 2007). The R package **DRIP** is created to fill this gap. Specifically, it implements two-dimensional jump regression methods with an emphasis on image processing applications. Regarding its computational performance, some of **DRIP**'s source code, large loops in particular, are written in Fortran or C to offer high execution speed. Parallel computing is also enabled in several routines to gain further acceleration. On the other hand, with the R interface, functions in DRIP are convenient to use, and their outputs are easy to visualize. As for its limitations, DRIP currently only supports analysis of grayscale images. This is because the existing jump regression literature has only considered cases of univariate responses. Much research is still needed on multivariate jump regression. As a practical workaround, our methods can be applied separately to each color channel when handling color images.

## 2 Package overview

The **DRIP** package provides functionality for three categories of image analysis problems: edge detection, edge-preserving noise removal and blind image deblurring. Edge detection is often used in image data compression. The following functions in the package are related to the task of edge detection: `stepEdge`, `roofEdge`, `dKQ`, `modify1` and `modify2`. Noise removal and image deblurring are important for improving human and machine perception of the images. The `JPLLK_surface`, `surfaceCluster` and `threeStage` functions in the package are related to noise removal. The `jpex` and `cv.jpex` functions are related to image deblurring. Furthermore, the package is designed for general image analysis without restrictions on the specific image type. However, the current use of **DRIP** focuses on grayscale images only, due to the aforementioned limitation in the jump regression literature. To use **DRIP** in practice, the workflow often involves parameter tuning before model estimation. This intended workflow is illustrated in the section of Workflow Illustration.

The remaining part of this article is organized as follows. Section 3 gives a short primer on 1-dimensional jump regression. Section 4 introduces 2-dimensional jump regression models, the related terminologies and their connections with image processing techniques. Section 5 describes edge detection routines in **DRIP**. Section 6 discusses methods included in the package for edge-preserving surface reconstruction from noisy images. Section 7 describes the problem of image deblurring and a blind deblurring algorithm in **DRIP**. Section 8 illustrates the workflow of using **DRIP**. Section 9 concludes the article and suggests future work.

## 3 A jump regression primer

The conventional 1-dimensional regression model has the following formulation:

$$Y_i = f(x_i) + \varepsilon_i, \quad i = 1, 2, \ldots, n,$$

where $\{(x_i, Y_i)\}$ are observed data, and $\varepsilon_i$'s are random errors with mean 0 and variance $\sigma^2$. This regression model implies that, at a given $x$ position, $Y$ is a random variable. Its mean value is a function of $x$. A major objective in regression analysis is to estimate the regression function $f(\cdot)$. A common assumption made by the conventional nonparametric regression methods (e.g., kernel smoothing) is that $f$ is continuous. Then a reasonable approach to estimating $f$ at position $x$ is a weighted average

of $Y_i$'s whose $x_i$ is near $x$. That is,

$$\widehat{f}(x) = \frac{\sum_i Y_i K((x_i - x)/h)}{\sum_i K((x_i - x)/h)},$$

where $h > 0$ is a bandwidth parameter, and $K(\cdot)$ is a kernel function with support $[0, 1]$. The kernel function is often chosen such that observations closer to $x$ receive heavier weights than those farther away. It can be proved that $\widehat{f}(x)$ is a consistent estimator of $f(x)$, provided that $f$ is a continuous function.

In the case where $f$ is piece-wise continuous (i.e., it has jumps), the conventional kernel smoothing estimator $\widehat{f}(x)$ is no longer consistent if $x$ is a jump position. In fact, it converges to a value between $f_-(x)$ and $f_+(x)$, where $f_\pm(x)$ denotes the one-sided limits of $f$ at $x$. Jump regression is designed to handle such scenarios and give rise to jump-preserving estimation. A fundamental idea in jump regression is to identify (explicitly or implicitly) the jump locations by estimating the one-sided limits $f_\pm$. For instance, position $x$ would be considered a continuity point if the estimated difference between the two one-sided limits, $|\widehat{f}_+(x) - \widehat{f}_-(x)|$, is smaller than a certain threshold. Then $f(x)$ can be estimated as done in the conventional kernel smoothing. On the other hand, if $|\widehat{f}_+(x) - \widehat{f}_-(x)|$ is large, then $x$ would be deemed a jump position, one of the one-sided estimates $\widehat{f}_\pm(x)$ should be chosen for estimating $f(x)$.

One significant application of jump regression is image processing where the regression function $f(\cdot, \cdot)$ is 2-dimensional. In such cases, the jump positions form jump location curves, and there are infinitely many directions for defining one-sided limits of $f$. Consequently, it becomes more complex to construct one-sided estimates in 2-dimensional jump regression. However, the fundamental idea of jump-preserving estimation remains the same: estimate the regression surface by one-sided smoothing at jump points and conventional smoothing at continuity points. The next section discusses in more detail the connection between 2-dimensional jump regression and digital image representation.

## 4 Image representation and jump regression models

Images generated by various devices such as cameras, scanners and sensors capture the spatial and amplitude behavior of the related physical scene. The spatial location is described by the x- and y-coordinates. The amplitude behavior is described by the amounts of illumination. The amplitude of color images consists of three channels: red, green and blue. For grayscale images, the amplitude is only a scalar, also known as intensity or gray level. In our discussion, all images are grayscale unless mentioned otherwise.

Images can be continuous with respect to the spatial coordinates and amplitude. In order for computers to handle an image, the image needs to be digitized. The process of digitizing the spatial location of an image is referred to as *sampling* (Gonzalez and Woods, 2018). It takes equally spaced x- and y- coordinates and represents the image by a matrix $\{f(x_i, y_j) : i = 1, \ldots, n_1, j = 1, \ldots, n_2\}$. Each $f(x_i, y_j)$, the amplitude value at location $(x_i, y_j)$, is called a pixel of the image. The sample size $n_1 \times n_2$ is the image resolution. It is noteworthy that popular image processing software programs (e.g., MATLAB) have the origin of the spatial coordinates at the upper-left corner of the image with the x-axis going vertically downward and the y-axis going horizontally to the right. In R and thus with **DRIP**, however, the spatial coordinates of an image have their origin at the lower-left corner of the image, with the x-axis going horizontally to the right and the y-axis going vertically upward. In addition to digitizing the spatial location, image digitization also requires converting the intensity values into discrete quantities, a process often referred to as *quantization* (Qiu, 2005). Conventionally, $f(x_i, y_j)$ is discretized to be an integer in the range $[0, M - 1]$, where $M$ is an integer power of 2. For instance, if $M = 2^8$, then a single pixel takes one byte in computer storage. Clearly, the quality of a digital image is affected by the sampling resolution and number of levels (i.e., $M$) used in quantization.

Observed digital images often contain noise which could be incurred from different sources. For example, noise can come from the digitization process described above. Noise can also be produced in poor lighting situations where images are taken. Assuming that the noise is additive to its true intensity values, an image can be described by the following two-dimensional regression model:

$$Z_{i,j} = f(x_i, y_j) + \varepsilon_{i,j},\ (x_i, y_j) \in \Omega,\ i = 1, \ldots,\ n_1,\ j = 1, \ldots, n_2, \tag{1}$$

where $f$ is the true image intensity function, $\varepsilon_{i,j}$ denotes the noise at the $(i, j)$-th pixel, $Z_{i,j}$ is the observed intensity value at the $(i, j)$-th pixel, and $\Omega$ is the design space. The image intensity function $f$ often has discontinuities. For instance, the boundary curves of objects in a photograph are positions where $f$ has jumps. Because our human-eye systems have evolved to make use of the boundary curves for recognizing objects, jumps in $f$ are important features of the image. This is illustrated in Figure




**Figure 1:** (a) The test image of peppers. (b) The boundary curves in the peppers image. (c) A SAR image of an area near Thetford forest, England.

1(a) – (b), which show the Lena image, a widely used test image in the image processing literature, and its jump positions (i.e., the boundary curves). Here two functions (`stepEdge` and `modify2`) have been used for extracting the jump positions. More details about these two functions and the topic of jump detection are provided in Section 4. It can be seen that much of the information regarding recognizing objects can be conveyed using jump locations only. Therefore, model (1) needs to be capable of handling discontinuous regression functions. The noise component is also necessary in (1). As an example, Figure 1(c) shows a synthetic aperture radar (SAR) image of an area near Thetford forest, England (Glasbey and Horgan, 1995). It can be seen that the SAR image is quite noisy. Both the Lena and SAR images are included in **DRIP** as test images. They can be accessed by the following code.

```
library(DRIP)
data("peppers")
data("sar")
```

The regression analysis in model (1) where the regression function is assumed to have jumps is referred to as *jump regression analysis* (Qiu, 2005). A major task in jump regression analysis is to estimate the jump locations. In some applications, it is also important to restore the true image from its observed but noisy version for better human and machine perception. Therefore, jump-preserving surface estimation is another important task in jump regression. The jump-preserving aspect is crucial due to the importance of jump locations. Formally, we assume that there exists a partition of the design space: $\Omega = \cup_{l=1}^{L} \Omega_l$ and $\Omega_{l_1} \cap \Omega_{l_2} = \emptyset$ if $l_1 \neq l_2$, and $f$ is continuous in $\Omega_l \backslash \partial\Omega_l$ for $l = 1, \ldots, L$. The point set $S = \cup_{l=1}^{L} \partial\Omega_l \cap \Omega^\circ$ are called the *jump location curves* of $f$ and $\cup_{l=1}^{L} \Omega_l^o$ are called the *continuity regions*.

In image processing, jump locations in $f$ are called *step edges*, and jump locations in the first-order derivatives of $f$ are called *roof edges*. Therefore, edge detection and edge-preserving image restoration in image processing are essentially the same tasks as jump detection and jump-preserving surface estimation in jump regression. However, jump regression is more general in two regards. First, image pixels are equally spaced, but jump regression model (1) does not require this regularity. The design points can be arbitrarily distributed and the design space $\Omega$ can be of any shape. This flexibility can be useful in some applications such as the estimation of equi-temperature surfaces in meteorology and oceanography (Kang et al., 2021). Second, intensity values of a digital image are discrete, and some image processing techniques make use of this discreteness, rendering them inappropriate for handling certain jump regression problems. By contrast, jump regression treats $Z_{i,j}$ in (1) as a numerical response and does not share this limitation. To facilitate our discussion in the remainder of this article, we assume that $n_1 = n_2$, $\Omega = [0,1] \times [0,1]$, and $\{\varepsilon_{i,j}\}$ are i.i.d. random errors with mean 0 and variance $\sigma^2$. In the following sections, we describe some jump regression methods and their implementations in **DRIP**.

## 5 Estimation of jump location curves

### 5.1 Step edge detection

First, we briefly describe the step edge detection method proposed in Kang and Qiu (2014). For any given point $(x, y) \in \Omega$, consider its circular neighborhood,

$$O(x, y; h) = \left\{(u, v) \in \Omega : (x-u)^2 + (y-v)^2 \leq h^2\right\},$$

where $h > 0$ is a bandwidth parameter, and the following local linear kernel smoothing procedure:

$$\min_{a,b,c} \sum_{(x_i,y_j)\in O(x,y;h)} \left\{Z_{i,j} - [a + b(x_i - x) + c(y_j - y)]\right\}^2 K_{i,j}, \tag{2}$$

where $K_{i,j} = K((x_i - x)/h, (y_j - y)/h)$ and $K$ is a circularly symmetric bivariate density kernel function defined on the unit disk centered at the origin. The solution to $(b, c)$ in (2), denoted by $(\widehat{b}(x, y), \widehat{c}(x, y))$, estimates the gradient of $f$ at $(x, y)$. If $(x, y)$ is near a jump location curve, $(\widehat{b}(x, y), \widehat{c}(x, y))$ indicates the direction in which $f$ increases the fastest, i.e., orthogonal to the jump location curve's tangent. Based on this intuition, we divide $O(x, y; h)$ into two halves along the direction perpendicular to $(\widehat{b}(x, y), \widehat{c}(x, y))$. Specifically, define (as illustrated in Figure 2)

$$O_1(x, y) = \left\{(x_i, y_j) \in O(x, y; h) : (x_i - x)\widehat{b}(x, y) + (y_j - y)\widehat{c}(x, y) \geq 0\right\},$$
$$O_2(x, y) = \left\{(x_i, y_j) \in O(x, y; h) : (x_i - x)\widehat{b}(x, y) + (y_j - y)\widehat{c}(x, y) < 0\right\}.$$

The change in the values of $f$ from $O_2(x, y)$ to $O_1(x, y)$ should be relatively large if $(x, y)$ is a jump point. To estimate this change, we fit two local linear kernel smoothing procedures in $O_1(x, y)$ and $O_2(x, y)$:

$$\min_{a_1,b_1,c_1} \sum_{(x_i,y_j)\in O_1(x,y)} \left\{Z_{i,j} - [a_1 + b_1(x_i - x) + c_1(y_j - y)]\right\}^2 K_{i,j}, \tag{3}$$

$$\min_{a_2,b_2,c_2} \sum_{(x_i,y_j)\in O_2(x,y)} \left\{Z_{i,j} - [a_2 + b_2(x_i - x) + c_2(y_j - y)]\right\}^2 K_{i,j}. \tag{4}$$

The solutions to $a_1$ of (3) and $a_2$ of (4) are denoted as $\widehat{f}_+(x, y)$ and $\widehat{f}_-(x, y)$, respectively. The design point $(x, y)$ is flagged as a detected jump point if

$$\left|\widehat{f}_+(x, y) - \widehat{f}_-(x, y)\right| > u, \tag{5}$$

where $u$ is a threshold parameter. This step edge detector is implemented by the `stepEdge` function, illustrated as follows.

```
stepedge <- stepEdge(image = sar, bandwidth = 10, thresh = 17,
                     degree = 1)
```

The output of `stepEdge` is a matrix of 0's and 1's. Figure 3(a) shows the estimated step edges by the function `stepEdge`. Here the parameter values of the bandwidth and threshold are user-specified. The data-driven parameter selection is discussed later in this section. It is worth noting that the bandwidth is specified in terms of the number of pixels. Also, we have exchanged 0's and 1's in the visualization with the `image` function in order to have the jump points shown in black.

Next, we compare our step edge detector `stepEdge` with the classic Sobel edge detector implemented by the function `edge_detection` in the **OpenImageR** package. The Sobel edge detector estimates the partial derivatives using two $3 \times 3$ Sobel masks and outputs the magnitude of the estimated gradient. In Figure 4, we apply both edge detectors to a test image of peppers and its noisy version. The noise is generated from the normal distribution $N(0, 30^2)$. It can be seen from the figure that the two edge detectors have rendered comparable results for the original test image, but the Sobel edge detector has produced many scattered false signals for the noisy version. This is because the Sobel edge detector has masks of fixed size. By contrast, `stepEdge` can adjust the bandwidth value (i.e., the neighborhood size) to accommodate different levels of noise. Additionally, the Sobel edge detector does not generate the binary output of edge detection. We have used the `90%` quantile as the threshold. In comparison, **DRIP** offers data-driven selection of the bandwidth and threshold parameters in `stepEdge`, which is described next.

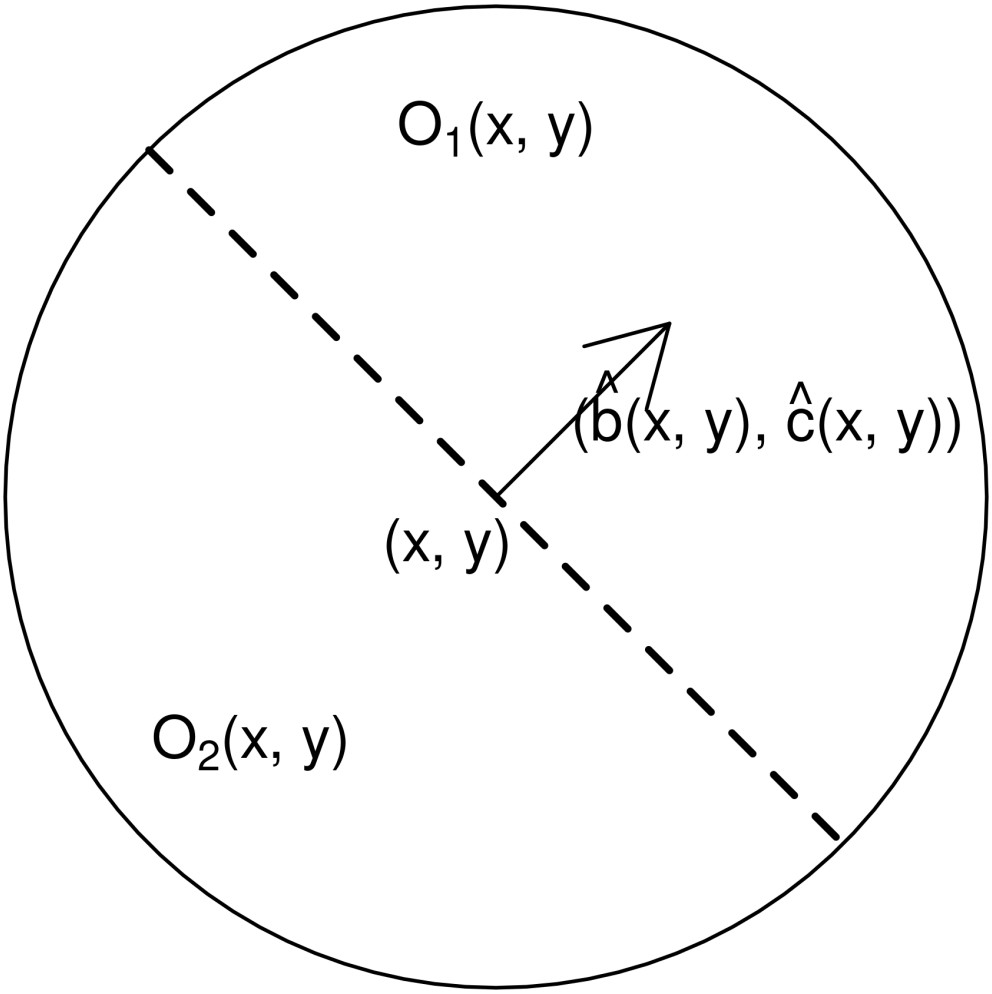



**Figure 2:** Neighborhood $O(x, y; h)$ of the design point $(x, y)$ is divided into two halves $O_1(x, y)$ and $O_2(x, y)$ along the direction perpendicular to the estimated gradient $(\widehat{b}(x, y), \widehat{c}(x, y))$.

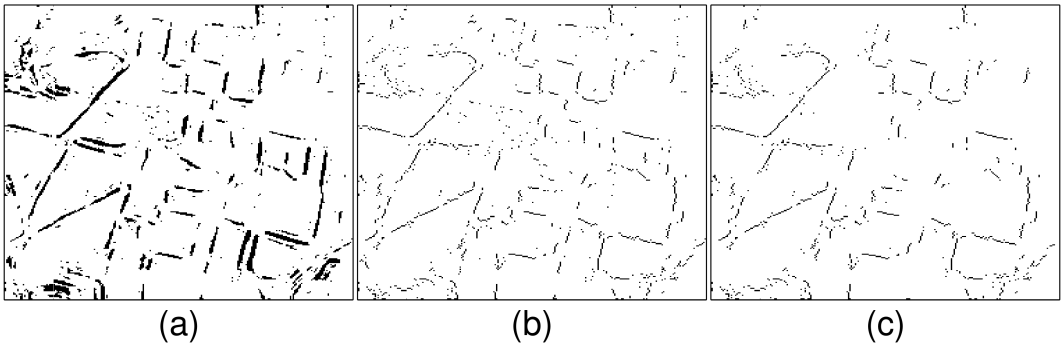



**Figure 3:** (a) The estimated step edges of SAR produced by `stepEdge`. (b) The estimated step edges after the first modification procedure described in the subsection of Two Modification Procedures. (c) The estimated step edges after the two modification procedures.

```
edge_peppers <- stepEdge(image = peppers, bandwidth = 5, thresh = 14, degree = 1)
sobel_peppers <- edge_detection(image = peppers, method = "Sobel")
par(mfrow = c(2, 3), mar = c(0.1, 0.1, 0.1, 0.1), xaxt = "n", yaxt = "n")
image(peppers, col = gray(c(0:255)/255))
image(1 - edge_peppers, col = gray(c(0:255)/255))
image(sobel_peppers < quantile(sobel_peppers, 0.9), col = gray(c(0:255)/255))
set.seed(42)
noise <- matrix(rnorm(prod(dim(peppers)), mean = 0, sd = 30),
                nrow = nrow(peppers), ncol = ncol(peppers))
edge_peppers <- stepEdge(image = peppers + noise, bandwidth = 11, thresh = 28,
                         degree = 1)
sobel_peppers <- edge_detection(image = peppers + noise, method = "Sobel")
image(peppers + noise, col = gray(c(0:255)/255))
image(1 - edge_peppers, col = gray(c(0:255)/255))
image(sobel_peppers < quantile(sobel_peppers, 0.9), col = gray(c(0:255)/255))
```

In order to properly select parameters for edge detection, Kang and Qiu (2014) has suggested the following performance measure for edge detectors.

$$d_{KQ}(\widehat{S}, S; h, u) = \frac{1}{2|\widehat{S}|} \sum_{(x',y')\in\widehat{S}} d_E\left((x',y'), S\right) + \frac{1}{2\,|S|} \sum_{(x,y)\in S} d_E\left((x,y), \widehat{S}\right), \tag{6}$$

where $d_E(\cdot,\cdot)$ denotes the Euclidean distance, and $S$ and $\widehat{S}$ are the true and estimated step edge points respectively. Here the Euclidean distance between a point $(x',y')$ and a set $S$ is defined to be $d_E((x',y'),S) = \inf_{(x,y)\in S} d_E((x',y'),(x,y))$. The first term in (6) includes all falsely detected jump points and calculates their average distance from $S$. The second term includes all missed true jump points and calculates their average distance from $\widehat{S}$. If $\widehat{S} = S$, we have $d_{KQ} = 0$. In practice, however, the true jump locations are unknown and thus $d_{KQ}$ cannot be computed. To address this issue, a bootstrap version of (6), denoted by $\widehat{d}_{KQ}(\widehat{S}, S; h, u)$, is proposed in Kang and Qiu (2014). It works as follows.

Bootstrap Procedure for Selecting $h$ and $u$

1. Apply the jump detector `stepEdge` to the original data set $\{(x_i, y_j, Z_{i,j}), i, j = 1, \ldots, n\}$. The set of detected jump points is denoted as $\widehat{S}$.
2. Apply a jump-preserving surface estimator and obtain $\widehat{f}(x,y)$. Define the residuals by $\widehat{\varepsilon}_{i,j} = Z_{i,j} - \widehat{f}(x_i, y_j)$.
3. Draw with replacement a bootstrap sample of size $n^2$, denoted as $\{\widehat{\varepsilon}^*_{i,j}, i, j = 1, \ldots, n\}$, from the residual set $\{\widehat{\varepsilon}_{i,j}, i, j = 1, \ldots, n\}$, and then obtain a bootstrap data set $\{(x_i, y_j, Z^*_{i,j}), i, j = 1, \ldots, n\}$, where $Z^*_{i,j} = \widehat{f}(x_i, y_j) + \widehat{\varepsilon}^*_{i,j}$.
4. Apply the jump detector with the same bandwidth and threshold as that in step 1

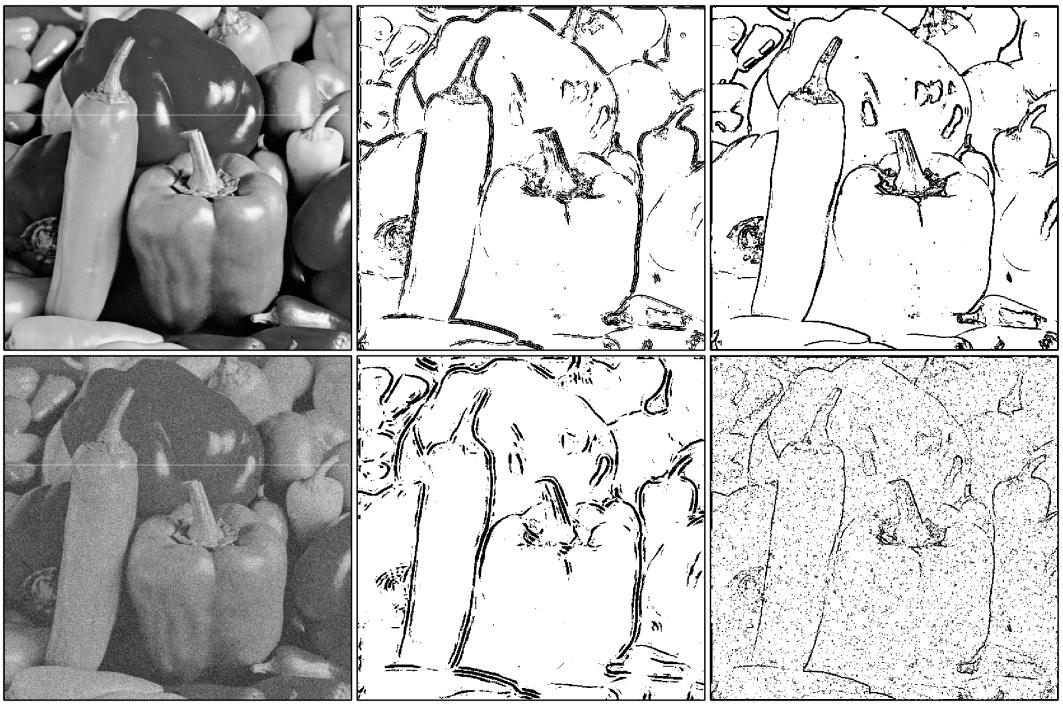


**Figure 4:** A comparison with the Sobel edge detector (from left to right). Top row: a test image of peppers, the edges detected by `stepEdge` and the edges detected by Sobel. Bottom row: a noisy version of the peppers image and the detected edges by `stepEdge` and Sobel.

to the bootstrap data set, and obtain a set of detected jump points from this bootstrap data.

5. Repeat step 3 and 4 for $B$ times. The sets of the detected jump points from the $B$ bootstrap data sets are denoted as $\widehat{S}_1^*, \widehat{S}_2^*, \ldots, \widehat{S}_B^*$, respectively. Then, the bootstrap estimator of $d_{KQ}(\widehat{S}, S; h, u)$ is defined by

$$\widehat{d}_{KQ}(\widehat{S}, S; h, u) = \frac{1}{B} \sum_{b=1}^{B} d_{KQ}(\widehat{S}_b^*, \widehat{S}; h, u).$$

The optimal bandwidth and threshold values are then approximated by the minimizer of $\widehat{d}_{KQ}(\widehat{S}, S; h, u)$ with respect to $(h, u)$:

$$\min_{h \in \mathscr{H}, u \in \mathscr{U}} \widehat{d}_{KQ}(\widehat{S}, S; h, u),$$

where $\mathscr{H}$ and $\mathscr{U}$ are candidates for the two parameters given by users.

In step 2 of the above procedure, $\widehat{f}(x, y)$ can be constructed by any jump-preserving surface estimator proposed in the literature (Hillebrand and Müller, 2007; Qiu, 2007). In **DRIP**, we adopt the estimator suggested in Qiu (2009) (see Section 5 for more details about this surface estimator). It is worth noting that this bootstrap procedure is computationally intensive. The computational cost of applying the step edge detector once to an image of resolution $n \times n = N$ is $O(N^2)$. Hence the total computational cost of the bootstrap procedure is $O(|\mathscr{H}|BN^2)$, where $|\mathscr{H}|$ denotes the number of candidate bandwidths. **DRIP** offers a pilot selector `edgeParSelPilot()` that helps narrow the scope of the parameter search. With the pilot selector, users can specify several preliminary bandwidths. Since the bandwidth specifies the neighborhood size for local smoothing (3) – (4), positive integers in the range of $[1, 20]$ are usually reasonable candidates. The `edgeParSelPilot()` function would plot the image of the edge detection statistics (i.e., the left hand side of (5) at each pixel) for each given bandwidth. Users can narrow down the choices based on the visual impression. In addition, the function outputs a few upper quantiles of the edge detection statistics for each bandwidth. Those quantiles offer clues to what reasonable threshold values can be, as edges are usually sparse in an image. In the following illustration with the SAR image, we provide three bandwidths and three probabilities to `edgeParSelPilot()`.

```
edgeParSelPilot(sar, edgeType = "step", degree = 1,
                bandwidth = c(6, 8, 10), probs = c(0.75, 0.85, 0.95))
```




**Figure 5:** The images of the edge detection statistics for specified bandwidths.

```
#>              probs=0.75 probs=0.85 probs=0.95
#> bandwidth=6   10.688440   13.79875   20.18085
#> bandwidth=8    9.610952   12.59395   19.38523
#> bandwidth=10   9.226387   12.21794   19.25910
```

Based on the visual impression given by Figure 5, bandwidth values around 10 seem reasonable as the visualization is less noisy. The quantiles suggest that the threshold should be around 20. Next, users can finalize the parameter choice using the function `stepEdgeParSel()`, which implements the above bootstrap procedure and estimates the edge detection performance for each combination of the bandwidth and threshold. Since this is a bootstrap procedure, the number of bootstrap samples needs to be given, and a random seed is required to ensure reproducibility. The function returns a matrix of $\widehat{d}_{KQ}$ values for each combination as well as the selected bandwidth and threshold parameters. The parameter selection in the following code chunk with the SAR image takes about 35 seconds to run on a single-core Intel(R) 4216 CPU @ 2.10GHz.

```
set.seed(24)
parSel <- stepEdgeParSel(image = sar, bandwidth = c(9, 10), degree = 1,
                         thresh = c(17, 21), nboot = 10)
print(parSel, type = "all")

#> The bootstrap matrix:
#>                thresh=17   thresh=21
#> bandwidth=9  0.009774721 0.012335871
#> bandwidth=10 0.008996036 0.011185702
#> The selected bandwidth:  10
#> The selected threshold:  17
```

**DRIP** has an alternative step edge detector, which is based on local constant kernel (LCK) smoothing. It is a more suitable choice if the intensity surface is piece-wise constant. The LCK-based edge detector can be called via `degree = 0`. The corresponding parameter selection can also be piloted by `edgeParSelPilot` and finalized by `stepEdgeParSel` with a `degree = 0` argument. In all these functions, the kernel $K$ is chosen to be the bivariate Epanechnikov kernel $K(x,y) = (2/\pi)(1-x^2-y^2)I\{x^2+y^2 \leq 1\}$ due to its nice theoretical properties (Fan and Gijbels, 2018).

### 5.2 Two modification procedures

There are two kinds of falsely flagged jump points. The first kind consists of those points near the real jumps. They occur due to the nature of local smoothing. This kind of deceptive jump points makes the detected edges thick. The second kind exists due to the nature of thresholding in (5). A design point can be flagged as a jump point with a small probability even if it is actually in a continuity region. Deceptive jump points of this kind are scattered in the entire design space. Qiu and Yandell (1997) has proposed two modification procedures to make the detected edges thin and to delete the scattered jump points. Their ideas are summarized below. Let $\theta_{i,j}$ denote the angle formed by the positive direction of the x-axis and the estimated gradient $(\widehat{b}(x_i,y_j),\widehat{c}(x_i,y_j))$ from solving (2) at $(x_i,y_j)$.

**MP$_1$** (to make the detected edges thin). For each $y_j$, consider the design points on the line $y = y_j$, $\{(x_i,y_j) : i = 1,\ldots,n\}$. Among them, the flagged jump points that satisfy $\theta_{i,j} \in [-\pi/4,\pi/4) \cup$

$[3\pi/4, 5\pi/4)$ are denoted by $\{(x_{i_s}, y_j) : s = 1, \ldots, m\}$. If there are integers $r_1 < r_2$ such that the increments of the sequence $x_{i_{r_1}} < x_{i_{r_1+1}} < \cdots < x_{i_{r_2}}$ are all less than the neighborhood width $2h$, but $x_{i_{r_1}} - x_{i_{r_1-1}} > 2h$ and $x_{i_{r_2+1}} - x_{i_{r_2}} > 2h$, then $\{(x_{i_s}, y_j) : r_1 \le s \le r_2\}$ form a *tie*, and the middle point $((x_{i_{r_1}} + x_{i_{r_2}})/2, y_j)$ replaces the tie as the new jump point. Thus, each tie is reduced to one representative, its middle point. Along the $y$ direction, for each $x_i$, consider the design points on the line $x = x_i$, $\{(x_i, y_j) : j = 1, \ldots, n\}$. Perform the same modification as that along the $x$ direction, except that only those jump points satisfying $\theta_{i,j} \in [\pi/4, 3\pi/4) \cup [5\pi/4, 7\pi/4)$ are considered.

When modifying the detected edges along the $x$ direction, the condition $\theta_{i,j} \in [-\pi/4, \pi/4) \cup [3\pi/4, 5\pi/4)$ ensures that the possible edge curve forms an acute angle with the y-axis of less than or equal to $\pi/4$. This avoids canceling the true edges that are parallel with the x-axis.

**MP$_2$** (to delete scattered jump points). For any jump point $(x_i, y_j)$, if the number of jump points in the neighborhood $O(x_i, y_j; h)$ is less than $nh$, then delete $(x_i, y_j)$ from the set of jump points.

Functions `modify1` and `modify2` implement these two procedures. We apply them to modify the detected jumps of the SAR image.

```
stepedge_m1 <- edgeModify1(bandwidth = 10, image = sar, edge = stepedge,
                       plot = FALSE)
stepedge_m2 <- edgeModify2(bandwidth = 10, edge = stepedge_m1, plot = FALSE)
```

The modified versions are shown in Figure 3(b) – (c). It can be seen from the figure that the edges are thinner and there are fewer scattered jump points.

### 5.3 Roof edge detection

Roof edge detection involves estimating the jump positions in the partial derivatives $f'_x$ and $f'_y$. In Qiu and Kang (2015), the following local quadratic smoothing procedure is considered.

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^6} \sum_{(x_i,y_j)\in O(x,y;h)} \Big\{ Z_{i,j} - \Big[\beta_0 + \beta_1(x_i - x) + \beta_2(y_j - y) + \beta_3(x_i - x)^2 + \beta_4(x_i - x)(y_j - y) + \beta_5(y_j - y)^2\Big]\Big\}^2 K_{i,j}, \tag{7}$$

where the bandwidth parameter $h$ may be different from that in (2). The solutions to $\beta_3$, $\beta_4$ and $\beta_5$ in (7) provide estimates for $f''_{xx}$, $f''_{xy}$ and $f''_{yy}$. Similar to constructing $|\widehat{f}_+ - \widehat{f}_-|$ in the subsection of Step Edge Detection, we divide the neighborhood $O(x, y; h)$ into two halves along the direction perpendicular to $(\widehat{\beta}_3(x, y), \widehat{\beta}_4(x, y))$ and run two local linear kernel smoothing procedures separately in the two half circles to obtain the estimate $\widehat{f}'_{x+} - \widehat{f}'_{x-}$. This is illustrated in Figure 6(a). The estimation of $f'_{y+} - f'_{y-}$ is the same except that the neighborhood is divided along the direction perpendicular to $(\widehat{\beta}_4(x, y), \widehat{\beta}_5(x, y))$, as shown in Figure 6(b).

A pixel is flagged as a roof edge point if $M(x, y) = \max\{|\widehat{f}'_{x+} - \widehat{f}'_{x-}|, |\widehat{f}'_{y+} - \widehat{f}'_{y-}|\}$ is greater than a threshold value. However, this criterion can be large around step edges too. Therefore, roof edge detection should be done only at places where there are no step edges. In **DRIP**, this roof edge detection method is implemented by `roofEdge`. We illustrate its use with a test image of peppers (Figure 7). The argument `edge1` specifies the step edge locations.

```
step.edges <- stepEdge(peppers, bandwidth = 6, thresh = 25)
roof.edges <- roofEdge(peppers, bandwidth = 9, thresh = 3000,
                       step_edge = step.edges)
```

Similar to the parameter selection for step edge detection, the bootstrap method works for roof edge detection. It is implemented by the `roofEdgeParSel` function, whose usage is identical to that of `stepEdgeParSel`. The pilot selector `edgeParSelPilot` can be used via `edgeType = "roof"`. The two modification procedures can also be applied to enhance the visual appearance of the detected roof edges.

## 6 Jump-preserving surface estimation

### 6.1 A three-stage approach

In some applications, e.g., image denoising, the ultimate goal is to estimate the regression surface. Since jumps are important features that dominate human visual perception, they should be preserved

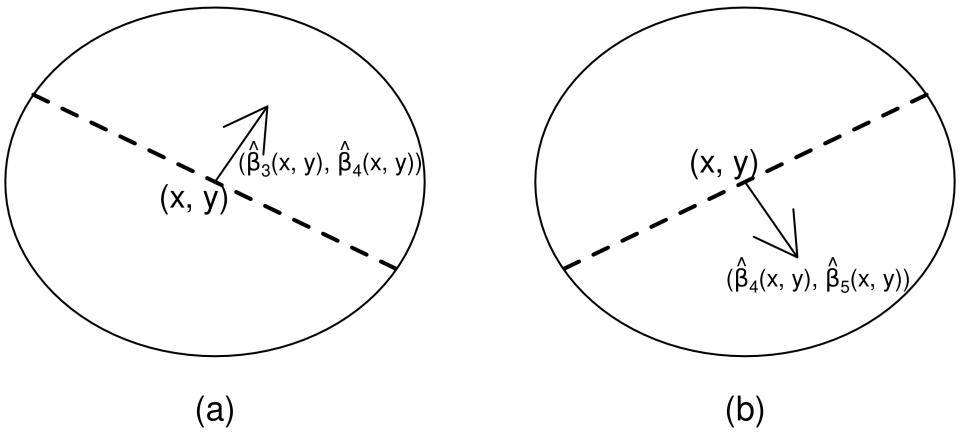



**Figure 6:** Neighborhood $O(x, y; h)$ of the design point $(x, y)$ is divided into two halves along the direction (a) perpendicular to $(\widehat{\beta}_3(x, y), \widehat{\beta}_4(x, y))$, (b) perpendicular to $(\widehat{\beta}_4(x, y), \widehat{\beta}_5(x, y))$.

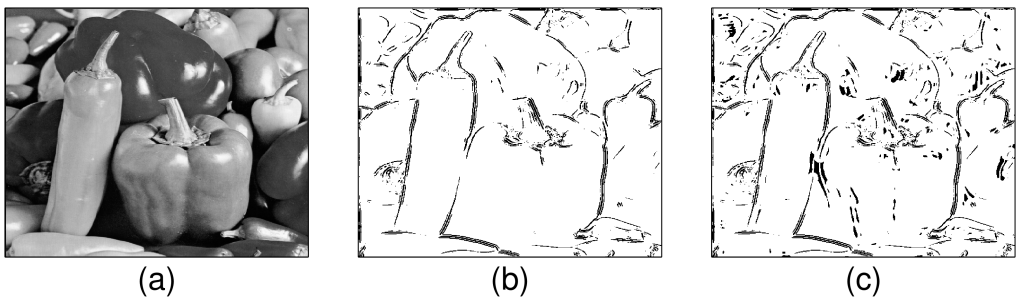



**Figure 7:** (a) A test image of peppers. (b) The detected step edges. (c) The detected step and roof edges.

by the estimation. Qiu (1998) has suggested a three-stage approach and its idea is outlined next. The first stage is to detect jumps and any reasonable jump detection procedure (e.g., the one implemented by `stepEdge`) can be used in this stage. In the second stage, at any given design point $(x, y)$, consider its neighborhood $O(x, y; h)$. Assume that the jump positions in $O(x, y; h)$ are denoted by $\{(w_r, v_r) : 1 \leq r \leq m\}$. Let $(W, V)$ denote a random vector taking values over $\{(w_r, v_r)\}$. Let $\overline{w}$ and $\overline{v}$ be the sample means of $\{w_r\}$ and $\{v_r\}$, respectively. Then search for a line $a(W - \overline{w}) + b(V - \overline{v}) = 0$ such that it attains the following minimum.

$$\min_{\tilde{a}^2+\tilde{b}^2=1} \text{Var}\left[\tilde{a}(W - \overline{w}) + \tilde{b}(V - \overline{v})\right].$$

The calculation of $a$ and $b$ is straightforward. We have

$$a = \sigma_{wv},\ b = \frac{1}{2}\left(\sigma_{vv} - \sigma_{ww} - \sqrt{(\sigma_{ww} - \sigma_{vv})^2 + 4\sigma_{wv}^2}\right),$$

where $\sigma_{ww}$, $\sigma_{vv}$ and $\sigma_{wv}$ denote the sample variances of $\{w_r\}$, $\{v_r\}$, and the sample covariance of $\{(w_r, v_r)\}$, respectively. Clearly, the line $a(W - \overline{w}) + b(V - \overline{v}) = 0$ is the first principal component line of the covariance matrix of $(W, V)$. It provides a first-order approximation to the jump location curve in $O(x, y; h)$. In the third stage, observations on the same side of the principal component line as $(x, y)$ are combined in a Nadaraya-Watson kernel estimator.

$$\widehat{f}(x, y) = \frac{\sum_{(x_i,y_j)\in\tilde{O}(x,y;h)} Z_{i,j}K_{i,j}}{\sum_{(x_i,y_j)\in\tilde{O}(x,y;h)} K_{i,j}}, \tag{8}$$

where $\tilde{O}(x, y; h)$ denotes the design points in $O(x, y; h)$ that are on the same side of the principal component line as $(x, y)$. That is, let $\zeta_{i,j} = a\,(x_i - \overline{w}) + b\left(y_j - \overline{v}\right)$, and

$$\tilde{O}(x, y; h) = \begin{cases} \left\{(x_i, y_j) \in O(x, y; h) : \zeta_{i,j} \geq 0\right\}, & \text{if } a\,(x - \overline{w}) + b\,(y - \overline{v}) \geq 0; \\ \left\{(x_i, y_j) \in O(x, y; h) : \zeta_{i,j} < 0\right\}, & \text{if } a\,(x - \overline{w}) + b\,(y - \overline{v}) < 0. \end{cases}$$

If there are no detected jump points in $O(x, y; h)$, then $\tilde{O}(x, y; h)$ is $O(x, y; h)$. This three-stage approach is implemented by `threeStage`, as illustrated below. The detected step edges are supplied using the argument `edge1`. Notably, users can also input the detected roof edges using the argument `edge2`. In cases where $O(x, y; h)$ contains roof edge points, a principal component line is fitted to approximate the roof edge curve and (8) is defined similarly using the observations on the same side of the fitted principal component line. If $O(x, y; h)$ contains both step and roof edge points, a principal component line would be fitted using the step edge points only and roof edge points are ignored. This is because step edges dominate human visual perception in comparison with roof edges. The idea of incorporating roof edge features in (8) was proposed in Qiu and Kang (2015), as an improvement of the approach initially suggested by Qiu (1998). The resulting estimator preserves both step and roof edges of the image. In cases where roof edge detection is deemed unnecessary, users can simply provide a matrix of zeros to `edge2`. Figure 8(a) shows the estimated SAR image after applying `threeStage`. It can be seen that the estimation preserves the discontinuities at places where edges have been successfully detected. At places where the edge detector fails to flag the edge points, the estimation still blurs the jumps.

```
fit <- restore3Stage(image = sar, bandwidth = 4, step_edge = stepedge_m2,
                     roof_edge = array(0, dim(sar)))
```

This three-stage approach requires a bandwidth parameter. It can be selected by minimizing the following leave-one-out cross validation score.

$$\min_{h\in\mathscr{H}} \text{CV}(h) = \frac{1}{n^2}\sum_{i,j=1}^{n}\left[Z_{i,j} - \widehat{f}_{-(i,j)}(x_i, y_j)\right]^2, \tag{9}$$

where $\mathscr{H}$ is a finite set of candidate bandwidths given by users and $\widehat{f}_{-(i,j)}$ denotes the estimator in (8) without using the $(i, j)$-th pixel. The cross validation procedure selects the bandwidth that minimizes the prediction error, as observed image intensities are readily available. In edge detection, however, there is no observed version of edges and thus cross validation cannot be utilized. Bootstrap estimates are constructed instead. It is also worth noting that, similar to the bootstrap procedure, the leave-one-out cross validation procedure is computationally expensive, as it carries out the estimation $|\mathscr{H}|$ times. The `threeStageParSel` function implements the cross validation procedure.

```
bw_3stage <- restore3StageParSel(image = sar, bandwidth = 4:5,
```

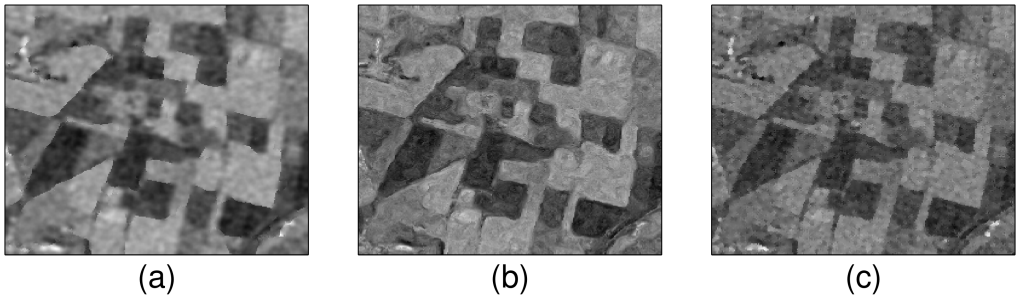



**Figure 8:** (a) The SAR image denoised by the three-stage approach. (b) The SAR image denoised by the gradient-based approach. (c) The SAR image denoised by the clustering-based approach.

```
                                step_edge = stepedge_m2,
                                roof_edge = array(0, dim(sar)))
print(bw_3stage, type = "all")

#> The cross validation scores:
#>          bandwidth=4 bandwidth=5
#> CV-score    294.7589    303.8463
#> The selected bandwidth:  4
```

### 6.2 A gradient-based approach

Although the three-stage approach is intuitively appealing, it is still complicated to use, as it requires explicit edge detection which involves careful selection of two important parameters (bandwidth and threshold). Qiu (2009) suggested a direct surface estimation approach without using any procedure parameters other than the bandwidth. Here we briefly describe its idea. As in the subsection of Step Edge Detection, $O(x,y;h)$ is divided into $O_1(x,y)$ and $O_2(x,y)$ along the direction perpendicular to the estimated gradient. Consider the local linear kernel smoothing procedures (3) and (4) and denote their solutions by $\{(\widehat{a}_k, \widehat{b}_k, \widehat{c}_k) : k = 1,2\}$. If there are no jumps in $O(x,y;h)$, both $\widehat{a}_1$ and $\widehat{a}_2$ are consistent for estimating $f(x,y)$. If there is a single step edge in $O(x,y;h)$, either $O_1(x,y)$ or $O_2(x,y)$ should be mostly on one side of the edge, and the corresponding one-sided estimator should be able to estimate $f(x,y)$ well. In such cases, Qiu (2009) proposes to choose between $\widehat{a}_1$ and $\widehat{a}_2$ based on their weighted residual mean square of the fitted plane. Specifically, their proposed surface estimator is defined as

$$\widehat{f}(x,y) = \begin{cases} \widehat{a}_1, & \text{if } e_1(x,y) < e_2(x,y), \\ \widehat{a}_2, & \text{if } e_1(x,y) > e_2(x,y), \\ (\widehat{a}_1 + \widehat{a}_2)/2, & \text{if } e_1(x,y) = e_2(x,y), \end{cases} \tag{10}$$

where $e_k(x,y) = \sum_{O_k(x,y)} \left\{ Z_{i,j} - \left[ \widehat{a}_k + \widehat{b}_k(x_i - x) + \widehat{c}_k(y_j - y) \right] \right\}^2 K_{i,j} / (\sum_{O_k(x,y)} K_{i,j})$ with $k = 1,2$. The surface estimator in (10) can be obtained by function `JPLLK_surface`. The resulting image after applying it to the SAR image is shown in Figure 8(b). It is worth noting that a numeric vector of positive integers can be supplied to the `bandwidth` argument. The one with the smallest leave-one-out cross validation score, which is similarly defined as in (9), is chosen for fitting the regression surface. The function returns a list of the fitted surface, residuals, selected bandwidth and estimated noise level. Here the residuals are defined as

$$\widehat{\varepsilon}_{i,j} = Z_{i,j} - \widehat{f}(x_i, y_j),\ i,j = 1,\ldots,n. \tag{11}$$

The residuals used in the bootstrap procedure in the subsection of Step Edge Detection are based on (11). The noise level $\sigma$ is estimated by $\widehat{\sigma} = (\sum_{i,j=1}^{n} \widehat{\varepsilon}_{i,j}^2 / n^2)^{1/2}$.

```
fit_gb <- JPLLK_surface(image = sar, bandwidth = 3:7)
print(fit_gb, type = "bandwidth")

#> The selected bandwidth:  6
```

### 6.3 A clustering-based approach

The surface estimation approaches introduced in the previous two subsections work well at places where the jump location curve is smooth. This is because both approaches approximate the jump curve locally using a straight line. At places where the jump location curve has corners or cusps, such approximations may be inadequate. To address this limitation, Kang et al. (2018) proposed a clustering-based approach, described as follows. Consider the local linear kernel smoothing procedure (2). The resulting estimator, $\widehat{a}(x,y)$, is good at removing noise but also blurs the edges. Center weighted median filtering (Ko and Lee, 1991; Sun et al., 1994) is a robust smoothing method in image processing and thus less sensitive to jumps in the regression surface. Apply the median filter with center weight $W_{0,0}$ to observations in $O(x,y;h)$ and denote its output by $\tilde{a}(x,y)$. Define

$$\Delta(x,y) = \widehat{a}(x,y) - \tilde{a}(x,y).$$

If $O(x,y;h)$ is in a continuity region, then the regression surface in $O(x,y;h)$ should be approximated well by the fitted plane described by $(\widehat{a}(x,y), \widehat{b}(x,y), \widehat{c}(x,y))$. As a result, $|\Delta(x,y)|$ should be small. If $O(x,y;h)$ contains a step edge, the fitted local plane cannot describe the regression surface in $O(x,y;h)$ well. Consequently, $|\Delta(x,y)|$ should be large. In other words, $|\Delta(x,y)|$ contains the jump information about $f$. Kang et al. (2018) has shown that, under the null hypothesis that $f$ is continuous at $(x,y)$,

$$nh\left[\widehat{a}(x,y) - \tilde{a}(x,y)\right] \xrightarrow{d} N(0, \Sigma_{11} + \Sigma_{22} - 2\Sigma_{12}),$$

where

$$\Sigma_{11} = \sigma^2 \int\int_{\mathbb{R}^2} K(x,y)^2\, dxdy,\ \Sigma_{12} = \frac{\mathrm{E}|\varepsilon_{1,1}|}{2\pi\phi(0)},\ \Sigma_{22} = \frac{1}{4\pi\phi(0)^2},$$

$\phi(\cdot)$ is the density function of $\varepsilon_{1,1}$, and $\xrightarrow{d}$ denotes convergence in distribution. Therefore, we flag $(x,y)$ as a jump point if

$$|\Delta(x,y)| > q_{1-\alpha}\sqrt{\frac{\Sigma_{11} + \Sigma_{22} - 2\Sigma_{12}}{nh}}, \tag{12}$$

where $q_{1-\alpha}$ denotes the $100(1-\alpha)$ percentile of the standard normal distribution and $\alpha \in (0,1)$ is the significance level. In cases where (12) holds, the estimation needs to accommodate the jump structure. Kang et al. (2018) has proposed to cluster the pixels in $O(x,y;h)$ into two groups based on their center weighted filtering outputs. Intuitively, pixels on the same side of the edge curve have similar filtering outputs. Hence, they can be put in the same group. Pixels on different sides of the edge curve have quite different filtering outputs, and they should be put in different groups. Clearly, it is not easy to specify the exact position of the edge curve within $O(x,y;h)$ and define the two groups of pixels accordingly. Nonetheless, an informative pixel clustering procedure can generate groups such that pixels within a group are similar in their filtering outputs and pixels in different groups have quite different filtering outputs. Such a pixel clustering procedure can reflect the local edge structure well without imposing restrictive conditions on the smoothness or shape of the edge curve. Specifically, define

$$\begin{aligned} O_{1,c}(x,y) &= \left\{(x_i,y_j) \in O(x,y;h) : \tilde{a}(x_i,y_j) \le c\right\}, \\ O_{2,c}(x,y) &= O(x,y;h) \backslash O_{1,c}(x,y), \end{aligned}$$

where $c$ is a cut-off constant and it is in the range

$$\left(\min_{(x_i,y_j)\in O(x,y;h)} \tilde{a}(x_i,y_j),\ \max_{(x,y)\in O(x,y;h)} \tilde{a}(x_i,y_j)\right).$$

The constant $c$ is chosen to maximize the following clustering criterion

$$\frac{|O_{1,c}(x,y)|\,(\overline{v}_1 - \overline{v})^2 + |O_{2,c}(x,y)|\,(\overline{v}_2 - \overline{v})^2}{\sum_{(x_i,y_j)\in O_{1,c}(x,y)}\left(\tilde{a}(x_i,y_j) - \overline{v}_1\right)^2 + \sum_{(x_i,y_j)\in O_{2,c}(x,y)}\left(\tilde{a}(x_i,y_j) - \overline{v}_2\right)^2}, \tag{13}$$

where $\overline{v}_s$ denotes the sample mean of the center weighted filtering outputs within $O_{s,c}(x,y)$ for $s = 1,2$, and $\overline{v}$ denotes the sample mean of the filtering outputs within $O(x,y;h)$. In (13), the numerator quantifies the dissimilarity between the two groups, and the denominator quantifies the dissimilarity within each of the two groups. Thus, it is reasonable to cluster the pixels by maximizing their ratio. It can be shown that (13) is the one-dimensional version of the clustering criterion suggested by Friedman

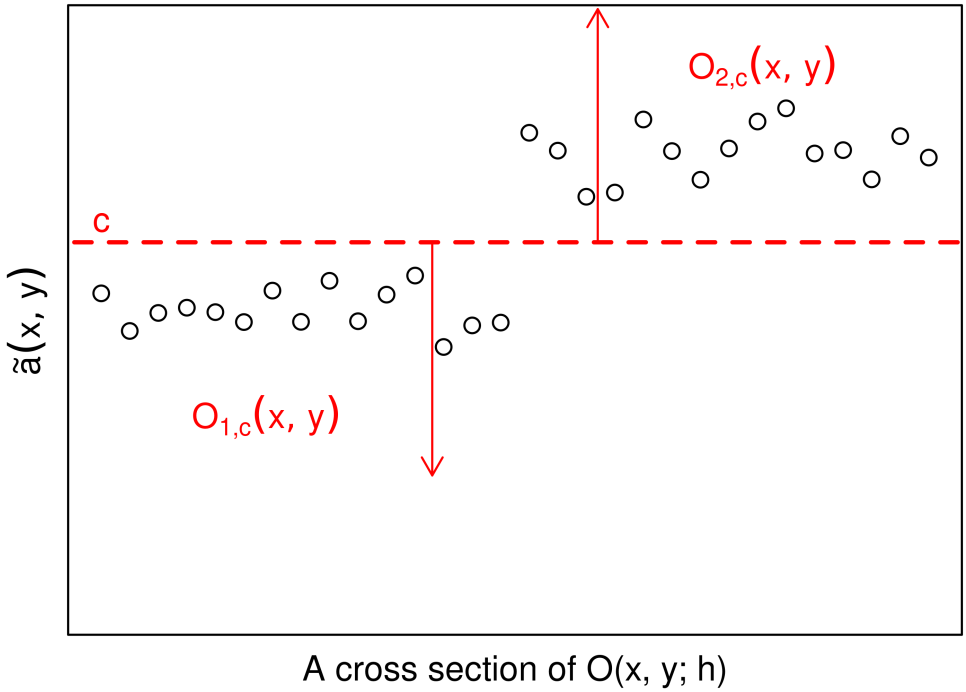



**Figure 9:** An illustration of local pixel clustering.

and Rubin (1967). Notably, there are only finitely many cut-off constants that can result in different partitions. Thus, it is sufficient to evaluate (13) on the finite set of $\{\tilde{a}(x_i, y_j) : (x_i, y_j) \in O(x, y; h)\}$ and the maximization can be solved by exhaustive search. This is illustrated in Figure 9. It can be seen that the values of $\{\tilde{a}(x_i, y_j)\}$ indirectly indicate the local edge structure.

Next, a weighted average of observations in the same group as $(x, y)$ should provide a good estimate for $f(x, y)$. Specifically,

$$\widehat{f}(x,y) = \sum_{s=1}^{2} \sum_{(x_i,y_j)\in O_{s,c}(x,y)} I\left\{(x,y) \in O_{s,c}(x,y)\right\} \frac{Z_{i,j}K_{i,j}}{\sum_{(x_i,y_j)\in O_{s,c}(x,y)} K_{i,j}}. \tag{14}$$

To use this estimator in practice, several unknown quantities in (12), $\sigma^2$, $\mathrm{E}|\varepsilon_{1,1}|$ and $\phi(0)$, need to be estimated. In the `surfaceCluster` function that implements this clustering-based approach, these unknown quantities are estimated using the residuals obtained from `JPLLK_surface`, as defined in (11). Specifically, the noise level $\widehat{\sigma}$ is estimated the same way as the previous subsection. As for the other two,

$$\widehat{\phi}(0) = \frac{1}{nb_n} \sum_{i,j=1}^{n} K_1\left(\frac{\widehat{\varepsilon}_{i,j} - 0}{b_n}\right), \ \widehat{\mathrm{E}|\varepsilon_{1,1}|} = \frac{1}{n^2} \sum_{i,j=1}^{n} \left|\widehat{\varepsilon}_{i,j}\right|,$$

where $K_1(\cdot)$ is the one-dimensional Gaussian kernel and $b_n = 1.06\widehat{\sigma}n^{-2/5}$ (Wand and Jones, 1994). Next, we apply the clustering-based estimator (14) to the SAR image. The argument `cw` specifies the center weight used in the center weighted median filter. Its default value is 3. The percentile $q_{1-\alpha}$ is derived from the argument `sig.level` whose value specifies $1 - \alpha$. Since the SAR image is quite noisy, setting a high significance level would blur many edges. We have used 0.7 in our illustration. The function returns a list of estimated image, $\widehat{\sigma}$, $\widehat{\phi}(0)$ and $\widehat{\mathrm{E}|\varepsilon_{1,1}|}$. The estimated image is shown in Figure 8(c). It can be seen that some corners that have been blurred by the three-stage and clustering-based approaches are now preserved.

```
fit_cl <- surfaceCluster(image = sar, bandwidth = 3, sig.level = 0.7, cw = 3)
```

The bandwidth parameter in (14) can be chosen by minimizing the leave-one-out cross validation score, defined similarly as that in (9). The `surfaceCluster_bandwidth` function implements the bandwidth selection for this clustering-based estimator.

```
parSel_cluster <- surfaceCluster_bandwidth(image = sar, bandwidths = 3:4,
                                           sig.level = 0.7, cw = 3)
print(parSel_cluster, type = "all")

#> The cross validation scores:
#>   bandwidths       cv
#> 1          3 267.9241
#> 2          4 284.7558
#> The selected bandwidth:  3
#> The estimated sigma:  16.80083
#> The estimated value of the density at 0:  0.4199855
#> The estimated mean of absolute error:  0.776125
```

### 6.4 Comparison with other denoising methods

We compare with two other denoising algorithms available in **R**. The first is the adaptive weights smoothing (AWS) suggested by Polzehl and Spokoiny (2000). The AWS method is an iterative algorithm and known for its good performance in practical settings. The second is the non-local means (NL-means) method proposed by Buades et al. (2005). The NL-means filter has been shown to enjoy some nice asymptotic properties. Both AWS and NL-means have been implemented in the package **aws** (Polzehl and Tabelow, 2023; Polzehl et al., 2020).

We apply the `JPLLK_surface` function in **DRIP** (i.e., the gradient-based approach), and the `aws` and `nlmeans` functions in **aws** to the SAR image and the noisy peppers image shown in Figure 4. The denoised images are presented in Figure 10. It can be seen that the images restored by NL-means are still somewhat noisy. The AWS method has rendered the sharpest results, benefiting from the algorithm's adaptive design. This suggests that incorporating a certain adaptive mechanism into the jump regression approach can further improve its performance.

```
sarhat_aws <- aws(sar, hmax = 8)
fit_gb <- JPLLK_surface(image = sar, bandwidth = 3:7)
sarhat_nlm <- nlmeans(sar, lambda = 0.85, sigma = fit_gb$sigma, patchhw = 3,
                      searchhw = 3)
par(mfrow = c(2, 3), mar = c(0.1, 0.1, 0.1, 0.1), xaxt = "n", yaxt = "n")
image(fit_gb$fitted, col = gray(c(0:255)/255), zlim = range(sar))
image(sarhat_aws@theta, col = gray(c(0:255)/255), zlim = range(sar))
image(sarhat_nlm$theta, col = gray(c(0:255)/255), zlim = range(sar))
set.seed(42)
noise <- matrix(rnorm(prod(dim(peppers)), mean = 0, sd = 30),
                nrow = nrow(peppers), ncol = ncol(peppers))
peppershat_aws <- aws(peppers + noise, hmax = 8)
peppers_gb <- JPLLK_surface(image = peppers + noise, bandwidth = 3:7)
peppershat_nlm <- nlmeans(peppers + noise, lambda = 0.85, sigma = fit_gb$sigma,
                          patchhw = 3, searchhw = 3)
image(peppers_gb$fitted, col = gray(c(0:255)/255), zlim = range(peppers + noise))
image(peppershat_aws@theta, col = gray(c(0:255)/255),
      zlim = range(peppers + noise))
image(peppershat_nlm$theta, col = gray(c(0:255)/255),
      zlim = range(peppers + noise))
```

## 7 Blind image deblurring

### 7.1 A deblurring model

In addition to having noise, images may also have blur involved. For instance, in astronomical imaging, ground-based imaging systems are subject to blurring due to the rapidly changing index of refraction of the atmosphere (Krist, 1995; Schulz, 1993). In photography, out-of-focus or camera shake often results in blurred images (Fergus et al., 2006). In medical imaging, blurred x-rays or mammograms are almost inevitable because the medical imaging systems limit the intensity of the incident radiation in order to protect the patient's health (Faulkner et al., 1989). In the image processing literature, a commonly used model to describe the relationship between the original image and its observed but



**Figure 10:** The denoised versions of the SAR image in Figure 1 and noisy peppers image in Figure 4 given by (from left to right) `JPLLK_surface`, `aws` and `nlmeans`, respectively.

blurred version is

$$Z(x_i, y_j) = G\{f\}(x_i, y_j) + \varepsilon_{i,j}, \tag{15}$$

where $G\{f\}$ is the convolution between $g$ and $f$ defined by

$$G\{f\}(x,y) = g \otimes f(x,y) = \int\int_{\Omega} g(x-u, y-v; x, y) f(u,v)\, dudv.$$

Here $g$ is called the point spread function. It describes how the original image is spatially degraded (i.e., blurred). In most references, it is further assumed that the blurring is location (or spatially) invariant. That is, $g(u, v; x, y)$ does not depend on $(x, y)$. It can be seen that model (15) includes model (1) as a special case when the point spread function is a two-dimensional Dirac delta function with a point mass at the origin.

Blind image deblurring is for estimating $f$ from $Z$ when the point spread function $g$ is not completely specified. This problem is ill-posed in nature because it is impossible to distinguish $(g, f)$ from $(ag, a^{-1}f)$ based on the observed image $Z$ alone, for any constant $a \neq 0$. Most existing blind image deblurring methods have been developed under the assumption that the point spread function $g$ does not depend on location because the ill-posed nature of the deblurring problem would be exacerbated if the blur is spatially varying, making the problem significantly more challenging. In practice, however, this assumption is often invalid. In photography, for instance, realistic camera shake includes both translations and in-plane rotations (Levin et al., 2011). Such blurring cannot be accurately described by a location-invariant point spread function. Kang (2020) has suggested a deblurring procedure without assuming a location-invariant point spread function. Their approach, which is implemented in **DRIP**, is described in the next subsection.

### 7.2 Image deblurring using JPEX

In image processing, it is common to assume that the blurring preserves average brightness, i.e., the point spread function $g(\cdot, \cdot; x, y)$ is a bivariate density function for each $(x, y)$. To understand the ill-posed nature of the deblurring problem, consider a special case in the one-dimensional version of (15)

$$\eta = g_0 \otimes f_0(\xi),$$

where $g_0(\zeta) = I(|\zeta| \leq s)/(2s)$, $s > 0$ and $f_0(\xi) = I(\xi > 0)$. In this ideal case, the observations $\{(\xi_i, \eta_i) : i = 1, \ldots, n\}$ contain no noise, there is a jump in $f_0$, and the point spread function $g_0$ is corresponding to the one-dimensional out-of-focus blur with blurring extent $s$. By some straightforward calculation, we have

$$\eta(\xi) = \begin{cases} 0, & \text{if } \xi < -s, \\ \frac{1}{2s}(\xi + s), & \text{if } -s \leq \xi \leq s \\ 1, & \text{if } \xi > s. \end{cases}$$

Clearly, $\eta$ is a piece-wise linear and continuous function. Without prior knowledge about $s$, it is impossible to distinguish the truth $(g_0, f_0)$ from the no-blur solution $(\delta_0, \eta)$ based on the observations $\{(\xi_i, \eta_i) : i = 1, \ldots, n\}$ alone, where $\delta_0(\cdot)$ denotes the Dirac delta function with the point mass at 0. It is also noteworthy that, because of its brightness-preserving property, the blurring does not alter $f_0(\xi)$ at places away from the jump location, where $f_0$ is flat. That is, $\eta(\xi) = f_0(\xi)$ for $\xi < -s$ or $\xi > s$. Therefore, the truth $f_0$ becomes estimable if we constrain the solution to the deblurring problem to be piece-wise constant, as non-zero gradients can only be caused by blurred jump structure in $f_0$. Motivated by this observation, Kang (2020) has proposed to impose the piece-wise constant constraint when estimating $f$ in (15). Specifically, it is assumed that $f$ is constant on each design space partition $\Omega_l$ for $l = 1, \ldots, L$. Let $\rho(x, y)$ denote the blurring extent at the pixel location $(x, y)$ defined by

$$\rho(x, y) = \inf\{r > 0 : supp\ g(\cdot, \cdot; x, y) \subseteq O(x, y; r)\},$$

where *supp* denotes the function's support and $O(x, y; r) = \{(u, v) : (u - x)^2 + (v - y)^2 \leq r^2\}$. The method suggested by Kang (2020) assumes that $\rho(x, y) \equiv \rho$ but allows $g(u, v; x, y)$ to have different forms as $(x, y)$ changes. Let $D = \{(x_i, y_j) \in \Omega, d_E((x_i, y_j), S) \leq \rho\}$. Obviously, $D$ is a collection of pixels that are no more than one blurring extent away from the jump location curves. Thus, their observed intensities are altered by blur and these pixels are called *blurry* pixels. With the brightness-preserving property and the piece-wise constant constraint, pixels outside $D$ (i.e., $(x_i, y_j) \in D^C$) are not affected by blur and they are called *sharp* pixels. The two kinds of pixels need to be handled differently. The key idea in Kang (2020) is to remove blur involved in blurry pixels using nearby sharp pixels.

For any given point $(x, y) \in \Omega$, consider its neighborhood $O(x, y; h)$, and fit the following local constant kernel smoothing and local linear kernel smoothing procedures:

$$\min_{\beta_0} \sum_{i,j=1}^{n} \left[Z_{ij} - \beta_0\right]^2 K_{ij}, \tag{16}$$

$$\min_{\beta_0, \beta_1, \beta_2} \sum_{i,j=1}^{n} \left[Z_{ij} - \beta_0 - \beta_1(x_i - x) - \beta_2(y_j - y)\right]^2 K_{ij}. \tag{17}$$

Let $\widehat{\beta}_0^{(0)}(x, y)$, and $(\widehat{\beta}_0^{(1)}(x, y), \widehat{\beta}_1^{(1)}(x, y), \widehat{\beta}_2^{(1)}(x, y))$ denote the solution to the minimization problem (16) and (17), respectively. Since $(\widehat{\beta}_1^{(1)}(x, y), \widehat{\beta}_2^{(1)}(x, y))$ estimates the gradient of $G\{f\}$ at $(x, y)$, its magnitude should be relatively small if $(x, y)$ is a sharp pixel. Based on this intuition, the null hypothesis that

$$\mathbf{d}\left(G\{f\}\right)(x, y) = (\partial_x G\{f\}(x, y), \partial_y G\{f\}(x, y)) = (0, 0) \tag{18}$$

can be tested using the following $F$ statistic

$$F_L(x, y) = \frac{\left(\text{WRSS}_0(x, y) - \text{WRSS}_1(x, y)\right)/2}{\widetilde{\sigma}_n^2},$$

where $\widetilde{\sigma}_n^2$ is a consistent estimate for $\sigma^2$, $\text{WRSS}_0(x, y)$ and $\text{WRSS}_1(x, y)$ denote the weighted residual sum of squares from (16) and (17), respectively. More specifically,

$$\text{WRSS}_0(x, y) = \sum_{i,j=1}^{n} \left[Z_{ij} - \widehat{\beta}_0^{(0)}(x, y)\right]^2 K_{ij},$$

$$\text{WRSS}_1(x, y) = \sum_{i,j=1}^{n} \left[Z_{ij} - \widehat{\beta}_0^{(1)}(x, y) - \widehat{\beta}_1^{(1)}(x, y)(x_i - x) - \widehat{\beta}_2^{(1)}(x, y)(y_j - y)\right]^2 K_{ij},$$

$$\widetilde{\sigma}_n^2 = \frac{1}{n^2} \sum_{i,j=1}^{n} \left[Z_{ij} - \widehat{\beta}_0^{(1)}(x_i, y_j)\right]^2.$$

Here $\widetilde{\sigma}_n^2$ is different from $\widehat{\sigma}^2$ in Section 4 and Section 5. It is directly based on the conventional

local linear kernel smoothing, whereas $\widehat{\sigma}^2$ is based on jump-preserving surface estimation. This is because the blurred regression surface $G\{f\}(x,y)$ is continuous and thus the residuals from (17) are appropriate representations of the random errors. This was not the case with jump regression problems in the previous two section.

Kang (2020) has shown that, under the null hypothesis (18), $2F_L(x,y)$ asymptotically follows a $\chi^2(2)$ distribution. Let $\chi^2_{1-\alpha}(2)$ denote the $100(1-\alpha)$ percentile of $\chi^2(2)$, where $\alpha \in (0,1)$ denotes the significance level . If $2F_L(x,y) > \chi^2_{1-\alpha}(2)$, then we flag $(x,y)$ as a blurry pixel. Otherwise we conclude that $f$ is constant on a neighborhood of $(x,y)$ and the blur does not affect its observed intensity value. Then $(x,y)$ is flagged as a sharp pixel. Let

$$\widehat{D}_n = \left\{(x_i,y_j) : 2F_L(x_i,y_j) > \chi^2_{1-\alpha}(2), i,j = 1,\ldots,n\right\}. \tag{19}$$

The deblurring estimate for $f$ is defined as follows.

$$\widehat{f}(x,y) = \begin{cases} \widehat{\beta}_0^{(0)}(x,y), & \text{if } 2F_L(x,y) \le \chi^2_{1-\alpha}(2), \\ \widehat{f}(\widehat{x}^*,\widehat{y}^*), & \text{otherwise}, \end{cases} \tag{20}$$

where $(\widehat{x}^*,\widehat{y}^*) \in \widehat{D}_n^C$ such that $d_E((x,y),(\widehat{x}^*,\widehat{y}^*)) = d_E((x,y),\widehat{D}_n^C)$. It can be seen that, in cases when $(x,y)$ is flagged as a sharp pixel, the local constant kernel estimate in (16) is used for $\widehat{f}(x,y)$. In cases when $(x,y)$ is flagged as a blurry pixel, its own observed intensity was not used in the estimation. Instead, $\widehat{f}(x,y)$ is a constant extrapolation from the closest detected sharp pixel. The idea is to make use of the sharp pixels in the same segment $\Omega_{l_0}$ as $(x,y)$ for estimating $f(x,y)$, where $l_0$ is some integer in $1,\ldots,L$ such that $(x,y) \in \Omega_{l_0}$. In order to obtain the correct extrapolation, it requires that (i) there exists some sharp pixels in $\Omega_{l_0}$, and (ii) $(\widehat{x}^*,\widehat{y}^*) \in \Omega_{l_0}$ almost surely. If the first condition is violated, certain image segment is completely blurred with no sharp pixels in it and thus it cannot be estimated well by (20). The second condition is to ensure that the extrapolation is done within the correct image segment. This condition would hold if the blurring extent is constant.

The deblurring estimate (20) is essentially a jump-preserving extrapolation (JPEX) and is implemented by the `jpex` function. We apply it to a blurry stop-sign image, which is shown in Figure 11(a). The function returns a list of two: `deblurred`, the deblurred image, and `edge`, the detected blurry pixels. The deblurred stop-sign image and the detected blurry pixels are shown in Figure 11(b) and (c), respectively. It can be seen that the JPEX method is able to flag most blurry pixels (i.e., those in close vicinity of the object boundaries) and deblur the image well. The arguments `alpha` and `sigma` specify the significance level and noise level, respectively. The details about estimating the noise level in this stop-sign image are provided in the next subsection.

```
data("stopsign", package = "DRIP")
deblur <- jpex(image = stopsign, bandwidth = 2, sigma = 0.00623, alpha = 0.001)
names(deblur)
```

```
#> [1] "deblurred" "edge"
```

For comparison, in Figure 11(d) we also include the result given by the unsupervised Wiener-Hunt deconvolution function in **scikit-image**. It is a Bayesian method suggested by Orieux et al. (2010) that performs joint estimation of the point spread function and underlying image. It can be seen that the restored image still contains blur and has some ringing artifacts.

```
if (reticulate::py_module_available("skimage")) {
  skimage <- reticulate::import("skimage")
} else {
  message("Python module 'scikit-image' is required and being installed.")
  reticulate::py_install("scikit-image", pip = TRUE)
  skimage <- reticulate::import("skimage")
}
np <- import("numpy")
restoration <- import("skimage.restoration")
psf0 <- np$ones(c(3L, 3L)) / 9
result <- restoration$unsupervised_wiener(stopsign, psf = psf0)
pyout <- result[[1]]
```

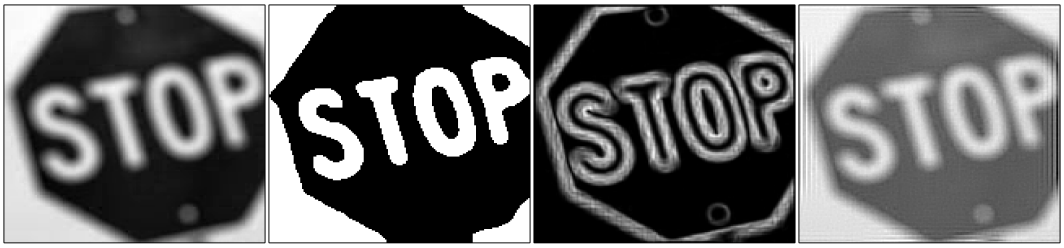



**Figure 11:** (a) The image of a blurry stop sign. (b) The deblurred image given by `jpex`. (c) The blurry pixels flagged by `jpex`. (d) The deblurred image given by **scikit-image**.

### 7.3 Bandwidth selection in JPEX

In the JPEX method, there is a bandwidth parameter $h$ to choose. The parameter is used for two purposes: to flag possible blurry pixels in (19) and to estimate $f$ at sharp pixel locations in (20). The purpose that $h$ serves the JPEX method is similar to what the bandwidth parameter does in the conventional kernel smoothing. Thus, it is reasonable to select $h$ by minimizing the leave-one-out cross validation score:

$$\min_h \sum_{i,j=1}^{n} \left[ Z_{i,j} - \widehat{\beta}_{0,-(i,j)}^{(1)}(x_i, y_j) \right]^2, \tag{21}$$

where $\widehat{\beta}_{0,-(i,j)}^{(1)}$ denotes the estimate $\widehat{\beta}_0^{(1)}$ from (17) when the $(i,j)$-th observation $(x_i, y_j, Z_{ij})$ is omitted. Since $\widehat{\beta}_0^{(1)}$ is a linear combination of $Z_{i,j}$, (21) can be written as

$$\min_h \sum_{i,j=1}^{n} \left[ \frac{Z_{i,j} - \widehat{\beta}_0^{(1)}(x_i, y_j)}{1-\omega} \right]^2, \tag{22}$$

where $\omega = K(0,0)/r_{00}$ and $r_{00} = \sum_{i,j=1}^{n} K_{i,j}$. The computation in (22) is greatly simplified as the local linear kernel smoothing procedure (17) does not need to be carried out $n^2$ times. In the **DRIP** package, this is implemented by the `cv.jpex` function, as illustrated below. There is an option in `cv.jpex` for parallel computing. The argument `ncpus` gives the number of cores to use, i.e., at most how many child processes will be run simultaneously. The value must be at least one. The output of `cv.jpex` includes (i) `LLK`, the estimated surface given by $\widehat{\beta}_0^{(1)}$ from (17), (ii) `sigma`, the estimated noise level $\tilde{\sigma}_n$, defined in the previous subsection, (iii) `cv`, the cross validation scores corresponding to the bandwidth candidates, (iv) `bandwidths`, the input bandwidth candidates, and (v) `band.min`, the bandwidth with the smallest cross validation score.

```
cv.out <- cv.jpex(image = stopsign, bandwidths = c(2, 3), ncpus = 2)
print(cv.out, type = "all")

#> The cross validation scores:
#> [1] 5.591376e-05 1.421656e-04
#> The selected bandwidth:  2
#> The estimated sigma:  0.006231292
```

## 8 Workflow illustration

The workflow of using functions in **DRIP** often involves tuning some parameters first and then estimating the image features. Here we illustrate this process using edge-preserving noise removal of the SAR image as an example. Specifically, in the edge detection step, the edge detector `stepEdge` requires a bandwidth parameter and a threshold value. We first establish an initial range for the parameters. The visualization provided by the `edgeParSelPilot` function is useful for this purpose.

```
edgeParSelPilot(sar, edgeType = "step", degree = 0,
                bandwidth = c(3, 6, 9), probs = c(0.75, 0.8, 0.9, 0.95))
```

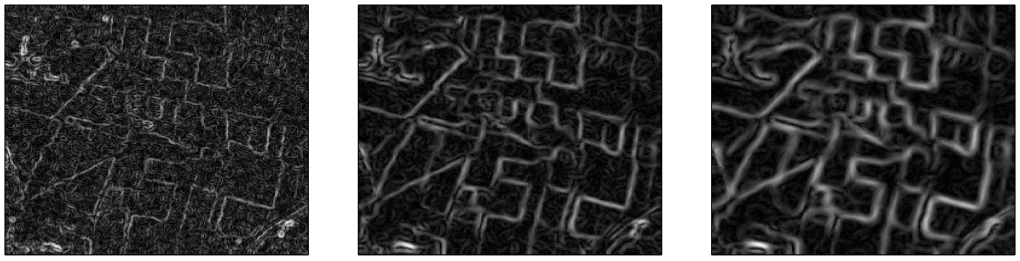



**Figure 12:** The images of the step edge detection statistics with the specified bandwidth values.

```
#>             probs=0.75 probs=0.8 probs=0.9 probs=0.95
#> bandwidth=3   15.82019  17.36624  22.11585   26.87138
#> bandwidth=6   13.31046  15.28433  21.32755   26.51575
#> bandwidth=9   14.72569  16.76905  22.61165   27.09358
```

It can be seen from Figure 12 that the bandwidth value of 3 seems too small (the corresponding image is still noisy). The bandwidth value of 9 appears to be too large (minor edges are lost in the resulting image). Therefore, bandwidth values around 6 are reasonable choices. Next, we determine the initial range for the threshold parameter. Based on the upper quantiles of the detection statistics with bandwidth 6, threshold values around 15 are good starting points. Next, we can finalize our parameter selection using the function `stepEdgeParSel`.

```
set.seed(24)
bandthresh <- stepEdgeParSel(sar, bandwidth = 4:5, thresh = 15:16,
                             nboot = 10, degree = 0)
edge1 <- stepEdge(sar, bandwidth = bandthresh$bandwidth,
                  thresh = bandthresh$threshold, degree = 0)
```

With the selected bandwidth and threshold, we have detected the step edges in the SAR image (shown in Figure 13. In the noise removal step, we apply the three-stage approach described earlier. Since the three-stage approach also requires a bandwidth parameter, we determine its value by cross validation (CV) before estimating the image. The denoised image is shown in Figure 13 and this completes the workflow.

```
cv3stage <- restore3StageParSel(sar, bandwidth = 4:9, step_edge = edge1,
                             roof_edge = array(0, dim(sar)))
sar_out <- restore3Stage(sar, bandwidth = cv3stage$bandwidth,
                         step_edge = edge1, roof_edge = array(0, dim(sar)))
par(mfrow = c(1, 2), mar = c(1, 1, 1, 1), xaxt = "n", yaxt = "n")
image(1 - edge1, col = gray(c(0:1)))
image(sar_out, col = gray(c(0:255)/255))
```

## 9 Concluding remarks

We have described **DRIP**, an R package that implements a number of jump regression methods with an emphasis on image processing applications. The package provides functions for step edge detection, roof edge detection, image denoising, image deblurring and the related data-driven parameter selection. Much of **DRIP**'s source code is written in Fortran or C to achieve good computational performance. Its R interface enables easy adoption and adaptation of the methods. It is intended to further broaden the scope of the package in the future. In particular, jump regression-based image registration (Qiu and Xing, 2013) has been an active research area. Including those routines in the package is certainly an interesting direction to pursue. Adding methods for image comparison (Feng and Qiu, 2018) and image monitoring (Kang, 2022, 2023) is also planned for the future.

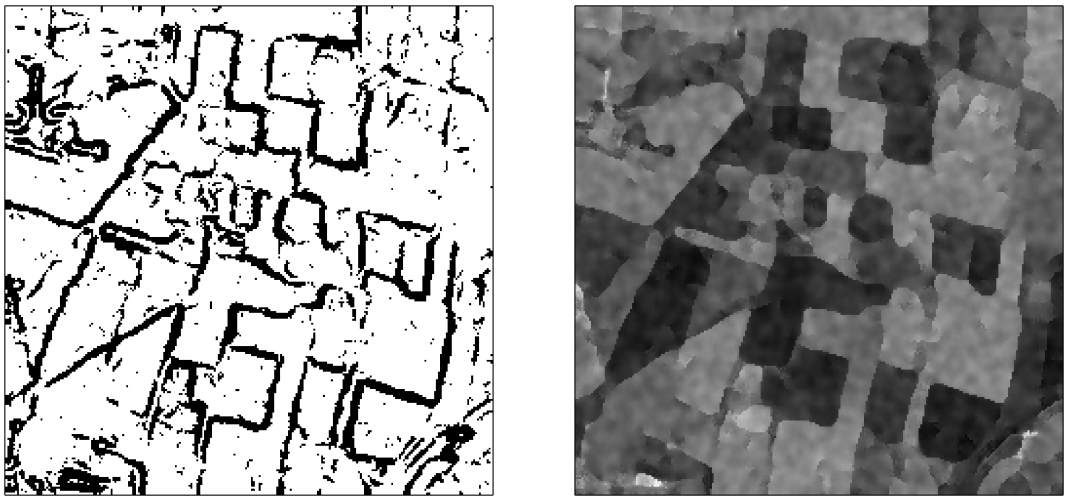


**Figure 13:** The detected step edges using the bootstrap-selected bandwidth and threshold values (left). The denoised SAR image using the CV-selected bandwidth (right).

## References


Y. Bai, G. Cheung, X. Liu, and W. Gao. Graph-based blind image deblurring from a single photograph. *IEEE Transactions on Image Processing*, 28(3):1404–1418, 2018. doi: TIP.2018.2874290. [p72]

S. Boonprong, C. Cao, W. Chen, X. Ni, M. Xu, and B. K. Acharya. The classification of noise-afflicted remotely sensed data using three machine-learning techniques: Effect of different levels and types of noise on accuracy. *International Journal of Geo-Information*, 7(7), 2018. doi: 10.3390/ijgi7070274. [p72]

G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008. URL https://opencv.org/. [p72]

A. Buades, B. Coll, and J.-M. Morel. A review of image denoising algorithms, with a new one. *Multiscale modeling & simulation*, 4(2):490–530, 2005. doi: 10.1137/040616024. [p72, 87]

J. Fan and I. Gijbels. *Local Polynomial Modelling and Its Applications: Monographs on Statistics and Applied Probability 66*. Routledge, New York, 2018. doi: 10.1201/9780203748725. [p80]

K. Faulkner, C. Kotre, and M. Louka. Veiling glare deconvolution of images produced by X-ray image intensifiers. In *Third International Conference on Image Processing and its Applications*, pages 669–673. IET, 1989. URL https://ieeexplore.ieee.org/abstract/document/132213. [p87]

L. Feng and P. Qiu. Difference detection between two images for image monitoring. *Technometrics*, 60 (3):345–359, 2018. doi: 10.1080/00401706.2017.1356378. [p92]

R. Fergus, B. Singh, A. Hertzmann, S. T. Roweis, and W. T. Freeman. Removing camera shake from a single photograph. In *Acm Siggraph 2006 Papers*, pages 787–794. ACM, 2006. doi: 10.1145/1179352.1141956. [p87]

H. P. Friedman and J. Rubin. On some invariant criteria for grouping data. *Journal of the American Statistical Association*, 62(320):1159–1178, 1967. doi: 10.2307/2283767. [p85]

A. Georgantzoglou, J. da Silva, and R. Jena. Image processing with matlab and gpu. In *MATLAB Applications for the Practical Engineer*. IntechOpen, 2014. doi: 10.5772/58300. [p72]

I. Gijbels, A. Lambert, and P. Qiu. Edge-preserving image denoising and estimation of discontinuous surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(7):1075–1087, 2006. doi: 10.1109/TPAMI.2006.140. [p73]

C. A. Glasbey and G. W. Horgan. *Image Analysis for The Biological Sciences*. John Wiley & Sons, Hoboken, New Jersey, 1995. URL https://dl.acm.org/doi/abs/10.5555/208252. [p75]

R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Pearson, New York, NY, 4 edition, 2018. URL https://elibrary.pearson.de/book/99.150005/9781292223070. [p74]

S. Goslee. *landsat: Radiometric and Topographic Correction of Satellite Imagery*, 2023. URL https://CRAN.R-project.org/package=landsat. **R** package version 1.1.2. [p73]

S. C. Goslee. Analyzing remote sensing data in **R**: the landsat package. *Journal of Statistical Software*, 43: 1–25, 2011. doi: 10.18637/jss.v043.i04. [p73]

P. Hall and P. Qiu. Blind deconvolution and deblurring in image analysis. *Statistica Sinica*, pages 1483–1509, 2007. URL http://www.jstor.org/stable/24307685. [p73]

M. Hillebrand and C. H. Müller. Outlier robust corner-preserving methods for reconstructing noisy images. *The Annals of Statistics*, 35(1):132 – 165, 2007. doi: 10.1214/009053606000001109. [p79]

Y. Kang. Consistent blind image deblurring using jump-preserving extrapolation. *Journal of Computational and Graphical Statistics*, 29(2):372–382, 2020. doi: 10.1080/10618600.2019.1665536. [p88, 89, 90]

Y. Kang. Statistical quality control using image intelligence: A sparse learning approach. *Naval Research Logistics (NRL)*, 69(7):996–1008, 2022. doi: 10.1002/nav.22069. [p92]

Y. Kang. Statistical process monitoring approaches for image data. *Wiley StatsRef: Statistics Reference Online*, pages 1–15, 2023. doi: 10.1002/9781118445112.stat08263. [p92]

Y. Kang and P. Qiu. Jump detection in blurred regression surfaces. *Technometrics*, 56(4):539–550, 2014. doi: 10.1080/00401706.2013.844732. [p76, 78]

Y. Kang, P. S. Mukherjee, and P. Qiu. Efficient blind image deblurring using nonparametric regression and local pixel clustering. *Technometrics*, 60(4):522–531, 2018. doi: 10.1080/00401706.2017.1415975. [p85]

Y. Kang, Y. Shi, Y. Jiao, W. Li, and D. Xiang. Fitting jump additive models. *Computational Statistics & Data Analysis*, 162:107266, 2021. doi: 10.1016/j.csda.2021.107266. [p75]

A. Klein, J. Andersson, B. A. Ardekani, J. Ashburner, B. Avants, M.-C. Chiang, G. E. Christensen, D. L. Collins, J. Gee, P. Hellier, et al. Evaluation of 14 nonlinear deformation algorithms applied to human brain mri registration. *Neuroimage*, 46(3):786–802, 2009. doi: 10.1016/j.neuroimage.2008.12.037. [p72]

S.-J. Ko and Y. H. Lee. Center weighted median filters and their applications to image enhancement. *IEEE Transactions on Circuits and Systems*, 38(9):984–993, 1991. doi: 10.1109/31.83870. [p85]

J. Krist. Simulation of HST PSFs using tiny tim. In R. Shaw, H. Payne, and J. Hayes, editors, *Astronomical Data Analysis Software and Systems IV*, volume 77 of *ASP Conference Series*, pages 349 – 352, 1995. URL https://adsabs.harvard.edu/full/record/seri/ASPC./0077/1995ASPC...77..349K.html. [p87]

A. Levin, Y. Weiss, F. Durand, and W. T. Freeman. Understanding blind deconvolution algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(12):2354–2367, 2011. doi: 10.1109/TPAMI.2011.148. [p88]

L. Mouselimis. *OpenImageR: An Image Processing Toolkit*, 2023. URL https://CRAN.R-project.org/package=OpenImageR. R package version 1.3.0. [p73]

F. Orieux, J.-F. Giovannelli, and T. Rodet. Bayesian estimation of regularization and point spread function parameters for wiener–hunt deconvolution. *Journal of the Optical Society of America A*, 27(7): 1593–1607, 2010. doi: 10.1364/JOSAA.27.001593. [p90]

G. Pau, F. Fuchs, O. Sklyar, M. Boutros, and W. Huber. EBImage - an **R** package for image processing with applications to cellular phenotypes. *Bioinformatics*, 26(7):979–981, 2010. doi: 10.1093/bioinformatics/btq046. [p73]

J. Polzehl and V. G. Spokoiny. Adaptive weights smoothing with applications to image restoration. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 62(2):335–354, 2000. doi: 10.1111/1467-9868.00235. [p72, 87]

J. Polzehl and K. Tabelow. Adaptive smoothing of digital images: The R package adimpro. *Journal of Statistical Software*, 19(1):1–17, 2007. doi: 10.18637/jss.v019.i01. [p73]

J. Polzehl and K. Tabelow. *Magnetic Resonance Brain Imaging: Modeling and Data Analysis Using R*. UseR! Springer, 2023. doi: 10.1007/978-3-030-29184-6. [p73, 87]

J. Polzehl, K. Papafitsoros, and K. Tabelow. Patch-wise adaptive weights smoothing in R. *Journal of Statistical Software*, 95(6):1–27, 2020. doi: 10.18637/jss.v095.i06. [p73, 87]

P. Qiu. Discontinuous regression surfaces fitting. *The Annals of Statistics*, 26(6):2218–2245, 1998. doi: 10.1214/aos/1024691468. [p83]

P. Qiu. *Image Processing and Jump Regression Analysis*. John Wiley & Sons, Hoboken, New Jersey, 2005. URL https://www.wiley.com/en-be/Image+Processing+and+Jump+Regression+Analysis-p-9780471420996. [p73, 74, 75]

P. Qiu. Jump surface estimation, edge detection, and image restoration. *Journal of the American Statistical Association*, 102(478):745–756, 2007. doi: 10.1198/016214507000000301. [p79]

P. Qiu. Jump-preserving surface reconstruction from noisy data. *Annals of the Institute of Statistical Mathematics*, 61:715–751, 2009. doi: 10.1007/s10463-007-0166-9. [p79, 84]

P. Qiu and Y. Kang. Blind image deblurring using jump regression analysis. *Statistica Sinica*, 25: 879–899, 2015. doi: 10.5705/ss.2014.054. [p81, 83]

P. Qiu and C. Xing. On nonparametric image registration. *Technometrics*, 55(2):174–188, 2013. doi: 10.1080/00401706.2012.727768. [p92]

P. Qiu and B. Yandell. Jump detection in regression surfaces. *Journal of Computational and Graphical Statistics*, 6(3):332–354, 1997. doi: 10.2307/1390737. [p80]

Y. Quan, Z. Wu, R. Xu, and H. Ji. Deep single image defocus deblurring via gaussian kernel mixture learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):11361–11377, 2024. doi: 10.1109/TPAMI.2024.3457856. [p72]

**R** Core Team. ***R**: A Language and Environment for Statistical Computing*. **R** Foundation for Statistical Computing, Vienna, Austria, 2023. URL https://www.R-project.org/. [p72]

T. J. Schulz. Multiframe blind deconvolution of astronomical images. *Journal of the Optical Society of America A*, 10(5):1064–1073, 1993. doi: 10.1364/JOSAA.10.001064. [p87]

J. Sun and P. Qiu. Jump detection in regression surfaces using both first-order and second-order derivatives. *Journal of Computational and Graphical Statistics*, 16(2):289–311, 2007. doi: 10.1198/106186007X204753. [p73]

T. Sun, M. Gabbouj, and Y. Neuvo. Center weighted median filters: some properties and their applications in image processing. *Signal Processing*, 35(3):213–229, 1994. doi: 10.1016/0165-1684(94)90212-7. [p85]

K. Tabelow and J. Polzehl. Beyond the gaussian model in diffusion-weighted imaging: The package dti. *Journal of Statistical Software*, 44(12):1–26, 2011. URL https://www.jstatsoft.org/v44/i12/. [p73]

K. Tabelow, J. Polzehl, V. Spokoiny, and H. U. Voss. Diffusion tensor imaging: Structural adaptive smoothing. *NeuroImage*, 39(4):1763–1773, 2008. doi: 10.1016/j.neuroimage.2007.10.024. [p73]

. The MathWorks Inc. ***MATLAB** version: 23.2.0 (R2023b)*. The MathWorks Inc., Natick, Massachusetts, United States, 2023. URL https://www.mathworks.com. [p72]

S. Van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014. doi: 10.7717/peerj.453. [p72]

M. P. Wand and M. C. Jones. *Kernel smoothing*. CRC Press, Boca Raton, FL, 1994. doi: 10.1201/b14876. [p86]

C. Wang, L. Sun, P. Cui, J. Zhang, and S. Yang. Analyzing image deblurring through three paradigms. *IEEE Transactions on Image Processing*, 21(1):115–129, 2011. doi: 10.1109/TIP.2011.2159985. [p72]

Z. Yue, J. Wang, and C. C. Loy. Efficient diffusion model for image restoration by residual shifting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(1):116–130, 2025. doi: 10.1109/TPAMI.2024.3461721. [p72]

B. Zitova and J. Flusser. Image registration methods: a survey. *Image and vision computing*, 21(11): 977–1000, 2003. doi: 10.1016/S0262-8856(03)00137-9. [p72]

*Yicheng Kang*
*Miami University*
*Department of Information Systems and Analytics*
*Oxford OH, USA*
https://miamioh.edu/fsb/directory/?up=/directory/kangy10
*ORCID: 0000-0002-8512-2722*
kangy10@miamioh.edu

# rSRD: An R Package for the Sum of Ranking Differences Statistical Procedure

*by Balázs R. Sziklai, Attila Gere, Károly Héberger and Jochen Staudacher*

**Abstract** Sum of Ranking Differences (SRD) is a relatively novel, non-parametric statistical procedure that has become increasingly popular recently. SRD compares solutions via a reference by applying a rank transformation on the input and calculating the distance from the reference in $L_1$ norm. Although the computation of the test statistics is simple, validating the results is cumbersome—at least by hand. There are two validation steps involved. Comparison of Ranks with Random Numbers, which is a permutation-test, and cross-validation combined with statistical testing. Both options impose computational difficulties albeit different ones. The rSRD package was devised to simplify the validation process by reducing both validation steps into single function calls. In addition, the package provides various useful tools including data preprocessing and plotting. This paper aims to serve as a guide for practitioners by offering a detailed presentation of the features.

## 1 Introduction

Sum of Ranking Differences first appeared in the field of analytical chemistry (Héberger, 2010; Kollár-Hunek and Héberger, 2013). In chemometrics, comparing methods or models via a reference is quite natural, as materials are often tested against some industry standard, a benchmark provided by some accredited laboratory, or some known theoretical value.

The idea caught on quickly, as the use of references is also common in other fields. Recommendation systems utilize consumers' preferences to generate content. In choice modeling, the consensus ranking or ground truth is considered the ideal or true ranking of the evaluated items. In web searches, a queried image can serve as a reference. A predefined set of features also plays a role in pattern recognition and computer vision. Consequently, SRD has a wide range of applications, from machine learning (Moorthy et al., 2017) and multi-criteria decision-making (Petchrompo et al., 2022) to network science (Sziklai and Lengyel, 2024), outlier detection (Brownfield and Kalivas, 2017), pharmacology (Vajna et al., 2012), political science (Sziklai and Héberger, 2020), ecology (Wątróbski et al., 2022), and even sports (West, 2018; Gere et al., 2022).

The test statistics of SRD is computed as follows. A rank transformation is performed on the input, which is a numerical table. Both in the solution vectors and in the reference, values are replaced by ranks. The smallest value of each vector obtains a rank of 1, while the largest $n$ (the size of the input). Ties are resolved by fractional ranking, that is, tied values are replaced by the arithmetic mean of their corresponding ranks. After rank transformation, the distances between the solutions and the reference are computed. SRD uses the $L_1$-norm. We discuss the advantage of this distance metric as well as possible alternatives in Section 2.

SRD values are normalized by the largest possible difference between two rankings of size $n$. In this way, SRD values of different problems can be compared with each other. The obtained values already rank the solutions—the one with the smallest SRD score is the closest to the reference, hence the best. However, the real benefit comes from the validation options.

SRD values are validated in two ways. The first option is the Comparison of Ranks with Random Numbers (CRRN). Informally, it is a permutation-test measuring the probability that a distance between a random ranking and the reference is as extreme as the solution's SRD score (Kollár-Hunek and Héberger, 2013). By convention, we accept solutions (i.e. reject $H_0$) below the 5% significance threshold. Between 5-95% solutions might have only artificial resemblance to the reference, but in fact they are not distinguishable from a random ranking. Above 95% the solutions seem to rank the objects in a reverse order—again $H_0$ is rejected with 5% significance, although the solution is not accepted since it is closer to the exact opposite of the reference. Thus, SRD can detect significant similarity as well as dissimilarity.

The difficulty of this option is that there is more than one way to assign a distribution to the SRD values. If neither the solution nor the reference ranking contains ties and the number of elements is sufficiently high, the distribution is asymptotically normal. Should any of these criteria (no ties, high $n$) not be met, the distribution can change substantially. Unfortunately, ties are quite common in applications and so is a small sample size. The derivation of thresholds can only be achieved by simulating the empirical distribution, since in such cases, the SRD values do not follow

any widely recognized distribution.

The second validation option is cross-validation combined with statistical testing. Although the SRD values rank the solutions, it is unclear what equal or similar SRD scores signify. Just because two solutions are of the same distance from the reference it doesn't mean they are close to each other. During cross-validation we take random samples from the rows and calculate the SRD scores for each sample. Then, with the help of a statistical test (e.g. Wilcoxon, Alpaydin or Dietterich-test) we can determine whether the two rankings induced by the solution vectors come from the same distribution, and if not, which one should be ranked lower in the SRD ranking. The choice of statistical test has a huge impact on type 1 and type 2 errors. Given the trade-off between the two error rates, ultimately only the practitioner can decide which one is best suited for her use case.

Due to the popularity of the procedure there are various software implementations available. Dávid Bajusz published a python based application uploaded to github[1], see also (Gere et al., 2021). John Kalivas' homepage offers a MATLAB code[2]. Attila Gere implemented a platform independent shiny application[3]. Finally, Klára Kollárné Hunek and Károly Héberger developed an MS Excel macro[4]. While these existing packages are undoubtedly valuable, they often lack thorough documentation, and none of them provide the comprehensive toolset found in rSRD. Our package excels in various aspects, offering a broad range of features for data preprocessing, SRD computation, validation, and plotting. As a result, the rSRD package emerges as a much-needed solution, bridging the gap for users of this statistical test. In the following, we formally introduce the SRD test, describe the functions featured in the package, and demonstrate their use through examples.

## 2 Related literature

In this section, we review the literature based on the three components of SRD analysis: rank transformation, reference ranking, and distance measures employed in testing. In addition, we briefly present the theoretical work that has been done related to SRD's validation options.

### 2.1 Ranking transformation in statistical testing

Sum of Ranking Differences is conceptually similar to Kendall's Tau (Kendall, 1938) and Spearman's Rank Correlation Coefficient (Spearman, 1904), both of which measure ordinal associations between variables. However, there are a few notable differences.

Firstly, SRD uses a fixed reference, whereas Kendall's Tau or Spearman's Rho are typically employed in either pairwise or multivariate comparisons. SRD is a proper distance metric, where 0 indicates perfect positive concordance, and 1 represents perfect negative concordance. In contrast, both Kendall's Tau and Spearman's Rho range from $-1$ to 1, where the scales are opposite, with $-1$ signifying the largest possible discrepancy and 1 representing perfect correlation.

The most significant difference lies in how the statistics are computed. In the case of SRD, distance is measured in the $L_1$-norm, while Kendall's Tau uses the number of inversions (also called Kemeny-distance or Kendall Tau distance), and Spearman's Rho computes the product-moment correlation coefficient. In the absence of ties, this coefficient resembles the Euclidean distance between rankings, in the sense that it uses the squares of rank differences in its computation.

Robustness can be enhanced by applying ranking transformation, especially when the data may not meet the assumptions of parametric methods. Ranking helps mitigate the impact of outliers and skewed distributions, making the analysis more robust. For a comprehensive mathematical discussion the reader is referred to (Kendall, 1970).

### 2.2 Reference values in statistical testing

The use of reference comes natural in many applications. In econometrics there is often a base period or control group to which data is compared. In recommendation systems, user preferences (Bhowmik and Ghosh, 2017), while in web searches, the queried subject (Dwork et al., 2001) can serve as a reference. Sometimes a reference is not given externally but is drawn from data. This happens, for instance, in marketing and advertisement research where consumer preferences are

[1] https://github.com/davidbajusz/srdpy
[2] https://www.isu.edu/chem/faculty/staffdirectoryentries/name-22144-en.html
[3] https://attilagere.shinyapps.io/srdonline/
[4] http://aki.ttk.mta.hu/srd

aggregated to create the ideal ranking (Lin, 2010; Haghani et al., 2021; Schuster et al., 2024). The most popular statistical models are the Mallows (Mallows, 1957), the Thurstone (Thurstone, 1927) and Plackett-Luce (Luce, 1959; Plackett, 1975).

The Thurstone model was developed in experimental psychology to handle cases when subjects have to arrange a series of stimuli in absolute rank order according to the sensation it prompted. The distribution of sensations from a particular stimulus is assumed to be normal. Furthermore, it assumes equal standard deviations of sensations corresponding to stimuli and equal correlations between pairs of stimuli sensations (Mosteller, 1951).

The Plackett-Luce model is a probabilistic model used to represent the likelihood of observed rankings. Plackett was originally motivated by a problem of calculating the odds of racehorses, in particular the probability of whether a horse finishes in the top 3 (Plackett, 1975). Holman and Marley proved that if the underlying random variables in Thurstone's approach have an extreme value distribution, the resulting choice probabilities are given by the Plackett-Luce model as well (Diaconis, 1988).

Finally, in the Mallows model, there is a central ranking $\pi_0$ and a scale parameter $\lambda$. The highest probability is assigned to $\pi_0$, with probabilities diminishing geometrically as the distance from $\pi_0$ increases. A higher value of $\lambda$ leads to a distribution that is increasingly peaked around $\pi_0$. Mallows initially focused on two specific distance metrics: Spearman's rho distance and Kendall's tau distance.

Mallows, Plackett-Luce, and Thurstone are burdened by both assumptions about the distribution of the underlying data and computational difficulties. In the Thurstone model, non-linear least squares approach is used to estimate the means of the normal distributions, while the standard deviations are assumed to be unit (Lin, 2010). In the Mallows model finding the Maximum Likelihood Estimate of $\pi_0$ is the Kemeny's consensus ranking problem, known to be $NP$-hard (Tang, 2019).

Fortunately, there are excellent software packages available to address these issues. The *PlackettLuce* package provides functions for preparing ranking data to fit the Plackett-Luce model or Plackett-Luce trees (Turner et al., 2023). *BayesMallows* offers a Bayesian version of the Mallows rank model (Sorensen et al., 2024). The *rankdist* package serves as a general platform for distance-based ranking models, including Mallows' (Qian and Yu, 2019). In addition, the *hyper2* package provides parametrized Plackett-Luce likelihood functions for Bradley-Terry strengths (Hankin, 2017, 2024).

Finally, it's important to note that the shared feature among these ranking models is the assumption of a reference ranking. However, the objectives of the Mallows, Plackett-Luce, and Thurstone models differ considerably from those of the Sum of Ranking Differences. Consequently, they cannot serve as substitutes for SRD and *vice versa*. Recently, however, Héberger (2026) proposed distance-based tests that also use a reference but do not require a ranking transformation.

### 2.3 Distance metrics in ranking

A commonly used distance metric is Kendall tau, quantifying the number of inversions between two rankings. In simpler terms, it measures the number of adjacent transpositions required to transform one ranking into another. The attractiveness of this metric lies in its inherent connection to rankings; it does not make sense with real valued arrays. Both the Mallows model and, unsurprisingly, the Kendall Tau Rank Correlation test uses the Kendall Tau measure.

The $L_1$-norm, sometimes also referred to as Manhattan-distance, was also proposed early on. Following Spearman's work, the $L_1$-norm of rankings, in the absence of tied values, is commonly referred to as Spearman's footrule. A comprehensive exploration of the statistical properties of the footrule is provided by Diaconis and Graham (1977). Empirical evidence suggests that $L_1$-norm is a sensible choice, comparable to other commonly used distance metrics. Héberger and Škrbić (2012) found that SRD (which uses the $L_1$-norm) is slightly stricter than Spearman rho and Kendall tau, that is, it rejects more models. Sipos et al. (2018) on the other hand compared SRD, Kendall tau, Caley distance, and a combination of Caley and SRD, and found that SRD and Kendall tau gives virtually the same results.

In various applications, such as sports or information retrieval, the top of a ranking often carries more significance than the bottom. Arguably, the accuracy of the first page of search results is far more critical for a search engine than the accuracy of subsequent pages. Similarly, sports enthusiasts tend to prioritize the top three positions over the last three. Consequently, distance metrics considering the top-$k$ positions have been developed for rank aggregation purposes (Fagin et al., 2003). Another solution that takes position into consideration without discarding some parts of the ranking, is weighting. Weighted distance-metrics for rankings are considered for instance in refs. (Gere et al., 2022; Chatterjee et al., 2018) and (Sziklai et al., 2022). However, imposing a weighting always adds a subjective element to the analysis as different weightings will benefit

different solutions. Moreover, the weighting will deteriorate the power of statistical tests. Thus, weighted distances should be adopted with care. Finally, incomplete rankings and their aggregation are also studied (Rodrigo et al., 2024).

### 2.4 SRD validation

Comparison of Ranks with Random Numbers (CRRN) is a type of permutation test. The latter is a test where the $p$-value is the proportion of data configurations yielding a test statistic as extreme as the value observed in the research results (Edgington and Onghena, 2007). In our case, this translates to the proportion of permutations that are at most as far from the reference as the solution's ranking.

If there are no ties in the ranks, the $L_1$-norm is commonly referred to as Spearman's footrule. Initially dismissed by Kendall (1970) due to a lack of known statistical properties, later research by Diaconis and Graham (1977) proved that the distribution composed of footrule distances is asymptotically normal. In combinatorics, this distance is known as the total displacement of a permutation, and its distribution is of independent interest. Generating the exact distribution is computationally intractable, as the problem is equivalent to counting weighted Motzkin paths of a given area (Guay-Paquet and Petersen, 2014). The distribution for $n \leq 50$ is available in the On-Line Encyclopedia of Integer Sequences (OEIS Foundation Inc., 2024). However, if ties are present SRD scores do not follow any special distribution and its density curve becomes jagged.

CRRN enables the grouping of solutions, distinguishing acceptable methods similar to the reference from those that are not. The second validation option in SRD is cross-validation, aiding in identifying the true order of solutions with the same or very close SRD scores. Originally, Kollár-Hunek and Héberger (2013) proposed the Wilcoxon signed-rank test for cross-validation purposes. The rSRD package implements two additional statistical tests: the Dietterich $t$-test (Dietterich, 1998) and Alpaydin's $\mathcal{F}$-test (Alpaydin, 1999), both popular cross-validation tools in machine learning. Sziklai et al. Sziklai et al. (2025) compared the performance of all three tests on synthetic and real data under various parameterizations and input sizes. The best-performing method was the Wilcoxon test with 8 folds, which is the default method of the package. Nevertheless, rSRD provides all three tests as they drastically differ in type I and type II error rates. Although the Wilcoxon-test proved to be a bit too sensitive in type I scenarios, it is the only method that performed well in type II situations. On real data, the advantage of Alpaydin and Dietterich methods in type I cases diminishes.

## 3 SRD

### 3.1 Getting and installing rSRD

The rSRD package is downloadable from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=rSRD` with this paper referring to package version 0.2.0. For efficiency, computationally intensive tasks are implemented in C++. This reduces overhead associated with repeated calls and memory management in R. In particular, the main functionalities (`calculateSRDValues, calculateSRDDistribution` and `calculateCrossValidation`) as well as one of the utility functions (`utilsRankingMatrix`) are programmed in C++. The C++ code is interfaced to R using the established package Rcpp by Dirk Eddelbuettel (Eddelbuettel, 2013). This setup implies that Windows users need to have Rtools with a version greater than or equal to 4.5 installed to build rSRD from source. Furthermore, rSRD imports functionality from the R packages dplyr, ggplot2, ggrepel, gplots, janitor, rlang, stringr and tibble.

### 3.2 Mathematical description of SRD

The input of the SRD-test is an $n \times m$ matrix $A = [a_{ij}]$. The first $m-1$ columns represent the models or methods (*solutions*) that we would like to compare, while rows represent measurements or features (*objects*). The last column has a special role, it contains the reference values for each row. This can be an external reference: a gold standard, a benchmark value, or a previous measurement. In the absence of a known gold standard, the reference can be extracted from the data. This step is referred sometimes to as data fusion or preference aggregation depending on the field. A common solution is to create a new column by taking the average of the row values for each row. The underlying idea is that the random errors of the measurements follow normal distribution and cancel each other out. In the presence of outliers, the row median is also a sensible choice. Furthermore, if the row medians form a ranking then it is the ranking that minimizes the total distance from the

| A | B | C | ref | | | | A | B | C | ref |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 6 | 6 | ⟶ | | ⟶ | 1 | 2 | 4 | 3 |
| 5 | 1 | 3 | 1 | ⟶ | rank | ⟶ | 2 | 1 | 2.5 | 1 |
| 7 | 6 | 2 | 5 | ⟶ | transformation | ⟶ | 3 | 3 | 1 | 2 |
| 8 | 10 | 3 | 7 | ⟶ | | ⟶ | 4 | 4 | 2.5 | 4 |
| | | | | | | $SRD$ | 4 | 2 | 5 | |
| | | | | | | $\underline{SRD}$ | 0.5 | 0.25 | 0.625 | |

**Table 1:** Example SRD computation with three solution columns (A,B,C) and a reference. Left: input table; Right: the computation of absolute and normalized SRD values after the rank transformation.

solution rankings in $L_1$-norm (Dwork et al., 2002), in other words, it is the ranking closest to the solutions. Depending on the use cases, the row minimum or maximum can also serve as a reference.

After the reference is fixed, a ranking transformation is performed on the input matrix. Values in each column are replaced by ranks. The smallest value receives a rank of 1, the second smallest gets a rank of 2, and so forth. The largest value is replaced by $n$. Ties are resolved by fractional ranking, tied values are replaced by the arithmetic mean of their corresponding ranks. For instance, if $m_{ij} = m_{kj}$ are tied for the 6th and 7th places, they are each replaced by a rank of 6.5. Let us denote the rank-transformed matrix by $R = [r_{ij}]$. The SRD score of the $j$th solution is the distance between the $j$th column and the reference column in $L_1$-norm

$$SRD_j = \sum_{i=1}^{n} |r_{ij} - r_{im}|.$$

SRD values are normalized by the maximum distance between rankings of size $n$, which has the following explicit formula

$$f(n) = \left\lfloor \frac{n^2}{2} \right\rfloor = \begin{cases} \frac{n^2}{2} & \text{if } n \text{ is even} \\ \frac{n^2-1}{2} & \text{if } n \text{ is odd} \end{cases} \tag{1}$$

Normalized SRD, denoted by $\underline{SRD}_j = SRD_j / f(n)$ is a number between 0 and 1, where 0 means that the solution produces the same ranking as the reference, while 1 means the solution ranks the objects in reverse order. Table 1 features a small example that demonstrates the ranking transformation and the SRD calculation.

### 3.3 Validation options

SRD scores already establish a ranking between the solutions. With the validation steps, we are able to extract more information.

One immediate question is whether there are unsuitable solutions. In many applications, the objective is not solely to identify the single best option but rather to have the flexibility to select from a set of solutions that meet certain criteria or requirements. In other scenarios, solutions may represent possible underlying factors of the reference ranking, and our goal is to identify the relevant ones. In such cases, we are interested in distinguishing the good solutions from the bad ones—those that have no significant connection to the reference. Comparison of Ranks with Random Numbers (CRRN) aims to address this issue.

Another interesting question is how reliable the SRD ranking is or how to group the solutions. If there is a huge gap between the SRD scores of the two solutions, by all likelihood their order is correct. But what happens if the SRD values fall close to each other? In particular, what should we do with tied values? To determine whether two rankings are essentially the same or not cross-validation is applied combined with statistical testing (CVST).

#### CRRN

One validation option of SRD is Comparison of Ranks with Random Numbers (CRRN), where SRD scores of the solutions are compared with those of random rankings. The permutation test, a standard technique in statistics, helps determine whether the results occurred by chance. By analogy, consider two binary sequences, $A = 0000000000$ and $B = 1010100011$. While binary sequences of the

same length have an equal probability of emerging as a result of a random coin toss, we might still intuitively feel that sequence $A$ is less random than $B$. From a statistical standpoint, our intuition is not entirely unfounded. Coin tosses with very few heads are rare, and the longer the sequence, the rarer they become. For instance, the probability of a ten-bit long sequence containing less than 3 heads is around 0.055, slightly higher than the usual significance threshold. In this example, a person would rightfully be cautious about accepting sequence $A$ as a result of fair coin tosses.

The concept behind CRRN is that a solution closely related to the reference will rank the objects more or less the same way. The less association they have, the less likely they are to rank objects similarly. To illustrate, consider a preference elicitation setting where users rate products or services (e.g. movies). Two users may have similar tastes but different evaluation habits; for instance, one may be stricter than the other. Despite having a large absolute difference in item scores, they might still rank items in the exact same way. Discordant tastes will result in different rankings. If the tastes have no relation to each other, we expect the corresponding ranking to show no pattern. Therefore, the distance between the solution and the reference ranking should be comparable to that of a random ranking and the reference ranking. This can be formulated as a null-hypothesis:

$H_0$: *The distance between a solution ranking and the reference ranking is not different from the distance between a random ranking and the reference ranking.*

A valuable feature of SRD is that it measures both similarity and dissimilarity. Consequently, we can reject $H_0$ in two occasions. A normalized SRD score close to 0 implies that the solution is similar to the reference—it is improbable that the resemblance happened by chance. On the other hand if the score is almost 1 then the solution vector ranks the components in a reverse order compared to the reference, which is equally unlikely.

Note that the SRD distribution depends heavily on the size of the input $n$ and the number of ties occurring in the solutions and the reference. Also, there is more than one meaningful way to derive the SRD distribution.

Diaconis and Graham (1977) provided a characterization for the SRD distribution when ties are not present. Let $S_n$ be the set of permutations of the first $n$ natural numbers and let $p, r \in S_n$ be chosen independently and uniformly. Furthermore let $D(p, r)$ be the random variable corresponding to the distance between $p$ and $r$ in $L_1$-norm. Then the following is true.

**Theorem 1** *(Diaconis and Graham, 1977)*
If $n$ tends to infinity

$$\mathbb{E}\left(D(p,r)\right) = \frac{1}{3}n^2 + \mathcal{O}(n)$$
$$var\left(D(p,r)\right) = \frac{2}{45}n^3 + \mathcal{O}(n^2)$$

and $D(p, r)$ follows asymptotically normal distribution.

This implies that when there are no ties, the expected value of a normalized SRD score of two random rankings is around $\frac{2}{3}$ (cf. Eq. 1). There are a few caveats here. The theorem does not help us when $n$ is small. In practice, the approximation error becomes sufficiently small when $n > 13$. For smaller $n$ the exact distribution can be calculated by considering every permutation of length $n$. However, the presence of ties spoils the normality property and produces a deformed (zigzagged) probability density function (see Section 5).

In addition, it is not clear how the ties should be handled in the empirical SRD distribution. Should we fix the reference ranking and only generate random rankings representing the solutions? Should the number of ties in the random ranking follow the frequency of ties in the reference or the frequency of ties in the solution vectors? If there is one solution that has many ties, while the others do not, should this affect the CRRN test of all solutions or just the one which exhibits many ties? There is no universally correct answer to these questions. The rSRD package is specifically designed to accommodate such situations, allowing the practitioner to have control over how to handle them. It provides the flexibility for the practitioner to choose a distribution that best describes their specific use case, enabling them to make informed decisions based on their requirements and preferences.

## CVST

It is not always straightforward to rank the solutions based on their distance from the reference. Solutions with the same or very close SRD scores are not necessarily similar to each other. Rankings

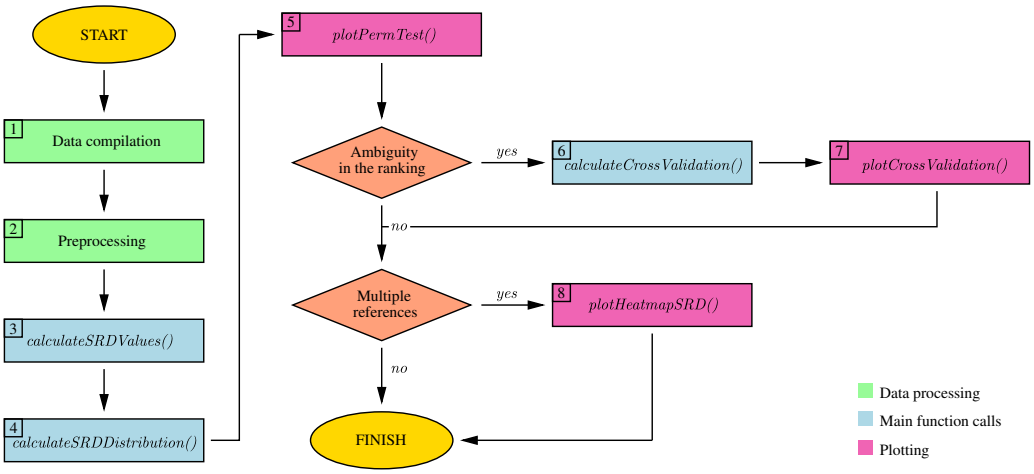



**Figure 1:** Typical workflow

can differ from the reference in different sections and still be of the same distance. Cross-Validation combined with Statistical Testing (CVST) is designed to determine whether two solutions are inherently the same or not. Consequently, the null-hypothesis is formulated as follows. Let $r^i$ and $r^j$ be two rankings corresponding to solution $i$ and $j$, in other words $r^j$ is the $j$th column vector of $R$.

$H_0$: *$r^i$ and $r^j$ come from the same distribution.*

Since it is impossible to draw a statistical conclusion by comparing two single SRD values, a sample is created using cross-validation. Randomly selected rows are discarded, and the SRD values are re-calculated on the remaining table. The SRD scores vary a bit depending on whether the omitted section agrees with the reference or not. In this way, we can assign a set of SRD values to the solutions and infer their relationship by comparing the samples. Solutions are ranked based on the median of the computed SRD scores with ties broken sequentially using the first quartile, third quartile, minimum, and finally maximum values.

Ranking the solutions is a delicate issue as we have to strike a balance between type 1 and type 2 errors. The literature suggests a sample size between 5 and 10 (Hastie et al., 2009). Increasing the sample size (*i.e.* the number of folds), above 10 will increase the bias but lower the variance[5].

The rSRD package features the three most widely used statistical tests that are coupled with cross-validation: Wilcoxon, Dietterich, and Alpaydin. The reason why three different tests are provided is that they handle type 1 and type 2 situations drastically differently (Sziklai et al., 2025). Wilcoxon excels in type 2 scenarios but it is inefficient in type 1 situations. In other words, it effectively identifies when solutions come from different distributions, but sometimes differentiates between solutions that come from the same distribution. Dietterich and Alpaydin perform quite well in type 1 cases but fail badly in type 2 scenarios. Ultimately only the practitioner knows which type of error is more costly to her, hence we implemented all three tests in the package.

Describing the exact mathematical formulation of the three statistical tests as well as discussing their performance on various data structures goes well beyond the scope of the manuscript. A detailed description can be found in ref. (Sziklai et al., 2025).

### 3.4 Typical workflow

Figure 1 illustrates the typical workflow of an SRD analysis. The rSRD package expects a data frame as input. Once the data have been compiled into the appropriate format (step 1), the user must decide how to preprocess the data (step 2). Standardization is not necessary—although the package provides tools for this purpose—because the statistical procedure involves a ranking transformation anyway. For more information on when standardization is useful, see Section 4.

The distances from the reference establish a ranking of the solutions (step 3). Generating a

[5]Note that Hastie et al. (2009)'s results relate to a slightly different setting. In our framework, the rows discarded in different cross-validation folds might partially overlap, hence the validation process resembles to bootstrapping. Moreover, although CV is conventionally used in predictive analyses, here its purpose is closer to a resampling-based stability analysis.

| | Botenga | Bompard | Ernst | Chahim | Pereira | Buzek | Groothuis | Kaljurand | Rego |
|---|---|---|---|---|---|---|---|---|---|
| Budget/ Costs | 12 | 20 | 17 | 0 | 7 | 17 | 8 | 1 | 13 |
| Climate/ Environment | 137 | 293 | 232 | 43 | 44 | 136 | 82 | 0 | 174 |
| Economy | 48 | 79 | 46 | 14 | 14 | 35 | 54 | 13 | 20 |
| Energy | 121 | 244 | 169 | 48 | 68 | 204 | 153 | 0 | 145 |
| Enterprise | 79 | 71 | 79 | 12 | 82 | 80 | 164 | 8 | 82 |
| Future | 17 | 66 | 65 | 4 | 11 | 35 | 75 | 40 | 16 |
| Geography | 53 | 121 | 85 | 13 | 28 | 49 | 39 | 13 | 44 |
| Health | 135 | 168 | 39 | 1 | 52 | 0 | 6 | 0 | 107 |
| Industry | 99 | 211 | 105 | 25 | 47 | 75 | 63 | 4 | 107 |
| Infrastructure | 14 | 50 | 32 | 10 | 17 | 54 | 83 | 5 | 24 |
| Labour | 66 | 111 | 84 | 9 | 40 | 3 | 5 | 2 | 40 |
| Mining/ Resources | 27 | 155 | 138 | 11 | 0 | 112 | 50 | 0 | 115 |
| Mode of conduct | 35 | 48 | 34 | 9 | 6 | 7 | 16 | 13 | 28 |
| Science | 47 | 59 | 28 | 6 | 21 | 13 | 30 | 0 | 46 |
| Security | 18 | 67 | 39 | 1 | 5 | 18 | 27 | 76 | 20 |
| Social | 65 | 74 | 80 | 16 | 37 | 16 | 13 | 4 | 49 |
| $\underline{SRD}_i$ | 0.234 | 0.297 | 0.312 | 0.352 | 0.352 | 0.484 | 0.547 | 0.891 | 0 |

**Table 2:** Profiles of the Members of the European Parliament (MEP). Breakdown of the number of expressions used by MEPs in amendments they co-sponsored by categories. MEP Sira Rego's profile serves as a reference. The last row displays the derived SRD scores.

distribution of random distances allows statistical significance to be assessed (step 4). Plotting the result of the permutation test helps visualize the obtained ranking (step 5). If there is any ambiguity, such as similar or tied SRD scores, cross-validation can help clarify the ranking (step 6). A box-and-whisker plot provides a visual summary of the cross-validation results (step 7). Finally, if more than one column qualifies as a reference, a heatmap can help compare the resulting distances and assist in choosing the right reference (step 8).

### 3.5 Demonstrative example

In this section, we present a small case study to demonstrate SRD and its validation options. The data is courtesy of Eulytix. For presentational reasons, figures were re-created and enhanced by Graphics Layout Engine (Pugmire et al., 2022).

The major part of the EU's legislative work is done in the European Parliament's Committees. Members of the European Parliament (MEPs) draw up, amend, and adopt legislative proposals and work out reports to be presented to the plenary assembly. To boost the chance of acceptance, MEPs routinely approach their fellow members to co-sponsor their initiative. EP Committees suffer from partisanship much less than national assemblies. This allows MEPs to draw supporters from a wider pool, quite possibly from parties not belonging to the same ideological family. One way to identify potential allies is to analyze the language they are using. In the following example, the texts of the amendments of the Committee on Industry, Research and Energy (ITRE) are analyzed in the 2014-2019 legislative term to profile MEPs. Expressions are classified into 16 categories (climate, infrastructure, security, etc.) and counted how many times an MEP used an expression belonging to a certain category in the amendments she co-sponsored. This reveals which areas the MEP deems important. Our assumption is that MEPs that have a similar profile are the most probable allies.

Table 2 compiles the language profile of 9 MEPs of the ITRE Committee. The last column shows the number of expressions the MEP Sira Rego used in the amendments she co-sponsored. The last row shows the normalized SRD scores. MEP Botenga's profile is the closest to the reference, which is MEP Rego's language-pattern. Both of them find Climate/Environment the most important while contribute to Budget/Costs category the least relatively to other topics. In general, they rank the topics the same way, which suggests that they might be ideologically close.

Figure 2 shows the result of the CRRN test. For sake of simplicity, SRD distribution was generated by assuming there are no ties, although this is not true. Both the reference and the solution vectors contain tied values. Six MEPs, Botenga, Bompard, Ernst, Chahim, Pereira, and Buzek pass the test, meaning the SRD value corresponding to their profile is below the 5% significance threshold, marked by the first dashed line labeled XX1.

There are a few things worth noting. First, MEP Buzek's SRD value falls exactly on the 5% threshold. Had we considered the ties when generating the SRD distribution, Buzek's profile would have fallen to the right of the threshold[6]. Second, MEP Kaljurand's profile is so different from Rego's,

[6]Technically, this makes no difference as we cannot reject the null hypothesis on either case. However, it shows that with enough data, some bound to fall near to the threshold, and a change in the distribution could mean pushing a solution over the threshold.

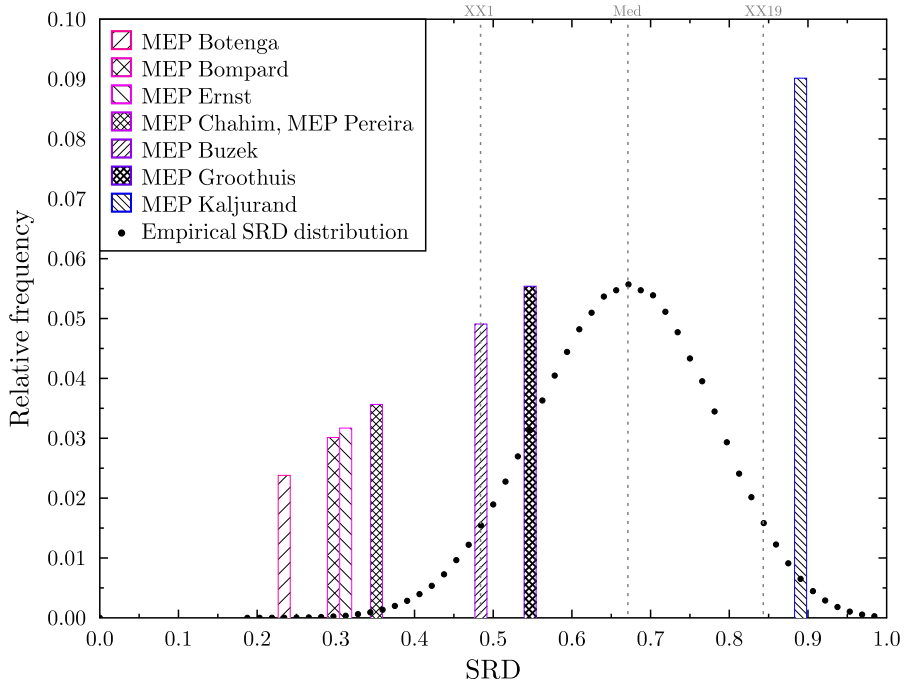



**Figure 2:** CRRN test: The colored bars from left to right follow the same order as the legend from top to bottom, and their heights are equal to their normalized SRD values, hence expressing the distance from zero. The further they fall from the origin, the less they resemble the reference. The vertical lines, XX1 and XX19 correspond to the 5% and 95% threshold respectively.

that it cannot even be considered random, but rather Rego's exact opposite. Reverse orderings are always revelatory, in our case it probably indicates that Kaljurand's and Rego's worldview is not compatible. This should be taken with a grain of salt as we have much fewer observations for Kaljurand, than for other MEPs. Groothuis' SRD score falls between the 5% and 95% significance thresholds meaning it cannot be distinguished from a random ranking. That suggests, that Groothuis is neither a probable ally nor an adversary of Rego. Finally, MEPs Chahim and Pereira's rankings are of the same distance from the reference and Bompard's and Ernst's SRD scores are also very close. This calls for cross-validation.

Figure 3 displays the result of the cross-validation coupled with an 8-fold Wilcoxon test. We took 8 samples from the input matrix, by repeatedly removing two random rows, calculated the SRD values on all the samples, then ordered the solutions (*i.e.* the MEPs) by the median of the SRDs, finally performed a Wilcoxon test between the adjacent solution-pairs. The box-and-whisker plot provides a visual summary of the SRD values obtained during cross-validation. Non-overlapping boxes may suggest that the corresponding SRD distributions differ substantially. The test found that there is no significant difference between Bompard and Ernst and Chahim and Pereira. The choice of the test matters. Wilcoxon is much more sensitive than Alpaydin or the Dietterich test. A 10-fold Alpaydin-test finds only the last difference (between Groothuis and Kaljurand) significant, while a 10-fold Dietterich finds no significant difference between the *consecutive* solution-pairs.

## 4 Package features

In this section, we review the features of the package through examples. The functions of rSRD are divided into three categories. Core functions that relate to SRD computation and validation are given a name that starts with the '`calculate`' prefix. Plot generating functions start with the '`plot`', while utility functions start with the '`utils`' prefix.

Throughout this section, we will demonstrate package features using the MEP profiles we introduced in the previous section and a football dataset compiled from https://www.whoscored.com/. Table 3 presents various game statistics aggregated over the 2020/21 season of the Bundesliga. The last column displays the points gathered by the teams. The final position in the league table is a natural reference that encompasses how strongly a team performed during the season. Some game statistics closely follow, while others seem to be independent of this ranking. With an SRD analysis we can uncover the style of play in the Bundesliga and explore which game elements

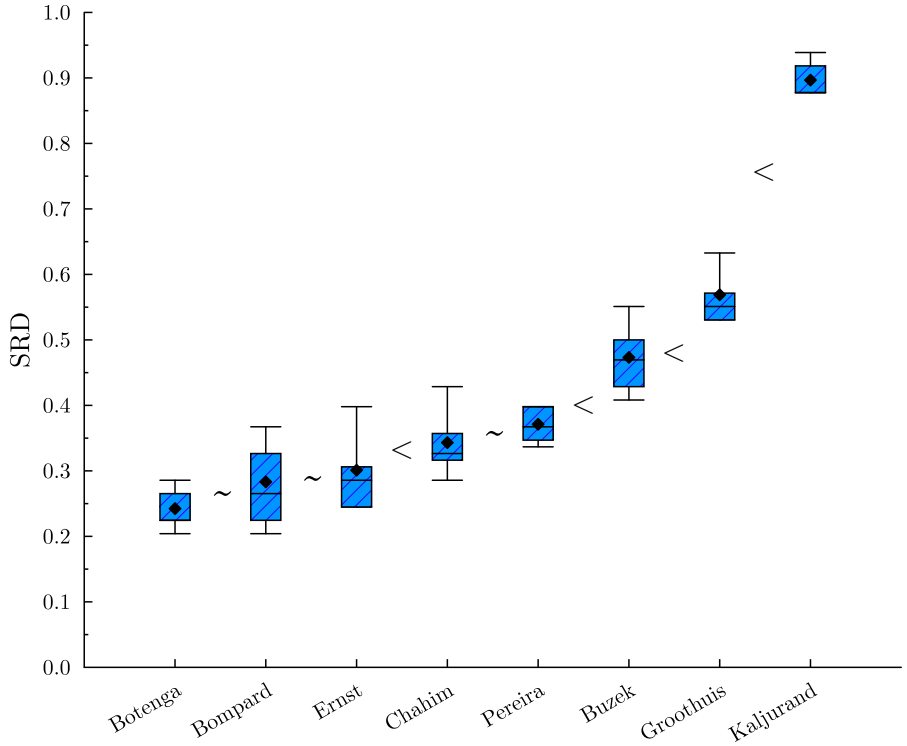



**Figure 3:** Cross-validation combined with Wilcoxon test: Box whisker plot representing the SRD values calculated on the different folds. The < symbol denotes significant difference between the values at 5% significance level, while ∼ marks that the null-hypothesis could not be rejected.

dominate and which ones are not pertinent to the success of the teams. In the following, we will use '`mep_profiles.csv`' and '`bundesliga20_21.csv`' to demonstrate package features. Both files are accessible from the package.

```
R> path <- system.file("extdata", "mep_profiles.csv", package = "rSRD")
R> profiles_df <- read.csv(path, header = TRUE, sep = ";", row.names = 1)
R> path <- system.file("extdata", "bundesliga20_21.csv", package = "rSRD")
R> bundesliga_df <- read.csv(path, header = TRUE, sep = ";", row.names = 1, check.names = FALSE)
```

### 4.1 Preprocessing

The rSRD package offers a variety of utility functions. The Sum of Ranking Differences relies on the existence of a reference vector. One issue that often comes up in applications is the need for creating a reference. If an external reference (e.g. industry standard, known theoretical value) is not readily available, we might still be able to extract one from the data. For instance, if the solutions represent independent measurements then taking the average of the columns (*i.e.* the average of the row values for each row) may work as a reference as the random errors cancel out. The median is often preferred in the presence of outliers. If objects represent performance properties that the solution needs to optimize, then row minimum or maximum might be a good choice. Yet sometimes we have a mix of these: some rows correspond to measurements, while other rows are performance traits.

In such cases, there are two issues we need to resolve. The first issue is that objects are not necessarily of the same kind; hence, their values do not fall on the same scale. For instance, the source data for the football leagues dataset measured the number of red and yellow cards in bulk for the whole season, while the number of offsides was averaged over the number of games. Even if the objects are on the same scale, it might be beneficial to standardize the values. For instance, in the MEP profiles, not every representative was equally productive. MEPs who submitted more amendments inevitably generated more phrases in each category, thus the absolute numbers are a bit misleading.

For the sake of convenience, rSRD includes standardization functions. Four methods are implemented: `scale_to_unit`, `standardize`, `range_scale` and `scale_to_max` to meet the different

| Teams | Shots pg | RY cards | Possession% | Pass | Dribbles pg | Offsides pg | Fouls pg | pts |
|---|---|---|---|---|---|---|---|---|
| Bayern Muenchen | 19.8 | 38 | 64.8 | 86 | 14.5 | 2.2 | 9 | 77 |
| Bayer Leverkusen | 13.5 | 66 | 53.7 | 81.8 | 11.8 | 1.9 | 10.7 | 64 |
| Borussia Dortmund | 13.3 | 62 | 59.4 | 84 | 10.4 | 2.1 | 10.6 | 69 |
| RB Leipzig | 12.9 | 49 | 56.5 | 83.1 | 10.2 | 1.9 | 10.6 | 58 |
| SC Freiburg | 13.6 | 34 | 48.6 | 76.2 | 6.6 | 1.7 | 11.5 | 55 |
| Borussia M.Gladbach | 14.8 | 67 | 54.1 | 82 | 10.9 | 2.2 | 10.6 | 45 |
| 1. FC Koeln | 13.8 | 67 | 54.8 | 77.4 | 7.4 | 2 | 12.3 | 52 |
| FSV Mainz 05 | 13.8 | 62 | 46 | 74.1 | 7.9 | 1.9 | 14.6 | 46 |
| VfL Wolfsburg | 12.4 | 61 | 50.2 | 78.6 | 9.4 | 2.2 | 12.1 | 42 |
| VfB Stuttgart | 13.3 | 64 | 50.4 | 80.7 | 11 | 1.6 | 10.9 | 33 |
| Union Berlin | 12.1 | 62 | 43.3 | 73.6 | 7.2 | 1.9 | 12.4 | 57 |
| Eintracht Frankfurt | 13.2 | 60 | 49.4 | 76.2 | 8.4 | 1.6 | 12.4 | 42 |
| TSG Hoffenheim | 13.3 | 75 | 53.2 | 80.7 | 7.4 | 2.1 | 12.6 | 46 |
| VfL Bochum | 12.1 | 55 | 44.5 | 72.1 | 6.8 | 2.2 | 12.5 | 42 |
| FC Augsburg | 10.8 | 74 | 40.6 | 72 | 7.7 | 2.1 | 13.2 | 38 |
| Arminia Bielefeld | 10.7 | 55 | 39.9 | 71.7 | 8.4 | 1.6 | 12.7 | 28 |
| Hertha BSC Berlin | 10.8 | 64 | 43.2 | 74.7 | 8.3 | 2.1 | 12.4 | 33 |
| Greuther Fuerth | 9.2 | 61 | 43 | 74.8 | 7.8 | 2 | 12.9 | 18 |

**Table 3:** Game statistics of Bundesliga teams aggregated over the season 2020/21 (pg stands for 'per game', RY abbreviates red and yellow).

needs.

`scale_to_unit` transforms each column vector into a vector of length 1.

`standardize` converts the values of each column vector into a standard normal scale.

`range_scale` casts the values of each column vector into the $[0, 1]$ interval.

`scale_to_max` divides each column vector by the maximum value of that column.

For instance, the practitioner may prefer to use `standardize` if there are both positive and negative values present in the data, or use `range_scale` if she wants to enforce each column into the same interval. Note that `utilsPreprocessDF()` transforms the columns. If the user would like to standardize the rows, the data frame should be transposed first.

```
R> utilsPreprocessDF(profiles_df, method="range_scale")
```

After range scaling the MEP profiles, it is immediately clear what topic is the most important for each representative. In the case of MEP Chahim, it is Energy, while the value 0.25 under Enterprise indicates that Chahim expressed a quarter of the amount of energy-related phrases about enterprises.

It is important to note, that each data preprocessing method is strictly monotonic. Hence, if we already have a fixed reference, the ranking transformation of SRD makes preprocessing *by columns* redundant, meaning we will obtain the same SRD values with or without using preprocessing. However, if we normalize the rows, or we obtain the reference values as a function of the columns then normalization alters the outcome, and different normalization processes may lead to different SRD scores.

The second issue is the extraction of a reference when the rows correspond to different properties. The rSRD package makes reference creation a flexible process. The `utilsCreateReference()` method appends a new column at the end of the data frame. The `method` parameter describes the aggregation process. Five possibilities are available: `max, min, median, mean` and `mixed`. In the case of the first four, the row maximum, row minimum, row median, or row mean value is appended respectively.

```
R> SRD_input <- data.frame(
  A=c(2, 5, 7, 8),
  B=c(5, 1, 6, 10),
  C=c(6, 3, 2, 3))

R> utilsCreateReference(SRD_input, method = "mean")
```

```
  A  B C  refCol
1 2  5 6   4.33
2 5  1 3   3.00
3 7  6 2   5.00
4 8 10 3   7.00
```

If the method is set to `mixed`, then we can specify an aggregation process for each row separately. For instance,

```
R> ref <- c("max","min","mean","mean")
R> utilsCreateReference(SRD_input, method = "mixed", ref)
```

```
  A  B C refCol
1 2  5 6    6
2 5  1 3    1
3 7  6 2    5
4 8 10 3    7
```

creates a reference for `SRD_input` by taking the maximum of the first row, the minimum of the second, and the mean of the last two. The size of the `ref` vector must match the size of the input. The need for such a composite reference may arise when the objects or variables are not all of the same type. For instance, if the rows represent performance indicators, the average may be a sensible reference for some indicators, whereas the best or worst observed performance may be more appropriate for others. Note that the function does not change the input, only returns with a new matrix.

rSRD provides a function for detailed SRD calculation, displaying the ranking transformation, the distance calculation, and the raw (unnormalized) SRD scores.

```
R> SRD_input = utilsCreateReference(SRD_input, method = "mixed", ref)
R> utilsDetailedSRD(SRD_input)
```

```
  A  A_Rank A_Dist  B B_Rank B_Dist  C  C_Rank C_Dist  refCol refCol_Rank
1 2      1      2     5      2      1   6      4    1.0      6        3
2 5      2      1     1      1      0   3    2.5    1.5      1        1
3 7      3      1     6      3      1   2      1    1.0      5        2
4 8      4      0    10      4      0   3    2.5    1.5      7        4
5 -      -      4     -      -      2   -      -    5.0      -        -
```

This utility serves multiple purposes—it allows users to visually check which part of the solution differs from the reference. Additionally, it is beneficial for pedagogical reasons, enabling users to quickly comprehend how the statistics are computed. Since the output of this function displays unnormalized SRD values, the user can promptly query the normalizing factor by calling

```
R> utilsMaxSRD(4)
```

```
8
```

which just returns the result of Eq. 1. The rSRD package also allows to perform the ranking transformation without the SRD calculation. This option was built in to ensure compatibility with other ranking-based packages. In this way, the user can export the ranking matrix without the need to format the output.

```
R> utilsRankingMatrix(SRD_input)
```

```
  A B   C
1 1 2 4.0
2 2 1 2.5
3 3 3 1.0
4 4 4 2.5
```

### 4.2 Core functions

There are three core functions, one corresponding to the SRD calculation and the other two for the validation steps. Each function is capable of exporting the results to a CSV file. First, let us look at the SRD values of the Bundesliga data.

```
R> calculateSRDValues(bundesliga_df)

[1] 0.3395062 0.7037037 0.3148148 0.3950617 0.6049383 0.6604938 0.8888889
```

The output follows the order of the columns in the input, in this case the column order of Table 3. The scores range over almost the entire $[0, 1]$ interval. The number of shots is a good indicator of the final position in the league table, as it ranks the football teams similarly as the obtained points. On the other hand, fouls committed per game seem to rank the teams in reverse order: worse teams tend to have more fouls. Which of these scores are significant?

```
R> calculateSRDDistribution(bundesliga_df, option = "f", seed = 42)

SRD_Distribution
    SRD_value   relative_frequency
1      0.0000         0.000000
2      0.1728         0.000001
3      0.1975         0.000002
4      0.2099         0.000001
...
127    0.9877         0.000025
128    0.9938         0.000005
129    1.0000         0.000018

xx1
[1] 0.4938
q1
[1] 0.5926
median
[1] 0.6667
q3
[1] 0.7346
xx19
[1] 0.8272
avg
[1] 0.6630454
std_dev
[1] 0.1021337
```

The XX1 and XX19 values specify the 5% and 95% significance thresholds. Thus, the number of shots, ball possession, successful passes, and fouls committed are statistically significant. The first three rank the teams similarly to the reference ($\underline{SRD}_i < XX1$), while fouls rank teams in reverse order ($\underline{SRD}_i \geq XX19$). The function `calculateSRDDistribution` has five parameters, of which only the first, the input dataframe, is compulsory. The second parameter specifies how the SRD distribution should be generated. The available options are the following:

**'n'** There are no ties for the solution vectors, the reference vector is fixed.

**'r'** There are no ties. Both the column vector and the reference are generated randomly.

**'t'** Ties occur with a fixed probability specified by the user for both the solution vectors and the reference vector.

**'p'** Ties occur with a fixed probability specified by the user for the solution vectors, the reference vector is fixed.

**'d'** Tie distribution reflects the tie frequencies displayed by the solution vectors, the reference vector is fixed.

**'f' (default)** Tie distribution reflects the tie frequencies displayed in the reference, the reference vector is fixed.

In each case, one million data points are produced by generating (either randomly or deterministically) solution and reference vectors and calculating their corresponding SRD scores. The third parameter, '`tie_probability`' only plays a role if either options 't' or 'p' were chosen, in which case they specify the tie frequencies occurring in the generated vectors. The package offers a utility function to check the number of ties present in a vector.

```
R> solution <- c(1,3,3,3,2,2,4,3)
R> utilsTieProbability(solution)

[1] 0.5714286
```

Note that in an $n$-long vector, ties can occur in $n-1$ places. In the above example, the values are ordered first to create a ranking. Hence, 4 out of the 7 places were tied and the result is $4/7 = 0.5714286$. If option 'r' is chosen, then both the solution and the reference are generated randomly without ties. In this case, the empirical distribution follows the Spearman's footrule distribution.

Since generating the distribution is a stochastic process, the function is equipped with a `seed` parameter, which allows the user to seed the random number generator and make the calculations reproducible. The user can also save the computation by setting the `output_to_file` parameter to TRUE.

The choice of how to generate the distribution can have a huge impact on its probability density function. Although the values of XX1 and XX19 are more robust, small changes can mean that solutions falling near the significance threshold can gain or lose significance based on the selected method of generation[7]. Therefore, the prudent way to design a hypothesis test is to select the generating method *before* calculating the SRD values. Consequently, it is worthwhile to contemplate which option is suitable for which circumstances.

If the reference is based on observations like in the case of MEP profiles or the points obtained by football teams during a season, then its realization is prone to change with repeated measurements. The number of observed ties would vary with each measurement, thus there is no reason to treat the reference during the distribution generation fixed (see options 'r' or 't'). If, however, the reference is derived from established behavior, like for instance from theoretical properties of a substance, then the distribution should reflect the fact that the reference will not be changing (options 'n', 'p', 'd' or 'f').

If the compared solutions come from the same distribution or their distributions are highly similar, we are not committing a significant error by generating the solutions in the same way. However, if the solutions display highly different tie frequencies, we introduce a slight bias when comparing them under the same distribution. The question is how we prefer to represent the underlying solution space. Suppose there are two solutions with tie frequencies $x$ and $y$, where $x << y$. Then we may consider generating a solution using the fixed tie probability $\frac{x+y}{2}$ (options 't' or 'p'). However, if the distance between $x$ and $y$ is too large, there is a chance that none of the generated vectors display a tie frequency as low as $x$ and as large as $y$. Another option would be to generate half the solutions with a tie frequency $x$ and the other half with a tie frequency $y$ (option 'd'), although in such cases we will end up with a hybrid distribution. If we have reason to believe that the solutions approximate or converge to the reference, using the same tie frequency for the solutions as for the reference is an adequate approach (default option). In such way, we will have an accurate estimation of the significance of solutions that have very low or very high SRD values. The point is, that we are not really interested in the solutions that behave randomly compared to the reference and exhibit an SRD value close to the expected value of the distribution.

Note that a fixed tie frequency does not mean that each vector is generated with the same number of ties. It simply means that each consecutive element in the vector is tied with the given probability. Finally, let us stress that these are recommendations and ultimately only the practitioner can decide which distribution fits best to her use case.

To take a closer look at the cross-validation function of the package, let us consider again the Bundesliga dataframe. The number of shots, ball possession, and the ratio of successful passes are all statistically significant elements with relatively close SRD values. Ball possession seems to be the most significant factor, followed by the number of shots and successful passes. Can we be definitive about their order? From a statistical point of view, the answer is no, at least without increasing the data size. However, we can extract more information from the existing data by examining how the ordering behaves under resampling. Let us compare the solutions based on the results of the Wilcoxon and Alpaydin tests.

[7]Although many authors warn that placing too much emphasis on the threshold is unreasonable and that one should simply report the $p$-value (Wasserstein and Lazar, 2016; Berrar, 2022), the economic literature continues the practice of only accepting an effect when the corresponding variable is statistically significant.

```
R> cv_Wilc <- calculateCrossValidation(bundesliga_df, seed = 137)
R> plotCrossValidation(cv_Wilc)
R> cv_Alp <- calculateCrossValidation(bundesliga_df, method =
   "Alpaydin", number_of_folds = 10, seed = 33)
R> plotCrossValidation(cv_Alp)
```

By default, the `calculateCrossValidation` function employs an 8-fold cross-validation combined with the Wilcoxon-test. Using the `method` and the `number_of_folds` parameters we can change the type of test and the number of folds used in the cross-validation. Figure 4 displays the box-whiskers plot created from the SRD scores of the different folds. Note that the SRD values vary much more under the Alpaydin-test than under the Wilcoxon-test. The reason stems from how the test performs cross-validation. Under Wilcoxon with $k$-fold cross-validation $\lceil \frac{n}{k} \rceil$ rows are removed in each fold, while under Alpaydin (and under Dietterich too) half the rows are removed in each fold.

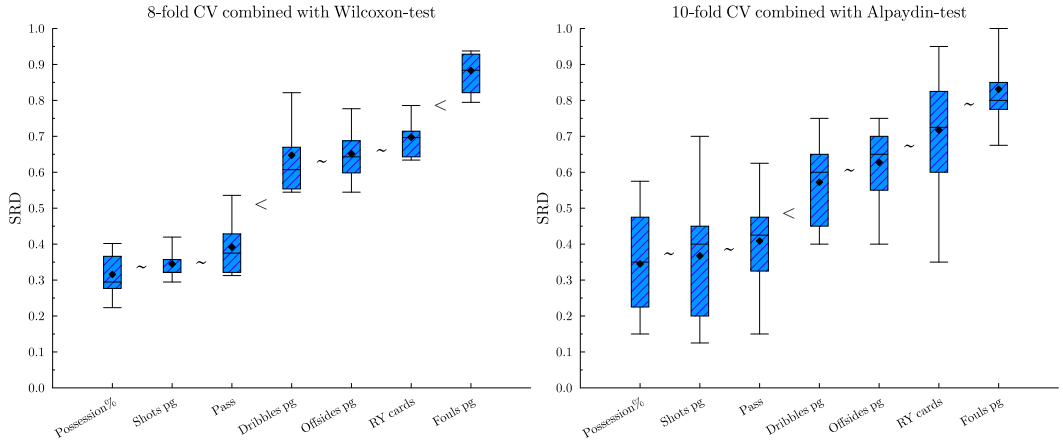



**Figure 4:** CVST. Comparing the 8-fold Wilcoxon test with the 10-fold Alpaydin test.

There are a couple of things worth noting. First, the result of the cross-validation is stochastic. Each time the function is run, new random rows are selected for each fold. To ensure reproducibility this function is also equipped with `seed` parameter. In addition, the function automatically saves not only the results but also the whole computation, including the selected rows. This feature can be disabled by setting the `output_to_file` parameter to FALSE. Some information is only saved to the CSV file, the function does not return the number and indices of the removed rows. Secondly, the function orders solution based on the median of the computed SRD scores, with ties broken by Q1, Q3, min and finally the max values. Thus, RY cards, which was the second column of our dataframe, was relayed as the penultimate column, while the number of shots switched places with ball possession. Only consecutive column pairs are tested. Thus, ball possession does not significantly differ from the number of shots, and the number of shots does not significantly differ from successful passes, but the relationship between ball possession and successful passes is not tested. Finally, significance is classified into three categories: `n.s.` (not significant), ($p < 0.1$) and ($p < 0.05*$).

```
R> cv_Wilc

new_column_order_based_on_folds
[1] 3 1 4 5 6 2 7

test_statistics
[1] 19 24 36 2 20 36

statistical_significance
[1] "n.s."  "n.s."  "(p<0.05*)"  "n.s."  "n.s."  "(p<0.05*)"

(the rest of the report can be found in the Appendix)
```

### 4.3 Plotting

The package allows plotting the results of the permutation test under the chosen SRD distribution. In the previous section, we discussed the available distributions and their advantages. Here, we

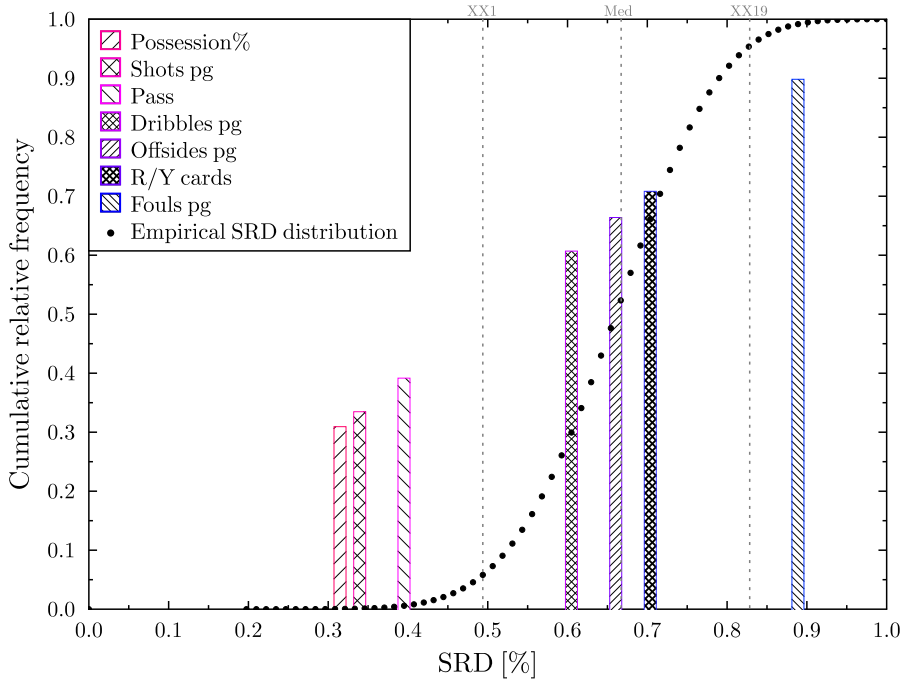



**Figure 5:** CRRN test: SRD values are compared to the cumulative distribution of random rankings generated by method 'r'.

simply note that the package enables the user to choose between plotting the probability density function and the cumulative distribution function. If the `densityToDistr` parameter is set to true, then the SRD values are compared to the cumulative distribution function (cf. Figure 5). Naturally, this does not affect the thresholds in any way.

```
R> dist_r <- calculateSRDDistribution(bundesliga_df, option = 'r')
R> plotPermTest(bundesliga_df, dist_r, densityToDistr = TRUE)
```

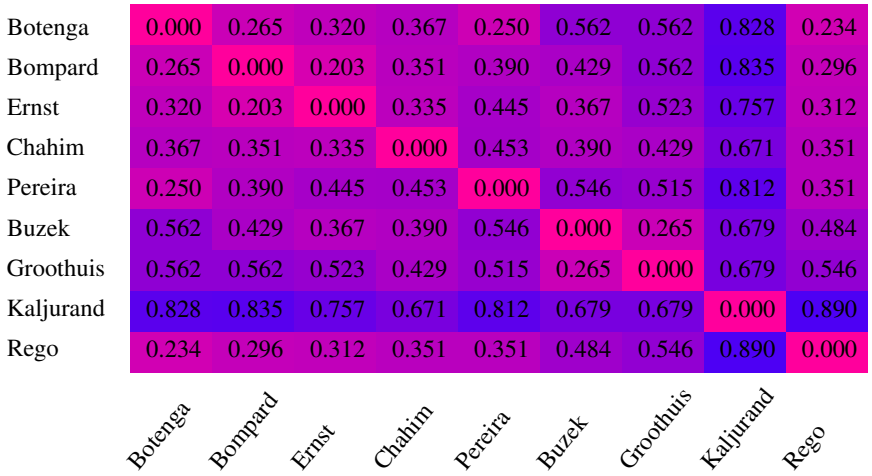



**Figure 6:** Heatmap featuring the pairwise SRD distances.

In a multiple comparison setting, each solution might serve as a potential reference. In such cases, we are often interested in seeing how far the solutions deviate from each other. For instance, in the MEP profiles dataset, we might be interested in generating recommendations not only for MEP Sira Rego but for every representative. Setting each solution as the reference manually is a cumbersome task; thus, the package offers a way to do it in one step. The `plotHeatmapSRD` function not only calculates the pairwise distances and exports them to a CSV file, but also illustrates the distance matrix using a color palette. The package offers a built-in palette where a shade of red corresponds to an SRD score of 0, while a shade of blue corresponds to an SRD score of 1.

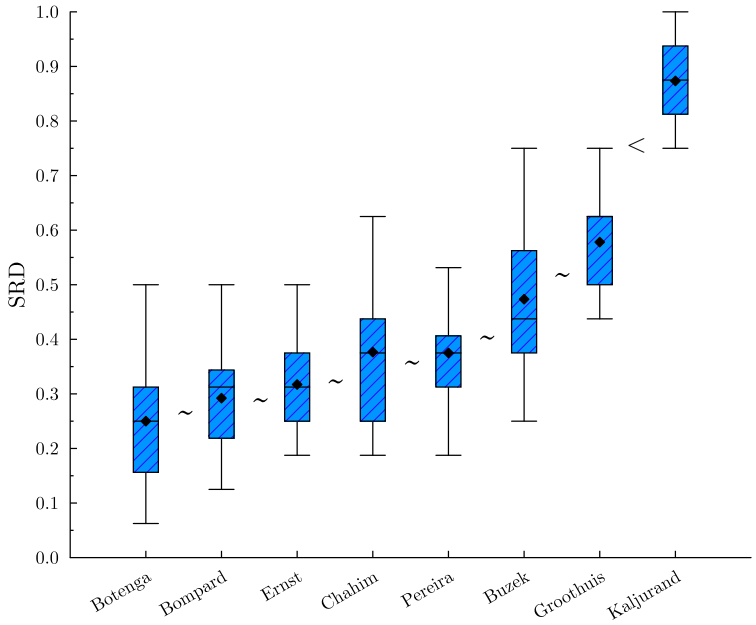



**Figure 7:** CVST. Dietterich test on the MEP profiles. The $<$ symbol denotes significant difference between the values, while $\sim$ marks that the null-hypothesis could not be rejected.

```
R> plotHeatmapSRD(profiles_df, output_to_file = TRUE, color =   utilsColorPalette)
```

The color palette can be easily customized. The size of the palette indicates how many categories the $[0, 1]$ interval is divided. For instance, the following code changes the color range from orange to green.

```
R> myPalette <- c("#eb9c34", "#ebba34", "#ebd634", "#ebe534",
                  "#d9eb34", "#b7eb34", "#99eb34", "#6beb34")
R> plotHeatmapSRD(profiles_df, color = myPalette)
```

Finally, the `plotCrossValidation` function produces a standard box-whisker plot (see Figure 7), where the whiskers mark the minimum and maximum of the SRD values calculated on the different folds. The box represents the first and third quartiles, the horizontal line inside the box is the median, while the crossmark/diamond symbol indicates the average.

```
cv <- calculateCrossValidation(profiles_df, method = "Dietterich",
      number_of_folds = 10, seed = 137)
plotCrossValidation(cv)
```

## 5 Robustness and scalability analysis

In this section, we discuss how the package behaves for untypical input.

### 5.1 Sensitivity to sample size and the presence of ties

The SRD distribution is asymptotically normal if the number of rows is sufficiently large ($n > 13$) and there are no ties. In practice, smaller data sets are common and so are tied values. Thus, it is worth to explore what happens under suboptimal data size. Let us return to the demonstrative example described in Table 2.

Suppose we only consider categories where the total number of expressions exceed 400. In this way, Climate and Environment, Energy, Enterprise, Geography, Health, Industry and Labour remains in our data frame—altogether seven rows from the original 16. In addition, there are quite a few tied values: the reference has two and the column of Kaljurand now features four.

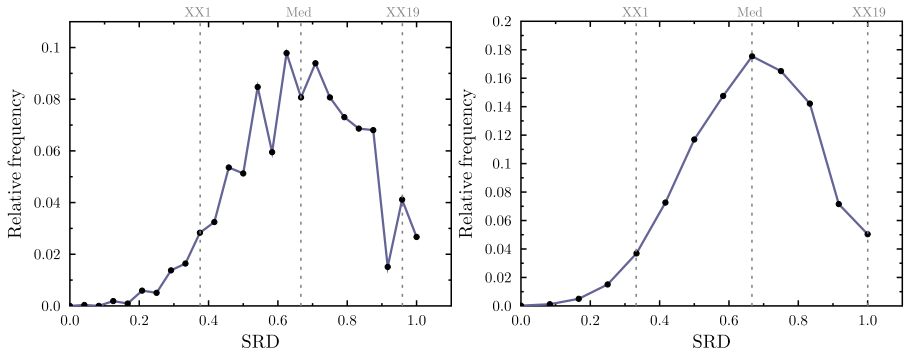



**Figure 8:** SRD distribution on the filtered MEP profiles with ties (on the left) and without ties (on the right). To help visualization the discrete relative frequencies were connected with line segments.

| | | Data Size (rows) | | | | |
|---|---|---|---|---|---|---|
| Function | Metric | 10 | 100 | 1 000 | 10 000 | 100 000 |
| calculateSRDValues | Time (s) | <0.001 | <0.001 | 0.04 | 2.74 | 136.64 |
| calculateSRDDistribution | Time (s) | 2.32 | 3.14 | 10.50 | 76.73 | 1184.30 |
| calculateCrossValidation | Time (s) | <0.001 | <0.001 | 0.05 | 2.65 | 162.67 |
| calculateSRDValues | RAM (MiB) | 0.10 | 0.70 | 57.50 | 96.0 | 90.50 |
| calculateSRDDistribution | RAM (MiB) | <0.001 | 0.30 | 23.1 | 95.90 | 90.10 |
| calculateCrossValidation | RAM (MiB) | 0.20 | 0.40 | 23.20 | 95.80 | 90.10 |

**Table 4:** Running time and Peak RAM consumption under different data sizes.

```
R> small_df <- profiles_df[rowSums(profiles_df) >= 400, ]
R> View(small_df)
R> dist_d <- calculateSRDDistribution(small_df, option = 'd', seed = 137)
R> plotPermTest(small_df, dist_d)
R> dist_r <- calculateSRDDistribution(small_df, option = 'r', seed = 137)
R> plotPermTest(small_df, dist_r)
```

The SRD distribution is asymmetric, its left tail is longer than its right, at least for small $n$. In case of ties, it also becomes less smooth. Figure 8 illustrates how the distribution looks like for options 'd' and 'r'. The 'd' option considers ties both for the solution vectors and the reference. First a probability is drawn uniformly from the set of tie probabilities featured in the solution vectors. Then a random ranking is generated with the selected tie probability. Finally, the distance between the generated ranking and the reference ranking is calculated. In contrast, option 'r' generates two random rankings of size $n$ without ties and calculates their distance. It follows, that the range of possible distance values is smaller for option 'r' than for option 'd'.

As with any other statistical test, a low sample size is problematic because it makes it difficult to separate a real effect from random noise. The $p$-values become unstable, and the test becomes highly sensitive to outliers. The asymmetry of the SRD distribution is also reflected in the uneven positions of the XX1 and XX19 thresholds: a significantly small distance can occur in various ways, but for a distance to be significantly large, it has to be near the maximum. Changing the type of SRD distribution from option 'd' to option 'r' also shifts the position of XX19 from 0.9583 to 1.

### 5.2 Performance scaling

Due to the fact that computationally demanding calculations are implemented in C++ most functions scale well with the data size. Table 4 summarizes the result of an experiment aimed to uncover how running time and peak RAM consumption changes while scaling the date size. For the experiment, we borrowed a social network dataset (Sziklai and Lengyel, 2022). The influence of the users (nodes of the network) was measured by various centrality measures. Fixing any of them as the reference, we can compare whether the other measures rank the users similarly. We took different sized samples from this data to simulate SRD calculation.

All benchmarks were performed on a laptop equipped with an AMD Ryzen 9 8940HX processor and 32 Gb RAM running Windows 11 Pro. The data frame had 10 columns. The longest running time in our experiment occurred when we calculated the SRD distribution of 100 000 data points.

One way to decrease this time would be to adjust the sample size for estimating the distribution, which is currently a hardcoded parameter.

### 5.3 Edge cases

The package is equipped with unit tests to ensure that updates do not break existing functionality or alter previously validated results. Nevertheless, it is worth exploring the behavior of the package in a few edge cases.

- Missing values generate an error message and break the function call.
- Choosing a non-existing option for `calculateSRDDistribution` (e.g. `option = 'x'`) triggers the default option.
- If the number of folds, $f$ does not fall into the $[5, 10]$ interval `calculateCrossValidation` does not display statistical significance, although the $p$-values are still determined for any fold number $f > 1$ (fractional fold numbers are rounded down).
- `calculateCrossValidation` throws an error if the CV output file from a previous run is still open.
- If the data frame contains constant columns `calculateSRDValues`, `calculateSRDDistribution`, `calculateCrossValidation` and `utilsRankingMatrix` stop with an informative error message. Note, however, that constant columns can produce missing or non-finite values when the "standardize" or "range_scale" preprocessing options are applied.

## 6 Summary and discussion

Comparing solutions through a reference is a common task that appears in various fields of science, from marketing through sports to machine learning and material science. Sum of Ranking Differences is a conceptually simple, non-parametric statistical procedure that, besides ranking the solutions, can also separate the statistically significant ones. The popularity of this comparative analytical tool stems from the fact that it was designed especially for comparing in the presence of a reference.

The rSRD package offers a comprehensive toolkit that encompasses data preprocessing, SRD computation, validation, and plotting features. Among the reviewed functions, validation is by far the most complex and demands the most consideration from the practitioner. The package features a variety of ways to compute the SRD distribution for the permutation test, and thus, can aid the practitioner in designing their hypothesis test. The choice may affect significance thresholds, so careful consideration is needed.

Since the validation options contain stochastic elements, they were designed to allow seeding. In addition the calculations can be exported, which allows for full reproducibility. Apart from simplifying the validation process, the package also offers plotting features. In the future, the package will be updated based on user feedback. In particular, radar plots are planned to highlight how the SRD scores vary between different datasets. Another useful idea is to indicate statistically significant differences in the boxplot, in a manner similar to that used in this paper; see, for example Figs. 3, 4, and 7.

Further updates aim to include an option to adjust the default sample size used for generating random rankings during the empirical distribution process. Currently, one million data points are generated, yielding precise results for all the examples tested so far. However, for very large $n$, the empirical distribution might be slightly inaccurate. Conversely, for smaller cases, generating one million data points is excessive, and reducing the sample size could speed up computation without sacrificing accuracy.

Another potential update pertains to cross-validation. Presently, cross-validation folds are generated by randomly discarding rows from the input table. In time-series analysis, it is common practice to exclude consecutive blocks rather than random rows. Incorporating block cross-validation, including predefined row sets, could be a straightforward and beneficial improvement.

## Acknowledgement

The authors are sincerely grateful to the reviewers for their expert comments, which substantially improved the paper, and to the editor for the exceptional care and attention she devoted to the review process. This work was supported by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, financed under the

K type funding scheme, K 134260 (Héberger), K 146320 (Sziklai) and the FK funding scheme 137577 (Gere). The last author thanks the Bavarian State Ministry of Science and Arts for partial funding.

## References

E. Alpaydin. Combined 5×2 cv F Test for Comparing Supervised Classification Learning Algorithms. *Neural Computation*, 11:1885–1892, 1999. [p100]

D. Berrar. Using p-values for the comparison of classifiers: pitfalls and alternatives. *Data Mining and Knowledge Discovery*, 36(3):1102–1139, May 2022. ISSN 1573-756X. [p110]

A. Bhowmik and J. Ghosh. LETOR methods for unsupervised rank aggregation. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 1331–1340, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee. ISBN 9781450349130. [p98]

B. Brownfield and J. H. Kalivas. Consensus outlier detection using sum of ranking differences of common and new outlier measures without tuning parameter selections. *Anal Chem*, 89(9): 5087–5094, 2017. Erratum in: Anal Chem. 2017 Sep 5;89(17):9609. [p97]

S. Chatterjee, A. Mukhopadhyay, and M. Bhattacharyya. A weighted rank aggregation approach towards crowd opinion analysis. *Knowledge-Based Systems*, 149:47–60, 2018. [p99]

P. Diaconis. *Group Representations in Probability and Statistics*, volume 11. Institute of Mathematical Statistics, 1988. [p99]

P. Diaconis and R. L. Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(2):262–268, Jan. 1977. ISSN 00359246. [p99, 100, 102]

T. G. Dietterich. Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Neural Computation*, 10(7):1895–1923, Oct. 1998. ISSN 0899-7667. [p100]

C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, page 613–622, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 1581133480. URL https://doi.org/10.1145/371920.372165. [p98]

C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation revisited. Manuscript, 2002. URL http://www.eecs.harvard.edu/~michaelm/CS222/rank2.pdf. [p101]

D. Eddelbuettel. *Seamless R and C++ integration with Rcpp*. Springer, New York, 2013. [p100]

E. Edgington and P. Onghena. *Randomization Tests*. Chapman and Hall/CRC, New York, 4th edition, 2007. [p100]

R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. *SIAM Journal on Discrete Mathematics*, 17(1):134–160, 2003. [p99]

A. Gere, A. Rácz, D. Bajusz, and K. Héberger. Multicriteria decision making for evergreen problems in food science by sum of ranking differences. *Food Chemistry*, 344:128617, 2021. ISSN 0308-8146. [p98]

A. Gere, D. Szakál, and K. Héberger. Multiobject optimization of national football league drafts: Comparison of teams and experts. *Applied Sciences*, 12(13), 2022. [p97, 99]

M. Guay-Paquet and K. Petersen. The generating function for total displacement. *Electronic Journal of Combinatorics*, 21:3–37, 2014. [p100]

M. Haghani, M. C. Bliemer, and D. A. Hensher. The landscape of econometric discrete choice modelling research. *Journal of Choice Modelling*, 40:100303, 2021. [p99]

R. K. S. Hankin. Partial rank data with the hyper2 package: Likelihood functions for generalized bradley-terry models. *The R Journal*, 9:429–439, 2017. ISSN 2073-4859. doi: 10.32614/RJ-2017-061. https://doi.org/10.32614/RJ-2017-061. [p99]

R. K. S. Hankin. Generalized plackett-luce likelihoods. *Journal of Statistical Software*, 109(8):1–17, 2024. doi: 10.18637/jss.v109.i08. URL https://www.jstatsoft.org/index.php/jss/article/view/v109i08. [p99]

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer New York, 2nd edition, 2009. [p103]

K. Héberger. Sum of ranking differences compares methods or models fairly. *TrAC Trends in Analytical Chemistry*, 29(1):101–109, Jan. 2010. ISSN 01659936. [p97]

K. Héberger. Sum of euclidean distance differences and sum of absolute manhattan distance differences: multicriteria decision making tools for small data tables. *Analytica Chimica Acta*, 1381:344649, 2026. ISSN 0003-2670. [p99]

K. Héberger and B. Škrbić. Ranking and similarity for quantitative structure–retention relationship models in predicting lee retention indices of polycyclic aromatic hydrocarbons. *Analytica Chimica Acta*, 716:92–100, 2012. ISSN 0003-2670. A selection of papers presented at the 12th International Symposium on Extraction Technologies (ExTech 2010). [p99]

M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938. [p98]

M. G. Kendall. *Rank Correlation Methods*. Griffin, London, 1970. [p98, 100]

K. Kollár-Hunek and K. Héberger. Method and model comparison by sum of ranking differences in cases of repeated observations (ties). *Chemometrics and Intelligent Laboratory Systems*, 127: 139–146, 2013. ISSN 0169-7439. [p97, 100]

S. Lin. Rank aggregation methods. *WIREs Computational Statistics*, 2(5):555–570, 2010. [p99]

R. D. Luce. *Individual Choice Behavior: A Theoretical Analysis*. Wiley, New York, 1959. [p99]

C. L. Mallows. NON-NULL RANKING MODELS. I. *Biometrika*, 44(1-2):114–130, June 1957. ISSN 0006-3444. [p99]

N. H. N. Moorthy, S. Kumar, and V. Poongavanam. Classification of carcinogenic and mutagenic properties using machine learning method. *Computational Toxicology*, 3:33–43, 2017. ISSN 2468-1113. [p97]

F. Mosteller. Remarks on the method of paired comparisons: I. the least squares solution assuming equal standard deviations and equal correlations. *Psychometrika*, 16(1):3–9, Mar 1951. [p99]

OEIS Foundation Inc. The total displacement of a permutation. *The On-Line Encyclopedia of Integer Sequences*, 2024. URL https://oeis.org/A062869. Entry A062869. [p100]

S. Petchrompo, D. W. Coit, A. Brintrup, A. Wannakrairot, and A. K. Parlikad. A review of pareto pruning methods for multi-objective optimization. *Computers & Industrial Engineering*, 167: 108022, 2022. ISSN 0360-8352. [p97]

R. L. Plackett. The analysis of permutations. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 24(2):193–202, 1975. ISSN 00359254, 14679876. [p99]

C. Pugmire, S. M. Mundt, V. P. LaBella, and J. Struyf. *Graphics Layout Engine (GLE) 4.3.4 User Manual*, 2022. URL https://glx.sourceforge.io/. Version 4.3.4, accessed on 2024/10/06. [p104]

Z. Qian and P. L. H. Yu. Weighted distance-based models for ranking data using the r package rankdist. *Journal of Statistical Software*, 90(5):1–31, 2019. [p99]

E. G. Rodrigo, J. C. Alfaro, J. A. Aledo, and J. A. Gámez. Label ranking oblique trees. *Knowledge-Based Systems*, 296:111882, 2024. [p100]

A. L. Schuster, N. L. Crossnohere, N. B. Campoamor, I. L. Hollin, and J. F. Bridges. The rise of best-worst scaling for prioritization: A transdisciplinary literature review. *Journal of Choice Modelling*, 50:100466, 2024. [p99]

L. Sipos, A. Gere, J. Popp, and S. Kovács. A novel ranking distance measure combining Cayley and Spearman footrule metrics. *Journal of Chemometrics*, 32(4):e3011, 2018. [p99]

O. Sorensen, W. Leoncio, V. Vitelli, M. Crispino, Q. Liu, C. Mollica, L. Tardella, and A. Stein. *BayesMallows: Bayesian Preference Learning with the Mallows Rank Model*. The Comprehensive R Archive Network (CRAN), 2024. https://CRAN.R-project.org/package=BayesMallows. [p99]

C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904. ISSN 00029556. [p98]

B. R. Sziklai and K. Héberger. Apportionment and districting by Sum of Ranking Differences. *PLOS ONE*, 15(3):e0229209, Mar. 2020. ISSN 1932-6203. [p97]

B. R. Sziklai and B. Lengyel. Finding early adopters of innovation in social networks. *Social Network Analysis and Mining*, 13(1):4, Dec 2022. ISSN 1869-5469. doi: 10.1007/s13278-022-01012-5. URL https://doi.org/10.1007/s13278-022-01012-5. [p114]

B. R. Sziklai and B. Lengyel. Audience selection for maximizing social influence. *Network Science*, 12(1):65–87, 2024. doi: 10.1017/nws.2023.23. [p97]

B. R. Sziklai, P. Biró, and L. Csató. The efficacy of tournament designs. *Computers & Operations Research*, page 105821, 2022. ISSN 0305-0548. [p99]

B. R. Sziklai, M. Baranyi, and K. Héberger. Does cross-validation work in telling rankings apart? *Central European Journal of Operations Research*, 33(4):1503–1528, Dec 2025. doi: 10.1007/s10100-024-00932-1. [p100, 103]

W. Tang. Mallows ranking models: Maximum likelihood estimate and regeneration. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6125–6134. PMLR, 2019. [p99]

L. L. Thurstone. A law of comparative judgment. *Psychological Review*, 34:273–286, 1927. [p99]

H. Turner, I. Kosmidis, D. Firth, and J. van Etten. *PlackettLuce: Plackett-Luce Models for Rankings*. The Comprehensive R Archive Network (CRAN), 2023. https://CRAN.R-project.org/package=PlackettLuce. [p99]

B. Vajna, A. Farkas, H. Pataki, Z. Zsigmond, T. Igricz, and G. Marosi. Testing the performance of pure spectrum resolution from raman hyperspectral images of differently manufactured pharmaceutical tablets. *Analytica Chimica Acta*, 712:45–55, 2012. ISSN 0003-2670. [p97]

R. L. Wasserstein and N. A. Lazar. The asa statement on p-values: Context, process, and purpose. *The American Statistician*, 70(2):129–133, 2016. [p110]

C. West. Statistics for analysts who hate statistics, Part VII: Sum of Ranking Differences (SRD)s. *LCGC North America*, 36:2–6, 2018. [p97]

J. Wątróbski, A. Bączkiewicz, R. Król, and W. Sałabun. Green electricity generation assessment using the codas-comet method. *Ecological Indicators*, 143:109391, 2022. ISSN 1470-160X. [p97]

## Appendix

```
R> cv_Wilc

$new_column_order_based_on_folds
[1] 3 1 4 5 6 2 7

$test_statistics
[1] 19 24 36  2 20 36

$statistical_significance
[1] "n.s."  "n.s."  "(p<0.05*)"  "n.s."  "n.s."  "(p<0.05*)"

$SRD_values_of_different_folds
       Possession% Shots pg   Pass      Dribbles pg Offsides pg RY cards Fouls pg
fold_1   0.2857143 0.3214286 0.3125000   0.5535714   0.6875000 0.7410714 0.9285714
fold_2   0.2232143 0.2946429 0.3214286   0.5446429   0.5982143 0.6964286 0.9107143
fold_3   0.4017857 0.3214286 0.5357143   0.8214286   0.6785714 0.6517857 0.7946429
fold_4   0.3035714 0.3839286 0.3303571   0.5535714   0.5446429 0.7142857 0.9375000
```

```
fold_5   0.2946429 0.3392857 0.3750000   0.6071429   0.7767857 0.6339286 0.9285714
fold_6   0.2767857 0.3214286 0.3839286   0.6696429   0.6428571 0.7857143 0.8839286
fold_7   0.3750000 0.3571429 0.4285714   0.6607143   0.5982143 0.6428571 0.8571429
fold_8   0.3660714 0.4196429 0.4464286   0.7678571   0.6875000 0.7142857 0.8214286

$boxplot_values
       Possession% Shots pg   Pass      Dribbles pg Offsides pg RY cards Fouls pg
min         0.2232   0.2946 0.3125      0.5446      0.5446   0.6339   0.7946
xx1         0.2232   0.2946 0.3125      0.5446      0.5446   0.6339   0.7946
q1          0.2768   0.3214 0.3214      0.5536      0.5982   0.6429   0.8214
median      0.2946   0.3214 0.3750      0.6071      0.6429   0.6964   0.8839
q3          0.3661   0.3571 0.4286      0.6696      0.6875   0.7143   0.9286
xx19        0.4018   0.4196 0.5357      0.8214      0.7768   0.7857   0.9375
max         0.4018   0.4196 0.5357      0.8214      0.7768   0.7857   0.9375
```

*Balázs R. Sziklai*
*ELTE Centre for Economic and Regional Studies*
*1097 Tóth Kálmán u. 4*
*Hungary*
*Corvinus University of Budapest*
*1093 Fővám tér 8.*
*Hungary*
sziklai.balazs@krtk.elte.hu

*Attila Gere*
*Hungarian University of Agriculture and Life Sciences*
*1118 Villányi út 29-43.*
*Hungary*
Gere.Attila@uni-mate.hu

*Károly Héberger*
*HUN-REN Research Centre for Natural Sciences*
*1117 Magyar tudósok körútja 2.*
*Hungary*
heberger.karoly@ttk.hu

*Jochen Staudacher*
*Fakultät Informatik, Hochschule Kempten*
*87435, Bahnhofstraße 61*
*Germany*
jochen.staudacher@hs-kempten.de

# echos: An R Package for Automatic Time Series Forecasting using Echo State Networks

*by Alexander Häußer*

**Abstract** The paper introduces echos, an R package for automatic univariate time series modeling and forecasting using Echo State Networks (ESNs). ESNs represent an efficient and flexible method that combines reservoir computing for dimensionality expansion with a simple linear model estimated via ridge regression. The package provides two complementary interfaces: a base R interface for modeling numeric vectors, integrating seamlessly into traditional R workflows; and a tidy interface, built upon the tsibble and fable frameworks, for streamlined forecasting, evaluation, and visualization in tidyverse-oriented workflows. echos implements lightweight and fully automated procedures, enabling accurate forecasting without extensive hyperparameter tuning or manual configuration. The package's capabilities are illustrated using synthetic and real-world data, accompanied by two case studies. The first case study demonstrates the base R interface, and the second showcases the tidy interface. Both case studies include reproducible coding examples, ensuring users can readily apply ESNs to their forecasting tasks.

## 1 Introduction

Time series forecasting is fundamental in domains such as finance, economics, and energy, where accurate forecasts support decision-making and resource planning. Over the last few decades, various methods for time series forecasting have been developed, ranging from statistical approaches like autoregressive integrated moving average (ARIMA) and exponential smoothing state space models (ETS) to more advanced techniques like machine learning algorithms and neural network models. Neural networks offer a flexible, data-driven approach for modeling linear and nonlinear relationships in time series data. Among the various neural network architectures, recurrent neural networks (RNNs) are particularly well-suited for sequential data due to their ability to process temporal dependencies in the hidden state, making them popular for time series applications.

An interesting variant of RNNs is the echo state network (ESN) (Jaeger, 2001), a type of reservoir computing (RC) model. ESNs expand the input signal into a high-dimensional feature space through a fixed, nonlinear reservoir and then use a comparatively simple readout mechanism, often a linear model estimated via ridge regression, to produce forecasts (Lukoševičius and Jaeger, 2009; Lukoševičius, 2012). By decoupling most of the network parameters from backpropagation, ESNs greatly reduce the computational overhead that characterizes many neural network approaches. Moreover, they simplify model building and hyperparameter tuning, making them appealing for large-scale, fully automatic forecasting tasks.

Despite the growing interest in RC and ESNs, there are relatively few mature and well-maintained software implementations available, particularly within R. **ReservoirPy** (Trouvain et al., 2020) is a Python library designed for efficient and flexible implementation of RC architectures, particularly ESNs. It supports both offline and online training, parallel computation, sparse matrix operations, and fast spectral initialization, and includes advanced features such as deep and feedback reservoirs, as well as graphical tools for hyperparameter optimization. **reservoirnet** is an R package that serves as a wrapper for **ReservoirPy**, enabling R users to access the full functionality of the Python library through the **reticulate** (Ushey et al., 2025) interface. It inherits all advanced features of **ReservoirPy**, including support for complex reservoir architectures, efficient computation, and hyperparameter exploration, while providing integration with R workflows.

In contrast, the R package **echos** (Häußer, 2026b) is a lightweight implementation focused on fast, fully automatic modeling and forecasting of univariate time series using ESNs. Its emphasis is on ease of use and automation: **echos** requires minimal manual configuration, offers seamless integration with R's time series ecosystem, and is optimized for practitioners seeking rapid and reliable forecasts rather than customizable RC research. Source code and the development version of **echos** are available in the GitHub repository.

In the R ecosystem for time series forecasting, there is a long-standing tradition of using the **forecast** package (Hyndman et al., 2024), which provides methods for statistical models (e.g., ARIMA, ETS) as well as some neural network approaches. In recent years, an alternative approach has emerged based on the so-called tidy philosophy: the **fable** package (O'Hara-Wild et al., 2024a), along with its supporting packages **tsibble** (Wang et al., 2020) and **fabletools** (O'Hara-Wild et al., 2024b). Unlike the

`ts` class, which underpins much of the traditional time series functionality in R, **tsibble** provides a data structure for temporal data that embraces tidy data principles, facilitating both user-friendliness and scalability. A `tsibble` is a special type of a `tibble` (or `data.frame`), specifically designed for time series data. It offers powerful features for time-based indexing, handling irregular time series, and performing group-wise forecasts in a straightforward manner. The **fable** package builds on **tsibble** to streamline forecasting workflows, model specification, and forecast reconciliation. This modern tidy approach has proven especially useful for handling datasets with multiple time series, enabling more user-friendly analysis compared to the traditional workflow.

The package **echos** offers two complementary interfaces:

1. **Base R interface:** The base approach accepts a `numeric` vector as input to integrate seamlessly into existing workflows relying on traditional data structures and types. A drawback of the base approach is that modeling multiple time series requires iterating over them explicitly (e.g., `for`-loops or functions from the `apply()` or `map()` family).
2. **Tidy interface:** The tidy approach connects directly to the **fable** framework based on **tsibble**, leveraging **fabletools**. The tidy approach enables streamlined modification, analysis, modeling, and forecasting of multiple series, even if they have different lengths (i.e., varying start and end dates), making the overall workflow more straightforward.

Both interfaces enable fully automatic model building and forecasting using ESNs, minimizing the need for extensive hyperparameter tuning. The **echos** package aims to make RC methods more accessible to both researchers and practitioners. Table 1 summarizes the key differences between **echos** and **ReservoirPy** in terms of scope, interface philosophy, and supported forecasting and evaluation functionality.

**Table 1:** Comparison between the R package **echos** and the Python library **ReservoirPy**, highlighting differences in scope, interface philosophy, tuning workflows, forecasting features, uncertainty quantification, and distribution.

| Dimension | echos | ReservoirPy |
|---|---|---|
| Primary scope | End-to-end ESN time series forecasting | General RC/ESN framework |
| Interface philosophy | High-level interface via base R and **fable**/**tsibble** ecosystem | Modular nodes and models via the **NumPy**/**SciPy** ecosystem |
| Model selection and tuning | Automated training, tuning, and time series cross-validation | Hyperparameter tools available; tuning and validation are user-assembled |
| Forecasting strategy | Built-in recursive multi-step forecasting | Multi-step forecasting implemented by the user |
| Multivariate and exogenous inputs | Primary focus on univariate forecasting | Multivariate and exogenous inputs supported |
| Uncertainty quantification | Forecast intervals via simulation using a moving block bootstrap of residuals | No dedicated interval interface; uncertainty typically implemented by the user |
| Data preprocessing and pipeline | Data preprocessing, reservoir generation, and estimation are handled automatically | Data preparation is typically explicit; inputs are provided as arrays |
| Diagnostics and evaluation | Built-in evaluation and plotting methods | Evaluation typically performed with external tooling |
| Performance | Uses compiled components via **Rcpp**/**RcppArmadillo** | Relies on **NumPy**/**SciPy**, supports sparse matrices and parallel computation |
| License and distribution | GPL-3; distributed on CRAN | MIT; distributed on PyPI |

The remainder of this paper is structured as follows: Section 2 covers the underlying methodology for applying ESNs in time series forecasting. Section 3 provides an overview of the available functions and methods as well as details on the implementation of the algorithm, from data preprocessing and reservoir generation to model estimation and selection. Section 4 shows illustrative examples of synthetic and real-world time series data and demonstrates the pattern recognition and learning capabilities of the proposed approach. Section 5 presents two case studies, one using base R and another using tidy R, to demonstrate the package's capabilities in real-world forecasting tasks. Finally, Section 6 summarizes the main findings.

## 2 Basic model

ESNs are an RC technique, and compared to other neural networks, they are similar in architecture but different in training. This approach leads to a fast, simple, and constructive supervised learning algorithm for neural networks. ESNs make a conceptual and computational separation between the hidden layer (dynamic reservoir as a nonlinear temporal dimensionality expansion) and a recurrence-free (usually linear) readout that produces the desired output from the expansion. This separation is based on the understanding that the input is expanded into a rich enough reservoir state space. These so-called internal states are combined into the desired output. The traditional training methods for RNNs often do not make a conceptual separation between the input, hidden, and output layers, where all the weights within a neural network are trained simultaneously (Lukoševičius and Jaeger, 2009).

The architecture of an ESN is similar to other neural networks: an input layer, a hidden layer, and an output layer. The idea of an ESN is the application of a large random reservoir (e.g., 100 units) as a source of the dynamic behavior of the network, from which the internal states are combined into the required output. Depending on the application, the reservoir size can be substantially larger. Unlike other neural networks, only the weights from the hidden layer to the output layer $\mathbf{W}^{\text{out}}$ are trained, while the input weights $\mathbf{W}^{\text{in}}$ (input-to-hidden connections) and the reservoir weights $\mathbf{W}$ (hidden-to-hidden connections) are randomly initialized and fixed (Lukoševičius, 2012). Figure 1 provides a schematic overview of the basic ESN architecture.

Formally, consider a univariate, (weak-) stationary time series $\{y_t\}_{t=1}^{T}$. An ESN maps an input $u_t$ (e.g., $y_{t-1}$) into a set of internal states, denoted by a vector $\mathbf{x}_t = (x_{1,t}, x_{2,t}, \ldots, x_{N_x,t})^\top$, where $N_x$ is the number of reservoir units. There is a distinction between non-leaky and leaky ESNs. In a non-leaky ESN, the reservoir state is replaced at each time step by the newly computed nonlinear update. In a leaky ESN (leaky-integrator), the state update includes an additional smoothing (integration) step, so the new state becomes a blend of the previous state and the newly computed update. For a standard non-leaky ESN, the reservoir update $\tilde{\mathbf{x}}_t$ is computed via

$$\tilde{\mathbf{x}}_t = \tanh\left(\mathbf{W}^{\text{in}}\, u_t + \mathbf{W}\, \mathbf{x}_{t-1}\right), \quad t = 1, 2, \ldots, T, \tag{1}$$

where $\mathbf{x}_0 = \mathbf{0}$ is the initial state and $\tanh(\cdot)$ is the hyperbolic tangent activation function applied element-wise. Functions other than the hyperbolic tangent can also be used as the activation function. However, it is the most common choice in practice. The hidden-to-hidden (reservoir) connections are encoded in a matrix $\mathbf{W} \in \mathbb{R}^{N_x \times N_x}$, while the input-to-hidden connections are encoded in $\mathbf{W}^{\text{in}} \in \mathbb{R}^{N_x \times N_u}$, where $N_u$ is the dimension of the input (e.g., $N_u = 1$ for a single-input unit $y_{t-1}$).

The input weight matrix $\mathbf{W}^{\text{in}}$ and the reservoir weight matrix $\mathbf{W}$ are generated randomly, i.e., both matrices are populated with random numbers, usually drawn from a uniform distribution. Nevertheless, other distributions are possible, like the normal distribution. Both matrices remain fixed after random initialization. For a leaky integrator ESN, a leakage rate $\alpha \in (0, 1]$ can additionally be introduced:

$$\mathbf{x}_t = (1 - \alpha)\, \mathbf{x}_{t-1} + \alpha \tilde{\mathbf{x}}_t. \tag{2}$$

The hidden state update then becomes a blend of the old internal state and the newly computed nonlinear transformation, thus providing a form of memory, similar to exponential smoothing applied to the internal states. $\alpha = 1$ recovers the non-leaky ESN, where each new reservoir state fully replaces the old one. $0 < \alpha < 1$ yields a weighted average between $\mathbf{x}_{t-1}$ and the $\tanh(\cdot)$ term, effectively slowing down changes in the reservoir states and allowing the ESN to capture longer-term dependencies (Lukoševičius, 2012).

A further hyperparameter within an ESN is the spectral radius of the reservoir weight matrix, i.e., the maximum absolute eigenvalue of this matrix. Rescaling $\mathbf{W}_1 = \max(|\lambda(W_0)|)^{-1}\mathbf{W}_0$ to a matrix with unit spectral radius and scaling $\mathbf{W}_1$ by the ultimate spectral radius $\rho$ gives the final reservoir weight matrix $\mathbf{W} = \rho\mathbf{W}_1$. Scaling the reservoir weight matrix to a spectral radius $\rho$ changes the width of the distribution of its elements. For an ESN to function correctly, it needs to satisfy the so-called echo state property, which ensures that the influence of previous inputs fades over time. This means the current state of the reservoir should be a reflection of recent inputs rather than distant past inputs. The spectral radius helps to ensure this property. If the spectral radius is too large (greater than 1), the states in the reservoir might not fade away but instead could grow without bound, leading to instability. If it is too small, the states might fade too quickly, losing important information. A spectral radius close to 1 means that the network can maintain a memory of past inputs for a longer time, which is often desirable (Lukoševičius, 2012).

The internal states are computed according to equations (1) and (2) and, together with an intercept term, stacked over time into the design matrix $\mathbf{X}$, whose rows are of the form $(1, x_{1,t}, x_{2,t}, ..., x_{N_x,t})$.

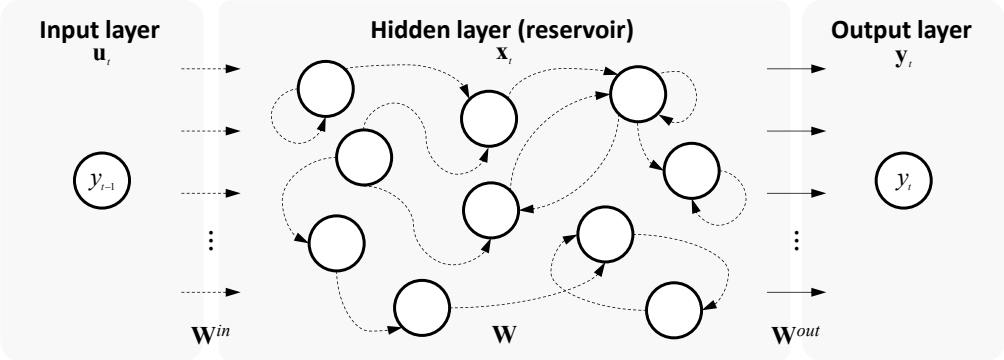



**Figure 1:** Basic ESN architecture with an input layer $\mathbf{u}_t$, a hidden layer (reservoir) $\mathbf{x}_t$ and an output layer $\mathbf{y}_t$. The figure illustrates the univariate first-order autoregressive case, where the single input is the lagged observation $y_{t-1}$ and the single output is $y_t$. Dashed lines indicate fixed, random weights, i.e., input weights $\mathbf{W}^{\text{in}}$ and reservoir weights $\mathbf{W}$. Solid lines indicate trainable output weights $\mathbf{W}^{\text{out}}$.

Hence, the first column of $\mathbf{X}$ is an intercept of ones, and the remaining columns contain the reservoir states over time. Note that the typical initial values $\mathbf{x}_0 = 0$ often introduce an unnatural starting point of the internal states (initial transient). Discarding these initial time steps is advisable until the reservoir has warmed up. The number of time steps to discard from the design matrix depends on the time series and reservoir parameters (Lukoševičius, 2012). For a univariate response time series $\{y_t\}_{t=1}^{T}$ (equivalently, a single-output setting with $N_y = 1$), the linear model can be written as:

$$\mathbf{y} = \mathbf{X}\,\mathbf{W}^{\text{out}} + \boldsymbol{\epsilon}, \tag{3}$$

where $\mathbf{W}^{\text{out}} \in \mathbb{R}^{(N_x+1)\times 1}$ are the readout weights to be estimated and $\boldsymbol{\epsilon}$ is an error term with $\mathbb{E}[\boldsymbol{\epsilon}] = \mathbf{0}$. A common issue for ESNs is overfitting, which occurs when a model is built that captures the signal and noise in the data. The task is to create a stable model that generalizes well while forecasting. Hence, mechanisms are required to mitigate overfitting and reduce prediction errors. In the original literature (Jaeger, 2001, 2002), ridge regression is used to train a linear model. With many highly correlated features, linear models estimated via ordinary least squares (OLS) can suffer from multicollinearity, leading to unstable estimates, overfitting, and poor forecasts. Ridge regression can reduce or eliminate these effects. Alternative approaches for model estimation in the existing literature are LASSO (least absolute shrinkage and selection operator) and Elastic-Net (Ceperic and Baric, 2014; Qiao et al., 2019; Xu and Han, 2016).

## 3 Implementation

ESNs in the **echos** package can be trained and used for forecasting through two complementary sets of functions: a base R interface and a tidy interface built on top of the **fable** framework and the **tsibble** data structure. By separating these interfaces, the package provides both flexibility and usability as standalone functions and a streamlined pipeline that leverages tidy data principles (Wickham et al., 2019). Table 2 gives an overview of the functions, methods, and datasets available.

**Table 2:** Overview of the functions and methods for model training, forecasting, evaluation, and visualization as well as the built-in datasets.

| Function | Description |
|---|---|
| **Base functions** | |
| `train_esn()` | Train an ESN on a univariate time series |
| `forecast_esn()` | Forecast a trained ESN |
| `tune_esn()` | Tune hyperparameters of an ESN |
| `print.esn()` | Print model specification |
| `summary.esn()` | Provide a detailed summary |
| `summary.tune_esn()` | Provide a summary of the hyperparameter tuning |
| `is.esn()` | Checks if the object is of class `esn` |
| `is.forecast_esn()` | Checks if the object is of class `forecast_esn` |
| `is.tune_esn()` | Checks if the object is of class `tune_esn` |

| Function | Description |
|---|---|
| `plot.esn()` | Plot internal states of a trained ESN model |
| `plot.forecast_esn()` | Plot `point` and `interval` forecasts, `actual` and `fitted` values |
| `plot.tune_esn()` | Plot forecasts from a tuned ESN object |
| `run_reservoir()` | Create the reservoir, i.e., the internal states |
| **Tidy functions** | |
| `ESN()` | Train an ESN on tidy time series data |
| `forecast.ESN()` | Forecast a trained ESN |
| `fitted.ESN()` | Extract fitted values from a trained ESN |
| `residuals.ESN()` | Extract residuals from a trained ESN |
| `model_sum.ESN()` | Print model specification |
| `tidy.ESN()` | Extract estimated coefficients |
| `glance.ESN()` | Summary statistics during random search |
| `report.ESN()` | Provide a detailed summary |
| `filter_esn()` | Filter ESN models from `mdl_df` (mable) |
| `reservoir()` | Return the reservoir from a trained ESN as a `tibble` |
| **Datasets** | |
| `m4_monthly_subset` | `tsibble` with six monthly time series |
| `synthetic_data` | `tibble` with ten synthetic time series |

### 3.1 Core functions

The core function `train_esn()` performs end-to-end training of an ESN model. It takes a `numeric` vector representing a univariate time series `y` as input and internally manages data preprocessing, reservoir generation, and model estimation and selection. Arguments with a default value of `NULL` are handled internally by heuristics or tuned automatically.

The default hyperparameter settings in `train_esn()` were chosen to reflect commonly recommended values and practical ranges reported in the ESN literature. In particular, the defaults follow established guidelines on leakage, spectral radius scaling for stable reservoir dynamics, and typical choices for reservoir size and sparsity, providing a robust starting point for automatic forecasting. Overall, these defaults are intended to be conservative and broadly applicable, while remaining consistent with ranges that have been found effective in prior studies (Viehweg et al., 2023; Lukoševičius, 2012).

```
train_esn(
  y,
  lags = 1,
  inf_crit = "bic",
  n_diff = NULL,
  n_states = NULL,
  n_models = NULL,
  n_initial = NULL,
  n_seed = 42,
  alpha = 1,
  rho = 1,
  tau = 0.4,
  density = 0.5,
  lambda = c(1e-04, 2),
  scale_win = 0.5,
  scale_wres = 0.5,
  scale_inputs = c(-0.5, 0.5)
  )
```

The input data `y` must be preprocessed before training and forecasting, as is common with many neural networks. Most time series exhibit nonstationarity due to trends, seasonality, or heteroscedasticity, with trends being especially problematic for ESNs due to the activation function and the resulting reservoir dynamics. Therefore, the Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test (Kwiatkowski et al., 1992) is employed to assess the stationarity of the output variable $y_t$. If the series is nonstationary, first differences ($z_t = y_t - y_{t-1}$) are calculated to stabilize the mean of the time series by removing changes in the level, thus reducing or eliminating trends (Hyndman and Athanasopoulos, 2021). In **echos**, differencing is handled as a lightweight preprocessing step aimed at reducing nonseasonal

nonstationarity prior to reservoir generation. Concretely, the number of differences `n_diff` is determined by applying a KPSS test for level stationarity and, if the null is rejected, applying a single first difference to mitigate level/trend effects that can be problematic for ESN reservoir dynamics. This procedure is intentionally conservative and does not attempt automatic seasonality detection or seasonal differencing (e.g., via `forecast::nsdiffs()`). For strongly seasonal series, seasonal adjustment or seasonal differencing is therefore recommended as an upstream preprocessing step using established tooling in the **forecast** or **fable** ecosystem before calling `train_esn()` or `ESN()`.

After detrending, the output variable is scaled to the interval $[a, b]$ by

$$z'_t = a + \frac{(z_t - \min(z_t))(b-a)}{\max(z_t) - \min(z_t)}, \tag{4}$$

where $a$ is the lower limit and $b$ is the upper limit for the new scale of the time series. Empirical research indicates good results when scaling inputs to the symmetric interval $[-0.5, 0.5]$. Other scaling methods, such as min-max or standardization (z-score normalization) could also be used, but are not implemented. For simplicity, $y_t$ (or matrix notation $\mathbf{y}$) is used in the following, assuming input data is already preprocessed. After training and forecasting, fitted values and forecasts are inversely differenced and rescaled back to the original scale.

Key arguments of `train_esn()` are defined as follows: Differencing (`n_diff`) specifies the number of differences to remove nonstationarity, automatically determined via KPSS if set to `NULL`. After differencing, input scaling (`scale_inputs`) stabilizes the input data in the interval $[-0.5, 0.5]$. Embedding (`lags`) defaults to 1, meaning the first autoregressive lag is used as input ($u_t = y_{t-1}$).

The reservoir generation phase involves several hyperparameters and the internal states are calculated according to equations (1) and (2). By default, the number of internal states `n_states` is determined by the following simple heuristic: $N_x = \min(\lfloor \tau T \rfloor, 200)$, where $T$ is the length of the time series and $\tau$ is the so-called reservoir scaling parameter to dynamically control the reservoir size. The default is $\tau = 0.4$, i.e., the number of internal states is set to 40% of the number of observations $T$ of the time series. To ensure that the output is an integer value, the floor function $\lfloor \cdot \rfloor$ is applied, rounding the output down to the nearest integer. Additionally, the value is capped at 200 to limit the reservoir size for long time series. This heuristic is an implementation choice intended to provide a simple, data-adaptive default that balances model flexibility and runtime; it should be understood as a reasonable starting point rather than a theoretically optimal rule, and users can override `n_states` (or tune `tau` if required).

The input weight matrix $\mathbf{W}^{\text{in}}$ is a dense matrix populated with random numbers (i.e., fully connected), drawn from a uniform distribution, where the interval is defined by `scale_win`, defaulting to $[-0.5, 0.5]$. The (initial) reservoir weight matrix $\mathbf{W}$ is a (structurally) sparse matrix with random numbers, drawn from a uniform distribution, where the interval is defined by `scale_wres`. By default, the reservoir weight matrix has a `density` of 50%, i.e., half of the matrix elements are (randomly) padded with random numbers, while the other half is filled with zeros. This approach ensures that the reservoir remains well-connected while avoiding a fully dense weight matrix. The spectral radius (`rho`) defaults to 1, balancing retaining memory of past inputs and ensuring the stability of the network. Leaky integration (`alpha`) defaults to 1 (non-leaky), but values between 0 and 1 can blend past and current information to enhance memory. Finally, a warm-up period (`n_initial`) discards the initial 5% of internal states by default to reduce transient effects.

For model estimation and selection, ridge regression is employed to mitigate overfitting due to the typically large reservoir and correlated internal states. The ridge regression estimator for the linear model in equation (3) has a closed-form solution:

$$\widehat{\mathbf{W}}^{\text{out}} = \left(\mathbf{X}^\top \mathbf{X} + \mathbf{R}_\lambda\right)^{-1} \mathbf{X}^\top \mathbf{y}, \tag{5}$$

where $\mathbf{R}_\lambda = diag(0, \lambda, ..., \lambda)$ is the regularization matrix. The 0 in the diagonal matrix ensures that the intercept term is untouched by the regularization, while the other $N_x$ coefficients are penalized with a common penalty $\lambda$. The regularization matrix is added along the diagonal of $\mathbf{X}^\top \mathbf{X}$ before inversion. If the predictor variables in the design matrix are highly correlated with each other, which is almost always the case within an ESN if the reservoir is large enough, then the matrix $\mathbf{X}$ does not have full rank and $\mathbf{X}^\top \mathbf{X}$ is not invertible. There is no unique solution to the linear regression problem. Adding the small term $\lambda$ along the diagonal of $\mathbf{X}^\top \mathbf{X}$ makes all columns linearly independent; hence the matrix $(\mathbf{X}^\top \mathbf{X} + \mathbf{R}_\lambda)$ is invertible.

Ridge regression navigates the bias-variance trade-off by penalizing large coefficients, balancing goodness-of-fit and complexity. Ridge regression penalizes the coefficients such that less influential internal states are more strongly shrunk toward zero. For $\lambda = 0$, the estimator in (5) is equal to OLS, and for $\lambda \to \infty$, all coefficients are zero (except the coefficient for the intercept term). For $\lambda$ in

between, the two ideas are balanced: goodness-of-fit and shrinking coefficients towards zero, thereby introducing bias but reducing the estimate's variance (Hastie et al., 2009).

Now that the estimation method is defined, a disciplined way to determine the regularization parameter $\lambda$ is required. A simple but effective approach is to search different values for the regularization parameter and choose the value that results in a model that achieves the best performance on a given time series. The best performance here implies a model with high goodness-of-fit but at the same time low model complexity to avoid overfitting (parsimonious model).

A popular strategy is to choose the regularization parameter $\lambda$ by information criteria (Burnham and Anderson, 2002; Hastie et al., 2009). Information criteria measure the balance between model fit and model complexity. By default, `train_esn()` uses the Bayesian Information Criterion (BIC) (Schwarz, 1978), but other information criteria like the Akaike Information Criterion (AIC) (Akaike, 1974) or the Hannan-Quinn Criterion (HQC) (Hannan and Quinn, 1979) are also implemented. Information criteria measure model fit by the maximum value of the log-likelihood function, while the number of parameters measures model complexity. The number of model parameters in OLS regression corresponds to the number of predictor variables in the model or, equivalently, to the degrees of freedom consumed by the model, which is equivalent to the trace of the projection matrix (hat matrix).

For ridge regression, it is natural to define model complexity analogously by the effective degree of freedom (or effective number of parameters). The trace of the hat matrix $\mathbf{H}_\lambda$ gives the effective degrees of freedom of the ridge regression:

$$df_\lambda = tr\left(\mathbf{H}_\lambda\right) = tr\left(\mathbf{X}(\mathbf{X}^\top\mathbf{X} + \mathbf{R}_\lambda)^{-1}\mathbf{X}^\top\right). \tag{6}$$

The effective degrees of freedom is a monotone decreasing function of $\lambda$. Usually, in a linear regression fit with $N_x$ variables, the degrees of freedom of the fit is $N_x$, the number of free parameters. The idea is that although all $N_x$ coefficients in a ridge fit will be non-zero, they are fit in a restricted fashion controlled by $\lambda$. Note that $df_\lambda = N_x + 1$ when $\lambda = 0$ (no regularization) and $df_\lambda = 1$ as $\lambda \to \infty$, i.e., all coefficients are shrunk towards zero except for the intercept term (Hastie et al., 2009). For the ESN readout in **echos**, $\mathbf{X}$ contains $N_x$ reservoir states plus an intercept, so the reported `df` lies in $1 \leq df \leq N_x + 1$.

For example, the BIC can be calculated for a given value of the regularization parameter $\lambda$ as follows:

$$BIC_\lambda = -2L + \ln(T)df_\lambda, \tag{7}$$

where $L$ is the maximum value of the log-likelihood function, $T$ is the number of observations and $df_\lambda$ are the effective degrees of freedom.

Different optimization algorithms may be used to tune the regularization parameter. However, two of the simplest and most common methods in the machine learning literature are grid search and random search (Bergstra and Bengio, 2012). Grid search defines a search space as a grid of hyperparameter values and evaluates every position in the grid. In contrast, random search defines a search space as a bounded domain of hyperparameter values and randomly samples points in that domain. Usually, the values are sampled from a uniform distribution within a lower and an upper limit.

Bergstra and Bengio (2012) show empirically and theoretically that random search is more efficient for hyperparameter optimization than grid search. They demonstrate that random search over the same domain can find models that are as good or better within a small fraction of the computational time. Furthermore, random search has the same practical advantages as grid search, e.g., ease of implementation and trivial parallelism.

The function `train_esn()` uses random search to optimize the regularization parameter $\lambda$. By default, the number of candidate models `n_models` during the random search is defined internally as $K = 2N_x$, i.e., the value for $K$ is double the reservoir size $N_x$ (`n_models = 2*n_states`), because the larger the reservoir size, the larger the search space for the regularization parameter should be. Thus, setting `n_models` proportional to `n_states` provides a simple, budgeted default that increases the random search effort as model capacity grows, while keeping computational cost within a reasonable range; users can increase or decrease `n_models` depending on the trade-off between accuracy and runtime. The values for `lambda` are drawn from a uniform distribution $[10^{-4}, 2]$. The limits were determined through trial and error and yielded favorable results in practice. The candidate models are estimated for all the random $\lambda$ values, and the corresponding information criteria are calculated. Afterward, the models are ranked, and the model with the minimum information criterion is selected.

To ensure reproducibility, randomization is controlled by a fixed seed (`n_seed`). Internally, `train_esn()` relies on the function `run_reservoir()` written in C++ using **Rcpp** (Eddelbuettel et al.,

2024a) and **RcppArmadillo** (Eddelbuettel et al., 2024b), significantly accelerating reservoir state computations compared to pure R implementations, especially for long time series.

After fitting, `train_esn()` returns an object of class `esn`, including fitted values, internal states, model diagnostics, etc. Once an ESN model is trained, `forecast_esn()` can be used to generate forecasts over a user-defined forecast horizon (`n_ahead`).

```
forecast_esn(
 object,
 n_ahead = 18,
 levels = c(80, 95),
 n_sim = 100,
 n_seed = 42
)
```

The implemented approach for generating point forecasts is recursive forecasting: rather than predicting the entire forecast horizon $h$ in a single step, the network generates each successive forecast based on the most recent data, including its own predictions. Specifically, after estimating the one-step ahead forecast $\hat{y}_{T+1|T}$ (with information up to time $T$), it is treated as the new input $u_{T+1} = \hat{y}_{T+1|T}$. The reservoir is updated accordingly to produce $\hat{y}_{T+2|T}$ (two-step ahead forecast with information up to time $T$), and so forth, until the desired forecast horizon $h$ is reached. This scheme effectively transforms the ESN into an autoregressive-like system, where each new forecast depends on previous forecasts once real observations are no longer available. While recursive forecasting can accumulate errors over multiple steps, it remains a straightforward and commonly used forecasting strategy (Taieb and Hyndman, 2014).

To quantify forecast uncertainty, future sample paths are simulated via a moving block bootstrap based on the (in-sample) residuals and the empirical quantiles of the distribution are estimated. A moving block bootstrap is used because resampling contiguous residual blocks preserves short-range serial dependence, unlike an i.i.d. residual bootstrap (Künsch, 1989). The residual series $\epsilon_t$ is centered and resampled into contiguous blocks using a block length $\ell = \lfloor T^{1/3} \rfloor$, where $T$ is the number of residuals. Choosing $\ell$ involves a bias–variance trade-off: larger blocks preserve more serial dependence but reduce the number of effectively independent resampled blocks. The rule $\ell = \lfloor T^{1/3} \rfloor$ is a widely used default, consistent with theoretical results that suggest block lengths that are optimal in terms of mean squared error for block bootstrap procedures (Politis and Romano, 1994; Politis and White, 2004). For each simulated path, blocks are drawn repeatedly with replacement until their total length reaches the forecast horizon $h$. The selected blocks are concatenated in the order they are drawn, and any excess observations beyond the forecast horizon are discarded. This residual-based bootstrap assumes that the centered residual series is approximately weakly stationary. If residual diagnostics indicate remaining structure (e.g., strong autocorrelation or conditional heteroscedasticity), the resulting intervals may be miscalibrated and should be interpreted with caution.

Each sequence is added to the corresponding mean prediction inside the recursive loop, so that every shock is fed back into the input layer before the next step is generated. For example, let $\hat{y}_{T+1|T}$ denote the recursive point forecast and let $\epsilon^*_{T+1}$ be a bootstrapped residual. One simulated draw is then $y^*_{T+1} = \hat{y}_{T+1|T} + \epsilon^*_{T+1}$, which is fed back as input to produce $\hat{y}_{T+2|T}$, and so on. The process is repeated until the desired forecast horizon is reached to create one possible future sample path. Repeating this procedure many times creates the entire forecast distribution. This approach preserves the autocorrelation structure while allowing for random innovations (Hyndman and Athanasopoulos, 2021). The resulting matrix of simulated future sample paths is finally rescaled and, if the series was differenced during preprocessing, integrated back to its original level. Prediction intervals are obtained by taking the empirical quantiles of the distribution: for an 80% interval, the 10th and 90th percentiles are reported at each horizon; for a 95% interval, the 2.5th and 97.5th percentiles are reported, and so on. Because no distributional form is assumed, the method adapts automatically to skewness or heteroscedasticity in the forecast errors. The reproducibility is controlled by the random seed argument.

The function `tune_esn()` implements hyperparameter selection based on time series cross-validation. The time series is partitioned into multiple expanding windows, i.e., train/test splits, where the training sample starts at the beginning of the series and grows over time, and each test window has a constant length equal to the forecast horizon (`n_ahead`). For each split, the function performs a grid search over hyperparameters `alpha`, `rho`, and `tau`. Each hyperparameter combination is fit on the current training window and evaluated by producing genuine out-of-sample forecasts for the subsequent test window; performance is quantified via mean squared error (MSE) and mean absolute error (MAE). By repeating this evaluation across several forecast origins (`n_split`) and aggregating results per hyperparameter setting, `tune_esn()` prioritizes configurations that deliver consistent forecast accuracy across time, reducing the risk of selecting parameters that work well only for particular

historical segments.

```
tune_esn(
  y,
  n_ahead = 12,
  n_split = 5,
  alpha = seq(0.1, 1, by = 0.1),
  rho = seq(0.1, 1, by = 0.1),
  tau = c(0.1, 0.2, 0.4),
  min_train = NULL,
  ...
)
```

A dedicated benchmarking study (Häußer, 2026a) investigates how core hyperparameters affect both forecast accuracy and runtime on a representative subset of the M4 data (monthly and quarterly series, up to 20 years of history). The study uses disjoint samples per frequency: a Parameter dataset for tuning (2,400 monthly and 1,200 quarterly time series) and an independent Forecast dataset of the same size for evaluation. On the Parameter set, a grid is evaluated over leakage $\alpha \in \{0.1, \ldots, 1.0\}$, spectral radius $\rho \in \{0.2, 0.3, \ldots, 1.2\}$, reservoir scaling $\tau \in \{0.2, 0.4, 0.6\}$ (with $N_x = \min(\lfloor \tau T \rfloor, 200)$), and the information criterion (AIC, AICc, BIC, HQC) used to select the ridge penalty, totaling 1,320 configurations (4,752,000 fits). The hyperparameter sweep shows consistent frequency-specific preferences: high leakage ($\alpha \approx 0.9 - 1.0$) in both cases, near-edge dynamics for monthly series ($\rho \approx 0.8 - 1.0$, typically $\tau \approx 0.4$), and more contractive reservoirs for quarterly series ($\rho \approx 0.3 - 0.5$, often larger $\tau$). Using the best settings from the hyperparameter sweep, ESN forecasts are benchmarked against common automatic forecasting methods such as ARIMA, ETS, the Theta method, and TBATS (trigonometric, Box-Cox transformation, ARMA errors, trend, and seasonal components), as well as standard baselines such as (seasonal) naive, drift, and mean (Hyndman and Athanasopoulos, 2021). Performance is evaluated using mean absolute scaled error (MASE) and symmetric mean absolute percentage error (sMAPE), and computational runtimes are recorded. For monthly series, the selected ESN (AICc, $\alpha = 1.0$, $\rho = 0.9$, $\tau = 0.4$) achieves mean MASE = 0.898 with 0.34s per series, comparable accuracy to ARIMA (MASE = 0.897; 0.45s) and TBATS (MASE = 0.899; 0.96s). For quarterly series, the selected ESN (AIC, $\alpha = 1.0$, $\rho = 0.4$, $\tau = 0.6$) obtains the best mean MASE = 1.111 with 0.12s per series (ARIMA MASE = 1.139; 0.13s; TBATS MASE = 1.160; 0.51s). Overall, these results indicate that ESNs can deliver competitive accuracy at modest computational cost and that lightweight, frequency-aware tuning of $\alpha$ and $\rho$ (and, optionally, $\tau$) can improve robustness across different time series structures.

Before turning to the remaining functions, a few practical considerations regarding input assumptions and computational resources are summarized. **echos** assumes complete, numeric input without missing values. Accordingly, `train_esn()` (and downstream or wrapper functions like `tune_esn()` or `ESN()`) require that gaps or missing observations are handled prior to model fitting (e.g., via imputation or gap-filling). To improve robustness, `train_esn()` includes an explicit check to ensure that the time series is sufficiently long given the chosen differencing order, lag structure, and initial warm-up period. Constant (or near-constant) time series are supported; in this case scaling becomes degenerate and is handled without affecting feasibility, since the series contains no dynamic variation to be learned.

The dominant memory cost arises from storing the reservoir states (and associated design matrices) over time. In particular, the internal state matrix has dimension $T \times N_x$, where $T$ is the effective sample size after preprocessing and warm-up and $N_x$ is the reservoir size, implying $O(TN_x)$ memory. Additional memory is required for the weight matrices: the reservoir weight matrix has size $N_x \times N_x$ and the input weight matrix has size $N_x \times N_u$ (with $N_u$ the input dimension). Although the reservoir weights may be structurally sparse (many zeros depending on `density`), they are stored as standard matrices in the current implementation, so the memory footprint scales with $O(N_x^2)$ for the reservoir weights.

### 3.2 Utility functions and methods

Additional utility functions such as `print.esn()` and `summary.esn()` allow for a quick inspection of the model specification and a detailed summary of the trained model, respectively. The compact print format `ESN({n_states, alpha, rho}, {n_models, df})` summarizes the reservoir generation stage (reservoir size `n_states`, leakage rate `alpha`, and spectral radius `rho`) and the model selection stage (number of candidate ridge fits `n_models` and effective degrees of freedom `df` of the selected readout). The function `plot.forecast_esn()` produces visualizations of actual values, fitted values, and forecasts. Checks like `is.esn()`, `is.forecast_esn()` and `is.tune_esn()` are also available to verify object classes.

On the tidy side, `ESN()` is a modeling function designed to work seamlessly with **tsibble** and **fable** workflows. Users can specify a time series by converting their data to a `tsibble` (e.g., using `as_tsibble()`) and then call `model("ESN" = ESN(value))` to train an ESN on the column `value`. The resulting model object (class `ESN`) is then compatible with standard **fable** generics like `forecast.ESN()`, `fitted.ESN()`, `residuals.ESN()`, and `glance.ESN()`. These functions facilitate common modeling tasks such as generating forecasts, retrieving fitted values and residuals, and reviewing summary statistics (O'Hara-Wild et al., 2024a,b; Wang et al., 2020).

The tidy functions `ESN()`, `forecast.ESN()`, and `report.ESN()` are essentially wrapper functions around their base counterparts `train_esn()`, `forecast_esn()`, and `summary.esn()`. For example, when using `ESN()` on a `tsibble`, the function calls `train_esn()` behind the scenes and returns an S3 object of class `ESN` that is compatible with the **fable** workflow. Similarly, `forecast.ESN()` internally invokes `forecast_esn()`, returning a **fable**-compliant object that can be used with the suite of downstream functions (e.g., `fabletools::accuracy()` or `fabletools::autoplot()`). `report.ESN()` consolidates the functionalities of `summary.esn()` for a comprehensive summary. This design allows users to pick the interface that best suits their needs. Additionally, `reservoir()` returns the ESN's internal states as a `tibble`, enabling further custom analysis or visualization of the reservoir dynamics.

As a convenience, the package also includes two datasets. The dataset `m4_monthly_subset` is a `tsibble` with six illustrative monthly time series from the M4 Forecasting Competition (Makridakis et al., 2020). `synthetic_data` is a `tibble` with ten synthetic time series. These datasets can be used to quickly experiment with ESNs.

## 4 Illustrative applications

### 4.1 Example 1 - Synthetic data

To demonstrate the pattern recognition and learning capabilities of the proposed ESN approach, the dataset `synthetic_data` is used, comprising ten representative signals: (a) square wave, (b) sawtooth wave, (c) harmonic wave (a combination of sine and cosine components at varying frequencies), (d) harmonic wave with trend (incorporating a varying mean), (e) amplitude-modulated wave (with changing variance), (f) frequency-modulated wave (with changing autocovariance), (g) AR(1) process, (h) MA(2) process, (i) white noise process, and (j) random walk process. These signals exhibit diverse structural properties, including sharp discontinuities, smooth periodic behavior, modulation patterns, and randomness, offering a robust testbed for demonstrating the ESN's learning capabilities.

Figure 2 presents the model's performance in terms of model fit and forecasts. The black line indicates the actual values, the orange line corresponds to the fitted values in the training period, and the blue line shows the forecasts for the test period. The vertical dotted line marks the split between training and testing. Overall, the ESN effectively captures the principal features of the signals.

Cases (a) - (b) show persistent oscillations around the sharp transitions throughout both the training and forecast periods. This behavior looks similar to the Gibbs phenomenon, which typically arises when approximating discontinuous signals using smooth basis functions. This behavior is consistent with the ESN's smooth function approximation: the reservoir applies the smooth $\tanh(\cdot)$ nonlinearity and the linear readout combines these smooth internal states, so sharp discontinuities are hard to represent and overshoot near transitions can occur. In the context of the ESN, this effect likely stems from the model's inherent smoothness and limited ability to sharply transition between discrete levels. While the ESN captures the general periodicity and structure of both signals, the overshooting near discontinuities reflects the intrinsic limitation of reservoir-based approximations and the smooth activation function for non-smooth signals.

In contrast, for the smooth cases (c) - (f), the fitted values and forecasts are very close to the actual data, indicating high model accuracy. The harmonic wave and its variant with the trend are captured accurately, demonstrating the network's ability to adapt to periodic structures as well as changing means due to the automatic detrending. The amplitude-modulated wave also fits well, indicating that the ESN can learn patterns of shifting variance. Of particular interest is the frequency-modulated wave (f), which involves changes in autocovariance and is relatively uncommon outside of certain technical or engineering contexts. Despite its complexity, the ESN handles these modulations robustly.

The stochastic cases (g) - (j) yield results that are reasonable for each class of stochastic process. For the AR(1) and MA(2) processes, the ESN effectively learns the underlying autocorrelation patterns. In the white noise case, the ESN's fitted values and forecasts flatten around the mean of zero, showing small random fluctuations, consistent with the absence of any temporal structure in the data. Finally, the random walk process leads to fitted values that lag slightly behind the actual values, while the forecasts remain almost flat with minor movements toward the mean, aligning with how random walks are difficult or impossible to predict.

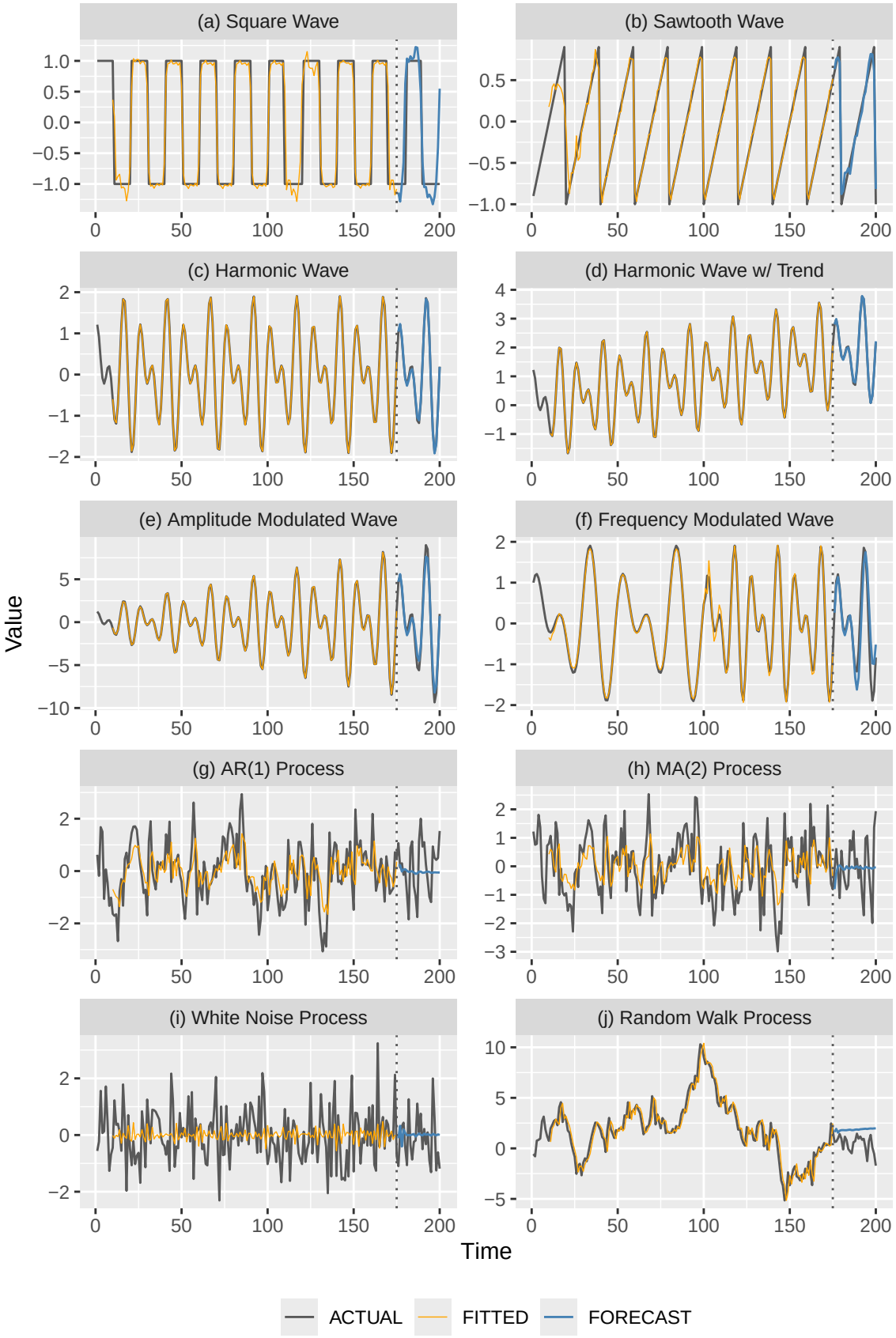



**Figure 2: Synthetic data:** Actual values (black), fitted values (orange), and out-of-sample forecasts (blue) from automatic ESN models (default setting) for the synthetic time series in the dataset `synthetic_data`. The vertical dotted line indicates the split of the actual values into training and testing data.

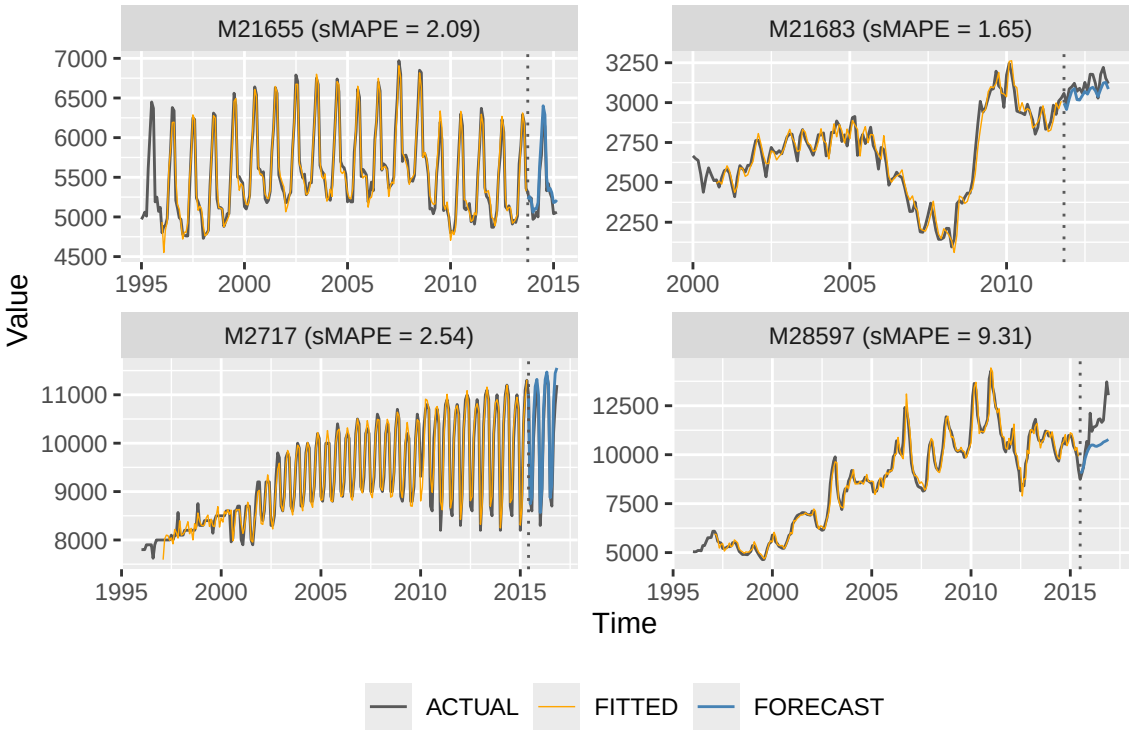



**Figure 3: Real-world data:** Actual values (black), fitted values (orange), and out-of-sample forecasts (blue) from automatic ESN models (default setting) for four selected time series from the dataset `m4_monthly_subset`. The vertical dotted line indicates the split of the actual values into training and testing data.

### 4.2 Example 2 - Real-world data

Figure 3 illustrates the behavior of the default ESN on four selected monthly time series from `m4_monthly_subset` covering different dynamics. The figure shows the actual values (black), fitted values (orange), and out-of-sample forecasts (blue); the vertical dotted line marks the split between the training and test periods. Overall, the default ESN adapts well to both seasonal and nonseasonal structure under the simple feedback-driven setup (single-input $y_{t-1}$) and minimal tuning. For M21655 (sMAPE = 2.09%), pronounced seasonality and longer-run variation are captured reasonably well. M21683 (sMAPE = 1.65%) shows no clearly visible seasonal pattern but exhibits a marked level change in the training sample; despite this structural break, the ESN produces sensible forecasts that track the post-break level. For M2717 (sMAPE = 2.54%), which combines strong seasonality with an upward trend, the ESN reproduces the recurring seasonal component and extrapolates the overall increase into the forecast horizon. Finally, M28597 (sMAPE = 9.31%) is dominated by an upward trend with no obvious seasonality; forecasts are initially close to the observed test values but increasingly underestimate the true series toward the end of the horizon, leading to larger errors.

### 4.3 Example 3 - Varying hyperparameters

Figures 4 and 5 show the reservoir dynamics and the resulting forecasts and fitted values for varying hyperparameters. For both figures, panel (a) shows the default setting with a leakage rate $\alpha = 1$ and a spectral radius $\rho = 1$; panel (b) lowers the leakage to $\alpha = 0.25$; panel (c) lowers the spectral radius to $\rho = 0.25$; and panel (d) increases the spectral radius to $\rho = 2$.

Figure 4 shows the internal states with varying leakage rate and spectral radius. For illustrative purposes, only 10 internal states and the first 100 observations per internal state are shown so that the visualization stays readable. Reducing the leakage rate (panel b) introduces a pronounced smoothing effect to the internal states. Lowering spectral radius (panel c) reduces the dynamics of the reservoir. By contrast, a large spectral radius (panel d) leads to chaotic oscillations between $-1$ and 1.

The figure illustrates how RC achieves nonlinear dimensionality expansion as a feature engineering technique for time series data. Some of the internal states are positive and others are negatively correlated with the output variable and with each other. Furthermore, some lead-lag relationships between the internal states and the output variable are introduced. The model can utilize these

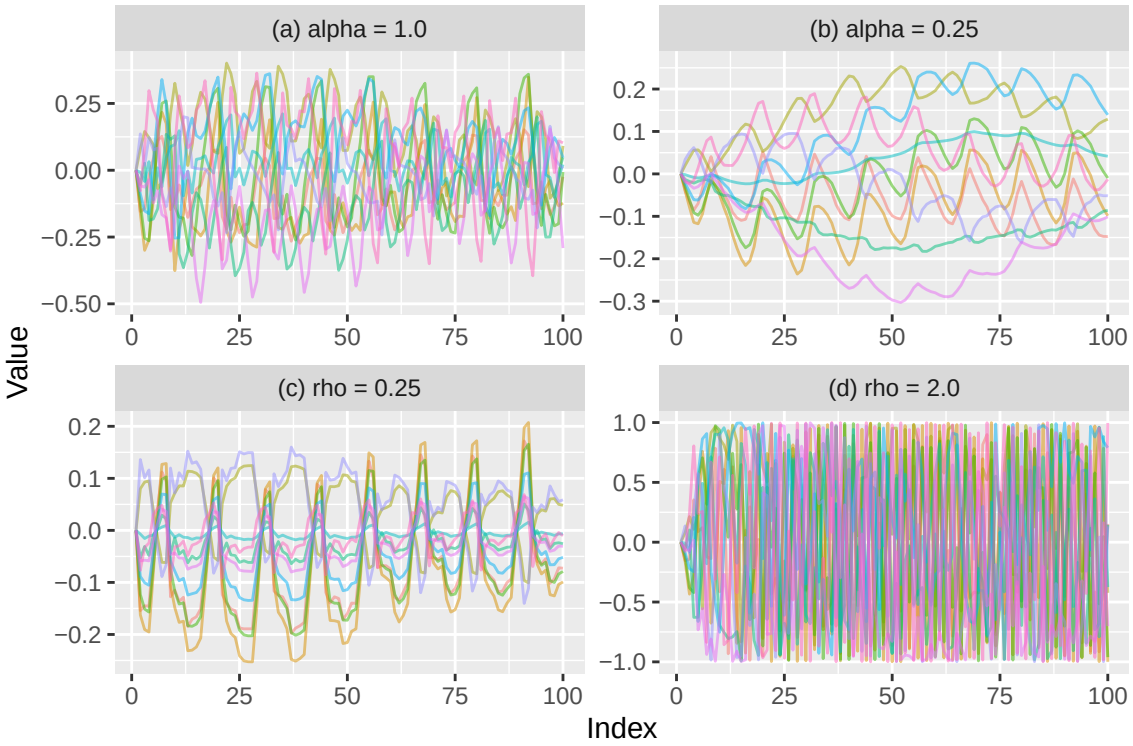



**Figure 4: Reservoir internal states** for different hyperparameters: (a) High leakage rate, (b) low leakage rate, (c) low spectral radius and (d) high spectral radius. Due to illustrative purposes, only 10 internal states and the first 100 observations are shown.

lead-lag relationships to capture autocorrelation, seasonality, and other patterns. The initial transient (warm-up) of the internal states is also visible at the start of the sequence.

Figure 5 shows how the reservoir dynamics translate into fitted values and forecasts. The baseline model (a) yields reasonable forecasts (sMAPE = 2.09%). Reducing the leakage rate (b) introduces stronger smoothing and slightly worsens accuracy (sMAPE = 3.94%). Lowering the spectral radius (c) dampens the reservoir dynamics and produces a suppressed forecast with substantially higher error (sMAPE = 7.12%). Conversely, a large spectral radius (d) induces overly reactive dynamics and an overreactive forecast (sMAPE = 7.01%).

### 4.4 Example 4 - Varying sample size

Figure 6 demonstrates the algorithm for automatic ESN model training for the time series M21655 with different sample sizes of training data and the effect on the corresponding forecast. Panel (a) shows the forecast with one year of training data ($T = 12$ observations), panel (b) shows the forecast with two years of training data ($T = 24$), etc. The results illustrate that with very small training samples the forecasts are necessarily limited (sMAPE = 13.10% for $T = 12$ and sMAPE = 9.40% for $T = 24$), as the seasonal pattern cannot be inferred reliably from only one or two annual cycles. As more history becomes available, forecast accuracy improves and the seasonal structure gradually emerges: sMAPE = 6.38% for $T = 36$, sMAPE = 5.28% for $T = 48$, and so on. A pronounced improvement occurs from about eight years onward, where the seasonal pattern is reflected much more clearly and errors drop substantially (sMAPE = 1.84% for $T = 96$ and sMAPE = 1.82% for $T = 108$; sMAPE = 2.18% for $T = 120$).

The forecasts show that it takes quite a long time with approximately eight years until the seasonal pattern is reflected by the model. Statistical models such as ARIMA or ETS usually require less history to reflect seasonality, at least two years, but it is explicitly required to specify the seasonality within the model. This is different within the presented ESN approach, where the model implicitly learns seasonality from the data if (i) seasonality is present and (ii) enough training data is available to learn from. The example shows the data-driven transition from a nonseasonal to a seasonal model. Furthermore, Figure 6 demonstrates that the proposed algorithm for automatic ESN model training can cope with short time series in the sense that it remains feasible and avoids overly erratic forecasts, while accuracy improves markedly once sufficient history is available.

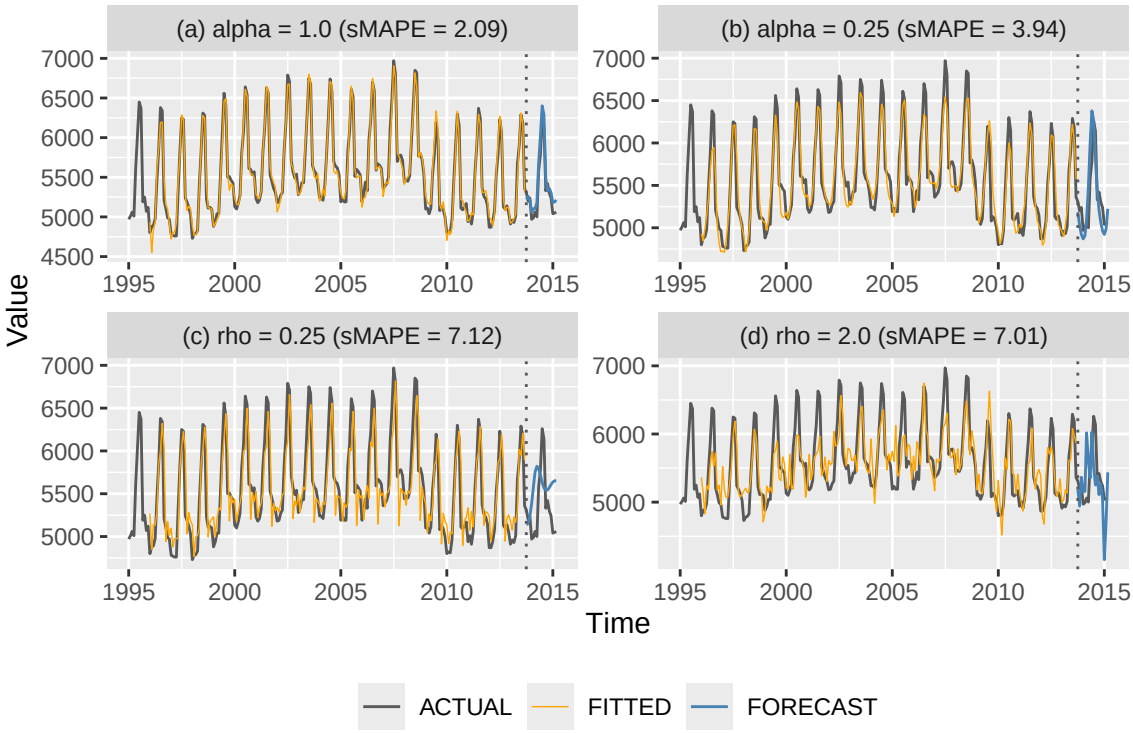



**Figure 5: Forecasts and hyperparameters:** Actual values (black), fitted values (orange), and out-of-sample forecasts (blue) for different hyperparameters: (a) High leakage rate, (b) low leakage rate, (c) low spectral radius and (d) high spectral radius. The vertical dotted line indicates the split of the actual values into training and testing data.

## 5 Case studies

### 5.1 Base functions

This case study models the well-known `AirPassengers` time series (`ts` object) from the package `datasets` (R Core Team, 2024). The dataset contains monthly totals of international airline passengers (in thousands) from January 1949 to December 1960 with 144 observations in total. The first 132 observations are used for model training (`n_train`) and the last 12 observations are used for testing, i.e., the forecast horizon `n_ahead`. `xtrain` and `xtest` are numeric vectors containing the training and testing data.

```
# Forecast horizon
n_ahead <- 12
# Number of observations (total)
n_total <- length(AirPassengers)
# Number of observations (training data)
n_train <- n_total - n_ahead

# Prepare train and test data as numeric vectors
xtrain <- AirPassengers[(1:n_train)]
xtest <- AirPassengers[((n_train+1):n_total)]
```

The function `train_esn()` is used to train three different ESN models on the input data `xtrain`: (a) the default configuration, (b) a more strongly regularized variant to show the effect of increased shrinkage, and (c) a low-leakage variant to demonstrate how leaky integration changes the reservoir dynamics.

For example, the object `model_a` is a list of class `esn` and contains the `actual` and `fitted` values, `residuals`, the internal states `states_train`, estimated coefficients from the ridge regression estimation, hyperparameters, etc. The model specification and a detailed summary can be obtained via the generic S3 methods `print()` and `summary()`.

```
# Train ESN models
```

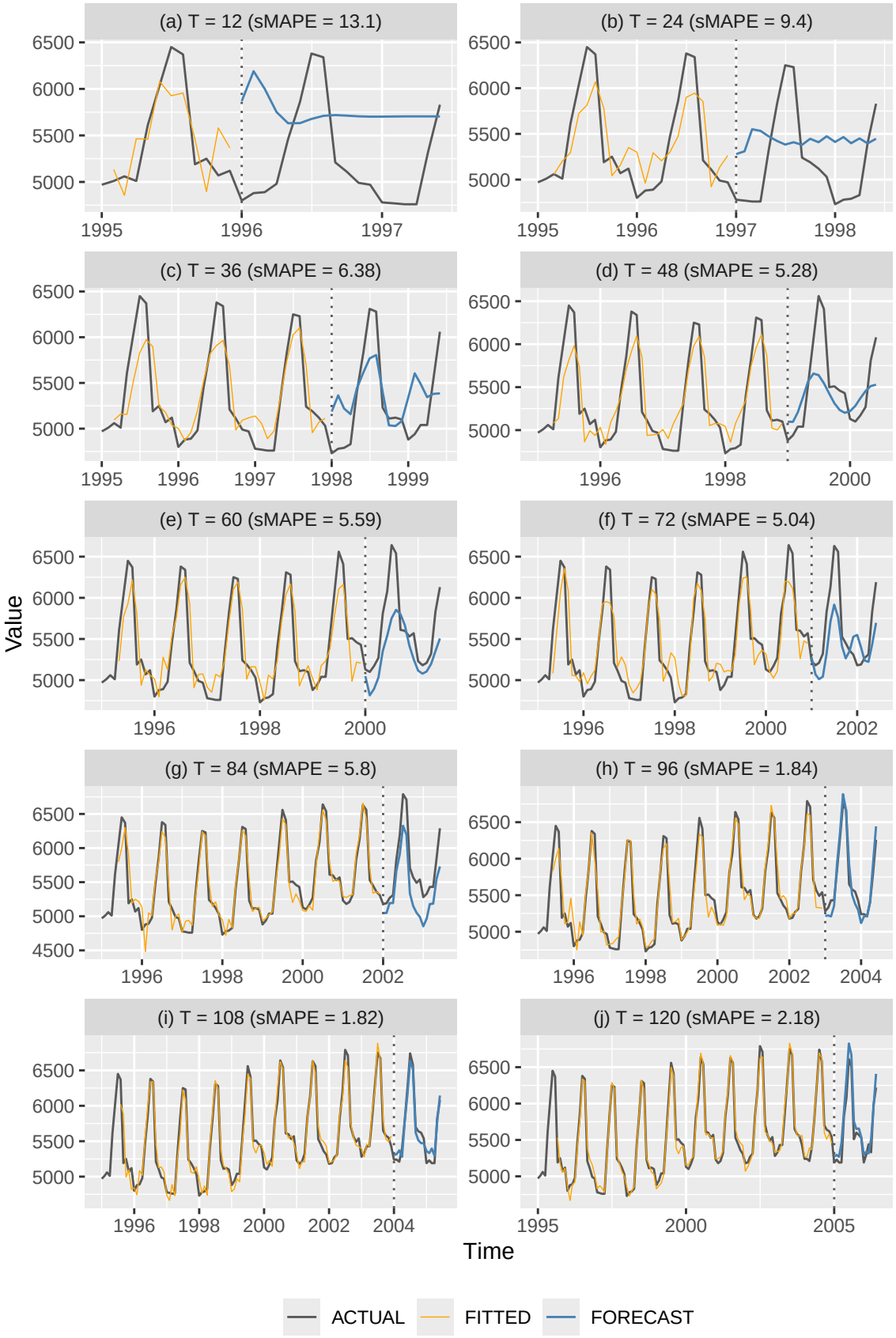



**Figure 6: Varying sample size:** Actual values (black), fitted values (orange), and out-of-sample forecasts (blue) from automatic ESN models (default setting) for the time series M21655 with different sample sizes ($T = 12, 24, ...$) of the training data. The vertical dotted line indicates the split of the actual values into training and testing data.

```
model_a <- train_esn(y = xtrain)                  # (a) Default setting
model_b <- train_esn(y = xtrain, lambda = c(1, 2)) # (b) High regularization
model_c <- train_esn(y = xtrain, alpha = 0.05)     # (c) Low leakage rate

# Print model specification
print(model_a)

#> ESN({52, 1, 1}, {104, 15.53})

# Summarize model
summary(model_a)

#>
#> --- Inputs ------------------------------------------------------
#> n_obs        = 132
#> n_diff       = 1
#> lags         = 1
#>
#> --- Reservoir generation ----------------------------------------
#> n_states     = 52
#> alpha        = 1
#> rho          = 1
#> density      = 0.5
#> scale_inputs = [-0.5, 0.5]
#> scale_win    = [-0.5, 0.5]
#> scale_wres   = [-0.5, 0.5]
#>
#> --- Model selection ---------------------------------------------
#> n_models     = 104
#> df           = 15.53
#> lambda       = 0.036
```

The output of `print(model_a)` provides a compact overview of the fitted model in the format `ESN({n_states, alpha, rho}, {n_models, df})`. For the example, the default model is printed as `ESN({52, 1, 1}, {104, 15.53})`, meaning that the reservoir uses 52 internal states with leakage rate `alpha = 1` and spectral radius `rho = 1`. Model selection is performed by fitting 104 candidate ridge models (random `lambda` draws) and choosing the best one by the selected information criterion; the resulting readout has effective degrees of freedom of 15.53, indicating moderate regularization (i.e., less shrinkage than a strongly penalized fit). When invoking `summary(model_a)`, more detailed output is presented, separated into inputs, reservoir generation, and model selection.

The function `forecast_esn()` is used to forecast the trained models for `n_ahead` steps into the future. The output (e.g., `fcst_a`) is a list of class `forecast_esn`, containing the `point` forecasts, `actual` and `fitted` values, the forecast horizon `n_ahead` and the model specification `model_spec`. The generic S3 method `plot()` is used to visualize the point forecast, the fitted values as well as the holdout test data `xtest` (optional).

Figure 7 shows the forecast from the ESN model (a) with the default setting and (b) with a stronger regularization. By increasing the lower bound of the `lambda` search space from its default 1e-04 to 1, the model applies stronger regularization to its estimated coefficients. In practical terms, this shrinks the degrees of freedom from 15.53 to 9.35, thereby reducing the model's complexity. As a result, forecasts appear more dampened, reflecting the stricter penalization of large coefficients. While this approach often leads to improved robustness, it can also limit the model's ability to capture patterns in the data, such as seasonality.

```
# Forecast ESN models
fcst_a <- forecast_esn(model_a, n_ahead = n_ahead)
fcst_b <- forecast_esn(model_b, n_ahead = n_ahead)

# Extract values
fcst_a$point    # Point forecasts

#>  [1] 420.5592 402.2205 449.4405 438.6973 466.9279 513.2675 580.4146 588.4173
#>  [9] 497.5575 448.8337 405.7796 443.9338

fcst_a$interval # Interval forecasts
```

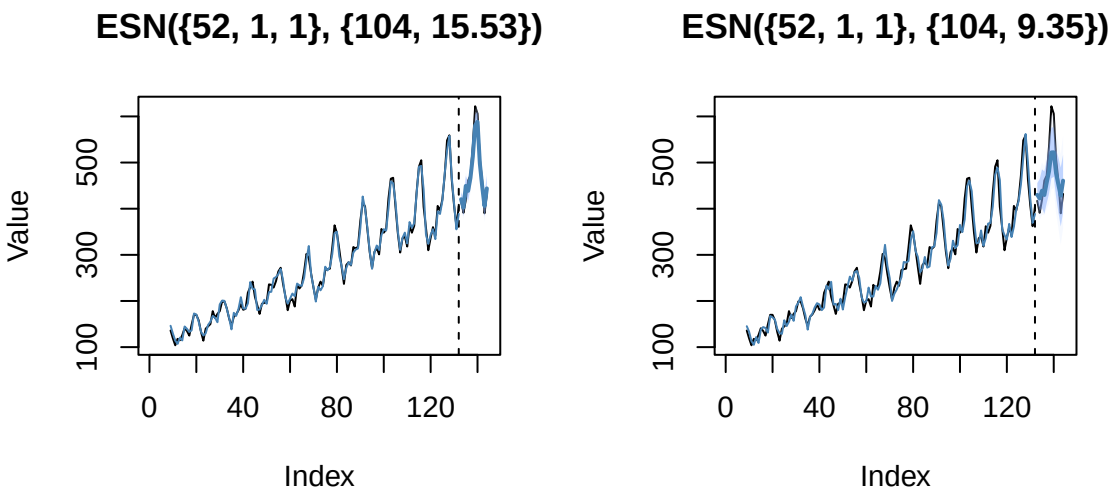



**Figure 7: Forecasts:** Actual values (black) and fitted/out-of-sample forecasts (blue) from model (a) with the default setting (left, sMAPE = 3.51%) and model (b) with stronger regularization (right, sMAPE = 7.75%). The vertical dashed line marks the split between training and test data. Stronger regularization yields a more dampened forecast compared to the default model.

```
#>       lower(80) lower(95) upper(80) upper(95)
#>  [1,]  410.1288  401.0506  435.6200  439.5025
#>  [2,]  383.7836  376.8707  422.2915  427.7195
#>  [3,]  428.0697  420.4539  470.5884  477.6391
#>  [4,]  417.4741  407.8879  459.1989  465.2480
#>  [5,]  444.3180  434.5537  489.6266  495.6535
#>  [6,]  488.8532  479.1714  534.6986  544.8950
#>  [7,]  556.2674  541.6526  603.6298  615.8202
#>  [8,]  568.7352  550.4265  611.3039  620.8753
#>  [9,]  478.2108  460.5468  520.0018  536.9712
#> [10,]  425.2954  415.8190  471.0781  490.6883
#> [11,]  382.5814  371.6403  433.4297  443.4057
#> [12,]  419.3160  406.4784  470.5589  481.1878

# Plot forecasts (1 x 2 plotting matrix)
par(mfrow = c(1, 2), cex = 0.9)
plot(fcst_a, test = xtest)
plot(fcst_b, test = xtest)
```

The generic S3 method `plot.esn()` can be used to visualize the internal states from a trained model. Figure 8 shows the internal states with a high leakage rate $\alpha = 1$ (left panel) and with a low leakage rate $\alpha = 0.05$ (right panel).

```
# Plot reservoirs (1 x 2 plotting matrix)
par(mfrow = c(1, 2), cex = 0.9)
plot(model_a)
plot(model_c)
```

`tune_esn()` performs time series cross-validation using five splits by default (with forecast horizon `n_ahead`, as in the holdout setup above). In this example, `tune_a` evaluates a fixed configuration with a low spectral radius $\rho = 0.3$, while `tune_b` evaluates a small grid over $\rho \in \{0.3, 0.6, 0.9\}$ (with `alpha` and `tau` held fixed) and selects the best-performing setting based on the default accuracy measure (MSE) averaged across splits. The call `summary(tune_b)` reports the cross-validation results for the selected configuration (`alpha = 1`, `rho = 0.9`, `tau = 0.4`) across the splits, including the training and test windows and out-of-sample error measures (MSE and MAE) for each forecast origin. Finally, the generic S3 method `plot.tune_esn()` is used to visualize the results and Figure 9 shows the corresponding forecasts: the setting with the low spectral radius yields an overly damped forecast (left), whereas the tuned configuration aligns much better with the observed dynamics (right).

```
# (a) Fixed hyperparameter (low spectral radius)
tune_a <- tune_esn(
```

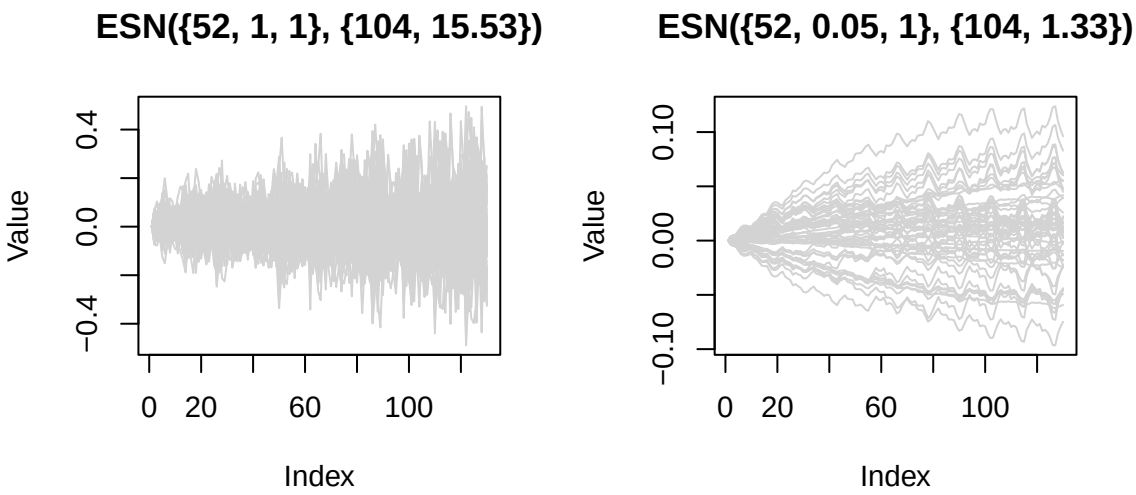



**Figure 8: Reservoir internal states** from model (a) with the default setting (left) and model (c) with a low leakage rate (right). The default setting produces more dynamic, rapidly changing state trajectories, whereas the low-leakage setting yields smoother, more slowly adapting reservoir dynamics.

```
  y = as.numeric(AirPassengers),
  n_ahead = n_ahead,
  alpha = 1.0,
  rho   = 0.3, # Low spectral radius
  tau   = 0.4)

# (b) Hyperparameter tuning
tune_b <- tune_esn(
  y = as.numeric(AirPassengers),
  n_ahead = n_ahead,
  alpha = 1.0,
  rho   = c(0.3, 0.6, 0.9),
  tau   = 0.4)

# Summarize optimal hyperparameters
summary(tune_b)

#> # A tibble: 5 x 11
#>   alpha   rho   tau split train_start train_end test_start test_end   mse   mae
#>   <dbl> <dbl> <dbl> <int>       <int>     <int>      <int>    <int> <dbl> <dbl>
#> 1     1   0.9   0.4     1           1        84         85       96  412.  16.7
#> 2     1   0.9   0.4     2           1        96         97      108  437.  15.0
#> 3     1   0.9   0.4     3           1       108        109      120  631.  22.6
#> 4     1   0.9   0.4     4           1       120        121      132  940.  28.1
#> 5     1   0.9   0.4     5           1       132        133      144  537.  20.1
#> # i 1 more variable: id <int>

# Plot model tuning (1 x 2 plotting matrix)
par(mfrow = c(1, 2), cex = 0.9)
plot(tune_a)
plot(tune_b)
```

### 5.2 Tidy functions

In this example, the dataset `m4_monthly_subset` is used to demonstrate how multiple models can be trained to forecast multiple time series using the **fable** framework. The data form a monthly `tsibble` and are filtered to include only the series `"M21655"` and `"M2717"`. The resulting object `train_frame` contains the training data. Forecasts are generated by separating the last `n_ahead` observations from each time series rather than using the full dataset for model fitting.

```
# Forecast horizon
```

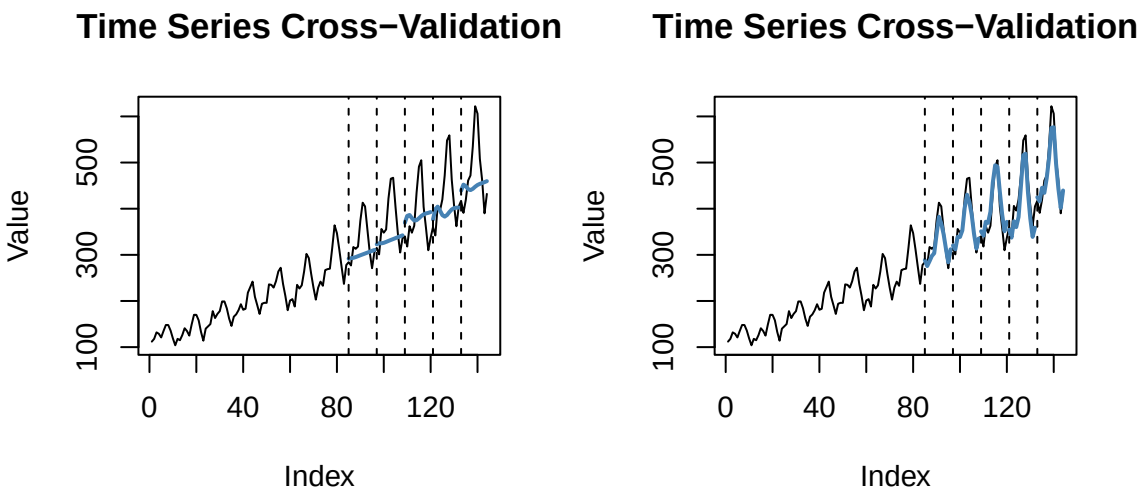



**Figure 9: Hyperparameter tuning:** Time series cross-validation results from `tune_esn()` with out-of-sample forecasts and five splits by default: (a) Evaluation of a single fixed setting with low spectral radius (left) and (b) tuning the spectral radius, where the best configuration is selected automatically based on average MSE across splits (right).

```
n_ahead <- 12

main_frame <- m4_monthly_subset %>%
  filter(series %in% c("M21655", "M2717"))

# Prepare train data
train_frame <- main_frame %>%
  group_by_key() %>%
  filter(row_number() <= n() - n_ahead) %>%
  ungroup()
```

The function `ESN()` is used in combination with `fabletools::model()` to train an ESN for the variable `value`. Changes to the default arguments in `ESN()` are passed to the underlying function `train_esn()`. The trained models are stored as a `mable` (i.e., model table). Additionally, an `ARIMA()` model is trained as a benchmark.

```
# Train ESN and ARIMA model
mable_frame <- train_frame %>%
  model(
    "ESN" = ESN(value),
    "ARIMA" = ARIMA(value)
    )

mable_frame

#> # A mable: 2 x 3
#> # Key:     series [2]
#>   series                              ESN                    ARIMA
#>   <chr>                            <model>                <model>
#> 1 M21655 <ESN({92, 1, 1}, {184, 24.27})> <ARIMA(1,0,1)(2,1,2)[12]>
#> 2 M2717  <ESN({95, 1, 1}, {190, 25.29})> <ARIMA(2,1,4)(0,1,0)[12]>
```

Forecasts are generated via the function `fabletools::forecast()`, where the forecast horizon is set to `h = 12` (i.e., 12-month ahead forecasts). The forecasts are stored as `fable` (i.e., forecast table) and visualized along the historical training data. Figure 10 shows that the ESN models (blue) produce reasonable forecasts and perform comparably to the ARIMA benchmark (orange).

```
# Forecast ESN and ARIMA model
fable_frame <- mable_frame %>%
  forecast(h = n_ahead)
```

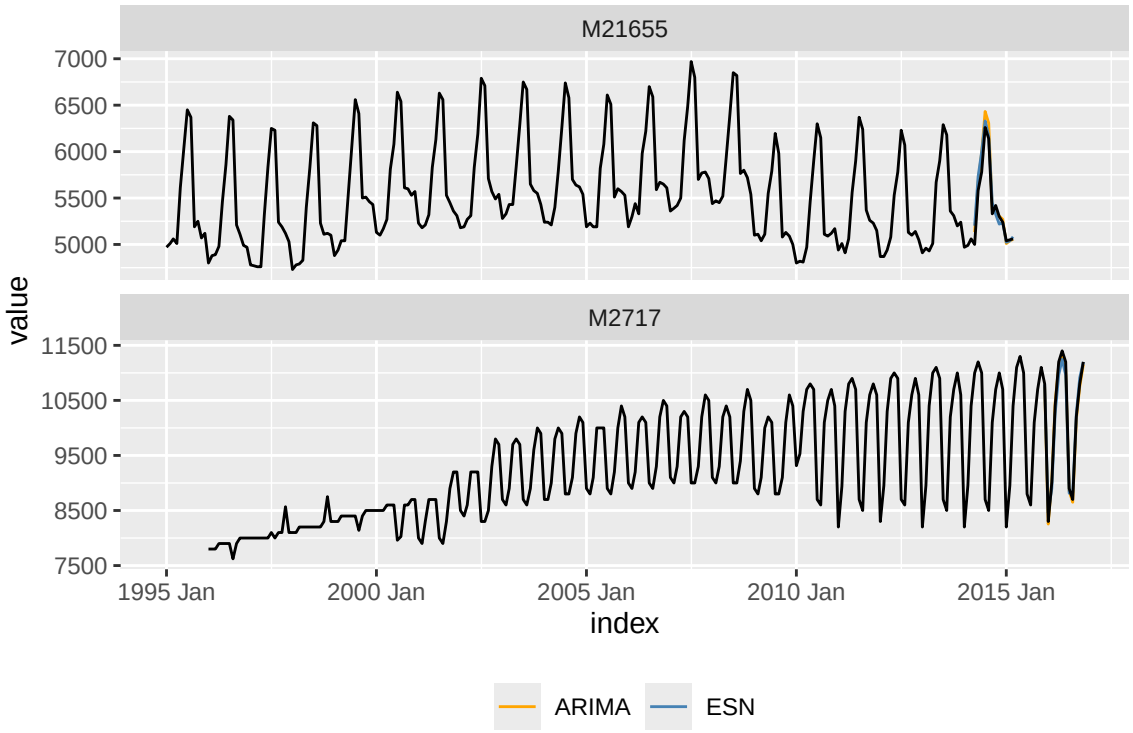



**Figure 10:** Actual values (black) and out-of-sample forecasts from the ESN model (blue) and ARIMA model (orange) for the time series M21655 and M2717.

```
# Plot forecast and test data
fable_frame %>%
  autoplot(main_frame, level = NULL) +
  scale_colour_manual(
    values = c(
      ESN = "steelblue",
      ARIMA = "orange"
      )
    )


# Evaluate forecast accuracy
fable_frame %>%
  accuracy(main_frame)

#> # A tibble: 4 x 11
#>   .model series .type    ME  RMSE   MAE    MPE  MAPE  MASE RMSSE  ACF1
#>   <chr>  <chr>  <chr> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 ARIMA  M21655 Test  -61.7 107.   81.0 -1.06  1.43  0.613 0.579 0.644
#> 2 ARIMA  M2717  Test   42.1  64.4  58.5  0.402 0.570 0.325 0.248 0.286
#> 3 ESN    M21655 Test  -50.5 111.   85.0 -0.910 1.56  0.643 0.600 0.499
#> 4 ESN    M2717  Test   44.3 138.  117.   0.363 1.16  0.651 0.530 0.478
```

The function `feasts::gg_tsresiduals()` (O'Hara-Wild et al., 2024c) is used to visualize the residuals of the ESN model for time series M21655. Note that the function `gg_tsresiduals()` works only for a single model and a single time series, so the `mable_frame` is filtered before the function call. In contrast, the generic S3 function `residuals()` can be used to extract the residuals for both models and both time series as `tsibble` (not shown).

Figure 11 indicates that the residuals look (weak-) stationary with a mean close to zero and constant variance. The histogram suggests an approximately normal distribution. While the fit is reasonably good overall, the (sample) autocorrelation function (ACF) indicates that there is still some autocorrelation present at lags 2 and 9, suggesting that some structure remains unmodeled in the data.

```
mable_frame %>%
```

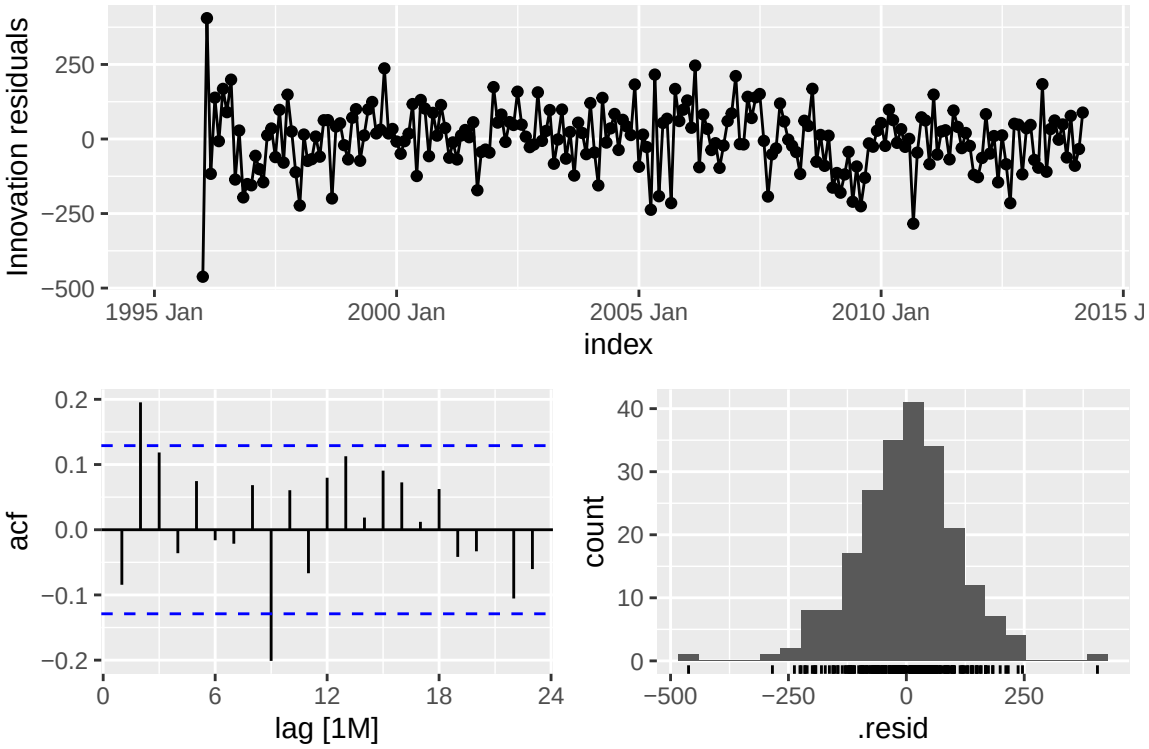



**Figure 11:** Residual diagnostics for the ESN model applied to time series M21655. The top panel displays the residuals over time, the bottom-left panel presents the (sample) autocorrelation function, revealing any remaining correlation structure in the residuals, and the bottom-right panel shows a histogram of their distribution.

```
  select("ESN") %>%
  filter(series == "M21655") %>%
  gg_tsresiduals()
```

The call to `glance()` displays results from the model's random search hyperparameter optimization. Each row corresponds to a unique model configuration, with `model` serving as an identifier. During the random search, multiple values of the regularization parameter `lambda` are drawn from the defined search space, and each candidate ESN is evaluated. The column `df` reports the effective degrees of freedom, reflecting how regularization shrinks the coefficients. Additionally, columns `aic`, `aicc`, `bic` and `hqc` give the corresponding information criteria, while the columns `mse` and `mae` offer in-sample accuracy measures (not shown in the excerpt). By default, the `tibble` is sorted according to `bic`, which is the default information criterion used by `ESN()` and `train_esn()` to select the final model. This sorting places the best-performing model (i.e., that with the smallest BIC value) at the top of the table.

```
mable_frame %>%
  select("ESN") %>%
  filter(series == "M21655") %>%
  glance()
```

```
#> # A tibble: 184 x 13
#>    series .model model loglik  nobs    df lambda   aic  aicc   bic   hqc     mse
#>    <chr>  <chr>  <chr>  <dbl> <int> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
#>  1 M21655 ESN    mode~   355.   219  24.3 0.0212 -662. -656. -580. -629. 0.00228
#>  2 M21655 ESN    mode~   349.   219  23.1 0.0273 -652. -646. -573. -620. 0.00242
#>  3 M21655 ESN    mode~   338.   219  21.3 0.0423 -633. -628. -561. -604. 0.00268
#>  4 M21655 ESN    mode~   321.   219  18.9 0.0808 -604. -600. -540. -578. 0.00313
#>  5 M21655 ESN    mode~   320.   219  18.8 0.0828 -603. -599. -539. -577. 0.00315
#>  6 M21655 ESN    mode~   316.   219  18.3 0.0949 -596. -592. -534. -571. 0.00326
#>  7 M21655 ESN    mode~   316.   219  18.3 0.0967 -595. -591. -533. -570. 0.00328
#>  8 M21655 ESN    mode~   315.   219  18.2 0.0978 -594. -591. -533. -569. 0.00329
#>  9 M21655 ESN    mode~   309.   219  17.6 0.119  -584. -580. -524. -560. 0.00347
```

```
#> 10 M21655 ESN    mode~   309.   219  17.5 0.121  -583. -580. -524. -559. 0.00348
#> # i 174 more rows
#> # i 1 more variable: mae <dbl>
```

Finally, the function `reservoir()` is used to extract the internal states for all models and series from the object `mable_frame` as a `tibble`. The columns `series` and `state` are unique identifiers for the time series itself and the internal state $1, 2, ..., N$. The column `index` represents the time steps and `value` the corresponding measurement. The values shown below are the same as visualized in Figure 4.

```
mable_frame %>%
  reservoir()
```

```
#> # A tibble: 43,675 x 5
#>    series model index state       value
#>    <chr>  <chr> <int> <chr>       <dbl>
#>  1 M21655 ESN       1 state(01)  0
#>  2 M21655 ESN       2 state(01) -0.154
#>  3 M21655 ESN       3 state(01) -0.148
#>  4 M21655 ESN       4 state(01) -0.140
#>  5 M21655 ESN       5 state(01) -0.109
#>  6 M21655 ESN       6 state(01) -0.0244
#>  7 M21655 ESN       7 state(01)  0.0886
#>  8 M21655 ESN       8 state(01)  0.165
#>  9 M21655 ESN       9 state(01) -0.0413
#> 10 M21655 ESN      10 state(01) -0.0557
#> # i 43,665 more rows
```

## 6 Summary

ESNs are a special type of RNN suitable for modeling nonlinear temporal dependencies in time series data. They consist of an input layer, a large, randomly initialized hidden layer (reservoir), and an output layer. Unlike traditional neural networks, ESNs use a fixed reservoir to project inputs into a high-dimensional nonlinear feature space, where only the output weights connecting reservoir states to the output are trained. Typically, ridge regression is used for estimating these output weights, greatly reducing computational complexity. The dynamics of the reservoir depend on key hyperparameters like spectral radius, reservoir size, and leaky integration, controlling stability, memory capacity, and dynamic behavior, making ESNs highly effective for time series modeling and forecasting.

The **echos** package provides a lightweight implementation of ESNs within R, designed to be both user-friendly and computationally efficient. It includes two complementary interfaces: a base R interface operating on numeric vectors, suitable for traditional workflows, and a tidy R interface fully integrated with the modern **tsibble** and **fable** frameworks, facilitating intuitive handling and forecasting of multiple time series simultaneously. The package automates the complete modeling pipeline, from data preprocessing (stationarity tests via KPSS, automatic differencing, and scaling) through reservoir generation (random initialization with controlled spectral radius and sparsity) to the estimation of output weights via ridge regression. Hyperparameters are either fixed (e.g., leakage rate), automatically determined via heuristics (e.g., number of internal states) or optimized through random search (e.g., regularization parameter). For computational efficiency, critical reservoir computations are implemented in C++ through **Rcpp** and **RcppArmadillo**. Illustrative examples on synthetic data (e.g., harmonic waves and stochastic processes) and real-world time series showcase the model's robustness and ability to capture diverse time series patterns.

Two case studies illustrate the practical application of the package. The base R case study utilizes the classic `AirPassengers` dataset to demonstrate straightforward model training, forecasting, and visualization. Adjusting hyperparameters, such as increasing regularization, reveals their impact on forecast smoothness and model complexity. The tidy R case study employs the embedded `m4_monthly_subset` to illustrate how multiple models are easily trained and used for forecasting simultaneously. The tidy integration simplifies comparison with ARIMA benchmarks, residual diagnostics, and exploration of internal reservoir states for further analysis. Overall, the **echos** package makes RC methods accessible through automated workflows, benefiting both practitioners and researchers engaged in time series modeling and forecasting.

## References


H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974. doi: 10.1109/TAC.1974.1100705. [p126]

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012. URL http://jmlr.org/papers/v13/bergstra12a.html. [p126]

K. P. Burnham and D. R. Anderson. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Springer, New York, 2nd edition, 2002. ISBN 978-0387953649. [p126]

V. Ceperic and A. Baric. Reducing complexity of echo state networks with sparse linear regression algorithms. In *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*, pages 26–31, 2014. doi: 10.1109/UKSim.2014.36. [p123]

D. Eddelbuettel, R. Francois, J. J. Allaire, K. Ushey, Q. Kou, N. Russell, I. Ucar, D. Bates, and J. Chambers. *Rcpp: Seamless R and C++ Integration*, 2024a. URL https://CRAN.R-project.org/package=Rcpp. R package version 1.0.13-1. [p126]

D. Eddelbuettel, R. Francois, D. Bates, B. Ni, and C. Sanderson. *RcppArmadillo: 'Rcpp' Integration for the 'Armadillo' Templated Linear Algebra Library*, 2024b. URL https://CRAN.R-project.org/package=RcppArmadillo. R package version 14.2.2-1. [p127]

E. J. Hannan and B. G. Quinn. The determination of the order of an autoregression. *Journal of the Royal Statistical Society. Series B (Methodological)*, 41(2):190–195, 1979. [p126]

T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, New York, 2 edition, 2009. URL http://www-stat.stanford.edu/~tibs/ElemStatLearn/. [p126]

A. Häußer. Echo state networks for time series forecasting: Hyperparameter sweep and benchmarking, 2026a. URL https://arxiv.org/abs/2602.03912. [p128]

A. Häußer. *echos: Echo State Networks for Time Series Modeling and Forecasting*, 2026b. URL https://github.com/ahaeusser/echos. R package version 1.0.4, https://ahaeusser.github.io/echos/. [p120]

R. Hyndman, G. Athanasopoulos, C. Bergmeir, G. Caceres, L. Chhay, M. O'Hara-Wild, F. Petropoulos, S. Razbash, E. Wang, and F. Yasmeen. *forecast: Forecasting functions for time series and linear models*, 2024. URL https://pkg.robjhyndman.com/forecast/. R package version 8.23.0. [p120]

R. J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, Melbourne, Australia, 3rd edition, 2021. URL https://OTexts.com/fpp3. [p124, 127, 128]

H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001. [p120, 123]

H. Jaeger. Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach. 2002. [p123]

H. R. Künsch. The jackknife and the bootstrap for general stationary observations. *The Annals of Statistics*, 17(3):1217–1241, 1989. doi: 10.1214/aos/1176347265. [p127]

D. Kwiatkowski, P. C. Phillips, P. Schmidt, and Y. Shin. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of Econometrics*, 54(1-3):159–178, 1992. [p124]

M. Lukoševičius. A practical guide to applying echo state networks. In *Neural Networks: Tricks of the Trade: Second Edition*, pages 659–686. Springer, 2012. [p120, 122, 123, 124]

M. Lukoševičius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009. [p120, 122]

S. Makridakis, E. Spiliotis, and V. Assimakopoulos. The M4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54–74, 2020. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2019.04.014. URL https://www.sciencedirect.com/science/article/pii/S0169207019301128. M4 Competition. [p129]

M. O'Hara-Wild, R. Hyndman, and E. Wang. *fable: Forecasting Models for Tidy Time Series*, 2024a. URL https://CRAN.R-project.org/package=fable. R package version 0.4.1. [p120, 129]

M. O'Hara-Wild, R. Hyndman, and E. Wang. *fabletools: Core Tools for Packages in the 'fable' Framework*, 2024b. URL https://CRAN.R-project.org/package=fabletools. R package version 0.5.0. [p120, 129]

M. O'Hara-Wild, R. Hyndman, and E. Wang. *feasts: Feature Extraction and Statistics for Time Series*, 2024c. URL http://feasts.tidyverts.org/. R package version 0.4.1, https://github.com/tidyverts/feasts/. [p139]

D. N. Politis and J. P. Romano. The stationary bootstrap. *Journal of the American Statistical Association*, 89(428):1303–1313, 1994. doi: 10.1080/01621459.1994.10476870. [p127]

D. N. Politis and H. White. Automatic block-length selection for the dependent bootstrap. *Econometric Reviews*, 23(1):53–70, 2004. doi: 10.1081/ETC-120028836. [p127]

J. Qiao, L. Wang, and C. Yang. Adaptive lasso echo state network based on modified Bayesian information criterion for nonlinear system modeling. *Neural Computing and Applications*, 31(10): 6163–6177, Oct 2019. ISSN 1433-3058. doi: 10.1007/s00521-018-3420-6. URL https://doi.org/10.1007/s00521-018-3420-6. [p123]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2024. URL https://www.R-project.org/. [p133]

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, pages 461–464, 1978. [p126]

S. B. Taieb and R. J. Hyndman. Recursive and direct multi-step forecasting: the best of both worlds. *International Journal of Forecasting*, 30(3):491–505, 2014. doi: 10.1016/j.ijforecast.2014.01.001. [p127]

N. Trouvain, L. Pedrelli, T. T. Dinh, and X. Hinaut. ReservoirPy: An efficient and user-friendly library to design echo state networks. In *Artificial Neural Networks and Machine Learning – ICANN 2020*, pages 494–505. Springer International Publishing, 2020. doi: 10.1007/978-3-030-61616-8_40. URL https://doi.org/10.1007/978-3-030-61616-8_40. [p120]

K. Ushey, J. Allaire, and Y. Tang. *reticulate: Interface to 'Python'*, 2025. URL https://rstudio.github.io/reticulate/. R package version 1.42.0, https://github.com/rstudio/reticulate. [p120]

J. Viehweg, K. Worthmann, and P. Mäder. Parameterizing echo state networks for multi-step time series prediction. *Neurocomputing*, 522:214–228, 2023. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2022.11.044. URL https://www.sciencedirect.com/science/article/pii/S0925231222014291. [p124]

E. Wang, D. Cook, and R. J. Hyndman. A new tidy data structure to support exploration and modeling of temporal data. *Journal of Computational and Graphical Statistics*, 29(3):466–478, 2020. doi: 10.1080/10618600.2019.1695624. URL https://doi.org/10.1080/10618600.2019.1695624. [p120, 129]

H. Wickham, M. Averick, J. Bryan, W. Chang, L. D. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. L. Pedersen, E. Miller, S. M. Bache, K. Müller, J. Ooms, D. Robinson, D. P. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani. Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686, 2019. doi: 10.21105/joss.01686. [p123]

M. Xu and M. Han. Adaptive elastic echo state network for multivariate time series prediction. *IEEE Transactions on Cybernetics*, 46(10):2173–2183, 2016. doi: 10.1109/TCYB.2015.2467167. [p123]

*Alexander Häußer*
*Justus Liebig University Giessen*
*Faculty of Economics and Business Studies*
*Chair of Statistics and Econometrics*
*Licher Strasse 64*
*35394 Giessen, Germany*
*ORCID: 0009-0000-5419-8479*
alexander.haeusser@wirtschaft.uni-giessen.de

# ldmppr: Location Dependent Marked Point Processes in R

*by Lane Drew and Andee Kaplan*

**Abstract** In this article, we present ldmppr, an R package for estimating, evaluating, simulating from, and visualizing location-dependent marked spatial point processes. To date, it has commonly been assumed that the marks associated with a point process are independent of the locations. However, when dealing with many point processes, such as those arising in forestry applications, the independence assumption proves unreasonable. We introduce a practical framework for generating marked point processes with dependence between the marks and locations. We provide a brief discussion of the theory underpinning our modeling approach and outline the use of the package in a typical scenario involving real data. We highlight the functionality of the package for both generating from and assessing the goodness-of-fit of a given model, enabling users to generate realistic point patterns given a reference pattern or parameter values of interest.

## 1 Introduction

Point process models are a rich and complex class of models encompassing processes that occur over time, space, and potentially include additional information about the process in the form of marks. Marks, which are attributes associated with each point (i.e., size or type), add another layer of complexity, particularly when they depend on spatially or temporally varying covariates. The behavior of a point process is typically characterized by the relationship between points. For example, the simplest process is a homogeneous Poisson process, which assumes a constant intensity and no interaction between points and may be described as complete spatial randomness (CSR), where intensity describes the expected number of points per unit of area. More structurally involved processes may exhibit regularity (or inhibition), where points tend to repel each other, or clustering, where collections of points tend to occur in proximity to each other within the pattern. Capturing the dynamics of these processes can be challenging even without the inclusion of marks, and notably more so when this type of auxiliary information is present. Consequently, researchers often make the simplifying assumption that the marks associated with a point process are independent of the primary process itself. However, this assumption is often hard to justify from a scientific perspective, as in the case of trees within a forest where the sizes of the trees (i.e., canopy volumes) play a role in their distribution over space, and where the sizes themselves are likely to depend on location-specific information such as elevation, soil moisture, or sunlight availability, which vary over space. Marked point processes have been used across a variety of fields including epidemiology, seismology, criminology, ecology, and the health sciences. Some examples of marked point processes include ambulance call locations with call severity and patient gender as marks (Bayisa et al., 2023), and locations of crime incidences with type of crime as a mark (Mohler et al., 2011).

In this paper, we present **ldmppr** (Drew and Kaplan, 2025), an R package that provides a suite of tools for working with location-dependent marked point processes characterized by regularity in the point pattern. While a wealth of R packages exist for working with point processes, such as **spatstat** (Baddeley and Turner, 2005) and **ptProcess** (Harte, 2010), their focus is often broad in scope and they fail to incorporate dependence in the mark distribution with a high degree of flexibility. In contrast, **ldmppr** is designed specifically for working with spatial marked point processes with dependence between the marks and locations that demonstrate inhibitory behavior in the spatial pattern. The package is structured to deliver a straightforward and modular workflow that simplifies the process of model estimation, evaluation, simulation, and visualization given a reference dataset and a set of corresponding covariate surfaces in the form of raster images. In addition to the included models, users can easily adapt the workflow to their specific needs by substituting appropriate components, while still taking advantage of the overall package structure.

While inhibitory marked point processes are commonly modeled using Gibbs processes, they are known to be computationally expensive and difficult to evaluate efficiently. Instead, we take a likelihood optimization approach that is computationally scalable, generally tractable, and enables a straightforward mechanism for simulating from and evaluating the goodness-of-fit of a model. To achieve this, we employ the mechanistic approach for equating a marked spatial point process with a spatio-temporal point process outlined by Møller et al. (2016), who demonstrated the utility of the self-correcting process introduced by Isham and Westcott (1979) for modeling marked point processes with regularity. We extend this framework to include location dependence in the mark distribution using a class of flexible non-linear models, improving the applicability of the original approach to a

wider variety of biologically plausible and scientifically interesting processes.

The remainder of the paper is organized as follows. Section 2 provides an overview of the relevant point process theory and modeling framework underlying **ldmppr**. Section 3 describes the package structure and functionality. Section 4 showcases the standard workflow for the package on an example dataset, with an emphasis on forestry applications. Section 5 concludes with a discussion of the methodological contributions, strengths, limitations, and potential directions for future development of the package.

## 2 Mathematical background

We begin this section with a brief introduction of the theory of marked point processes to provide context for our modeling approach. We follow this with a discussion of our proposed approach for mapping a marked point process to a marked spatio-temporal process that preserves the dependence between marks and the process and the self-correcting model that we employ for point processes characterized by regularity.

### 2.1 Marked point processes

Marked point processes are an extension of point processes that include additional point-specific information in the form of marks. Marks may represent continuous or discrete quantities (i.e., sizes or species of trees in a forest), and may be dependent or independent of the primary generating process. In a process with dependence, the distribution of the marks depends on the locations of the points, while independent marks occur independently of the points themselves. In practice, marked point processes may be modeled by a joint distribution for the points and their associated marks, or alternatively by the conditional distribution of the marks given the locations, often described by their first and second-order characteristics. The first-order characteristic is the mark intensity function which describes the expected mark value per unit of area, while the second-order characteristic, the mark correlation function, captures the dependence between marks at different times or locations (Schlather et al., 2004).

For purposes of illustration, we consider a marked spatial point process over a finite domain $\mathscr{S} \subset \mathbb{R}^2$ defined as $\mathbf{\Phi} = \{(\boldsymbol{U}_i, \boldsymbol{M}_i) : i = 0, \ldots, N\}$ with a two-dimensional nonnegative mark (though this may be extended to a higher-dimensional mark space with additional real valued or discrete marks). We assume that the process is characterized by regularity such that points in the process tend to repel each other resulting in higher interpoint distances than would be observed in a pattern with CSR. We define $\boldsymbol{U}_i \in \mathscr{S}$ to be the spatial location of the $i$-th point and $\boldsymbol{M}_i = (M_{i1}, M_{i2}) \in \mathbb{R}^+ \times \mathbb{R}^+$ to be the mark vector associated with the $i$-th point, where $M_{i1} = M_1(\boldsymbol{U}_i)$ is taken to be a measure of size or age and $M_{i2} = M_2(\boldsymbol{U}_i)$ is a secondary mark that may be of additional scientific interest (i.e., height or dbh in forestry applications). We allow for the possibility that $M_{i1} = M_{i2}$, which may be of interest when utilizing our modeling approach to capture location dependence in the primary mark distribution. Additionally, $\boldsymbol{Z}(\boldsymbol{U}_i)$ represents a vector of topographic (or location-specific) covariates at location $\boldsymbol{U}_i$. We assume that the mark vector $\boldsymbol{M} = \{\boldsymbol{M}_i : i = 0, \ldots, N\}$ is ordered according to the first mark such that the $M_{i1}$ are increasingly ordered continuous random variables with $0 \leq M_{01} \leq \cdots \leq M_{N1} \leq \tau < \infty$ and where $\tau$ is an unknown upper bound for the marks.

A common approach for modeling a process like $\mathbf{\Phi}$ with regularity in the spatial pattern would be to adopt a Gibbs process model as in Møller and Waagepetersen (2003). However, these models typically include an intractable normalizing constant that makes them difficult to estimate efficiently and often rely on computationally expensive MCMC algorithms. Alternatively, assuming that we have a point process with a structure like $\mathbf{\Phi}$, we can use a likelihood-based approach that allows for straightforward estimation, model evaluation, and simulation by extending the mechanistic framework introduced by Møller et al. (2016). The method maps a spatial marked point process onto a marked spatio-temporal process conditional on the history of the process by incorporating location dependence in the mark distribution and allowing for higher dimensional marks, as described in the following section.

### 2.2 Spatio-temporal process mapping with location dependent marks

While marked point processes naturally capture the relationship between event locations and their associated attributes, directly modeling these dependencies can be challenging. By mapping a marked spatial point process onto a spatio-temporal process, we introduce a structured mechanism for estimation that preserves the dependence between marks and locations while leveraging established likelihood-based methods for estimation. Spatio-temporal point processes are a class of models that

represent the occurrences of random events over time and space and may be extended to include additional marks that depend on space, time, or both domains simultaneously (Rathbun and Cressie, 1994). These processes are driven by a conditional intensity function

$$\lambda^*(t, \boldsymbol{y} \mid \mathscr{F}_t), \quad t > 0,\ \boldsymbol{y} \in \mathscr{S},$$

where $\mathscr{F}_t$ is the $\sigma$-algebra generated by the oldest point in the process $\boldsymbol{M}_N$, which we define as the anchor point for the process, and the points in the process $(T_i, \boldsymbol{Y}_i)$ with $T_i < t$ (a more thorough discussion may be found in Daley and Vere-Jones (1988)). The conditional intensity function describes the instantaneous expected rate of events at time $t$ and at location $\boldsymbol{y}$, conditional upon all of the points in the process occurring before time $t$.

This process may be described as a mechanistic model (Diggle, 2013), and for a marked point process of the form of $\boldsymbol{\Phi}$, defined in Section 2.1, we can equate the process with an infinite marked spatio-temporal point process $\{(T_1, \boldsymbol{Y}_1, X_1), (T_2, \boldsymbol{Y}_2, X_2), \dots\}$, where $\mathscr{T} \subset \mathbb{R}$ is the temporal domain. To facilitate this transformation, we introduce a mapping from the original marked point process notation $\boldsymbol{\Phi} = \{(\boldsymbol{U}_i, \boldsymbol{M}_i) : i = 0, \dots, N\}$ to a spatio-temporal representation $\{(T_i, \boldsymbol{Y}_i, X_i)\}$. First, we associate each point $(\boldsymbol{U}_i, \boldsymbol{M}_i)$ with a derived time $\widetilde{T}_i \overset{def}{=} T(M_{i1}) \in \mathscr{T}$ via

$$\widetilde{T}_i = 1 - \left[\frac{M_{i1} - \min_j(M_{j1})}{\max_j(M_{j1}) - \min_j(M_{j1})}\right]^{\delta}, \qquad i = 0, \dots, N, \tag{1}$$

where $\delta$ controls the shape of the mapping function and $\delta = 1$ is a 1-to-1 linear mapping. This construction yields $\min_i \widetilde{T}_i = 0$ and $\max_i \widetilde{T}_i = 1$, so the observed times occur on the interval $[0, 1]$, which standardizes the times and ensures comparability across datasets for the temporal parameters while reducing sensitivity to the scale of the mark variable chosen for the temporal mapping.

We then relabel points according to their derived times, so that the spatio-temporal index corresponds to the ordered event times. Let $\pi$ be a permutation of $\{0, \dots, N\}$ such that $\widetilde{T}_{\pi(1)} < \widetilde{T}_{\pi(2)} < \cdots < \widetilde{T}_{\pi(N+1)}$. Define, for $i = 1, \dots, N+1$,

$$T_i \overset{def}{=} \widetilde{T}_{\pi(i)}, \qquad \boldsymbol{Y}_i \overset{def}{=} \boldsymbol{U}_{\pi(i)}, \qquad X_i \overset{def}{=} M_{\pi(i),2}.$$

Thus, $\boldsymbol{Y}_i \in \mathscr{S}$ is the spatial location of the $i$-th event in time, and $X_i$ is the associated secondary mark, where $X_i$ may depend on $\boldsymbol{Y}_i$ and $\boldsymbol{Z}(\boldsymbol{Y}_i)$. Under Equation (1) (for $\delta > 0$), larger primary marks correspond to smaller derived times, so ordering by $T_i$ corresponds to processing points from largest to smallest $M_{i1}$, consistent with conditioning on the anchor point $(\boldsymbol{U}_N, \boldsymbol{M}_N)$. This representation allows us to generate a point pattern conditional on $(\boldsymbol{U}_N, \boldsymbol{M}_N)$, i.e., the point with the largest primary mark, where the behavior of the process after the final observed event time is ignored as we exploit the relationship between the observed finite processes.

We define the conditional intensity function of our marked spatio-temporal process as follows

$$\lambda^*(t, \boldsymbol{y}, x \mid \mathscr{F}_t), \quad t > 0,\ \boldsymbol{y} \in \mathscr{S},\ x \geq 0,$$

such that the intensity depends on the points arriving in the process before time $t$. The notation $\lambda^*$ signifies the dependence on the history of the process, $\mathscr{F}_t$, which is suppressed in the notation going forward for concision. We note that under the assumption of conditional independence between the arrival times and spatial locations, the conditional intensity function may be decomposed as follows

$$\lambda^*(t, \boldsymbol{y}, x) = \lambda^*(t) h_t^*(\boldsymbol{y}, x), \quad t > 0,\ \boldsymbol{y} \in \mathscr{S},\ x \geq 0,$$

where $\lambda^*(t)$ is the temporal intensity and $h_t^*(\boldsymbol{y}, x)$ is the likelihood for the marked spatial process. We note that $h_t^*(\boldsymbol{y}, x)$ may be decomposed further if we assume conditional independence between the locations and the secondary mark characteristic. This will extend the framework of Møller et al. (2016) to allow for spatio-temporal dependence in the secondary mark distribution and enable flexible estimation of the conditional mark distribution incorporating potentially complex location dependence.

We specify a general parametric model for the conditional intensity, i.e., $\lambda_{\boldsymbol{\theta}}^*(t, \boldsymbol{y}, x)$, where $\boldsymbol{\theta}$ is an unknown set of parameters and obtain the spatio-temporal log-likelihood conditional on $(\boldsymbol{U}_N, \boldsymbol{M}_N) = (\boldsymbol{u}_n, \boldsymbol{m}_n)$ such that

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log \lambda_{\boldsymbol{\theta}}^*(t_i, \boldsymbol{y}_i, x_i) - \iiint_{(0,t_n)\times\mathscr{S}\times\mathbb{R}^+} \lambda_{\boldsymbol{\theta}}^*(t, \boldsymbol{y}, x)\, dt\, d\boldsymbol{y}\, dx\,.$$

To facilitate model fitting and simulation, we introduce the temporal-integrated intensity function

$$\Lambda^*(t) = \int_0^t \lambda^*(s)\, ds, \quad t > 0, \tag{2}$$

such that $S_i = \Lambda^*(T_i)$, $i = 1, 2, \ldots$ form a unit rate Poisson process on $(0, \infty)$. This relationship allows us to easily simulate from the process once the model is estimated, which is a key component in assessing the goodness-of-fit of the estimated process. Details for the simulation algorithm are provided in Section 2.5.

### 2.3 Model exhibiting regularity

The framework we have introduced in this section applies to a general marked point process. We now turn our attention to processes exhibiting regularity in their spatial pattern as described in Section 2.1. Point processes may be described as self-correcting when the intensity function decreases as the number of events increases, resulting in an inhibitory effect that counterbalances event clustering and enforces regularity in the spatial pattern over time. We define a modified version of the self-correcting model introduced by Isham and Westcott (1979) with the following intensity function

$$\lambda^*(t, \boldsymbol{y}, x) = \lambda^*_{\boldsymbol{\theta}_1}(t) h^*_{\boldsymbol{\theta}_2, t}(\boldsymbol{y}) g^*_{\boldsymbol{\theta}_3, t, \boldsymbol{y}, \boldsymbol{z}(\boldsymbol{y})}(x) f^*_{\boldsymbol{\theta}_4}(t, \boldsymbol{y}).$$

This model incorporates an arrival time process $(\lambda^*_{\boldsymbol{\theta}_1}(t))$, spatial process $(h^*_{\boldsymbol{\theta}_2, t}(\boldsymbol{y}))$, conditional mark process $(g^*_{\boldsymbol{\theta}_3, t, \boldsymbol{y}}(x))$, and spatio-temporal interaction $(f^*_{\boldsymbol{\theta}_4}(t, \boldsymbol{y}))$. We discuss the formulations for each component as follows.

#### Arrival time process

The arrival time process $T$ is modeled with intensity function $\lambda^*_{\boldsymbol{\theta}_1}(t)$, following Møller et al. (2016), given by

$$\lambda^*_{\boldsymbol{\theta}_1}(t) = \exp(\alpha_1 + \beta_1 t - \gamma_1 N(t)),$$

where $\boldsymbol{\theta}_1 = (\alpha_1, \beta_1, \gamma_1)$ such that $\alpha_1 \in \mathbb{R}$ is a baseline rate, $\beta_1 \in [0, \infty)$ is a log-linear function of $t$, and $\gamma_1 \in [0, \infty)$ is the scaling factor for $N(t)$ where $N(t) = \sum_{i=1}^{N} \mathbb{I}\{i \geq 0 : t_i < t\}$.

#### Spatial process

We model the spatial process with interaction function $\psi(r)$ such that

$$\psi_{\boldsymbol{\theta}_2}(r) = \mathbb{I}[r \leq \alpha_2](r/\alpha_2)^{\beta_2} + \mathbb{I}[r > \alpha_2], \quad r \geq 0,$$

where $r$ is the Euclidean distance between two points (i.e., $r_{ij} = \|\boldsymbol{y}_i - \boldsymbol{y}_j\|$). This formulation is analogous to the pairwise interaction function in Diggle and Gratton (1984). The density for the process is given by

$$h^*_{\boldsymbol{\theta}_2, i}(\boldsymbol{y}_i) = \frac{1}{c^*_{\boldsymbol{\theta}_2, i}} \prod_{j:j<i} \psi_{\boldsymbol{\theta}_2}(r_{ij}), \tag{3}$$

where $c^*_{\boldsymbol{\theta}_2, i}$ is the normalizing constant for the spatial density, which is defined as

$$c^*_{\boldsymbol{\theta}_2, i} = \int_{\mathscr{S}} \prod_{j:j<i} \psi_{\boldsymbol{\theta}_2}(r_{ij})\, d\boldsymbol{y},$$

and $\boldsymbol{\theta}_2 = (\alpha_2, \beta_2) \in [0, \infty)^2$. The spatial density $h^*_{\boldsymbol{\theta}_2, i}$ defines an inhibitive circular region around each larger point $\boldsymbol{y}_j$, for all points with $t_j < t_i$, such that the strength of the interaction diminishes at a polynomial rate for interpoint distances less than $\alpha_2$ and disappears for distances greater than $\alpha_2$.

#### Conditional mark process

We model the conditional mark process using a flexible non-linear model such that

$$g^*_{\boldsymbol{\theta}_3}(x_i \mid t_i, \boldsymbol{y}_i, \boldsymbol{z}(\boldsymbol{y}_i)) = \mathrm{G}_{x_i}(t_i, \boldsymbol{y}_i, \boldsymbol{z}(\boldsymbol{y}_i)),$$

where G is trained on the feature set $\{T_i, \boldsymbol{Y}_i, \boldsymbol{Z}(\boldsymbol{Y}_i)\}$, and $\boldsymbol{Z}(\boldsymbol{Y}_i)$ is the set of covariate values at the location $\boldsymbol{Y}_i$. We specify this component in generality, and note that any suitably flexible model may be chosen to model the conditional mark process (e.g., a random forest (Breiman, 2001) or gradient boosted

tree model (Chen and Guestrin, 2016)). We assume that the mark process is conditionally independent given the times and locations and is parameterized by a non-overlapping set of parameters $\boldsymbol{\theta}_3$ when estimating the model. For example, in a forestry application, assuming conditional independence of marks (i.e., sizes) given spatial and temporal covariates implies that tree size at a given location and time is determined by local environmental factors and possible interpoint competition indices.

### Spatio-temporal interaction

The final component of the model is the spatio-temporal interaction term, as introduced by Møller et al. (2016), which we define as

$$f^*_{\boldsymbol{\theta}_4}(t_i, \boldsymbol{y}_i) = \exp\left(-\alpha_4 \sum_{j:j<i} \mathbb{I}[\|\boldsymbol{y}_j - \boldsymbol{y}_i\| \leq \beta_4, t_i - t_j \geq \gamma_4]\right),$$

where $\boldsymbol{\theta}_4 = (\alpha_4, \beta_4, \gamma_4) \in [0, \infty)^3$ with $\beta_4 = \gamma_4 = 0$ if $\alpha_4 = 0$. This term connects the temporal and spatial process components of the full model and captures the dependence between these components. The form of the interaction term regulates the development of the process by limiting the occurrence of points that arrive in close proximity to each other in both space and time.

## 2.4 Parameter estimation

We now consider the procedure for estimating the parameters of the marked spatio-temporal process. We leverage the separability of the full log-likelihood under the assumption of conditional independence of the marks given the locations and arrival times and estimate the conditional mark process and the spatio-temporal process individually. This approach allows us to take advantage of highly flexible non-linear machine learning models to capture the conditional mark process, while maintaining the computational efficiency of maximum likelihood estimation for the spatio-temporal process. We note that both processes depend on the mapping from the original mark space to arrival times, which is controlled by the parameter $\delta$ in Equation (1). However, we do not necessarily require the mappings to be identical for both processes, as the conditional mark process may exhibit different behavior than the spatial process. We provide an overview of the estimation procedure for the full model in practice in Section 3.

For the self-correcting model described in Section 2.3, we estimate the parameters of the process by optimizing the spatio-temporal component of the log-likelihood, which may be expressed as

$$L(\boldsymbol{\theta_1}, \boldsymbol{\theta_2}, \boldsymbol{\theta_4}) \propto \sum_{i=1}^{n} \left[\log \lambda^*_{\boldsymbol{\theta}_1}(t_i) + \log h^*_{\boldsymbol{\theta}_2, i}(\boldsymbol{y}_i) + \log f^*_{\boldsymbol{\theta}_4}(t_i, \boldsymbol{y}_i)\right] - \iint_{(0,t_n)\times\mathscr{S}} \lambda^*_{\boldsymbol{\theta}_1}(t) h^*_{\boldsymbol{\theta}_2, t}(\boldsymbol{y}) f^*_{\boldsymbol{\theta}_4}(t, \boldsymbol{y})\, dt\, d\boldsymbol{y}.$$

The parameter sets $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_4$ can be estimated using any suitable optimization algorithm that allows for bound constraints on the parameter space.

For the conditional mark process, we estimate the parameter set $\boldsymbol{\theta}_3$ using a flexible model that allows for the incorporation of spatial and temporal covariates. The set of features used in training the model which correspond to location-specific covariates $\boldsymbol{Z}(\boldsymbol{Y}_i)$ may be derived from raster images or other spatial data sources and may include interpoint competition metrics, as discussed in Section 3.3.

We combine the estimated parameter values for the conditional mark process and the spatio-temporal process to obtain the full set of estimated parameters for the marked spatio-temporal process. This approach is modular and allows for a wide variety of models to be substituted for the conditional mark process. In theory, this framework also holds for alternative spatio-temporal processes, such as those exhibiting clustering or other spatial patterns, though the specific form of the likelihood components would need to be adjusted accordingly.

## 2.5 Simulating from the spatio-temporal process

Utilizing the framework outlined in this section and estimates of the parameter sets $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$, $\boldsymbol{\theta}_3$, $\boldsymbol{\theta}_4$, we can easily generate realizations from the marked spatio-temporal point process on $[0, 1] \times \mathscr{S}$. Given $S_i = \Lambda^*(T_i)$, $i = 1, 2, \ldots$ as defined in Equation (2), we have that

$$T_1 = \frac{1}{\beta_1} \log\{1 + \beta_1(\gamma_1 - \alpha_1)S_1\}$$

and for $i = 2, 3, \ldots$

$$T_i = \frac{1}{\beta_1} \log \Bigg[ \exp(\beta_1 T_{i-1}) + \beta_1 \exp(\gamma_1 i - \alpha_1) S_i - \sum_{j=1}^{i-1} \{\gamma_1 (i-j)\} \{\exp(\beta_1 T_j) - \exp(\beta_1 T_{j-1})\} \Bigg]$$

such that we can generate a realization $t_1 < \cdots < t_n$ under the self-correcting model on $[0, 1]$ as follows.

1. Generate a draw from a Poisson distribution with rate $\rho^*_{\widehat{\boldsymbol{\theta}}_1}(t) = \exp\{\widehat{\alpha}_1 + \widehat{\beta}_1 t_n\}$, where $t_n = 1$ to obtain the number of possible arrival times $n^*$.
2. For $i = 1, \ldots, n^*$, generate a draw from the uniform distribution on $(0, t_n)$.
3. Order the uniform draws such that $u_1 < \cdots < u_{n^*}$ to obtain the candidate arrival times $t^*_1, \ldots, t^*_{n^*}$.
4. For $i = 1, \ldots, n^*$, calculate the acceptance ratio

$$r^*_i = \frac{\lambda^*_{\widehat{\boldsymbol{\theta}}_1}(t^*_i)}{\exp\{\widehat{\alpha}_1 + \widehat{\beta}_1 t_n\}},$$

   then generate a draw $u^*_i$ from the uniform distribution on $(0, 1)$ and accept $t^*_i$ if $u^*_i < r^*_i$.
5. Given the set of accepted arrival times $\{t_i\}_{i=1}^n$, generate points $\boldsymbol{y}_i$ from the density $h^*_{\widehat{\boldsymbol{\theta}}_{2,i}}(\boldsymbol{y}_i)$ using rejection sampling with acceptance probability derived from the unnormalized component of the spatial interaction in Equation (3) and a uniform proposal distribution over $\mathscr{S}$.
6. Given the set of $\{t_i, \boldsymbol{y}_i\}_{i=1}^n$, thin the process such that each point is kept with probability $f^*_{\widehat{\boldsymbol{\theta}}_4}(t_i, \boldsymbol{y}_i)$ and the kept points are a realization of the spatio-temporal process.
7. For $i = 1, \ldots, n$, given $(t_i, \boldsymbol{y}_i, \boldsymbol{z}(\boldsymbol{y}_i))$, generate $x_i$ from the density $g^*_{\widehat{\boldsymbol{\theta}}_3}(x_i \mid t_i, \boldsymbol{y}_i, \boldsymbol{z}(\boldsymbol{y}_i))$.

This modeling approach allows us to employ interpretable and computationally tractable likelihood methods for estimating and simulating from the spatio-temporal process and provides a high degree of flexibility when selecting and training a model for the conditional mark distribution as discussed in the following section.

## 3 Package structure

In this section, we introduce the core functionality for the **ldmppr** package and detail the key functions associated with spatio-temporal process and mark model estimation, model goodness-of-fit evaluation, simulation, and visualization. We note that the package is currently designed for working with marked point processes that can be mapped onto spatio-temporal processes where the pattern is characterized by regularity and the mark distribution is dependent on location- specific covariate information.

### 3.1 Workflow

We begin by describing the standard workflow that we envision for using the package. We decompose the task of working with marked point processes into a handful of straightforward and manageable steps supported by an intuitive set of functions. The process is outlined as follows, where the output of each step can be passed as the input to successive stages where appropriate.

1. Estimate the parameters of a self-correcting point process given a reference dataset.
2. Train a mark model given the reference data set and topographic covariate surfaces in the form of rasters.
3. Check the fit of the estimated model using various non-parametric summaries for point processes and global envelope tests.
4. Simulate and visualize datasets from the fitted model.

We anticipate that users of the package will have a point process that they are interested in investigating, and we provide the tools to facilitate that exploration. We also note the modular structure of the package such that a user may provide their own estimated mark model in lieu of one of the currently available options in the package. In the remaining portions of this section, we detail the key functionality for each step given above.

### 3.2 Self-correcting model estimation

The first step in the model estimation procedure is to estimate the parameters of the self-correcting model detailed in Section 2.3 that captures the spatio-temporal process in the data. Given a reference dataset, we must define a mapping between our initial mark (i.e., size) and the arrival time in the process by specifying a value for $\delta$. This allows the user to specify how the marks are mapped to arrival times using the transformation given in Equation (1). Once a mapping is chosen, the user should select an optimization strategy ("`local`", "`global_local`", or "`multires_global_local`") that determines how the optimization is performed. For the `"local"` strategy, the optimization is run using a single layer with a local algorithm. For the "`global_local`" strategy, the optimization is run using two layers that start with a global optimization that is then updated in a local sweep. For the "`multires_global_local`" strategy, this extends the "`global_local`" approach to allow users to further refine the second stage local estimate at higher resolution. Once a strategy has been selected, users must specify both the integration grid and the optimization budget using `ldmppr_grids()` and `ldmppr_budgets()`, respectively. The grid controls the numerical resolution used to approximate the likelihood integral at each fitting level, while the budget controls how much optimization effort is allocated to each stage (global search, first-level local search, and refinement-level local search), including stopping criteria such as evaluation limits and convergence tolerances. In this context, "budget" refers to computational effort, not model complexity. With both components defined, parameter estimation for the self-correcting process proceeds via `estimate_process_parameters()`, where the arguments of the function are defined as follows.

| Argument | Description |
|---:|---|
| `data` | a data.frame or matrix. Must contain either columns `(time, x, y)` or `(x, y, size)`. If a matrix is provided without time, it must have column names `c("x","y","size")`. |
| `process` | type of process used (currently supports "`self_correcting`"). |
| `grids` | a `ldmppr_grids` object specifying the integration grid schedule (single-level or multi-resolution). The integration bounds are taken from `grids$upper_bounds`. |
| `budgets` | a `ldmppr_budgets` object controlling optimizer options for the global stage and local stages (first level vs refinement levels). |
| `parameter_inits` | (optional) numeric vector of length 8 giving initialization values for the model parameters. If `NULL`, defaults are derived from `data` and `grids$upper_bounds`. |
| `delta` | (optional) numeric scalar or vector. Used only when `data` does not contain `time` (i.e., data has `(x,y,size)`).<br>• If `length(delta) == 1`, fits the model once using `power_law_mapping(size, delta)`.<br>• If `length(delta) > 1`, performs a delta-search by fitting the model for each candidate value and selecting the best objective. If `refine_best_delta = TRUE` and multiple grid levels are used, the best delta is refined on the remaining (finer) grid levels.<br>If `data` already contains `time`, `delta` is ignored when `length(delta)==1` and is an error when `length(delta)>1`. |
| `parallel` | `TRUE` or `FALSE` specifying **furrr**/**future** to parallelize either:<br>a) over candidate `delta` values (when `length(delta) > 1`), and/or<br>b) over local multi-start initializations (when `starts$local > 1`), and/or<br>c) over global restarts (when `starts$global > 1`). |
| `num_cores` | number of workers to use when `set_future_plan = TRUE`. |
| `set_future_plan` | `TRUE` or `FALSE`, temporarily sets `future::plan(multisession, workers = num_cores)` and restores the original plan on exit. |

| | |
|---|---|
| `strategy` | character string specifying the estimation strategy:<br>• "`local`": local optimization only (single-level or multi-level polish).<br>• "`global_local`": global optimization then local polish (single grid level).<br>• "`multires_global_local`": multi-resolution (coarsest uses global+local; refinements use local only). |
| `global_algorithm`, `local_algorithm` | NLopt algorithms to use for the global and local optimization stages, respectively. |
| `starts` | a list controlling restart and jitter behavior:<br>• `global`: integer, number of global restarts at the first/coarsest level (default 1).<br>• `local`: integer, number of local multi-starts per level (default 1).<br>• `jitter_sd`: numeric SD for jittering (default 0.35).<br>• `seed`: integer base seed (default 1). |
| `finite_bounds` | (optional) list with components `lb` and `ub` giving finite lower and upper bounds for all 8 parameters. If `NULL`, bounds are derived from `parameter_inits`. Global algorithms and select local algorithms in NLopt require finite bounds. |
| `refine_best_delta` | `TRUE` or `FALSE`. If `TRUE` and `length(delta) > 1`, performs refinement of the best `delta` across additional grid levels (if available). |
| `rescore_control` | controls candidate rescoring and bound-handling behavior in multi-resolution fitting. Can be either:<br>• a single logical value (toggle rescoring on/off while keeping defaults), or<br>• a named list with any of: `enabled`, `top`, `objective_tol`, `param_tol`, `avoid_bound_solutions`, `bound_eps`.<br>Defaults are `list(enabled = TRUE, top = 5L, objective_tol = 1e-6, param_tol = 0.10, avoid_bound_solutions = TRUE, bound_eps = 1e-8)`. |
| `verbose` | `TRUE` or `FALSE` indicating whether to show progress of model estimation. |

This function makes use of the R implementation of the NLopt library (Johnson, 2008) and returns an `ldmppr_fit` object containing the estimated parameters and optimization details. The optimization procedure is designed to be flexible and allows for a variety of strategies and algorithm choices and combinations through the `nloptr` engine. The optimal choice of strategy and global/local algorithm may depend on the dataset, but the recommended default is a "`global_local`" strategy using the `NLOPT_GN_CRS2_LM` global algorithm paired with the `NLOPT_LN_BOBYQA` local algorithm. The `NLOPT_GN_CRS2_LM` algorithm is a global optimization algorithm that implements the "controlled random search" method, originally introduced by Price (1983), with the "local mutation" modification defined by Kaelo and Ali (2006). This algorithm is designed to be robust and efficient for global optimization problems, as it is less dependent on the initialization values and has been found to perform well in our testing. The recommended local optimization algorithm `NLOPT_LN_BOBYQA` is a derivative-free optimization algorithm that is designed for bound-constrained optimization problems through iteratively building a quadratic approximation of the objective function (Powell, 2009). Another local optimization option is `NLOPT_LN_SBPLX`, which is an implementation of the "Subplex" algorithm introduced by Rowan (1990) that incorporates explicit bound constraints. This algorithm is a more efficient and robust implementation of the original Nelder-Mead algorithm (Nelder and Mead, 1965). If the user wishes to test a set of different $\delta$ values for the mapping function, the `delta` argument may be specified as a vector and the `refine_best_delta` argument enables refinement of the optimal $\delta$ from the initial stage in additional layers.

In practice, we recommend a staged setup for `estimate_process_parameters()`. First, set `delta = 1` and obtain a baseline fit with a coarse grid and moderate optimization budgets. If diagnostics suggest poor fit, then increase optimization effort by adding additional starts and moving to the "`multires_global_local`" strategy with stricter tolerances. If users wish to tune `delta`, in the absence of prior knowledge of the relationship, we recommend searching a small neighborhood around 1

(rather than a wide range), selecting the best objective, and then refining only that candidate with finer grids. This strategy typically improves robustness while keeping computational cost manageable and allows users to refine the fit as needed based on the available diagnostics.

In addition to the optimization strategy, the user may also specify the number of global and local multi-starts and the amount of jittering to add to the initial parameter values for each start within the `starts` argument. Jittering multiple starts can help to improve the optimization results, and reduce dependence on the initial values. This will increase the computational time for estimation, though this can be offset by parallelization over the multi-starts using the `parallel` and `num_cores` arguments. The `estimate_process_parameters()` function is constructed to derive reasonable initial values and bound constraints based on the provided reference dataset, and we encourage users to use these defaults unless they have substantial knowledge about the reasonable values and parameter bounds for their specific application. We also note that the success of the optimization procedure may depend on the number of function evaluations and tolerance settings for the various algorithms, which can be controlled through the `budgets` argument. We recommend starting with a moderate number of function evaluations (e.g., 1000 for the global stage and 500 for the local stage) and adjusting as needed based on the results and computational resources available. The choice of integration grid granularity, specified through `grids`, for the spatial and temporal components should reflect the scale of the data and the expected range of interactions. See Section 4 for an example of the effect of the choice of strategy, grids, and budgets on the model fit.

### 3.3 Mark model training

The second step in the model estimation procedure is to train the conditional mark model. We use the reference data with the mapped arrival times derived in the self-correcting model estimation step, or an alternate mapping, in concert with a set of covariate surfaces in the form of raster images. The raster images may be pre-processed using the `scale_rasters()` function, or may be provided in their raw form. To train the model, we use the `train_mark_model()` function, where the arguments of the function are provided as follows.

| Argument | Description |
|---:|---|
| `data` | a data.frame or a `ldmppr_fit` object. |
| `raster_list` | a list of raster objects. |
| `scaled_rasters` | `TRUE` or `FALSE` indicating whether the rasters have been scaled. |
| `model_type` | the machine learning model type (“`xgboost`” or “`random_forest`”). |
| `xy_bounds` | a vector of domain bounds (2 for x, 2 for y). If `data` is an `ldmppr_fit` and `xy_bounds` is `NULL`, defaults to `c(0, b_x, 0, b_y)` derived from fit. |
| `delta` | (optional) numeric scalar used only when `data` contains `(x,y,size)` but not `time`. If `data` is an `ldmppr_fit` and time is missing, the function will infer the `delta` value from the fit. |
| `save_model` | `TRUE` or `FALSE` indicating whether to save the generated model. |
| `save_path` | the path for saving the generated model. |
| `parallel` | `TRUE` or `FALSE` indicating whether to use parallelization in model training. |
| `num_cores` | number of cores to use in parallel model training (if `parallel` is `TRUE`). |
| `include_comp_inds` | `TRUE` or `FALSE` indicating whether to generate and use competition indices as covariates. |
| `competition_radius` | distance for competition radius if `include_comp_inds` is `TRUE`. |
| `edge_correction` | type of edge correction to apply (“`none`”, “`toroidal`”, or “`truncation`”). |
| `selection_metric` | metric to use for identifying the optimal model (“`rmse`”, “`mae`”, or “`rsq`”). |
| `cv_folds` | number of cross-validation folds to use in model training. If `cv_folds <= 1`, tuning is skipped and the model is fit once with default hyperparameters. |
| `tuning_grid_size` | size of the tuning grid for hyperparameter tuning. |
| `seed` | integer seed for reproducible resampling/tuning/model fitting. |
| `verbose` | `TRUE` or `FALSE` indicating whether to show progress of model training. |

The `train_mark_model()` function allows users to select between a random forest or gradient boosted tree model, using the **ranger** (Wright and Ziegler, 2017) and **xgboost** (Chen et al., 2026) engines respectively, where these models are effective at capturing potentially complex non-linear relationships. Users choose a model selection criterion, either root mean squared error (RMSE) or mean absolute error (MAE), and may employ cross validation and hyperparameter tuning using a grid design that optimizes the maximum entropy of the hyperparameter space. As a general rule, we recommend that users start with a moderate number of cross-validation folds (e.g., 5) and a tuning grid size of 100, and adjust as needed based on the results and computational resources available, where parallelization can be used to speed up the tuning process. The `train_mark_model()` function is designed to be flexible and allows users to specify a variety of options for model training, including the ability to save the trained model for future use.

The function also allows users to incorporate a collection of interpoint competition metrics at a specified neighborhood size to capture additional trends that are not accounted for by the topographic covariates. The metrics include nearest neighbor distance, number of neighbors, average neighbor distance, nearest neighbor arrival time, sum of neighbor arrival times, and the ratio of nearest neighbor distance and arrival time. For an in- depth discussion of competition indices and their construction, see Pommerening and Sánchez Meador (2018) and Contreras et al. (2011). Finally, users may select an edge correction mechanism when training the model to account for the possibility that the reference dataset provided is a subset of a larger dataset and that unobserved points are impacting the observed mark values (i.e., sizes).

### 3.4 Goodness-of-fit checks for the fitted model

Once the self-correcting model parameters are estimated and the mark model is trained, we turn our attention to assessing how well the fitted models capture the dynamics observed in the reference dataset. To accomplish this, we use the `check_model_fit()` function, which provides global envelope tests, using the **GET** package (Myllymäki et al., 2017), for a collection of standard non-parametric marked point process summary functions. We include the $L$, $F$, $G$, $J$, $E$, and $V$ functions as evaluation metrics, together with a combined global envelope test that provides a single Monte Carlo $p$-value across the full set of summaries.

The $L$-function is a variance-stabilized transformation of Ripley's $K$-function and is commonly interpreted as a second-order measure of spatial interaction: relative to CSR, $L(r)$ above (below) its theoretical reference suggests clustering (inhibition) at spatial scale $r$ (Ripley, 1976, Diggle (2013)). The $F$ and $G$ functions summarize complementary nearest-neighbor behavior: $F(r)$ (the empty-space function) describes distances from arbitrary locations in $\mathscr{S}$ to the nearest event, whereas $G(r)$ (the nearest-neighbor distribution function) describes distances from events to their nearest neighboring event (Diggle, 2013). The $J$-function combines these summaries via $J(r) = \frac{1-G(r)}{1-F(r)}$; values $J(r) > 1$ are typically associated with inhibition/regularity and values $J(r) < 1$ with clustering (van Lieshout, 2006). To assess the mark component, we use the mark summary functions $E$ and $V$ (implemented as `Emark` and `Vmark` in **spatstat**), which describe how the conditional mean and conditional variability of marks change with interpoint distance, thereby providing diagnostics for mark–location dependence beyond the unmarked spatial structure (Schlather et al., 2004, Baddeley et al. (2015)). Finally, global envelope testing provides a principled way to control multiplicity across distances (and across multiple functions in the combined test) when comparing the observed summaries to simulations from the fitted model (Myllymäki et al., 2017).

The arguments of the `check_model_fit()` function are defined as follows.

| Argument | Description |
|---|---|
| `reference_data` | (optional) a marked ppp object for the reference dataset. If `NULL`, the reference pattern is derived from `process_fit` when `process_fit` is an `ldmppr_fit` and contains `data_original` (preferred) or `data` with columns `(x, y, size)`. |
| `t_min` | minimum value for time. |
| `t_max` | maximum value for time. |
| `process` | type of process used (currently supports "`self_correcting`"). |
| `process_fit` | either an `ldmppr_fit` object (from `estimate_process_parameters`) or a numeric vector of length 8 giving the process parameters. |
| `anchor_point` | (optional) vector of `(x, y)` coordinates of the point to condition on. If `NULL`, inferred from the reference data (largest mark if available) or from the `ldmppr_fit`. |

| | |
|---|---|
| raster_list | (optional) a list of raster objects used for predicting marks. Required when mark_mode = "mark_model" unless rasters are stored in mark_model. |
| scaled_rasters | TRUE or FALSE indicating whether rasters are already scaled. Ignored when mark_mode = "time_to_size". |
| mark_model | (optional) a mark model object used when mark_mode = "mark_model". May be an ldmppr_mark_model, model_fit, or workflow. |
| xy_bounds | (optional) vector of bounds as c(a_x, b_x, a_y, b_y). If NULL, inferred from reference_data's window when reference_data is provided; otherwise inferred from the ldmppr_fit with lower bounds assumed to be 0. |
| include_comp_inds | TRUE or FALSE indicating whether to compute competition indices. |
| competition_radius | distance for competition radius if include_comp_inds = TRUE. |
| thinning | TRUE or FALSE indicating whether to use the thinned simulated values. |
| edge_correction | type of edge correction to apply ("none" or "toroidal"). |
| n_sim | number of simulated datasets to generate. |
| save_sims | TRUE or FALSE indicating whether to save and return the simulated metrics. |
| verbose | TRUE or FALSE indicating whether to show progress of model checking. When TRUE, progress is reported via progressr (if available) and is compatible with parallel execution. |
| seed | integer seed for reproducibility. |
| parallel | TRUE or FALSE. If TRUE, simulations are run in parallel via furrr/future. |
| num_cores | number of workers to use when parallel = TRUE. Defaults to one fewer than the number of detected cores. |
| set_future_plan | TRUE or FALSE. If TRUE and parallel = TRUE, set a temporary future plan internally and restore the previous plan on exit. |
| mark_mode | (optional) mark generation mode. "mark_model" uses predict() on a mark model, while "time_to_size" maps simulated times back to sizes via delta. If NULL, inferred as "mark_model" when mark_model is provided, otherwise "time_to_size". |
| fg_correction | correction used for F/G/J summaries ("km" or "rs"). |
| max_attempts | maximum number of simulation attempts when rejection occurs due to non-finite summaries. |

The check_model_fit() function allows users to simulate a collection of datasets using the estimated parameters from the self-correcting process and trained mark model, which can be passed directly as ldmppr_fit and ldmppr_mark_model objects through the process_fit and mark_model arguments respectively. The non-parametric summary functions are calculated for each dataset and the global envelopes are obtained across the whole collection of simulated datasets for each metric, as well as a combined test across all metrics. This allows the user to gauge whether the realizations from the estimated location dependent marked point process accurately reflect the dynamics observed in the reference dataset across a variety of different metrics. Each individual metric includes a $p$-value range indicating the compatibility of the reference dataset with the simulated datasets, and the combined test provides a single $p$-value across the entire set of metrics, allowing users to quickly gauge the overall fit of the model. Contrary to many testing procedures, a non-significant result (i.e., a large $p$-value) indicates that the reference dataset is not incompatible with the fitted model, whereas a significant result (i.e., a small $p$-value) indicates that the reference dataset is incompatible with the fitted model. We recommend starting with a moderate number of simulations (e.g., 200 or 500) to balance accuracy of the $p$-value estimates with computational time, and adjusting as needed based on the results and resources available. Parallelization can be employed to speed up the simulation and testing process when a larger number of simulations is desired. For additional discussion of the summary functions included in check_model_fit() see Møller and Waagepetersen (2003) and Schlather et al. (2004).

### 3.5 Simulation and visualization

When a user is satisfied with the estimated location dependent marked point process obtained in the model estimation, training, and goodness- of-fit checking steps, they can proceed with simulating

realizations from the process and visualizing the results. The `simulate_mpp()` function provides an efficient way to generate a realization from the process, where the arguments for the function are as follows.

| Argument | Description |
|---|---|
| `process` | type of process used (currently supports "`self_correcting`"). |
| `process_fit` | either (1) an `ldmppr_fit` object returned by `estimate_process_parameters`, or (2) a numeric vector of length 8 giving self-correcting parameters $(\alpha_1, \beta_1, \gamma_1, \alpha_2, \beta_2, \alpha_3, \beta_3, \gamma_3)$. |
| `t_min` | minimum value for time. |
| `t_max` | maximum value for time. |
| `anchor_point` | (optional) vector of `(x, y)` coordinates of the point to condition on. If `NULL`, inferred from the reference data (largest mark if available) or from `process_fit$data_original` (largest size). |
| `raster_list` | (optional) list of raster objects used for mark prediction. Required when `mark_mode = "mark_model"` unless rasters are stored in `mark_model`. |
| `scaled_rasters` | `TRUE` or `FALSE` indicating whether rasters are already scaled. Ignored when `mark_mode = "time_to_size"`. |
| `mark_model` | (optional) mark model object used when `mark_mode = "mark_model"`. May be an `ldmppr_mark_model`, `model_fit`, or `workflow`. |
| `xy_bounds` | (optional) vector of bounds as `c(a_x, b_x, a_y, b_y)`. If `NULL`, bounds are inferred from `process_fit` when available. |
| `include_comp_inds` | `TRUE` or `FALSE` indicating whether to compute competition indices. |
| `competition_radius` | distance for competition radius if `include_comp_inds = TRUE`. |
| `thinning` | `TRUE` or `FALSE` indicating whether to use the thinned simulated values. |
| `edge_correction` | type of edge correction to apply ("`none`" or "`toroidal`"). |
| `seed` | integer seed for reproducibility. |
| `mark_mode` | (optional) mark generation mode. "`mark_model`" uses `predict()` on a mark model, while "`time_to_size`" maps simulated times back to sizes via `delta`. If `NULL`, inferred as "`mark_model`" when `mark_model` is provided, otherwise "`time_to_size`". |
| `size_range` | (optional) numeric vector `c(smin, smax)` used for `mark_mode = "time_to_size"`. If `NULL`, inferred from `process_fit` when possible. |
| `delta` | (optional) positive scalar used for `mark_mode = "time_to_size"`. If `NULL`, inferred from `process_fit` when possible. |

The function returns a `ldmppr_sim` object containing the relevant generating information, which can be provided to the function directly or inferred by passing a `ldmppr_fit` and `ldmppr_mark_model` through the `process_fit` and `mark_model` arguments respectively, and realized simulation in both a `ppp` format and as a `data.frame`. With a realization of the process in hand, the user can easily visualize the marked point process object using the default `plot()` method, which has the following arguments.

| Argument | Description |
|---|---|
| `x` | an `ldmppr_sim` object |
| `pattern_type` | type of pattern to plot "`simulated`" (default). |

In the following section, we walk through the entire workflow described in Section 3.1 with an example forestry dataset.

## 4 Application

Equipped with an understanding of the package workflow and primary functionality, we now provide an example of using **ldmppr** to analyze a forest stand dataset comprised of canopy volumes (in cubic meters) and locations for conifer species in the Southern Rocky Mountains, obtained from Drew et al. (2024), that is included in the package. We incorporate four topographic covariates, in the form of raster surfaces, that have been previously found to be related to the processes of tree growth and

that capture key environmental conditions like energy and water availability (Drew et al., 2025). The covariates included in this analysis are Southness Aspect, Topographic Wetness Index, Elevation, and Slope which are derived from a LiDAR-based digital elevation model (DEM).

Following the steps outlined in Section 3, we begin by estimating the self-correcting model using $\delta = 1$ as the size to time mapping parameter. We load the data and specify the mapping as follows.

```
data("medium_example_data")
parameter_estimation_data <- medium_example_data

delta <- 1
```

Next, we define the grid values for the spatial and temporal components of the process and upper bounds using the `ldmppr_grids()` function, and the budgets for the optimization procedure using the `ldmppr_budgets()` function. To get a sense of where to start with the grid values, we recommend plotting the data and considering the spatial and temporal scales of the observed dynamics. For example, in this dataset, the spatial domain is approximately 50m by 50m, and the mapped arrival times range from 0 to 1. We want to ensure that the grid values capture the relevant scales of the observed dynamics, while balancing computational cost. The nearest neighbor distance distribution in our dataset can be used to identify a reasonable starting point for the spatial grid values.

```
summary(spatstat.geom::nndist(parameter_estimation_data[,-3]))

#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>  0.9415  2.3064  2.7598  2.9095  3.4448  6.4035
```

Given this distribution, we start with a single grid level, specified as a $10 \times 10 \times 10$ grid for the spatial and temporal components, respectively. We opt to employ the "`global_local`" strategy for optimization, which includes a global optimization stage followed by a local polish. We start with a moderate number of function evaluations and higher tolerance for both the global and local stages to facilitate a low-cost initial estimation.

```
upper_bounds <- c(1, 50, 50)

grids <- ldmppr_grids(
  upper_bounds = upper_bounds,
  levels = list(c(10, 10, 10))
)

budgets <- ldmppr_budgets(
  global_options = list(
    maxeval = 500,
    ftol_rel = 1e-3,
    xtol_rel = 1e-3
  ),
  local_budget_first_level = list(
    maxeval = 1000,
    ftol_rel = 1e-4,
    xtol_rel = 1e-4
  )
)

starts <- list(global = 1, local = 1, jitter_sd = 0.1, seed = 90210)
```

Using the `estimate_process_parameters()` function, we then estimate the parameters of the self-correcting process and obtain the optimal parameter estimates. We use the `NLOPT_GN_CRS2_LM` global and `NLOPT_LN_BOBYQA` local algorithms (described in Section 3.2) with the internally derived initial values and parameter bounds based on the reference data for the optimization. Details regarding the derivation of the initial values and bounds can be found in Appendix A of the Supplementary Materials.

```
estimated_sc <- estimate_process_parameters(
  data = parameter_estimation_data,
  process = "self_correcting",
```

```
  grids = grids,
  budgets = budgets,
  delta = 1,
  parallel = FALSE,
  strategy = "global_local",
  global_algorithm = "NLOPT_GN_CRS2_LM",
  local_algorithm = "NLOPT_LN_BOBYQA",
  starts = starts,
  verbose = FALSE
)

optimal_parameters <- coef(estimated_sc)
summary(estimated_sc)

#> Summary: ldmppr Fit
#>   process:         self_correcting
#>   engine:          nloptr
#>   strategy:        global_local
#>   starts:          global=1, local=1, jitter_sd=0.35, seed=90210
#>   status:          3
#>   outcome:         NLOPT_FTOL_REACHED: Optimization stopped because ftol_rel or
#>                    ftol_abs (above) was reached.
#>   objective:       432.2978
#>   selected_delta:  1
#>   elapsed_sec:     0.038
#>   coefficients:
#> [1] 4.516536e+00 2.495385e-01 8.525285e-08 1.129760e-07 1.105553e-07
#> [6] 1.095338e-07 1.067978e-07 1.205356e-01
```

We obtain the estimated parameter set ($\alpha_1 = 4.5165$, $\beta_1 = 0.2495$, $\gamma_1 = 0$, $\alpha_2 = 0$, $\beta_2 = 0$, $\alpha_3 = 0$, $\beta_3 = 0$, $\gamma_3 = 0.1205$). In practice, we recommend starting with a low-budget initial fit to gauge the need for increasing the number of iterations and reducing the fractional tolerance levels for the optimization procedure, and then adjusting these settings as needed to improve the fit.

In addition to estimating the self-correcting process, we need to train the conditional mark model for canopy volume ($m^3$) given the arrival times, locations, location-specific topographic covariates, and interpoint competition metrics in a $10m$ neighborhood. We begin by loading the example raster images and scaling them.

```
raster_paths <- list.files(system.file("extdata", package = "ldmppr"),
                           pattern = "\\.tif$", full.names = TRUE)
raster_paths <- raster_paths[grepl("_med\\.tif$", raster_paths)]
rasters <- lapply(raster_paths, terra::rast)
scaled_rasters <- scale_rasters(rasters)
```

Next, to train the mark model, we opt for a gradient boosted tree model using the **xgboost** engine with 5-fold cross-validation and a hyperparameter tuning grid of size 50. We can pass the prepared data directly from the previous step, and specify the use of parallelization to speed up the training process. We use the selection metric of mean absolute error (MAE) to identify the optimal model from the tuning grid.

```
example_trained_mark_model <- train_mark_model(
  data = estimated_sc,
  raster_list = scaled_rasters,
  scaled_rasters = TRUE,
  model_type = "xgboost",
  parallel = TRUE,
  include_comp_inds = TRUE,
  competition_radius = 10,
  edge_correction = "none",
  selection_metric = "mae",
  cv_folds = 5,
  tuning_grid_size = 50,
  verbose = FALSE
)
```

With the estimated parameters for the self-correcting process and a trained conditional mark model in hand, we now assess how well the estimated process reflects the dynamics in the original dataset using the `check_model_fit()` function. In order to obtain valid $p$-values at the $\alpha = .05$ level for the global envelope tests, Myllymäki et al. (2017) recommend using a minimum of 2499 realizations from the process, however for our initial fit we reduce the number of realizations to get an approximate sense of our performance. We pass the reference data and estimated process parameters from our estimation step through the `process_fit` argument, and the trained mark model and scaled rasters through the `mark_model` argument. We also include the competition indices in the simulations to match the training of the mark model.

```
example_model_fit <- check_model_fit(
  process = "self_correcting",
  process_fit = estimated_sc,
  mark_model = example_trained_mark_model,
  include_comp_inds = TRUE,
  thinning = TRUE,
  edge_correction = "none",
  competition_radius = 10,
  n_sim = 499,
  save_sims = FALSE,
  verbose = FALSE,
  parallel = FALSE,
  fg_correction = "km"
)
```

Once we have run the `check_model_fit()` function, we can plot the combined global envelope test, or any of the individual tests for the summary functions ($L$, $F$, $G$, $J$, $E$, or $V$), using the default `plot()` method as seen in Figure 1.

```
plot(example_model_fit)
```

In this example, our initial estimated model does not actually provide evidence of an adequate fit to the reference dataset as evidenced by the small $p$ value ($p = 0.002$) and the number of points from the reference data that fall outside of the simulated envelopes (as highlighted in red). In particular, the nearest neighbor distance distribution function, $G(r)$, from the original data is not well captured by realizations from the fitted process. The $G(r)$ function is the distribution function of the distance from an arbitrary event to its nearest neighboring event, and is commonly used to assess the degree of clustering or inhibition in a point pattern relative to complete spatial randomness (CSR) (Ripley, 1988). We also note that the function $V(r)$, which captures the variance of the mark associated with a typical random point given that another random point exists at distance $r$, is not well captured by the fitted process. This suggests that the estimated mark model may not be capturing the dynamics of the mark process well, or that the self-correcting process is not adequately capturing the spatial dynamics of the reference dataset.

This result suggests that we may need to increase the budget and strategy for our self-correcting model estimation step when using the `estimate_process_parameters()` function, and potentially expand the training regime for our conditional mark model.

Given that we obtained a less than optimal fit with our initial modeling attempt, we revisit both the self-correcting process and mark models to see if we can improve them. A standard approach for improving estimation of the self-correcting process is to increase the granularity of the grid used to estimate the model, while increasing the number of iterations and potentially reducing the tolerance threshold. We can also leverage the "`multires_global_local`" strategy to improve the estimate iteratively, while making use of multiple starts at each level to expand our search area to identify a global optimum. We refit the model with a sequence of finer grid levels and increase the budget for the optimization procedure to allow for a more thorough search of the parameter space, and reduce the tolerance levels to facilitate convergence to a more polished solution. We also introduce additional starts at each optimization level to explore the parameter space more effectively and reduce our dependence on the initial values. We maintain the same optimization strategy and algorithms as before, and use the internally derived initial values and parameter bounds based on the reference data for the optimization.

```
grids <- ldmppr_grids(
  upper_bounds = upper_bounds,
  levels = list(c(10, 10, 10),
```

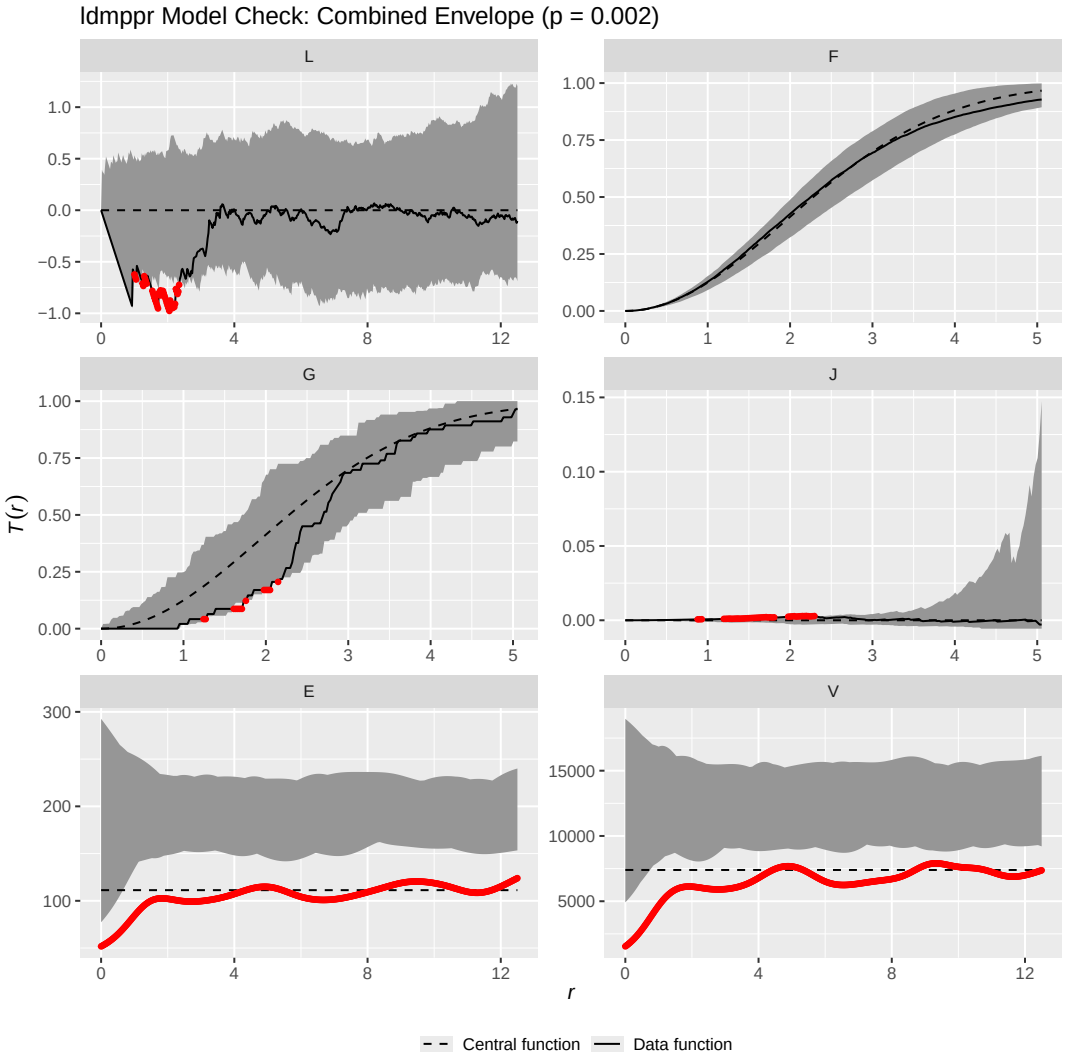



**Figure 1:** Combined global envelope test for the realizations from the fitted process. Solid black lines represent the reference process, dashed black lines represent a homogeneous Poisson process (or CSR), and the colored band represents the global envelope for the simulated datasets at the $\alpha = .05$ level. Reference values outside the envelope are highlighted in red and suggest a poor fit.

```
                c(12, 12, 12),
                c(16, 16, 16))
)

budgets <- ldmppr_budgets(
  global_options = list(
    maxeval = 1000,
    ftol_rel = 1e-6,
    xtol_rel = 1e-6
  ),
  local_budget_first_level = list(
    maxeval = 1500,
    ftol_rel = 1e-8,
    xtol_rel = 1e-8
  ),
  local_budget_refinement_levels = list(
    maxeval = 2000,
    ftol_rel = 1e-10,
    xtol_rel = 1e-10
  )

)

estimated_sc_update <- estimate_process_parameters(
  data = parameter_estimation_data,
  process = "self_correcting",
  grids = grids,
  budgets = budgets,
  delta = 1,
  parallel = FALSE,
  strategy = "multires_global_local",
  global_algorithm = "NLOPT_GN_CRS2_LM",
  local_algorithm = "NLOPT_LN_BOBYQA",
  starts = list(global = 5, local = 3, jitter_sd = 0.15, seed = 90210),
  verbose = TRUE
)

improved_optimal_parameters <- coef(estimated_sc_update)
summary(estimated_sc_update)
```

We obtain the following improved parameter estimates from the optimization.

```
improved_optimal_parameters <- if (inherits(estimated_sc_update, "ldmppr_fit")) {
  coef(estimated_sc_update)
} else {
  as.numeric(estimated_sc_update$solution)
}
summary(estimated_sc_update)

#> Summary: ldmppr Fit
#>   process:         self_correcting
#>   engine:          nloptr
#>   strategy:        multires_global_local
#>   starts:          global=5, local=3, jitter_sd=0.15, seed=90210
#>   status:          3
#>   outcome:         NLOPT_FTOL_REACHED: Optimization stopped because ftol_rel or
#>                    ftol_abs (above) was reached.
#>   objective:       387.0972
#>   selected_delta:  1
#>   elapsed_sec:     1.935
#>   coefficients:
#> [1] 1.4405824 7.3756717 0.0314348 1.6980704 0.9817515 2.4976881 1.8469612
#> [8] 0.0832517
```

We see that the improved parameter estimates set ($\alpha_1 = 1.4406$, $\beta_1 = 7.3757$, $\gamma_1 = 0.0314$, $\alpha_2 = 1.6981$, $\beta_2 = 0.9818$, $\alpha_3 = 2.4977$, $\beta_3 = 1.847$, $\gamma_3 = 0.0833$) has shifted compared to the initial estimates.

Next, we retrain the mark model using two changes. First, we increase the size of the hyperparameter tuning grid to 150 to better explore the hyperparameter space and increase the number of cross-validation folds from 5 to 10, which typically reduces the variance of performance estimates to provide a more stable estimate of model generalization.

We then retrain the mark model using the updated estimated process and adjusted model specification.

```
improved_example_trained_mark_model <- train_mark_model(
  data = estimated_sc_update,
  xy_bounds = c(0, 50, 0, 50),
  raster_list = scaled_rasters,
  scaled_rasters = TRUE,
  model_type = "xgboost",
  parallel = TRUE,
  include_comp_inds = TRUE,
  competition_radius = 10,
  edge_correction = "none",
  selection_metric = "mae",
  cv_folds = 10,
  tuning_grid_size = 150,
  verbose = TRUE
)
```

We proceed by checking the fit of our model using the improved parameter estimates for the self-correcting process and the retrained mark model. We use the same reference dataset and raster images as before, and simulate a full 2500 realizations from the process to assess the fit of the updated model.

```
improved_example_model_fit <- check_model_fit(
  process = "self_correcting",
  process_fit = estimated_sc_update,
  mark_model = improved_example_trained_mark_model,
  include_comp_inds = TRUE,
  thinning = TRUE,
  edge_correction = "none",
  competition_radius = 10,
  n_sim = 2500,
  save_sims = FALSE,
  verbose = TRUE,
  parallel = FALSE,
  fg_correction = "km"
)
```

Next, we check the combined global envelope test for the updated model to assess how well the estimated model captures the dynamics of the reference dataset.

```
plot(improved_example_model_fit)
```

We see that the improved model provides a notably improved fit to the reference dataset, as seen in Figure 2, evidenced by the fact that the simulation envelopes contain the reference pattern across all six metrics. In addition, the $p$-value for the combined global envelope test is $p = 0.2819$, indicating that the estimated process is a markedly improved fit to the reference dataset compared to the fit that we initially achieved.

When evaluating the fit of a model, in addition to the non-parametric summary statistics and global envelope tests, it may also be useful to perform a visual comparison of a realization from the model and the reference dataset. To assess the agreement between the improved fitted model and the reference data, we simulate a realization from the improved model and compare it to the original reference dataset and a dataset simulated from the initial (poorly fitting) model.

```
improved_simulated_mpp <- simulate_mpp(
```

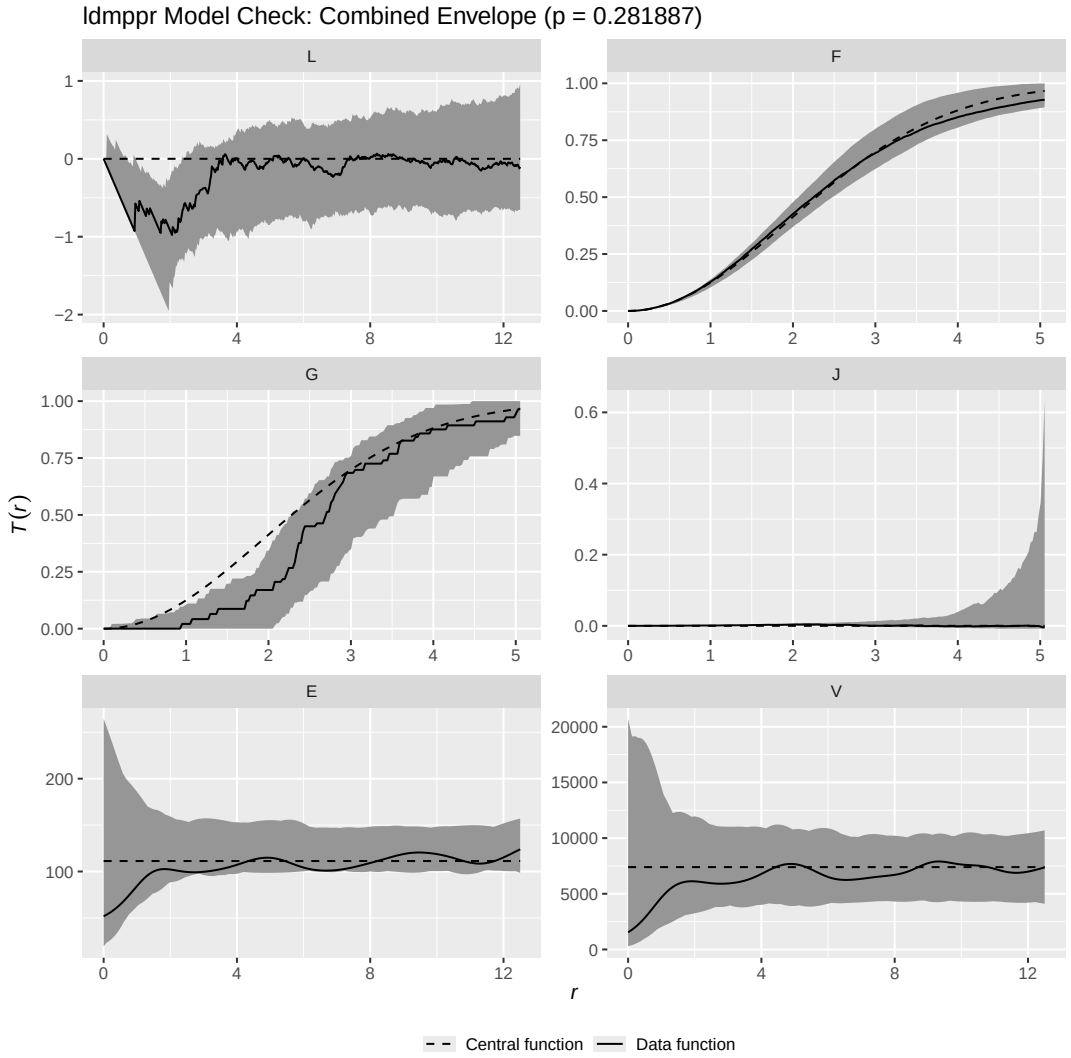



**Figure 2:** Combined global envelope test for the realizations from the improved fitted process. As in Figure 1, solid black lines represent the reference process, dashed black lines represent a homogeneous Poisson process (or CSR), and the colored band represents the global envelope for the simulated datasets at the $\alpha = .05$ level.

```
  process = "self_correcting",
  process_fit = estimated_sc_update,
  mark_model = improved_example_trained_mark_model,
  include_comp_inds = TRUE,
  competition_radius = 10,
  edge_correction = "none",
  thinning = TRUE
)

initial_simulated_mpp <- simulate_mpp(
  process = "self_correcting",
  process_fit = estimated_sc,
  mark_model = example_trained_mark_model,
  include_comp_inds = TRUE,
  competition_radius = 10,
  edge_correction = "none",
  thinning = TRUE
)

ref_plot <- plot_mpp(
  mpp_data = reference_data,
  pattern_type = "reference"
)

improved_sim_plot <- plot(
  mpp_data = improved_simulated_mpp,
  pattern_type = "simulated"
)

initial_sim_plot <- plot(
  mpp_data = initial_simulated_mpp,
  pattern_type = "simulated"
)
```

Figure 3 demonstrates that the improved model provides a more accurate representation of the reference dataset than we obtained from the initial model in terms of the spatial process and the conditional mark process. We also include histograms of the realized mark distributions to highlight the improvement in replicating the mark distribution of the reference process. This visualization provides additional evidence that the improved model captures the spatial and mark dynamics of the reference dataset more effectively, which results in the improved model being a better fit. This example highlights the utility of the package for working with a real marked point process of interest, and provides the intuition for using the main functionality of **ldmppr** to estimate, assess model fit, and simulate from a marked spatial point process.

We note that the computational cost of the workflow is driven primarily by three steps: self-correcting process estimation, mark-model training with cross-validation/tuning, and goodness-of-fit simulation in `check_model_fit()`. Runtime increases with grid resolution, optimizer budgets, number of starts, tuning grid size, cross-validation folds, and the number of simulations (`n_sim`). A practical approach is to begin with a low-cost baseline (coarser grid, fewer starts, smaller tuning grid, moderate `n_sim`) and increase complexity only when diagnostics indicate the need for refinement. Parallelization can also reduce wall-clock time for multi-start estimation, model tuning, and simulation-based checks when working with larger datasets, though the benefits are less pronounced for small datasets or lower budget runs. Concrete benchmark tables are provided in Appendices B and C of the Supplementary Materials for various usage scenarios, and Appendix D provides a contextual side-by-side comparison against a two-stage **spatstat** derived workflow under matched tuning/check settings.

## 5 Discussion

In this paper, we provide a novel framework for estimating location dependent marked point processes and introduce the **ldmppr** package, which contains a user-friendly modular suite of tools for model estimation, evaluation, simulation, and visualization for marked point processes with location dependence characterized by regularity in the spatial pattern. We outlined the typical workflow for using the package and discussed the key functions and their arguments in detail before providing an example of using the package with a real forestry dataset. **ldmppr** simplifies the process of working with marked

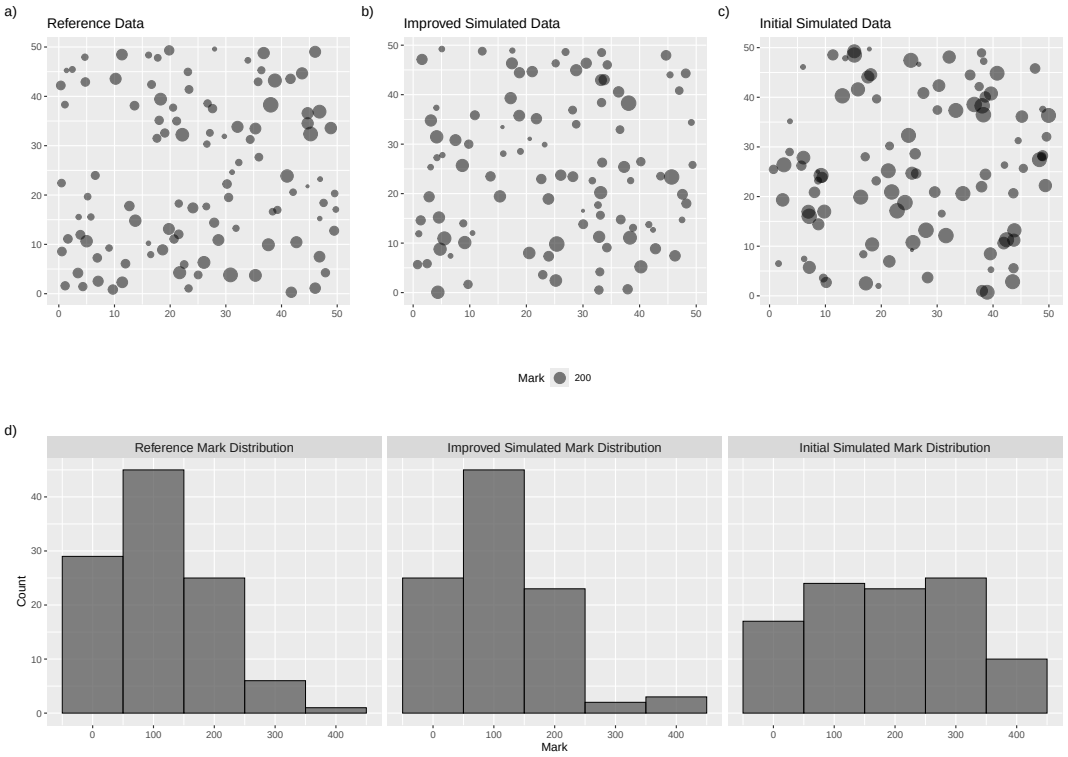



**Figure 3:** Plots a) - c) provide a comparison of a realization from the improved estimated process with the original reference dataset and a realization from the initial estimated process. Plot d) shows the corresponding observed mark distributions from the reference dataset and the simulation realizations.

point processes and provides a likelihood-based estimation approach that is computationally feasible. While the framework presented applies to a broader range of marked point process, the package in its current implementation is still somewhat limited in the types of patterns that it can address. As the package continues to develop, we would like to incorporate additional models that can address point process data that demonstrate clustering behavior, as opposed to regularity, while still maintaining the focus on location dependent marks.

## References

A. Baddeley and R. Turner. spatstat: An R package for analyzing spatial point patterns. *Journal of Statistical Software*, 12(6):1–42, 2005. doi: 10.18637/jss.v012.i06. [p144]

A. Baddeley, E. Rubak, and R. Turner. *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC, New York, Nov. 2015. ISBN 978-0-429-16170-4. doi: 10.1201/b19708. [p153]

F. L. Bayisa, M. Ådahl, P. Rydén, and O. Cronie. Regularised Semi-parametric Composite Likelihood Intensity Modelling of a Swedish Spatial Ambulance Call Point Pattern. *Journal of Agricultural, Biological and Environmental Statistics*, 28(4):664–683, Dec. 2023. ISSN 1537-2693. doi: 10.1007/s13253-023-00534-5. [p144]

L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, Oct. 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324. [p147]

T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, Aug. 2016. Association for Computing Machinery. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. [p148]

T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou, M. Li, J. Xie, M. Lin, Y. Geng, Y. Li, J. Yuan, and D. Cortes. *xgboost: Extreme Gradient Boosting*, 2026. [p153]

M. A. Contreras, D. Affleck, and W. Chung. Evaluating tree competition indices as predictors of basal area increment in western Montana forests. *Forest Ecology and Management*, 262(11):1939–1949, Dec. 2011. ISSN 0378-1127. doi: 10.1016/j.foreco.2011.08.031. [p153]

D. J. Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes*. Springer, New York, NY, 1 edition, 1988. ISBN 978-0-387-96666-3. URL https://link.springer.com/book/9780387966663. [p146]

P. J. Diggle. *Statistical Analysis of Spatial and Spatio-Temporal Point Patterns*. Chapman and Hall/CRC, New York, 3 edition, July 2013. ISBN 978-0-429-09809-3. doi: 10.1201/b15326. [p146, 153]

P. J. Diggle and R. J. Gratton. Monte Carlo Methods of Inference for Implicit Statistical Models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 46(2):193–212, Jan. 1984. ISSN 0035-9246. doi: 10.1111/j.2517-6161.1984.tb01290.x. [p147]

L. Drew and A. Kaplan. *ldmppr: Estimate and Simulate from Location Dependent Marked Point Processes*, 2025. [p144]

L. Drew, A. Kaplan, and I. Breckheimer. Data from "A Bayesian Record Linkage Approach to Applications in Tree Demography Using Overlapping LiDAR Scans", 2024. [p155]

L. Drew, A. Kaplan, and I. Breckheimer. A Bayesian Record Linkage Approach to Applications in Tree Demography Using Overlapping LiDAR Scans. *The Annals of Applied Statistics*, 0(0):1–26, June 2025. URL https://arxiv.org/search/?query=A+Bayesian+Record+Linkage+Approach+to+Applications+in+Tree+Demography+Using+Overlapping+LiDAR+Scans&searchtype=title. [p156]

D. S. Harte. PtProcess: An R package for modelling marked point processes indexed by time. *Journal of Statistical Software*, 35(8):1–32, 2010. doi: 10.18637/jss.v035.i08. [p144]

V. Isham and M. Westcott. A self-correcting point process. *Stochastic Processes and their Applications*, 8 (3):335–347, May 1979. ISSN 0304-4149. doi: 10.1016/0304-4149(79)90008-5. [p144, 147]

S. G. Johnson. *The NLopt Nonlinear-Optimization Package*, 2008. URL https://nlopt.readthedocs.io/. [p151]

P. Kaelo and M. M. Ali. Some Variants of the Controlled Random Search Algorithm for Global Optimization. *Journal of Optimization Theory and Applications*, 130(2):253–264, Aug. 2006. ISSN 1573-2878. doi: 10.1007/s10957-006-9101-0. [p151]

G. O. Mohler, M. B. Short, P. J. Brantingham, F. P. Schoenberg, and G. E. Tita. Self-Exciting Point Process Modeling of Crime. *Journal of the American Statistical Association*, 106(493):100–108, Mar. 2011. ISSN 0162-1459. doi: 10.1198/jasa.2011.ap09546. [p144]

J. Møller and R. P. Waagepetersen. *Statistical Inference and Simulation for Spatial Point Processes*. Chapman and Hall/CRC, New York, Sept. 2003. ISBN 978-0-203-49693-0. doi: 10.1201/9780203496930. [p145, 154]

J. Møller, M. Ghorbani, and E. Rubak. Mechanistic spatio-temporal point process models for marked point processes, with a view to forest stand data. *Biometrics*, 72(3):687–696, 2016. ISSN 1541-0420. doi: 10.1111/biom.12466. [p144, 145, 146, 147, 148]

M. Myllymäki, T. Mrkvička, P. Grabarnik, H. Seijo, and U. Hahn. Global envelope tests for spatial processes. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 79(2):381–404, 2017. ISSN 1369-7412. doi: 10.1111/rssb.12172. [p153, 158]

J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4): 308–313, Jan. 1965. ISSN 0010-4620, 1460-2067. doi: 10.1093/comjnl/7.4.308. [p151]

A. Pommerening and A. J. Sánchez Meador. Tamm review: Tree interactions between myth and reality. *Forest Ecology and Management*, 424:164–176, Sept. 2018. ISSN 0378-1127. doi: 10.1016/j.foreco.2018.04.051. [p153]

M. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. *Technical Report, Department of Applied Mathematics and Theoretical Physics*, Jan. 2009. URL https://optimization-online.org/2010/05/2616/. [p151]

W. L. Price. Global optimization by controlled random search. *Journal of Optimization Theory and Applications*, 40(3):333–348, July 1983. ISSN 1573-2878. doi: 10.1007/BF00933504. [p151]

S. L. Rathbun and N. Cressie. A space-time survival point process for a longleaf pine forest in southern georgia. *Journal of the American Statistical Association*, 89(428):1164–1174, 1994. ISSN 0162-1459. doi: 10.2307/2290979. [p146]

B. D. Ripley. The second-order analysis of stationary point processes. *Journal of Applied Probability*, 13 (2):255–266, June 1976. ISSN 0021-9002, 1475-6072. doi: 10.2307/3212829. [p153]

B. D. Ripley. *Statistical Inference for Spatial Processes*. Cambridge University Press, Cambridge, 1988. ISBN 978-0-521-42420-2. doi: 10.1017/CBO9780511624131. [p158]

T. H. Rowan. *Functional Stability Analysis of Numerical Algorithms*. PhD thesis, University of Texas at Austin, USA, 1990. URL https://search.worldcat.org/search?q=Functional+Stability+Analysis+of+Numerical+Algorithms+Rowan. [p151]

M. Schlather, P. J. Ribeiro, and P. J. Diggle. Detecting Dependence between Marks and Locations of Marked Point Processes. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 66(1): 79–93, 2004. ISSN 1369-7412. doi: 10.1046/j.1369-7412.2003.05343.x. [p145, 153, 154]

M. N. M. van Lieshout. A J-Function for Marked Point Patterns. *Annals of the Institute of Statistical Mathematics*, 58(2):235–259, June 2006. ISSN 1572-9052. doi: 10.1007/s10463-005-0015-7. [p153]

M. N. Wright and A. Ziegler. ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77(1):1–17, 2017. doi: 10.18637/jss.v077.i01. [p153]

*Lane Drew*
*Colorado State University*
*Department of Statistics, Colorado State University*
*851 Oval Dr, Fort Collins, CO 80524*
https://lanedrew.com/
*ORCID: 0009-0006-5427-4092*
lane.drew@colostate.edu

*Andee Kaplan*
*Colorado State University*
*Department of Statistics, Colorado State University*
*851 Oval Dr, Fort Collins, CO 80524*
https://andeekaplan.com/
*ORCID: 0000-0002-0912-0225*
andee.kaplan@colostate.edu

# mtarm: Bayesian Analysis of Multivariate Threshold Autoregressive Models in R

*by L.H. Vanegas, S.A. Calderón V. and L.M. Rondón*

**Abstract** This paper introduces mtarm, an R package for Bayesian estimation, inference, and forecasting in multivariate Threshold Autoregressive (TAR) models. The package supports both standard m-step-ahead forecasting from a single model fit and rolling-origin forecast evaluation without repeated model re-estimation. These models provide a flexible framework for analyzing nonlinear multivariate time series and encompass important special cases such as multivariate Self-Exciting Threshold Autoregressive (SETAR) models and Vector Autoregressive (VAR) models. The package supports a broad class of innovation distributions beyond the Gaussian assumption, including the Student-t, slash, symmetric hyperbolic, Laplace, contaminated normal, skew-normal, and skew-t distributions, thereby accommodating heavy tails, skewness, and other forms of non-normality frequently observed in practice. Inference is performed within a Bayesian framework using a Markov chain Monte Carlo (MCMC) algorithm that jointly estimates all model parameters, except for the number of regimes. The resulting posterior samples can be readily summarized, visualized, and diagnosed through integration with the coda package. For model assessment and selection, mtarm implements several predictive and information-based criteria, including the Deviance Information Criterion (DIC), the Watanabe–Akaike Information Criterion (WAIC), the log-score, the Continuous Ranked Probability Score (CRPS), the Energy Score (ES), and a variety of forecast accuracy measures. Additional functionality includes graphical goodness-of-fit diagnostics and simulation of multivariate TAR processes. The capabilities of mtarm are illustrated through the analysis and forecasting of two real multivariate time series.

## 1 Introduction

Threshold Autoregressive (TAR) models were originally introduced by Tong (1978, 1983) and have become one of the most widely used frameworks for modeling nonlinear dynamics in time series. By allowing the autoregressive structure to vary across regimes defined by threshold variables, TAR models can effectively capture structural changes, asymmetric adjustments, and regime-switching behavior. In contrast, linear autoregressive (AR) models—obtained as a special case of a TAR model with a single regime—are generally unable to adequately represent such features. Owing to their flexibility, TAR models have been successfully applied in a wide range of fields. In economics, they have been used to model gross national product (GNP) growth, unemployment, and inflation rates; in finance, to analyze stock returns and insurance pricing; in hydrology, to describe river flow dynamics; and in ecology, to study annual records of lynx trapping, among many other applications (see Tong (1990); Hansen (2011); Chen et al. (2011); Tong (2011, 2015)).

Within the TAR family, several important extensions have been developed. In the univariate setting, these include the incorporation of moving average components through Threshold Autoregressive Moving Average (TARMA) models (Tong, 1990), the introduction of conditional heteroscedasticity via Threshold Autoregressive Conditional Heteroscedastic (TAR-ARCH) models (Liu et al., 1997), and recent advances in testing for threshold effects based on supremum Lagrange multiplier procedures (Goracci et al., 2023; Giannerini et al., 2024). In the multivariate context, Tsay (1998) introduced the Multivariate Threshold Autoregressive (MTAR) model, which was subsequently extended to the Threshold Vector Autoregressive Moving Average (TVARMA) framework by Niglio and Vitale (2015). More recently, Vanegas et al. (2025) further generalized the TAR models by allowing for non-Gaussian error distributions.

Despite the theoretical importance and wide applicability of TAR-type models, their statistical implementation in R remains relatively limited. For example, the package **tsDyn** (Stigler, 2018) provides a well-established framework for nonlinear time series analysis, including Self-Exciting Threshold Autoregressive (SETAR) models. Estimation is based on conditional least squares, a frequentist approach that typically assumes Gaussian innovations and determines threshold values through grid-search optimization, which can become computationally demanding in the presence of multiple thresholds or high-dimensional settings. Likewise, the package **NTS** (Liu et al., 2020) supports the estimation of multivariate TAR models and includes tools for simulation, prediction, and model identification. However, its methodology is also rooted in a frequentist framework, relies on Gaussian innovation assumptions, and offers limited flexibility for accommodating alternative error distributions. More recently, the package **tseriesTARMA** (Giannerini and Goracci, 2024) expanded the range of available nonlinear time series tools by implementing threshold ARMA models under a frequentist paradigm. In

contrast, the package **BAYSTAR** (Chen et al., 2022) adopts a Bayesian perspective, providing inference for univariate TAR models with flexible prior specifications. Nevertheless, its scope is restricted to the univariate case.

However, existing software solutions remain fragmented, typically focusing exclusively on either univariate or multivariate time series. Moreover, most available implementations are restricted to two-regime models and do not naturally accommodate more complex multi-regime structures. To the best of our knowledge, the **NTS** package currently provides the most comprehensive framework for handling both univariate and multivariate TAR models with multiple regimes. However, its inferential procedures are limited by Gaussian innovation assumptions and do not support the broad class of non-Gaussian error distributions frequently encountered in practice. Such distributions arise naturally in applications involving financial returns, river flows, unemployment rates, and many other phenomena characterized by nonlinear and regime-dependent dynamics. As a result, there is a growing demand for a unified statistical framework that can simultaneously accommodate univariate and multivariate TAR models, allow for multiple regimes, and provide the flexibility to model non-Gaussian innovations. Such a framework would be particularly valuable in fields such as finance, economics, hydrology, and other disciplines where departures from normality and regime-switching behavior are common.

To address these limitations, we introduce **mtarm**, a new `R` package for the Bayesian analysis of multivariate TAR-type models. The package is built upon the methodology developed by Vanegas et al. (2025) and extends the current `R` ecosystem by providing a flexible and unified framework for Bayesian inference in TAR models. Specifically, **mtarm** contributes in the following ways:

- **Unified modeling framework.** The package supports the estimation of multivariate TAR models and several important special cases, including Self-Exciting Threshold Autoregressive (SETAR) models and Vector Autoregressive (VAR) models, together with their univariate counterparts. This unified framework facilitates model comparison and allows users to seamlessly transition between linear and nonlinear specifications.
- **Flexible innovation distributions.** In contrast to most existing implementations, which are restricted to Gaussian innovations, **mtarm** accommodates a broad class of multivariate distributions, including the Student-$t$, slash, symmetric hyperbolic, Laplace, contaminated normal, skew-normal, and skew-$t$ distributions. These distributions provide increased robustness to heavy tails, skewness, and other forms of non-normality commonly observed in real-world applications.
- **Bayesian inference and forecasting.** All aspects of estimation, inference, and prediction are carried out within a Bayesian framework. Model parameters are estimated jointly through a Markov chain Monte Carlo (MCMC) algorithm, eliminating the need for the sequential grid-search procedures commonly employed in frequentist approaches. This strategy yields a coherent quantification of uncertainty through posterior distributions and naturally supports probabilistic forecasting. The package supports both standard $m$-step-ahead forecasting from a single model fit and rolling-origin forecast evaluation without repeated model re-estimation.

The remainder of the paper is organized as follows. Section 2 presents the statistical framework for multivariate TAR models and summarizes the Bayesian methodology implemented in **mtarm**. Section 3 describes the package architecture and its main functionalities through an empirical application involving rainfall and the flows of two rivers in Colombia. Section 4 further illustrates the use of the package through a second empirical application involving precipitation, temperature, and the flows of two rivers in Iceland.

## 2 Theoretical framework

### 2.1 Model formulation

Let $\{\boldsymbol{Y}_t\}_{t\geq 1}$ be the $k$-dimensional time series of interest (output), $\{\boldsymbol{X}_t\}_{t\geq 1}$ the $r$-dimensional time series of exogenous covariates, and $\{Z_t\}_{t\geq 1}$ the one-dimensional threshold time series. We say that $\{\boldsymbol{Y}_t\}_{t\geq 1}$ follows a multivariate Threshold Autoregressive (TAR) model with $l$ regimes, denoted TAR$(l; \boldsymbol{p}, \boldsymbol{q}, \boldsymbol{d})$, if

$$\boldsymbol{Y}_t = \sum_{j=1}^{l} I(Z_{t-h} \in (c_{j-1}, c_j]) \Big( \boldsymbol{\phi}_0^{(j)} \boldsymbol{D}_t + \sum_{i=1}^{p_j} \boldsymbol{\phi}_i^{(j)} \boldsymbol{Y}_{t-i} + \sum_{i=1}^{q_j} \boldsymbol{\beta}_i^{(j)} \boldsymbol{X}_{t-i} + \sum_{i=1}^{d_j} \boldsymbol{\delta}_i^{(j)} Z_{t-i} + \boldsymbol{\epsilon}_{t_j} \Big), \tag{1}$$

where $\boldsymbol{p} = (p_1, \ldots, p_l)$, $\boldsymbol{q} = (q_1, \ldots, q_l)$ and $\boldsymbol{d} = (d_1, \ldots, d_l)$ are vectors of non-negative integers specifying the regime-dependent lag orders. Specifically, in Regime $j$, $p_j$ denotes the autoregressive order of the output series $\{\boldsymbol{Y}_t\}_{t\geq 1}$, $q_j$ denotes the maximum lag of the exogenous series $\{\boldsymbol{X}_t\}_{t\geq 1}$,

and $d_j$ denotes the maximum lag of the threshold series $\{Z_t\}_{t\geq 1}$. The parameter $h \geq 0$ is the delay parameter, while $\boldsymbol{c} = (c_1, \ldots, c_{l-1})$ is the vector of threshold values satisfying $-\infty = c_0 < c_1 < \cdots < c_{l-1} < c_l = \infty$. For each regime ($j = 1, \ldots, l$), $\boldsymbol{\phi}_0^{(j)}$ is the coefficient matrix associated with the deterministic component of the model, which can include an intercept, a linear or quadratic time trend, and seasonal effects. The matrices $\boldsymbol{\phi}_i^{(j)}$, $\boldsymbol{\beta}_i^{(j)}$, and $\boldsymbol{\delta}_i^{(j)}$ denote the coefficients associated with the $i$-th lag of the response process $\{\boldsymbol{Y}_t\}_{t\geq 1}$, the exogenous process $\{\boldsymbol{X}_t\}_{t\geq 1}$, and the threshold process $\{Z_t\}_{t\geq 1}$, respectively. The vector of deterministic regressors is given by $\boldsymbol{D}_t = (1, a(t), b(t))^\top$, where $a(t) = t$ for a linear time trend and $a(t) = (t, t^2)$ for a quadratic time trend. The seasonal component $b(t)$ corresponds to the $((t \bmod n_s) + 1)$-th row of $\tilde{\boldsymbol{I}}_{n_s}$, where $n_s \geq 2$ denotes the number of seasonal periods, $a \bmod b$ is the remainder obtained when dividing the integer $a$ by the integer $b$, and $\tilde{\boldsymbol{I}}_{n_s}$ is the $n_s \times n_s$ identity matrix with its first column removed. Finally, for $t = 1, 2, \ldots$ and $j = 1, \ldots, l$, the innovation vectors $\boldsymbol{\epsilon}_{t_j}$ are assumed to be independent $k$-dimensional random variables with location parameter $\boldsymbol{0}$, regime-specific scale matrix $\boldsymbol{\Sigma}_j$, and, depending on the chosen distribution, additional unknown parameters that are common across regimes and time. Furthermore, the innovations are assumed to be mutually independent and independent of both the exogenous process $\{\boldsymbol{X}_t\}_{t\geq 1}$ and the threshold process $\{Z_t\}_{t\geq 1}$.

From the perspective of nonlinear dynamical systems, a multivariate TAR model can be interpreted as a stochastic piecewise-linear approximation to an unknown nonlinear vector process. Within each regime, the dynamics are governed by a linear VAR specification with regime-specific coefficients and innovation covariance matrices. However, the governing dynamics change whenever the threshold variable crosses a regime boundary, producing a globally nonlinear process in the sense of Tong's threshold dynamical-systems framework (Tong, 1990). This regime-dependent structure enables TAR models to capture features that are difficult to represent within a conventional linear VAR framework, including state-dependent persistence, regime-specific volatility patterns, asymmetric responses to shocks, and nonlinear impulse-response behavior. Moreover, threshold dynamics can generate richer forms of temporal dependence, such as limit cycles, time irreversibility, and non-Gaussian—potentially multimodal—stationary distributions. The flexibility of TAR models has led to their successful application in a wide range of fields. In economics, they have been used to study business-cycle asymmetries and nonlinear output dynamics (Potter, 1995); in finance, to model nonlinear behavior in monetary and asset-price processes, including speculative bubble dynamics (Tsay, 1998; Grynkiv and Stentoft, 2018); and in environmental sciences, to characterize complex hydrological processes such as river-flow dynamics. These applications highlight the value of TAR models as a flexible framework for representing nonlinear and regime-dependent behavior in multivariate systems.

When the autoregressive order of $\{\boldsymbol{Y}_t\}_{t\geq 1}$ and the maximum lags orders of $\{\boldsymbol{X}_t\}_{t\geq 1}$ and $\{Z_t\}_{t\geq 1}$ are unknown, model selection can be performed using the Stochastic Search Variable Selection (SSVS) methodology of George and McCulloch (1993, 1995, 1997). Under this approach, a vector of binary inclusion indicators, $\boldsymbol{\zeta}_j = (\zeta_{j,1}, \ldots, \zeta_{j,p_j}, \zeta_{j,p_j+1}, \ldots, \zeta_{j,p_j+q_j}, \zeta_{j,p_j+q_j+1}, \ldots, \zeta_{j,p_j+q_j+d_j})$, is introduced for each regime, where $\zeta_{j,i} \in \{0, 1\}$ for each $j = 1, \ldots, l$ and $i = 1, \ldots, p_j, p_j + 1, \ldots, p_j + q_j, p_j + q_j + 1, \ldots, p_j + q_j + d_j$. The indicators determine whether the corresponding lagged effects are included in the model. For the autoregressive component, $\zeta_{j,i} = 0$ implies that $\boldsymbol{\phi}_i^{(j)} = \boldsymbol{0}$, and therefore the $i$-th lag of $\{\boldsymbol{Y}_t\}_{t\geq 1}$ is excluded from Regime $j$. Conversely, $\zeta_{j,i} = 1$ implies that $\boldsymbol{\phi}_i^{(j)}$ is unrestricted and estimated as part of the MCMC algorithm. Analogously, for the exogenous component, $\zeta_{j,p_j+i} = 0$ implies $\boldsymbol{\beta}_i^{(j)} = \boldsymbol{0}$, whereas $\zeta_{j,p_j+i} = 1$ allows $\boldsymbol{\beta}_i^{(j)}$ to be estimated. Likewise, for the threshold component, $\zeta_{j,p_j+q_j+i} = 0$ implies $\boldsymbol{\delta}_i^{(j)} = \boldsymbol{0}$, while $\zeta_{j,p_j+q_j+i} = 1$ permits $\boldsymbol{\delta}_i^{(j)}$ to vary freely within the posterior sampling scheme. Consequently, the posterior distribution of $\boldsymbol{\zeta}_j$ provides a mechanism for identifying the relevant lagged effects of the response, exogenous, and threshold processes within each regime. In this way, the SSVS procedure simultaneously performs parameter estimation and regime-specific lag selection within the Bayesian framework.

Special cases of the model (1) include multivariate Self-Exciting Threshold Autoregressive (SETAR) models when $h > 0$ and $Z_t \equiv Y_{m,t}$ for some $m \in \{1, \ldots, k\}$ and all $t$, where $Y_{m,t}$ represents the $m$-th component of $\boldsymbol{Y}_t = (Y_{1,t}, \ldots, Y_{k,t})^\top$; and Vector Autoregressive (VAR) models when $l = 1$.

### 2.2 Noise process distribution

This section presents two representative innovation distributions implemented in **mtarm** for modeling the error process in model (1); the remaining distributions are described in Appendix .1. Throughout this section, $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_k)^\top$ and $\boldsymbol{\Sigma}$ denote the location vector and the positive-definite $k \times k$ scale matrix, respectively. Most of the innovation distributions implemented in **mtarm** belong to the class of Gaussian scale mixtures (McNeil et al., 2015, section 6.2), a broad family that includes the Gaussian distribution as a special (degenerate) case. Relative to the Gaussian model, these distributions are capable of accommodating heavier tails and, in some cases, a greater concentration of probability mass

around the center of the distribution. Consequently, they offer increased robustness to outliers and extreme observations while preserving a convenient hierarchical structure that facilitates Bayesian estimation and posterior computation.

- **Student-$t$ distribution**. If $\boldsymbol{Y} \sim \text{Student-}t_k(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu)$ then its probability density function is given by (McNeil et al. (2015, example 6.7); Fang et al. (2018, page 85))

$$f_Y(\boldsymbol{y}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu) = \frac{\Gamma(\frac{\nu+k}{2})}{\Gamma(\frac{\nu}{2})(\nu\pi)^{\frac{k}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}}\left(1 + \frac{1}{\nu}(\boldsymbol{y}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})\right)^{-\frac{\nu+k}{2}}, \qquad \nu > 0.$$

  The multivariate Student-$t$ distribution provides a flexible alternative to the Gaussian specification for modeling innovations in multivariate time series. Owing to its heavy-tailed nature, it can accommodate extreme observations more effectively than the Gaussian model, reducing the influence of outliers and improving robustness to departures from distributional assumptions (Lange et al., 1989; Abanto-Valle et al., 2010). The degree of robustness is governed by the degrees-of-freedom parameter, which controls tail heaviness and induces a smooth transition to the Gaussian distribution as its value increases (Kotz and Nadarajah, 2004). In addition to its robustness properties, the multivariate Student-$t$ distribution retains many of the attractive features of the multivariate normal distribution, including closure under linear transformations and a convenient hierarchical representation that facilitates Bayesian inference. At the same time, it offers greater flexibility for capturing heavy-tailed behavior and other forms of non-Gaussian dependence commonly encountered in practice (Abanto-Valle et al., 2012). These characteristics make the multivariate Student-$t$ distribution particularly appealing for multivariate TAR models and other nonlinear regime-switching frameworks, where deviations from normality are often present.

- **Skew-$t$ distribution**. If $\boldsymbol{Y} \sim \text{S}t_k(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\lambda}, \nu)$ then its probability density function is given by (Sahu et al., 2003, page 136)

$$f_Y(\boldsymbol{y}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\lambda}, \nu) = 2^k \frac{\Gamma\left(\frac{\nu+k}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)(\nu\pi)^{\frac{k}{2}}|\boldsymbol{\Sigma} + \boldsymbol{D}_{(\lambda)}^2|^{\frac{1}{2}}}\left(1 + \frac{(\boldsymbol{y}-\boldsymbol{\mu})^\top(\boldsymbol{\Sigma} + \boldsymbol{D}_{(\lambda)}^2)^{-1}(\boldsymbol{y}-\boldsymbol{\mu})}{\nu}\right)^{-\frac{\nu+k}{2}} \Pr(\boldsymbol{V} > 0),$$

  where

$$\boldsymbol{V} \sim \text{Student-}t_k\Big(\boldsymbol{D}_{(\lambda)}(\boldsymbol{\Sigma} + \boldsymbol{D}_{(\lambda)}^2)^{-1}(\boldsymbol{y}-\boldsymbol{\mu}), \frac{\nu + (\boldsymbol{y}-\boldsymbol{\mu})^\top(\boldsymbol{\Sigma}+\boldsymbol{D}_{(\lambda)}^2)^{-1}(\boldsymbol{y}-\boldsymbol{\mu})}{\nu+k}\big(\boldsymbol{I}_k - \boldsymbol{D}_{(\lambda)}(\boldsymbol{\Sigma} + \boldsymbol{D}_{(\lambda)}^2)^{-1}\boldsymbol{D}_{(\lambda)}\big), \nu+k\Big).$$

In this parametrization, the vector $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_k)^\top$ acts as a directional skewness parameter that determines both the magnitude and orientation of departures from symmetry, while the degrees-of-freedom parameter ($\nu > 0$) controls tail heaviness. When $\boldsymbol{\lambda} = \boldsymbol{0}$, the distribution reduces to the multivariate Student-$t$ distribution with $\nu$ degrees of freedom; conversely, as $\nu \to \infty$, it converges to the multivariate skew-normal distribution (Sahu et al., 2003, page 133). The multivariate skew-$t$ distribution may therefore be viewed as a flexible extension of both the Student-$t$ and skew-normal families, combining heavy-tailed behavior with directional asymmetry. The parameter $\boldsymbol{\lambda}$ allows the distribution to capture systematic departures from symmetry, while $\nu$ regulates the probability of extreme observations. This dual capacity to model skewness and heavy tails makes the skew-$t$ distribution particularly attractive for time-series applications in which innovations exhibit asymmetric behavior and occasional large shocks. Such features are frequently encountered in financial, economic, and environmental data, as well as in stochastic-volatility models and Bayesian VAR frameworks with non-Gaussian innovations (Abanto-Valle et al., 2015; Karlsson et al., 2023).

### 2.3 Prior distributions

The value of $l$ is assumed to be known. The model parameters are $h$, $\boldsymbol{c} = (c_1, \ldots, c_{l-1})$, $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_l$, $\boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_l, \boldsymbol{\zeta}_1, \ldots, \boldsymbol{\zeta}_l, \boldsymbol{\lambda}$ and $\nu$, where $\boldsymbol{\theta}_j = (\boldsymbol{\phi}_0^{(j)}, \boldsymbol{\phi}_1^{(j)}, \ldots, \boldsymbol{\phi}_{p_j}^{(j)}, \boldsymbol{\beta}_1^{(j)}, \ldots, \boldsymbol{\beta}_{q_j}^{(j)}, \delta_1^{(j)}, \ldots, \delta_{d_j}^{(j)})^\top$ is the regression parameter matrix in Regime $j$, whose dimension is $s_j \times k$, in which $s_j = 1 + (p_j \times k) + (q_j \times r) + d_j$. Henceforth, we will refer to $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_l$ and $\boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_l$ as location and scale parameters, respectively. The prior distribution is the following

$$\pi(h, \boldsymbol{c}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_l, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_l, \boldsymbol{\zeta}_1, \ldots, \boldsymbol{\zeta}_l, \boldsymbol{\lambda}, \nu) = \pi(h)\pi(\boldsymbol{c})\pi(\nu)\pi(\boldsymbol{\lambda})\prod_{j=1}^{l}\pi(\boldsymbol{\theta}_j|\boldsymbol{\Sigma}_j, \boldsymbol{\zeta}_j)\pi(\boldsymbol{\Sigma}_j)\pi(\boldsymbol{\zeta}_j),$$

where

(1) For the threshold vector we take

$$\pi(\boldsymbol{c}) \propto \begin{cases} 1 & \text{if} \quad q_z(\alpha_0) < c_1 < \ldots < c_{l-1} < q_z(\alpha_1), \\ 0 & \text{otherwise,} \end{cases}$$

in which $q_z(\alpha)$ represents the $100(\alpha)\%$ percentile of the realization of $\{Z_t\}_{t\geq 1}$, and $0 \leq \alpha_0 < \alpha_1 \leq 1$. This prior is uniform over ordered thresholds constrained to lie between percentiles of the threshold variable, thereby ensuring that the regime boundaries remain within a data-relevant region while excluding implausible configurations corresponding to extreme or nearly empty regimes;

(2) For the delay parameter we take

$$\pi(h) \propto I\{h_{\min}, \ldots, h_{\max}\},$$

in which $h_{\min}$ and $h_{\max}$ are integer values such that $0 \leq h_{\min} \leq h_{\max}$. This specification induces a discrete uniform prior over a restricted set of plausible delays, assigning equal prior probability to all candidate values within the range while excluding lag lengths that are empirically unrealistic or theoretically implausible;

(3) For the regime-specific scale matrices we assume

$$\pi(\boldsymbol{\Sigma}_j) = \mathrm{W}^{-1}(\boldsymbol{\Omega}_{0j}, \tau_{0j}), \qquad j = 1, \ldots, l,$$

in which $\mathrm{W}^{-1}(\boldsymbol{\Omega}, \tau)$ represents the inverse Wishart distribution (see, for instance, Gupta and Nagar (1999, Section 3.4)), with $\boldsymbol{\Omega}$ a $k \times k$ positive-definite matrix and $\tau > k - 1$. This prior regularizes the regime-specific covariance matrices by shrinking $\boldsymbol{\Sigma}_j$ toward $\boldsymbol{\Omega}_{0j}/(\tau_{0j} - k - 1)$, so that $\boldsymbol{\Omega}_{0j}$ encodes a plausible volatility–correlation structure for Regime $j$, while $\tau_{0j}$ determines the strength of the shrinkage. Moreover, the specification preserves conjugacy with the likelihood, thereby facilitating posterior simulation of $\boldsymbol{\Sigma}_j$ and contributing to computational efficiency;

(4) For the vector of inclusion indicators in Regime $j$, $\boldsymbol{\zeta}_j = (\zeta_{j,1}, \ldots, \zeta_{j,p_j+q_j+d_j})^\top$, we assume

$$\pi(\boldsymbol{\zeta}_j) = \prod_{i=1}^{p_j+q_j+d_j} \rho_{0j}^{\zeta_{j,i}} (1-\rho_{0j})^{1-\zeta_{j,i}}, \qquad j = 1, \ldots, l,$$

with $\rho_{0j} \in (0, 1)$. This specification corresponds to independent Bernoulli priors with a common inclusion probability $\rho_{0j}$, so that $\rho_{0j}$ directly governs the expected proportion of active coefficients, and thus the overall sparsity level in Regime $j$;

(5) $\pi(\boldsymbol{\theta}_j \mid \boldsymbol{\Sigma}_j, \boldsymbol{\zeta}_j)$ is $\mathrm{MN}_{s_{j\zeta},k}(\boldsymbol{\mu}_{0j\zeta}, \boldsymbol{\Delta}_{0j\zeta}, \boldsymbol{\Sigma}_j)$ for $j = 1, \ldots, l$, in which

$$s_{j\zeta} = 1 + k \sum_{i=1}^{p_j} \zeta_{j,i} + r \sum_{i=1}^{q_j} \zeta_{j,p_j+i} + \sum_{i=1}^{d_j} \zeta_{j,p_j+q_j+i},$$

and $\mathrm{MN}_{s,k}(\boldsymbol{\mu}, \boldsymbol{\Delta}, \boldsymbol{\Sigma})$ represents the matrix Gaussian distribution (see, for instance, Gupta and Nagar (1999, Chapter 2)), with $\boldsymbol{\mu}$ ($s \times k$) as its location parameter matrix, and the positive-definite matrices $\boldsymbol{\Delta}$ ($s \times s$) and $\boldsymbol{\Sigma}$ ($k \times k$) as its scale parameters. This matrix-normal prior jointly regularizes the active coefficients in Regime $j$ towards $\boldsymbol{\mu}_{0j\zeta}$, with $\boldsymbol{\Delta}_{0j\zeta}$ governing their prior dependence. The dependence on $\boldsymbol{\Sigma}_j$ ensures scale invariance across equations and preserves conjugacy with the likelihood, thereby streamlining posterior simulation;

(6) For the skew-normal and skew-$t$ cases, the skewness vector $\boldsymbol{\lambda}$ is assigned the prior $\pi(\boldsymbol{\lambda}) = \mathrm{Normal}_k(\mathbf{0}, \boldsymbol{\Lambda}_0)$, where $\boldsymbol{\Lambda}_0$ is a positive-definite $k \times k$ matrix. This prior centres the innovations at symmetry ($\boldsymbol{\lambda} = \mathbf{0}$) while $\boldsymbol{\Lambda}_0$ controls how strongly we shrink towards symmetry and how large skewness is allowed a priori; and

(7) Note that this prior depends on the assumed error distribution: $\pi(\nu)$ is

- for the Student-$t$, symmetric hyperbolic, and skew-$t$ cases, $\mathrm{Uniform}(\gamma_0, \eta_0)$;
- for the Slash case, $\mathrm{Gamma}(\gamma_0, \eta_0)$, that is, $\pi(\nu) \propto \nu^{\gamma_0 - 1} \exp(-\eta_0 \nu) I_\nu(0, \infty)$; and
- for the contaminated normal case, $\pi(\nu_1)\pi(\nu_2)$, in which $\pi(\nu_1)$ is $\mathrm{Beta}(\gamma_{01}, \eta_{01})$, that is, $\pi(\nu_1) \propto \nu_1^{\gamma_{01}-1}(1-\nu_1)^{\eta_{01}-1} I_{\nu_1}(0,1)$ and $\pi(\nu_2)$ is $\mathrm{Truncated\,Gamma}(\gamma_{02}, \eta_{02}; (0,1))$, that is, $\pi(\nu_2) \propto \nu_2^{\gamma_{02}-1} \exp(-\eta_{02}\nu_2) I_{\nu_2}(0,1)$.

These priors keep the parameters on their natural support and in plausible ranges, and they control the degree of tail thickness and hence the robustness of the innovations to extreme observations.

The package **mtarm** adopts the following simplified specification of the prior distributions. For each $j = 1, \ldots, l$: $\boldsymbol{\Omega}_{0j} = \mathbf{I}_k \otimes \omega_0$, $\tau_{0j} = \tau_0$, $\boldsymbol{\mu}_{0j\zeta} = \mathbf{1}_{s_{j\zeta},k} \otimes \mu_0$, $\boldsymbol{\Delta}_{0j\zeta} = \mathbf{I}_{s_{j\zeta}} \otimes \delta_0^{s_j/s_{j\zeta}}$ and $\rho_{0j} = \rho_0$. Moreover, $\boldsymbol{\Lambda}_0 = \mathbf{I}_k \otimes \lambda_0$, where $\mathbf{1}_{m,n}$ denotes the $m \times n$ matrix of ones and $\otimes$ denotes Kronecker product.

## 3 Main routines in the package mtarm

This section demonstrates the main capabilities of **mtarm** through a case study based on the bivariate time series contained in the `riverflows` dataset (see Table 1). The dataset was previously analyzed by Calderón and Nieto (2017) and Vanegas et al. (2025), making it a useful benchmark for illustrating the package's methodology and workflow. Throughout this section, the analysis is presented in a step-by-step manner, showing how the core functions of **mtarm** can be combined to perform model specification, estimation, model comparison, diagnostic assessment, and forecasting within a unified Bayesian framework. The objective is to study the dynamic relationship between rainfall and river flows over the period from January 1, 2006, to April 14, 2009. Rainfall, measured in millimeters ($mm$), was recorded at the San Juan meteorological station and is denoted by ($Z_t$). River flows, measured in cubic meters per second ($m^3/s$), were recorded at two hydrological stations: the Bedon River flow ($Y_{1,t}$) at El Trebol station and the La Plata River flow ($Y_{2,t}$) at Villalosada station. The San Juan station is located at an elevation of 2400 meters above sea level, whereas the El Trebol and Villalosada stations are situated at 1720 and 1300 meters, respectively. All stations are located in a dry equatorial region of Colombia, where hydrological and meteorological conditions are relatively stable, allowing for a clearer assessment of the nonlinear dynamic relationships captured by TAR models. The data were provided by the Colombian Institute of Hydrology, Meteorology and Environmental Studies (IDEAM). Missing observations were imputed using the methodology proposed by Calderón and Nieto (2017).

| Column | Role | Description |
|---|---|---|
| `Date` | Labels for time points | Dates when measurements were taken |
| `Rainfall` | Threshold series | Rainfall, in $mm$ |
| `Bedon` | First component of the output series | Bedon river flow, in $m^3/s$ |
| `LaPlata` | Second component of the output series | La Plata river flow, in $m^3/s$ |

**Table 1:** Description of the variables in the `riverflows` dataset included in the **mtarm** package.

Figure 1 displays the three time series described above. A clear association can be observed between rainfall and river flows: periods of elevated precipitation tend to coincide with increased river discharge, particularly in the Bedon River, whereas periods of low rainfall are generally associated with reduced flow levels. This relationship appears less pronounced for the La Plata River, suggesting that the hydrological response to precipitation may differ between the two river systems. Overall, these patterns indicate a strong dependence of river discharge on rainfall and suggest that the underlying dynamics may vary across hydrological conditions. Such behavior is consistent with the presence of threshold effects in the river-flow dynamics. In particular, the relationship between rainfall and river discharge may change when precipitation exceeds or falls below certain critical levels, giving rise to distinct dynamic regimes. This possibility motivates the use of a regime-switching framework. The multivariate TAR model is especially well suited to this setting, as it allows the joint dynamics of the Bedon and La Plata river flows to evolve differently across rainfall regimes. To reproduce this exploratory analysis, the following code can be used to generate the corresponding plots:

```
> data(riverflows)
> str(riverflows)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':   1200 obs. of  4 variables:
 $ Date    : Date, format: "2006-01-01" "2006-01-02" ...
 $ Bedon   : num  5.6 6.36 7.5 7.11 12.39 ...
 $ LaPlata : num  17.6 17.3 17.1 16.8 28.5 ...
 $ Rainfall: num  0 0 0 4 0 ...
>
> dev.new()
> par(mfrow=c(3,1))
> with(riverflows,{plot(Date, Rainfall, type="l", lty=1, col="black", xlab="",
+                       ylab="Millimeters", main="Rainfall")
+                  plot(Date, Bedon, type="l", lty=1, col="black", xlab="",
+                       ylab="Cubic meters per second", main="Bedon river")
+                  plot(Date, LaPlata, type="l", lty=1, col="black", xlab="",
+                       ylab="Cubic meters per second", main="La Plata river")})
```

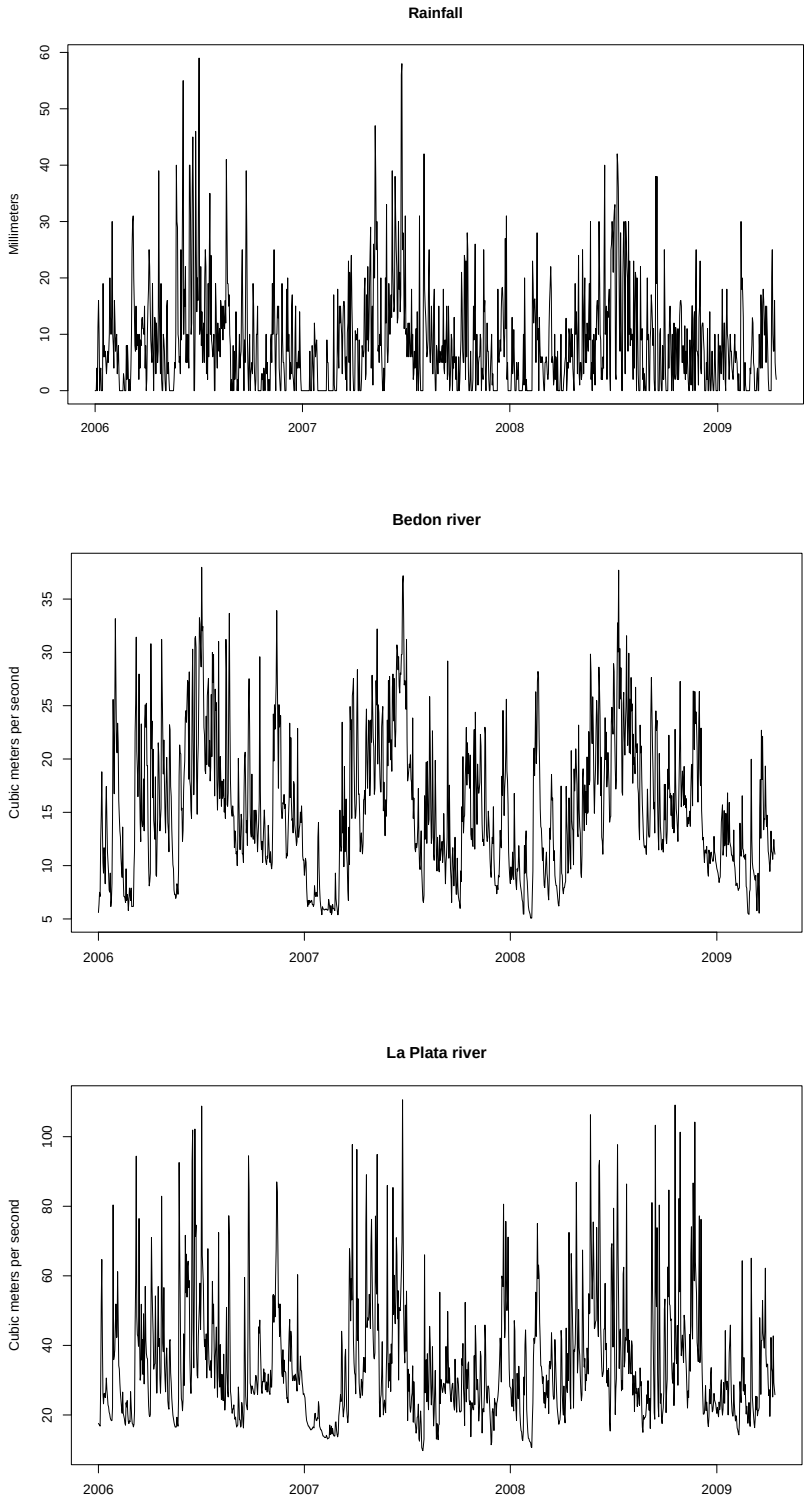



**Figure 1:** Time series plots of rainfall (a), Bedon River flow (b), and La Plata River flow (c).

### 3.1 Fitting a TAR model

```
> args(mtar)
function(formula, data, subset, Intercept=TRUE, trend=c("none","linear","quad"),
         nseason=NULL, ars=ars(), row.names, dist=dist=c("Gaussian","Student-t",
         "Hyperbolic","Laplace","Slash","Contaminated normal","Skew-Student-t",
         "Skew-normal"), prior=list(), n.sim=500, n.burnin=100, n.thin=1,
```

```
ssvs=FALSE, setar=NULL, progress=TRUE, ...)
```

The routine `mtar()` performs Bayesian estimation in multivariate TAR models, including SETAR and VAR models, and their univariate versions, as described by Vanegas et al. (2025). In that routine, the user can specify several features of both the TAR model to be fitted to the data and the MCMC algorithm to be used to draw a sample from the posterior distribution of the model parameters. Some of those features are the following: $(i)$ components of the model, that is, the output time series, the threshold time series, and the exogenous time series; $(ii)$ number of regimes; $(iii)$ the autoregressive order for $\{\boldsymbol{Y}_t\}_{t\geq 1}$ and the maximum lags for $\{\boldsymbol{X}_t\}_{t\geq 1}$ and $\{Z_t\}_{t\geq 1}$ within each regime; $(iv)$ noise process distribution; $(v)$ hyperparameter values; and $(vi)$ chain size, thinning interval and burn-in period for the MCMC algorithm. The main arguments in the routine `mtar()` are the following:

- **formula**. This Formula-type argument allows us to specify the output time series, the threshold time series (if any), and the exogenous time series (if any). Formula-type objects provided by the package `Formula` (Zeileis and Croissant (2010)) extend the base class `formula` by allowing for multiple responses and multiple parts of regressors. For example, `formula = ∼ y1 + y2 + y3 | z | x1 + x2` indicates the following: $(i)$ the output time series, $\{\boldsymbol{Y}_t\}_{t\geq 1}$, is tridimensional and its realization is composed of the columns named `y1`, `y2` and `y3`; $(ii)$ the realization of the threshold time series, $\{Z_t\}_{t\geq 1}$, is the column named `z`; and $(iii)$ the exogenous time series, $\{\boldsymbol{X}_t\}_{t\geq 1}$, is bidimensional and its realization is composed of the columns named `x1` and `x2`. The columns named `y1`, `y2`, `y3`, `z`, `x1` and `x2` are assumed to be included in the data.frame specified in the argument `data` in the call to the routine `mtar()`. Indeed, if the data.frame-type object specified in the argument `data` contains only the variables comprising the output time series, a `VAR(p)` model can be specified by `formula = ∼ .`
- **ars**. This list-type argument allows us to specify values for $l$ (number of regimes), $\boldsymbol{p} = (p_1, \ldots, p_l)$, $\boldsymbol{q} = (q_1, \ldots, q_l)$ and $\boldsymbol{d} = (d_1, \ldots, d_l)$. The helper function `ars()` simplifies the model setup specification in `ars`. For example, `ars(nregim=3,p=2)` sets a TAR model with 3 regimes and an autoregressive order of 2 within each regime. Similarly, `ars(nregim=3,p=2,q=3)` sets a TAR model with 3 regimes, an autoregressive order of 2 within each regime, and a maximum lag of 3 for the exogenous time series within each regime. Moreover, `ars(nregim=3,p=c(1,2,1),q=c(2, 0,3),d=c(1,1))` indicates the following: $(i)$ the number of regimes is 3; $(ii)$ the autoregressive order in the first, second and third regimes is 1, 2 and 1, respectively; $(iii)$ the maximum lag for the exogenous time series in the first, second and third regimes is 2, 0 and 3, respectively; $(iv)$ the maximum lag for the threshold time series in the first, second and third regimes is 1, 1 and 0, respectively. The inclusion of `p` in `ars()` is mandatory, whereas `q` and `d` are optional. Indeed, when a threshold series is specified in the argument `formula`, the inclusion of `d` becomes relevant, but it is not necessary. In addition, if `q/d` is missing, then `ars()` assumes that the maximum lag for the exogenous/threshold series is 0 in all regimes.
- **data**. A data.frame-type object with rows representing time points, arranged ascendingly according to the time and containing at least the following columns: $(i)$ all those used to specify the TAR model in the argument `formula`, that is, those composing the output time series, the threshold time series (if any), and the exogenous time series (if any); $(ii)$ that specified in the optional argument `row.names` to label the time points; $(iii)$ those utilized in the optional argument `subset` to choose a subset of the rows of the data.frame to be used in the fitting process. If the argument `data` is missing, then the columns are taken from `environment(formula)`, typically the environment from which the routine `mtar()` is called.
- **Intercept**. This logical-type argument allows us to specify whether there are intercept terms in the model, that is, if there are $\phi_0^{(j)} \neq 0$ for $j = 1, \ldots, l$. By default, `Intercept` is set to `TRUE`.
- **trend**. This character string allows users to specify the degree of a deterministic time trend to be included in each regime. The available options are: linear trend ("linear"), quadratic trend ("quadratic"), and no time trend ("none"). By default, `trend` is set to "none".
- **nseason**. This integer value allows us to specify the number of seasonal periods. If `nseason` is specified then `nseason`-1 seasonal dummies are added to the regressors within each regime. As described in section 2.2.1, those seasonal dummies are obtained by vertically stacking several `nseason`$\times$ `nseason` identity matrices with their first column removed.
- **dist**. A character-type argument that allows us to specify a multivariate distribution to describe the noise process behavior. Available options are: Gaussian ("`Gaussian`"), Student-$t$ ("`Student-t`"), Slash (``` ``Slash'' ```), symmetric Hyperbolic (``` ``Hyperbolic'' ```), contaminated normal (``` ``Contaminated normal'' ```), Laplace (``` ``Laplace'' ```), skew-normal (``` ``Skew-normal'' ```) and Skew-$t$ (``` ``Skew-Student-t'' ```). By default, `dist` is set to ``` ``Gaussian'' ```.
- **prior**. A list-type argument that allows us to specify the hyperparameter values described in section 2.2.3. By default, they are set to the following values, which ensure non-informative

prior distributions: $h_{\min} = 0$, $h_{\max} = 3$, $\alpha_0 = 0$, $\alpha_1 = 1$, $\omega_0 = 10^{-9}$, $\tau_0 = k$, $\mu_0 = 0$, $\delta_0 = 10^9$, $\rho_0 = 0.5$, $\lambda_0 = 10^9$, and

- for the Student-$t$ and skew-$t$ cases, $\gamma_0 = 1$ and $\eta_0 = 100$;
- for the Slash case, $\gamma_0 = 10^{-9}$ and $\eta_0 = 10^{-9}$;
- for the contaminated normal case, $\gamma_{01} = 10^{-9}$, $\eta_{01} = 10^{-9}$, $\gamma_{02} = 10^{-9}$ and $\eta_{02} = 10^{-9}$;
- for the symmetric hyperbolic case, $\gamma_0 = 0.1$ and $\eta_0 = 4$.

For example, `prior=list(hmin=0,hmax=3,theta0=0,delta0=1e+09,omega0=1e-09,tau0=k)` indicates that $h_{\min}$, $h_{\max}$, $\theta_0$, $\delta_0$, $\omega_0$ and $\tau_0$ are set to 0, 3, 0, $10^9$, $10^{-9}$ and $k$, respectively, while the other hyperparameters are set to their default values. Similarly, `prior=list(hmax=5)` indicates that $h_{\max}$ is set to 5, whereas the other hyperparameters are set to their default values. Finally, `prior=list(hmin=3,hmax=3)` indicates that the delay parameter $h$ is known and equal to 3. The routine `mtar()` uses the helper function `priors()` to validate the argument `prior`. Practical guidance on selecting the hyperparameters that determine the bounds of the delay and threshold parameters has been added to Appendix .2.

- **n.burnin, n.sim, n.thin**. These integer-type arguments allow us to specify the number of iterations that the MCMC-type algorithm must perform, where `n.sim`, `n.thin` and `n.burnin` represent, respectively, the chain size, the thinning interval, and the length of the burn-in period. Hence, the MCMC-type algorithm performs `n.burnin + n.sim` $\times$ `n.thin` iterations, from which `n.sim` iterations are returned, so that `n.burnin + n.sim` $\times$ (`n.thin - 1`) are discarded. The `n.sim` iterations that the routine `mtar()` returns correspond to those at the following positions:

| | |
|---|---|
| 1 | `n.burnin` + 1, |
| 2 | `n.burnin` + 1 + (1 $\times$ `n.thin`), |
| 3 | `n.burnin` + 1 + (2 $\times$ `n.thin`), |
| $\vdots$ | $\vdots$ |
| `n.sim` | `n.burnin` + 1 + ((`n.sim` - 1) $\times$ `n.thin`). |

- **ssvs**. If `ssvs=TRUE` is specified, then the Stochastic Search Variable Selection (SSVS) procedure is applied to identify relevant lags of the output, exogenous (if any), and threshold (if any) series. As a consequence, the components of $\boldsymbol{\zeta}_1, \ldots, \boldsymbol{\zeta}_l$ are estimated as part of the MCMC-type algorithm. The length and structure of each $\boldsymbol{\zeta}_j$ depends on the argument `ars`. Specifically, for $j = 1, ..., l$, $\boldsymbol{\zeta}_j$ is a vector combining three blocks: $(\zeta_{j,1}, ..., \zeta_{j,p_j})$, $(\zeta_{j,p_j+1}, ..., \zeta_{j,p_j+q_j})$, and $(\zeta_{j,p_j+q_j+1}, \ldots, \zeta_{j,p_j+q_j+d_j})$, so that $\boldsymbol{\zeta}_j = (\zeta_{j,1}, ..., \zeta_{j,p_j}, \zeta_{j,p_j+1}, ..., \zeta_{j,p_j+q_j}, \zeta_{j,p_j+q_j+1}, ..., \zeta_{j,p_j+q_j+d_j})$. If `ssvs=FALSE` then $\boldsymbol{\zeta}_1, \ldots, \boldsymbol{\zeta}_l$ are assumed to be known and set to vectors of ones. By default, `ssvs` is set to `FALSE`.
- **setar**. If `setar = m` is specified, a SETAR model is fitted, where `m` identifies the component of the multivariate time series used as the threshold variable. In this case, the hyperparameter $h_{\min}$ is automatically set to `1` (instead of its default value, `0`), since the delay parameter in a SETAR model must be positive. By default, `setar = NULL`, indicating that a general TAR model, rather than a SETAR model, is fitted.
- **progress**. If `progress=TRUE` is specified, then a progress bar is displayed during the execution of the MCMC-type algorithm. By default, `progress` is set to `TRUE`.

To demonstrate the use of this routine, we fit a bivariate TAR$(2; p = (5,5), d = (5,5))$ model to the original data, following the specification considered by Calderón and Nieto (2017). As a linear benchmark, we also estimate a standard VAR$(5)$ model with Gaussian innovations. In the TAR specification, rainfall serves both as the threshold variable and as an exogenous predictor. A notable feature of this analysis is the use of the Stochastic Search Variable Selection (SSVS) procedure implemented in `mtar()`, which enables automatic identification of the relevant lag structures within the Bayesian estimation framework. Although SSVS methods are available for linear models in packages such as `bvartools`, extending Bayesian lag selection to multivariate threshold autoregressive models constitutes an important feature of **mtarm**. For the empirical analysis, observations from January 1, 2006, to April 4, 2009, are used for model estimation, while the final 10 observations are reserved for out-of-sample forecast evaluation. Posterior inference is based on an MCMC algorithm with 9000 iterations. The first 3000 iterations are discarded as burn-in, and a thinning factor of 2 is applied to the remaining draws, resulting in a posterior sample of 3000 observations. The models can be estimated using the following code:

```
> set.seed(2000)
> model_TAR <- mtar(~ Bedon + LaPlata | Rainfall, row.names=Date, dist="Gaussian",
+                   data=riverflows, subset={Date<="2009-04-04"}, ssvs=TRUE,
+                   ars=ars(nregim=2,p=5,d=5), n.burnin=3000, n.sim=3000, n.thin=2)
> model_VAR <- update(model_TAR, ars=ars(nregim=1,p=5))
```

The `mtar()` function returns an object of class `mtar`, for which a range of standard methods are available, including `print()`, `summary()`, `coef()`, `vcov()`, `model.matrix()`, `fitted()`, `plot()`, `predict()`, and `residuals()`. In addition, fitted models can be compared using model assessment criteria such as the Deviance Information Criterion (DIC) (Spiegelhalter et al., 2002, 2014) and the Watanabe–Akaike Information Criterion (WAIC) (Watanabe, 2010), which are implemented in **mtarm** through the functions `DIC()` and `WAIC()`, respectively. To compare the in-sample performance of the fitted models and assess whether the additional flexibility of the nonlinear specification is supported by the data, we compute the DIC and WAIC for both estimated models as follows:

```
> DIC(model_TAR, model_VAR)
                DIC
model_TAR  14186.12
model_VAR  15654.09
>
> WAIC(model_TAR, model_VAR)
               WAIC
model_TAR  14369.19
model_VAR  15793.80
```

Both information criteria favor the TAR specification (`model_MTAR`) over the linear benchmark (`model_VAR`), as indicated by the lower DIC and WAIC values obtained for the former model. This result suggests that the additional flexibility introduced by the threshold structure is supported by the observed data and leads to an improved in-sample fit. From a modeling perspective, these findings provide evidence that allowing the dynamics to vary across rainfall regimes captures important features of the river-flow processes that are not adequately represented by a conventional linear VAR model. Consequently, the results support the use of a threshold autoregressive framework for describing the nonlinear relationship between rainfall and river discharge in this application.

### 3.2 Residual analysis

Next, we examine the residuals of the fitted multivariate TAR model to assess the adequacy of the Gaussian innovation assumption. Specifically, we compute the quantile-type residuals (Dunn and Smyth, 1996), including the joint residuals $r_t^{(2)}$ for $t = 1, \ldots, T$ and the componentwise residuals $r_{m,t}^{(1)}$ for $m = 1, 2$ and $t = 1, \ldots, T$. These residuals are obtained using the following code and stored in the list object `res`. The element `full` contains the joint residuals $r_t^{(2)}$, whereas `by.component` stores the componentwise residuals $r_{m,t}^{(1)}$. The residual diagnostics can then be performed as follows:

```
> set.seed(0202)
> res_model_TAR <- residuals(model_TAR)
> plot(res_model_TAR, col="blue")
```

Figure 2 displays the normal Q-Q plot and histogram of the quantile-type residuals. Since quantile-type residuals are expected to behave as a random sample from the standard normal distribution when the fitted model is correctly specified, these results provide additional evidence supporting the adequacy of the selected models. The diagnostic plots indicate noticeable departures from normality, particularly in the tails of the distribution, where several extreme observations are apparent. Such patterns suggest that the Gaussian innovation assumption may not adequately describe the residual behavior and motivate the consideration of more flexible distributions, such as the Student-$t$, Laplace, or skew-$t$ families. The residual diagnostics are readily obtained through the tools implemented in **mtarm**, which facilitate the computation and extraction of quantile-type residuals for multivariate TAR models. These diagnostics provide a useful basis for evaluating distributional assumptions and guiding the selection of alternative innovation distributions when warranted by the data.

To systematically explore candidate models with different innovation distributions, numbers of regimes, and lag structures, **mtarm** provides the function `mtar_grid()`. This routine serves as a wrapper around `mtar()`, automatically fitting a collection of models defined over a user-specified grid of configurations. In particular, it evaluates combinations of: $(i)$ the innovation distribution specified through `dist`; $(ii)$ the number of regimes, ranging from `nregim.min` to `nregim.max`; $(iii)$ the autoregressive order within each regime, ranging from `p.min` to `p.max`; $(iv)$ the maximum lag of the exogenous series, ranging from `q.min` to `q.max`; and $(v)$ the maximum lag of the threshold series, ranging from `d.min` to `d.max`. The arguments `dist`, `nregim.min`, `nregim.max`, `p.min`, `p.max`, `q.min`, `q.max`, `d.min`, and `d.max` therefore define the model search space explored by `mtar_grid()`. The function returns a list whose elements are objects of class `mtar`, each corresponding to a distinct model specification. This facilitates systematic model comparison and selection within a unified framework. As with any flexible threshold autoregressive model, increasing the number of regimes improves

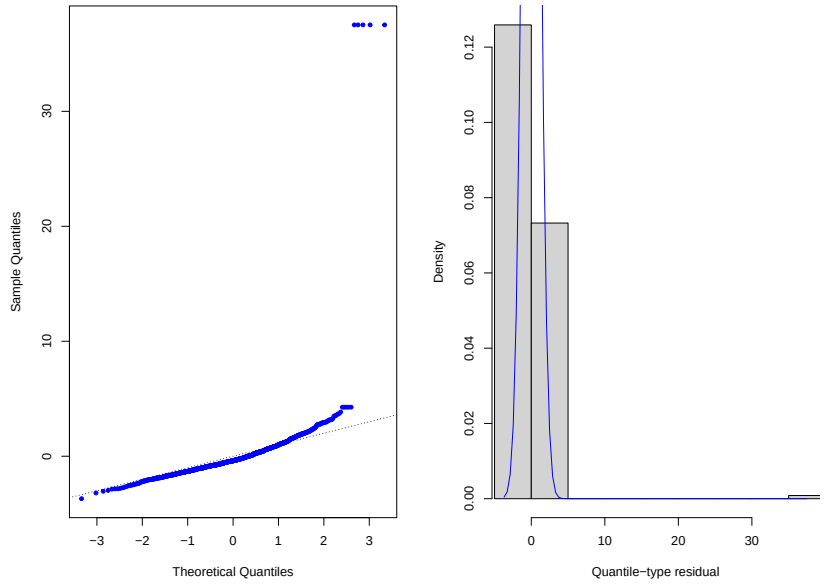



**Figure 2:** Diagnostic plots of the quantile-type residuals for the TAR$(2; \boldsymbol{p} = (5,5))$ model with Gaussian innovations: Normal Q-Q plot (a) and histogram (b).

model flexibility but also increases the number of parameters to be estimated. Consequently, regimes containing relatively few observations may lead to weakly identified parameters and greater posterior uncertainty. In practice, potential identifiability issues can be investigated using the convergence diagnostics available in **mtarm**, such as Geweke's diagnostic and effective sample size, together with posterior summaries including High Posterior Density (HPD) intervals. Unusually wide HPD intervals may indicate that a model is overly complex relative to the information available in the data. Users should also be aware of the computational trade-offs associated with richer model specifications. In particular, heavy-tailed innovation distributions typically require the introduction of latent variables within the MCMC algorithm, increasing computational cost per iteration. A practical strategy is to conduct an initial grid search using relatively short MCMC runs to eliminate clearly inferior specifications before refitting a smaller set of promising candidates using longer chains. In addition, fitted models can be compared using out-of-sample predictive performance measures, including the Absolute Error (AE), Absolute Percentage Error (APE), Squared Error (SE), log-score (Good, 1952), Energy Score (Gneiting et al., 2008)—a multivariate generalization of the Continuous Ranked Probability Score (CRPS) (Matheson and Winkler, 1976; Grimit et al., 2006)—, and other forecasting metrics available through the `out_of_sample()` function. Finally, the `plan_strategy` argument of `mtar_grid()` enables parallel computation through the packages **future** (Bengtsson, 2024a) and **future.apply** (Bengtsson, 2024b), substantially reducing computation times when a large number of candidate models are considered.

### 3.3 Improving the fit: exploration of alternative models

To improve model fit and assess the impact of model complexity, a total of 20 candidate models are fitted to the data. Specifically, we consider the families VAR$(p^*)$, TAR$(2; \boldsymbol{p} = (p^*, p^*))$, TAR$(3; \boldsymbol{p} = (p^*, p^*, p^*))$, and TAR$(4; \boldsymbol{p} = (p^*, p^*, p^*, p^*))$, with $p^* = 1, \ldots, 5$. This model set allows us to jointly evaluate the effects of autoregressive order and the number of regimes on predictive performance and goodness of fit. To accommodate the outliers identified in the residual diagnostics, the innovation process $\boldsymbol{\epsilon}_{t_j}$ is assumed to follow a multivariate Laplace distribution. This heavy-tailed specification provides increased robustness to extreme observations and departures from normality. More generally, one of the key strengths of **mtarm** is its ability to estimate multivariate TAR models under a broad range of non-Gaussian innovation distributions, thereby extending the scope of existing software for threshold autoregressive modeling in R. The candidate models considered in this analysis can be expressed as follows:

$$\boldsymbol{Y}_t = \sum_{j=1}^{l} I(Z_{t-h} \in (c_{j-1}, c_j]) \Big( \boldsymbol{\phi}_0^{(j)} + \sum_{i=1}^{p^*} \boldsymbol{\phi}_i^{(j)} \boldsymbol{Y}_{t-i} + \boldsymbol{\epsilon}_{t_j} \Big),$$

where $\boldsymbol{\epsilon}_{t_j} \overset{\text{ind}}{\sim} \text{Laplace}(\mathbf{0}, \boldsymbol{\Sigma}_j)$. Meaningful comparisons across fitted models require that all models be estimated using the same effective sample period. In autoregressive settings, however, the first $m$ observations of the observed series cannot be used for estimation because they are needed to construct the lagged predictors. Consequently, the effective sample size is $T - m$, where $T$ denotes the length of the observed series. For VAR($p^*$) models (i.e., a single-regime specification), $m = p^*$. For threshold autoregressive models with two or more regimes, the effective sample offset is given by $m = \max(p^*, h_{\max})$, where $h_{\max}$ denotes the upper bound of the prior support for the delay parameter $h$, whose default value in **mtarm** is 3. The `mtar_grid()` function automatically adjusts the effective sample period for each fitted model, ensuring that all model comparisons are based on the same set of observations.

### Model estimation

The following code uses the wrapper function `mtar_grid()` to systematically fit a collection of candidate models by repeatedly calling `mtar()` over all combinations of the number of regimes ($l = 1, \ldots, 4$) and autoregressive orders ($p^* = 1, \ldots, 5$). The resulting fitted models are stored in the list object `models`, whose elements are named `Laplace.`$l$`.`$p^*$, corresponding to the model with $l$ regimes and autoregressive order $p^*$. For each candidate model, `mtar()` is configured as follows: $(i)$ relevant lagged effects of $\{\boldsymbol{Y}_t\}_{t \geq 1}$ are identified separately within each regime using the Stochastic Search Variable Selection (SSVS) procedure (`ssvs=TRUE`); $(ii)$ default hyperparameter values are employed, yielding non-informative prior distributions for all model parameters; $(iii)$ posterior inference is based on an MCMC algorithm with 9000 iterations, of which the first 3000 are discarded as burn-in and the remaining draws are thinned by retaining every second sample; $(iv)$ the estimation period spans January 6, 2006, to April 4, 2009; $(v)$ the last ten observations (April 5, 2009, to April 14, 2009) are reserved for out-of-sample forecast evaluation; and $(vi)$ observation dates are provided through the `Date` column of the `riverflows` data frame. To reduce the overall computation time, `mtar_grid()` is instructed to fit the candidate models in parallel by specifying `plan_strategy = "multisession"`.

```
> set.seed(0220)
> models <- mtar_grid(~ Bedon + LaPlata | Rainfall, row.names=Date, dist="Laplace",
                      data=riverflows, subset={Date<="2009-04-04"}, nregim.min=1,
                      nregim.max=4, p.min=1, p.max=5, n.burnin=3000, n.sim=3000,
                      n.thin=2, ssvs=TRUE, plan_strategy="multisession")
> models

Sample size          : 1185 time points (2006-01-06 to 2009-04-04)
Output Series        : Bedon    |    LaPlata
Threshold Series     : Rainfall
Error Distribution   : Laplace
Number of regimes    : 1 to 4
Deterministics       : Intercept
Autoregressive orders: 1 to 5
```

### Model selection

The fitted models are then compared using adjusted within-sample predictive accuracy measures, such as DIC and WAIC. According to the following code, a first matrix, named `DICs`, stores the values of the DIC criterion for the twenty fitted models, whereas a second matrix, named `WAICs`, stores the values of the WAIC criterion for the twenty fitted models. In both cases, the rows represent the different values of $p^*$, whereas the columns represent the different values of the number of regimes $l$.

```
> DICs <- matrix(DIC(models),5,4)
> rownames(DICs) <- paste0("p*=",1:5)
> colnames(DICs) <- c("VAR(p*)","TAR(2;p*,p*)","TAR(3;p*,p*,p*)","TAR(4;p*,p*,p*,p*)")
> round(DICs,2)
      VAR(p*) TAR(2;p*,p*) TAR(3;p*,p*,p*) TAR(4;p*,p*,p*,p*)
p*=1 14490.90     13757.46        13573.22           13500.45
p*=2 14423.63     13726.43        13572.99           13519.93
p*=3 14422.83     13681.53        13568.18           13458.78
p*=4 14421.12     13680.52        13525.78           13447.31
p*=5 14385.23     13637.01        13504.93           13433.73
>
> WAICs <- matrix(WAIC(models),5,4)
> rownames(WAICs) <- rownames(DICs)
```

```
> colnames(WAICs) <- colnames(DICs)
> round(WAICs,2)
      VAR(p*) TAR(2;p*,p*) TAR(3;p*,p*,p*) TAR(4;p*,p*,p*,p*)
p*=1 14494.80     13765.46        13585.76           13540.82
p*=2 14428.96     13737.65        13587.31           13545.79
p*=3 14429.75     13695.53        13591.64           13479.35
p*=4 14429.73     13694.27        13564.89           13482.35
p*=5 14391.72     13737.57        13531.97           13499.51
```

Compared with linear specifications (i.e., VAR models), nonlinear alternatives, namely TAR models with two, three, and four regimes, exhibit superior performance under both criteria. For a fixed autoregressive order $p^*$, models with four regimes consistently yield the lowest DIC and WAIC values, whereas single-regime models perform worst. Likewise, for a fixed number of regimes $l$, models of order 5 generally attain the lowest DIC and WAIC values, while order 1 models tend to have the highest. According to the DIC, the preferred specification is TAR$(3; \boldsymbol{p} = (3,3,3))$, whereas the WAIC favors TAR$(4; \boldsymbol{p} = (3,3,3,3))$. To complement the in-sample model assessment, several out-of-sample predictive accuracy measures are also evaluated. Specifically, we consider the average log-score (LS), the average Energy Score (ES) and the Mean Absolute Percentage Error (MAPE) for each component of the multivariate response series. The following code computes these measures using the hold-out observations reserved for forecast evaluation. The resulting values are then aggregated using the function supplied through the `FUN` argument, which by default is set to `mean()`. This allows users to summarize predictive performance according to alternative criteria if desired. The resulting values are stored in the matrices `LSs`, `ESs`, `APE.1s`, and `APE.2s`. In each matrix, rows correspond to different autoregressive orders $p^*$, whereas columns correspond to different numbers of regimes $l$.

```
> set.seed(0220)
> future.obs <- subset(riverflows, Date>"2009-04-04")
> oos <- out_of_sample(models, newdata=future.obs, n.ahead=nrow(future.obs), FUN=mean)
>
> LSs <- matrix(oos[,1],nrow=5,ncol=4)
> rownames(LSs) <- rownames(DICs)
> colnames(LSs) <- colnames(DICs)
> round(LSs,2)
     VAR(p*) TAR(2;p*,p*) TAR(3;p*,p*,p*) TAR(4;p*,p*,p*,p*)
p*=1    6.27         6.14            5.96               6.02
p*=2    6.21         6.02            5.94               5.98
p*=3    6.22         5.99            5.94               5.97
p*=4    6.21         5.98            5.90               5.97
p*=5    6.10         5.98            5.80               5.92
>
> ESs <- matrix(oos[,2],nrow=5,ncol=4)
> rownames(ESs) <- rownames(DICs)
> colnames(ESs) <- colnames(DICs)
> round(ESs,2)
     VAR(p*) TAR(2;p*,p*) TAR(3;p*,p*,p*) TAR(4;p*,p*,p*,p*)
p*=1   12.56         9.76            8.24               8.46
p*=2   12.45         9.62            8.19               8.40
p*=3   12.44         9.51            8.25               8.15
p*=4   12.43         9.49            8.12               8.13
p*=5   11.80         9.35            7.95               8.10
>
> APEs.1 <- matrix(oos[,5],nrow=5,ncol=4)
> rownames(APEs.1) <- rownames(DICs)
> colnames(APEs.1) <- colnames(DICs)
> round(APEs.1,2)
     VAR(p*) TAR(2;p*,p*) TAR(3;p*,p*,p*) TAR(4;p*,p*,p*,p*)
p*=1   21.76        25.31           24.15              26.41
p*=2   23.16        22.41           23.23              25.97
p*=3   23.29        21.76           23.33              25.07
p*=4   22.33        21.33           22.33              24.85
p*=5   19.98        21.29           20.10              23.52
>
> APEs.2 <- matrix(oos[,6],nrow=5,ncol=4)
> rownames(APEs.2) <- rownames(DICs)
> colnames(APEs.2) <- colnames(DICs)
```

```
> round(APEs.2,2)
     VAR(p*) TAR(2;p*,p*) TAR(3;p*,p*,p*) TAR(4;p*,p*,p*,p*)
p*=1   33.61        13.94           11.30              11.06
p*=2   34.59        13.85           11.15              11.27
p*=3   33.91        13.19           11.16              10.28
p*=4   33.60        13.28           11.45              10.46
p*=5   31.75        13.82           11.23              10.56
```

In contrast to the conclusions suggested by the DIC and WAIC, the out-of-sample predictive measures favor models with three regimes over those with four regimes. For a fixed autoregressive order $p^*$, the three-regime specifications generally achieve the lowest average log-score (LS) and average Energy Score (ES) values, whereas the single-regime model consistently exhibits the weakest predictive performance. Similarly, for a fixed number of regimes $l$, models with $p^* = 5$ tend to outperform those with lower autoregressive orders, while models with $p^* = 1$ typically perform the worst. Regarding point forecast accuracy, the benefits of introducing TAR-type nonlinearity appear to differ across the two components of the response series. For the Bedon River flow, TAR models with two, three, and four regimes provide only little improvements relative to the corresponding VAR models. For the La Plata River flow, however, the gains are considerably more pronounced, with all TAR specifications substantially outperforming their linear VAR counterparts. Moreover, the average LS, average ES, and MAPE associated with the first response component (the Bedon River flow) consistently identify the TAR$(3; \boldsymbol{p} = (5,5,5))$ model as the most competitive specification. Taken together, these results suggest that the additional flexibility provided by a three-regime structure yields meaningful gains in predictive performance, while further increasing the number of regimes offers little benefit. This discrepancy between within-sample and out-of-sample criteria illustrates an important practical consideration in time-series modeling. Information criteria such as DIC and WAIC balance model fit and complexity and are computed using the data employed for model estimation. As a result, they may favor more flexible specifications, such as the four-regime model in this application. In contrast, out-of-sample predictive measures evaluate forecasting performance on observations that were not used during estimation and therefore provide a direct assessment of predictive generalization. For practitioners, this example highlights the importance of considering multiple model-selection criteria jointly rather than relying on a single measure. Although the four-regime specification achieved slightly better DIC and WAIC values, the three-regime model consistently outperformed it according to the out-of-sample predictive measures considered. Since forecasting is the primary objective of this application, we selected the three-regime model. More generally, this example illustrates that additional model complexity is not always accompanied by improved predictive performance, and that parsimonious specifications may be preferable when they provide comparable fit while yielding superior forecasts. Consequently, the TAR$(3; \boldsymbol{p} = (5,5,5))$ model is selected as the preferred specification for subsequent analysis.

### Overview of the chosen model

The following code requests a summary of the fitted TAR$(3; \boldsymbol{p} = (5,5,5))$ Laplace model, where the argument `credible`, which by default is set to 0.95, allows the user to specify the required level for the credible intervals of the model parameters.

```
> summary(models[["Laplace.3.5"]], credible=0.95)

Sample size          : 1185 time points (2006-01-06 to 2009-04-04)
Output Series (OS)   : Bedon    |    LaPlata
Threshold Series     : Rainfall with a estimated delay equal to 0
Error Distribution   : Laplace
Number of regimes    : 3
Deterministics       : Intercept
Autoregressive orders: 5 in each regime

Thresholds (Mean, HDI_low, HDI_high)
Regime 1     (-Inf,3.4337]     (-Inf,3.08834]     (-Inf,3.98057]
Regime 2 (3.4337,10.00928] (3.08834,10.00268] (3.98057,10.01671]
Regime 3    (10.00928,Inf)     (10.00268,Inf)     (10.01671,Inf)

Regime1:
     OS.lag(1) OS.lag(2) OS.lag(3) OS.lag(4) OS.lag(5)
SSVS         1         0         0         0         1
```

```
Autoregressive coefficients
                Mean  2(1-PD) HDI.Lower HDI.Upper     Mean  2(1-PD) HDI.Lower HDI.Upper
(Intercept)     1.28929 0.00001  1.08328  1.51231  |  3.59625 0.00001  2.91502  4.30655
Bedon.lag(1)    0.63936 0.00001  0.58745  0.69628  |  0.09655 0.17600 -0.05574  0.22998
LaPlata.lag(1)  0.02471 0.02600  0.00255  0.04455  |  0.61940 0.00001  0.55499  0.68540
Bedon.lag(5)    0.11205 0.00001  0.07241  0.15627  |  0.09219 0.04600  0.00395  0.18634
LaPlata.lag(5) -0.01256 0.02533 -0.02487 -0.00141  |  0.03783 0.03267  0.00244  0.06995

Scale parameter (Mean, HDI.Lower, HDI.Upper)
          Bedon LaPlata        Bedon LaPlata        Bedon LaPlata
Bedon   0.33997 0.37479    . 0.27289 0.25739    . 0.40063 0.49272
LaPlata 0.37479 2.49280    . 0.25739 2.04113    . 0.49272 2.98511

Regime2:
     OS.lag(1) OS.lag(2) OS.lag(3) OS.lag(4) OS.lag(5)
SSVS         1         0         0      0.08      0.88

Autoregressive coefficients
                Mean  2(1-PD) HDI.Lower HDI.Upper     Mean  2(1-PD) HDI.Lower HDI.Upper
(Intercept)     2.23360 0.00001  1.41838  3.14468  |  6.92708 0.00001  4.63591  8.94865
Bedon.lag(1)    0.64825 0.00001  0.55580  0.72742  |  0.21746 0.03800  0.02686  0.40818
LaPlata.lag(1)  0.00696 0.54667 -0.01773  0.02968  |  0.54337 0.00001  0.46861  0.61895
Bedon.lag(5)    0.09064 0.00200  0.02886  0.15241  | -0.19322 0.01600 -0.33803 -0.04437
LaPlata.lag(5)  0.00784 0.49800 -0.01506  0.03163  |  0.13070 0.00001  0.05150  0.18953

Scale parameter (Mean, HDI.Lower, HDI.Upper)
          Bedon LaPlata        Bedon LaPlata        Bedon LaPlata
Bedon   1.14169 1.33062    . 0.92934 0.94279    . 1.37166 1.70368
LaPlata 1.33062 6.62753    . 0.94279 5.43417    . 1.70368 7.95012


Regime3:
     OS.lag(1) OS.lag(2) OS.lag(3) OS.lag(4) OS.lag(5)
SSVS         1      0.01      0.01         0      0.01

Autoregressive coefficients
                Mean  2(1-PD) HDI.Lower HDI.Upper    Mean   2(1-PD) HDI.Lower HDI.Upper
(Intercept)     7.36682 0.00001  5.93633  8.74479  | 20.44985 0.00001 14.90725 26.53765
Bedon.lag(1)    0.64983 0.00001  0.55891  0.75016  |  0.37278 0.05267  0.01207  0.75264
LaPlata.lag(1)  0.01369 0.38000 -0.01968  0.04457  |  0.38793 0.00001  0.27306  0.51174

Scale parameter (Mean, HDI.Lower, HDI.Upper)
          Bedon  LaPlata        Bedon  LaPlata        Bedon  LaPlata
Bedon   3.02402  7.69892    . 2.40927  5.87866    . 3.62148  9.61347
LaPlata 7.69892 47.89816    . 5.87866 38.77202    . 9.61347 57.99085
```

Each model parameter is summarized by its posterior mean and a 95% credible interval, except for the delay parameter $h$ and the by-regime selection indicators $\zeta_1$, $\zeta_2$, and $\zeta_3$, for which only posterior means are reported. For the location parameters, the summary also includes the quantity $2(1 - \text{PD})$, which can be interpreted as a Bayesian analogue of the frequentist $p$-value (Makowski et al., 2019a,b). Here, PD denotes the Probability of Direction, defined as the proportion of the posterior distribution sharing the sign of its median. The columns labeled "Mean" in the summary for the location parameters correspond to the posterior means of the components of $(\boldsymbol{\phi}_0^{(1)}, \boldsymbol{\phi}_1^{(1)}, \boldsymbol{\phi}_5^{(1)})^\top$, $(\boldsymbol{\phi}_0^{(2)}, \boldsymbol{\phi}_1^{(2)}, \boldsymbol{\phi}_5^{(2)})^\top$, and $(\boldsymbol{\phi}_0^{(3)}, \boldsymbol{\phi}_1^{(3)})^\top$ for regimes 1, 2, and 3, respectively. For example, the posterior mean of the location and scale parameters in the first regime are the following:

$$\boldsymbol{\phi}_0^{(1)} = \begin{bmatrix} 1.28929 \\ 3.59625 \end{bmatrix}, \boldsymbol{\phi}_1^{(1)} = \begin{bmatrix} 0.63936 & 0.02471 \\ 0.09655 & 0.61940 \end{bmatrix}, \boldsymbol{\phi}_5^{(1)} = \begin{bmatrix} 0.11205 & -0.01256 \\ 0.09219 & 0.03783 \end{bmatrix} \text{ and } \boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.33997 & 0.37479 \\ 0.37479 & 2.49280 \end{bmatrix}$$

The extractor function `coef()` allows the user to compute summary statistics such as the mean, median, standard deviation, minimum, maximum, or any other from the posterior samples of the model parameters. For instance, the code '`posMED <- coef(models[["Laplace.3.5"]],FUN=median)`' requests the posterior medians, which are stored in a list object named `posMED`. This object contains three sublists, one for each regime: `posMED[[1]]`, `posMED[[2]]`, and `posMED[[3]]`. Each regime-specific

sublist consists of two matrices. In particular, `posMED[[1]]$location` contains the posterior medians of the location parameters $(\boldsymbol{\phi}_0^{(1)}, \boldsymbol{\phi}_1^{(1)}, \boldsymbol{\phi}_5^{(1)})^\top$, while `posMED[[1]]$scale` stores the posterior median of the corresponding scale matrix $\boldsymbol{\Sigma}_1$. Analogously, `posMED[[2]]` and `posMED[[3]]` summarize the posterior medians for regimes 2 and 3, respectively. Finally, the elements `posMED[["delay"]]` and `posMED[["thresholds"]]` provide the posterior medians of the delay parameter and the threshold values, respectively. The extraction function `vcov(models[["Laplace.3.5"]],FUN)` behaves similarly to `coef()` but is restricted to the scale parameters.

The fitted TAR$(3; \boldsymbol{p} = (5,5,5))$ Laplace model, represented by

$$\sum_{j=1}^{3} I(z_{t-\hat{h}} \in (\hat{c}_{j-1}, \hat{c}_j])\Big(\hat{\boldsymbol{\phi}}_0^{(j)} + \sum_{i=1}^{5} \hat{\boldsymbol{\phi}}_i^{(j)} \boldsymbol{y}_{t-i}\Big),$$

where $\hat{h}$, $\hat{c}_1$, $\hat{c}_2$, $\hat{\boldsymbol{\phi}}_0^{(1)}$, $\hat{\boldsymbol{\phi}}_0^{(2)}$, $\hat{\boldsymbol{\phi}}_0^{(3)}$, $\hat{\boldsymbol{\phi}}_i^{(1)}$, $\hat{\boldsymbol{\phi}}_i^{(2)}$ and $\hat{\boldsymbol{\phi}}_i^{(3)}$ denote the posterior means of $h$, $c_1$, $c_2$, $\boldsymbol{\phi}_0^{(1)}$, $\boldsymbol{\phi}_0^{(2)}$, $\boldsymbol{\phi}_0^{(3)}$, $\boldsymbol{\phi}_i^{(1)}$, $\boldsymbol{\phi}_i^{(2)}$ and $\boldsymbol{\phi}_i^{(3)}$, respectively, can be graphically compared with the observed output series for the last 250 observations using the following code. The choice of 250 observations is made solely for illustrative purposes; users may select any number of observations according to their visualization needs.

```
> a <- fitted(models[["Laplace.3.5"]])
> plot(a, observed=list(type="b",pch=20,col="black",lty=3), last=250,
+         fitted=list(type="l",col="blue",lty=3,ylab=rep("Cubic meters per second",2)))
```

### Integration with coda

The tools provided by the package **coda** (Plummer et al., 2006) can be used to summarize and visualize the MCMC chains obtained from the fitted TAR$(3; \boldsymbol{p} = (5,5,5))$ model with Laplace innovations. Specifically, the function `as.mcmc()` constructs a list of `mcmc` objects corresponding to the chains of the thresholds, location, and scale parameters generated by the fitting process. Subsequently, the `summary()` and `plot()` methods from **coda** are applied to each element of this list in order to summarize posterior samples. Similarly, the function `HPDinterval()` in **coda** can be applied to calculate the Highest Posterior Density (HPD) intervals for thresholds, location and scale parameters resulting from the fitted TAR$(3; \boldsymbol{p} = (5,5,5))$ Laplace model.

### Residual analysis

To evaluate the adequacy of the selected multivariate TAR model with Laplace innovations, we perform a residual analysis. In particular, we investigate whether the departures from normality observed in the preliminary model have been satisfactorily accommodated by the more flexible specification. The residuals are extracted using the following code:

```
> set.seed(0220)
> res <- residuals(models[["Laplace.3.5"]])
```

The following code builds a histogram with superimposed standard normal density and a normal Q-Q plot of quantile-type residuals (Figure 3).

```
> plot(res, col="blue")
```

The histogram shows a close agreement between the empirical distribution of the residuals and the standard normal distribution. Likewise, the Q-Q plot closely follows a straight line with near-zero intercept and unit slope, indicating that the fitted model provides an adequate description of the observed data. The autocorrelation and partial autocorrelation functions of the quantile-type residuals can easily be obtained using `acf(res[["full"]])` and `pacf(res[["full"]])`, respectively. The selected TAR model, characterized by three regimes and Laplace innovations, extends the specification previously considered by Calderón and Nieto (2017). This result illustrates the advantages of the **mtarm** framework, which enables the joint exploration of alternative regime structures and non-Gaussian innovation distributions. In this application, the additional modeling flexibility leads to a specification that achieves superior empirical performance while providing a richer characterization of the underlying hydrological dynamics.

### Convergence diagnostics

Geweke's convergence diagnostic, implemented in the routine `geweke_diag()` of the package **coda**, was originally proposed by Geweke (1992). The following code uses the wrapper function `geweke_diagTAR()`

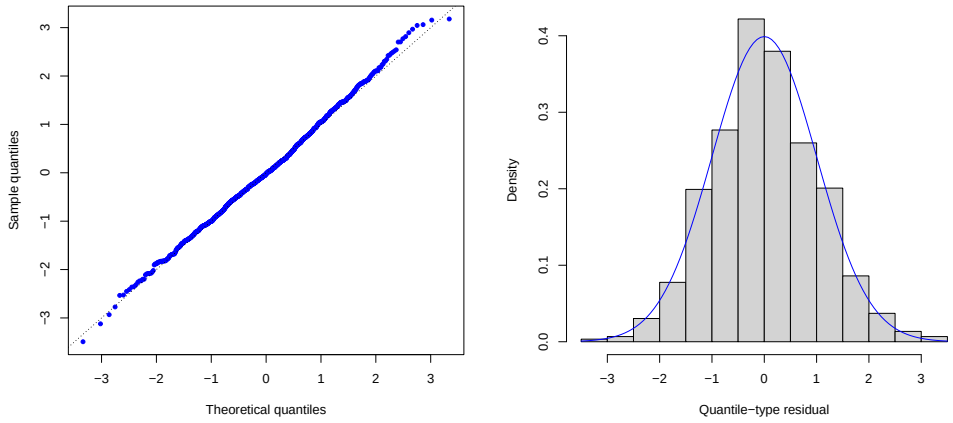



**Figure 3:** Diagnostic plots of the quantile-type residuals for the TAR$(3; \boldsymbol{p} = (5,5,5))$ model with Laplace innovations: Normal Q-Q plot (a) and histogram (b).

provided by **mtarm** to apply `geweke_diag()` to each MCMC chain obtained from the fitted TAR$(3; \boldsymbol{p} = (5,5,5))$ Laplace model. This procedure yields the corresponding $z$-scores that form the basis of Geweke's convergence assessment.

```
> geweke_diagTAR(models[["Laplace.3.5"]])

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

Thresholds:
Threshold.1 Threshold.2
    -1.0652     -1.9642

Regime 1

Autoregressive coefficients:
                  Bedon  LaPlata
(Intercept)    -1.08629 -0.14971
Bedon.lag(1)   -0.50880  0.70745
LaPlata.lag(1)  0.33666 -0.40158
Bedon.lag(5)    0.90577 -0.74717
LaPlata.lag(5) -0.46920  0.45755

Scale parameter:
           Bedon   LaPlata
Bedon    0.684164  0.059843
LaPlata 0.059843 -0.544754


Regime 2

Autoregressive coefficients:
                  Bedon  LaPlata
(Intercept)    -0.90517 -0.24451
Bedon.lag(1)    0.81444  2.10698
LaPlata.lag(1) -1.27630 -1.97840
Bedon.lag(5)    1.46737 -0.12233
LaPlata.lag(5)  1.17324  1.13680

Scale parameter:
           Bedon  LaPlata
Bedon    0.42967 -0.45240
LaPlata -0.45240  0.38793
```

```
Regime 3

Autoregressive coefficients:
                  Bedon  LaPlata
(Intercept)    -0.22473  1.48287
Bedon.lag(1)   -1.88483 -0.51478
LaPlata.lag(1)  1.97141 -0.93629

Scale parameter:
         Bedon LaPlata
Bedon   1.1593 1.51684
LaPlata 1.5168 0.80106
```

In most cases, the absolute values of the resulting $z$-scores are small relative to the 99.5% quantile of the standard normal distribution. Consequently, at the 1% significance level, Geweke's diagnostic provides not enough evidence against the convergence of the Markov chains. In addition, the **mtarm** package provides the wrapper function `geweke_plotTAR()`, which applies `geweke.plot()` from the **coda** package to each MCMC chain generated for the fitted TAR$(3; \boldsymbol{p} = (5,5,5))$ model with Laplace innovations, thereby providing a graphical assessment of convergence for all model parameters.

```
> geweke_plotTAR(models[["Laplace.3.5"]])
```

To assess the mixing and sampling efficiency of the MCMC algorithm, the `effectiveSize_TAR()` function computes the effective sample size (ESS) for all parameters estimated in an `mtar` object. The ESS measures the amount of independent information contained in the posterior sample after accounting for autocorrelation, making it a useful diagnostic for evaluating MCMC efficiency. The following code computes this diagnostic:

```
> effectiveSize_TAR(models[["Laplace.3.5"]])

Thresholds:
Threshold.1 Threshold.2
     8.8078     38.5229

Regime 1
Autoregressive coefficients:
                 Bedon LaPlata
(Intercept)    1718.53  1126.2
Bedon.lag(1)   1043.34  1301.9
LaPlata.lag(1) 1352.69  1159.9
Bedon.lag(5)    908.21  1431.4
LaPlata.lag(5) 1240.61  1224.7

Scale parameter:
         Bedon LaPlata
Bedon   1785.1  2029.6
LaPlata 2029.6  1758.3

Regime 2
Autoregressive coefficients:
                  Bedon  LaPlata
(Intercept)    324.056  709.963
Bedon.lag(1)   240.927  138.885
LaPlata.lag(1) 506.795  144.089
Bedon.lag(5)    65.441 1309.136
LaPlata.lag(5) 365.276   42.178

Scale parameter:
         Bedon LaPlata
Bedon   1620.0  2059.3
LaPlata 2059.3  1752.9
```

```
Regime 3
Autoregressive coefficients:
                 Bedon LaPlata
(Intercept)    1101.19  1099.7
Bedon.lag(1)    634.62  1185.8
LaPlata.lag(1) 1022.89  1294.8

Scale parameter:
         Bedon LaPlata
Bedon   1719.5  1940.2
LaPlata 1940.2  1777.4
```

The output provides a useful summary of the sampling efficiency achieved by the MCMC algorithm. The scale parameters and most autoregressive coefficients—particularly those associated with Regimes 1 and 3—exhibit relatively large effective sample sizes, often exceeding 1000 draws, indicating good posterior exploration and satisfactory mixing. In contrast, the threshold parameters and a small number of coefficients in Regime 2 (e.g., `Bedon.lag(5)` and `LaPlata.lag(5)`) display comparatively lower ESS values. Such behavior is not unexpected in threshold autoregressive models, as threshold parameters are typically more challenging to estimate due to the regime-switching nature of the likelihood function (Vanegas et al., 2025). Nevertheless, the ESS diagnostic provides valuable information for assessing the quality of the posterior sample and identifying parameters that may require additional simulation effort. In practice, if higher precision is desired for the slower-mixing components, the user may consider increasing the MCMC chain length or modifying the thinning strategy.

### Forecasting

The following code generates forecasts of the river flows and their corresponding 90% prediction intervals for the period from April 5, 2009, to April 14, 2009, conditional on the observed rainfall values over the same horizon.

```
> set.seed(0220)
> out <- predict(models[["Laplace.3.5"]], newdata=future.obs, n.ahead=nrow(future.obs),
+                credible=0.9, row.names=Date)
> round(cbind(out[["summary"]][,c(1:3)], Bedon=future.obs[,"Bedon"]),2)

           Bedon.Mean Bedon.Lower Bedon.Upper Bedon
2009-04-05       9.79        7.08       12.30  9.46
2009-04-06      14.11        6.57       22.91 10.40
2009-04-07      17.04        7.77       27.12 13.24
2009-04-08      14.78        6.54       22.59 11.36
2009-04-09      13.33        5.95       20.63 11.17
2009-04-10      12.29        5.69       19.51 10.59
2009-04-11      15.75        6.72       25.47 11.46
2009-04-12      14.72        6.52       22.86 12.45
2009-04-13      12.73        6.17       19.30 11.46
2009-04-14      11.26        5.52       16.44 11.07
>
> round(cbind(out[["summary"]][,c(4:6)], LaPlata=future.obs[,"LaPlata"]),2)

           LaPlata.Mean LaPlata.Lower LaPlata.Upper LaPlata
2009-04-05        24.15         16.80         31.22   19.60
2009-04-06        34.33          4.94         64.76   25.26
2009-04-07        38.69          3.82         74.13   42.28
2009-04-08        32.97          8.36         55.74   33.45
2009-04-09        29.68         10.32         48.07   32.40
2009-04-10        27.51         10.66         43.71   25.17
2009-04-11        35.73          1.42         67.58   42.73
2009-04-12        31.80          8.48         55.34   29.27
2009-04-13        27.20          9.02         44.12   27.38
2009-04-14        24.20         10.99         38.26   25.82
```

The river flow forecasts closely track the observed values, and all true observations lie within the associated 90% prediction intervals. Figure 4 displays the last 300 observations of the Bedon and La

Plata river flows, shown as solid black lines. Ten-step-ahead forecasts are indicated by dashed black lines, while the corresponding 90% prediction intervals are represented by light gray shaded bands. The following code produces this figure.

```
> plot(out,last=300,
+          historical=list(type="l",col="black",lty=1),
+       forecasts=list(type="l",col="black",lty=3,ylab=rep("Cubic meters per second",2)),
+          forecasts.PI=list(col="light gray",border=NULL))
```

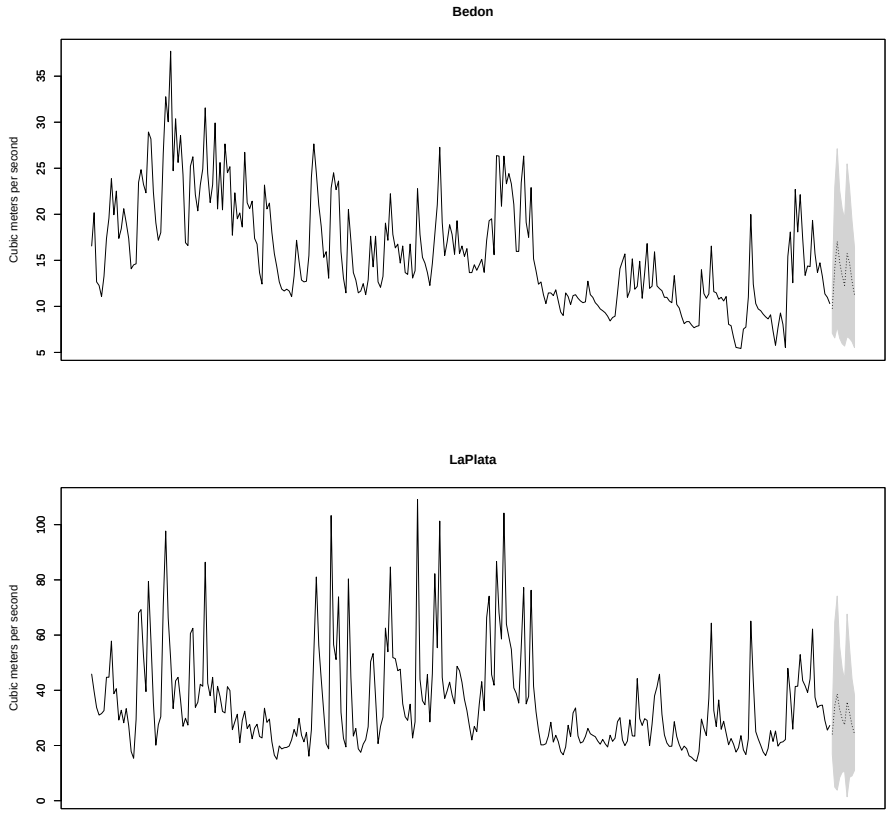



**Figure 4:** Observed and forecasted Bedon River flow (a) and La Plata River flow (b). The last 300 observations are shown together with 10-step-ahead forecasts and their associated 90% prediction intervals.

The `predict()` method for `mtar` objects generates point forecasts and predictive intervals for the multivariate response series. These forecasts are obtained from the posterior predictive distribution, thereby incorporating parameter uncertainty into the prediction process. Those forecasts are obtained from known or previously forecasted values for the input time series (if there is one in the fitted model) and the exogenous time series (if there is one in the fitted model) for the time points at which forecasts of the output time series are requested. This method can also be used to evaluate the out-of-sample predictive performance of a fitted model. For this purpose, it computes several forecast accuracy measures, including the Absolute Error (AE), Absolute Percentage Error (APE), and Squared Error (SE), which assess point forecast accuracy, as well as the log-score and Energy Score, which evaluate the quality of the predictive distribution. The main arguments of the `predict()` method are as follows:

- **object**. An `mtar`-type object resulting from a call to the routine `mtar()`, that is, an object in which the results of a fitted TAR model are stored.
- **newdata**. A `data.frame`-type object with rows representing time points, arranged in ascending order with respect to time, which includes the future values for the threshold time series (if there is one in the fitted model) and the exogenous time series (if there is one in the fitted model). The columns composing those series must be named equal to those used to call the routine `mtar()`.
- **n.ahead**. It is an integer value specifying the number of desired forecast steps.
- **row.names**. It is an optional column in `newdata` with the labels for the time points for which the forecast is requested.

- **credible**. The credible level for the prediction intervals. By default, `credible` is set to 0.95.
- **out.of.sample**. It is a logical type argument. If `TRUE` then the AE, APE, SE, log-score and Energy Score values are computed for each requested forecast step. Therefore, the `data.frame`-type object specified in the argument `newdata` must include the actual values of the output time series stored in columns whose names are equal to those used to call the routine `mtar()`.
- **rolling**. A positive integer specifying the size of the rolling window used for out-of-sample forecasting with fixed model parameters when `out.of.sample = TRUE`. By default, `rolling = NULL`, indicating that forecasts are generated without updating the estimation window.

The method `predict()` returns a list-type object that includes the following objects: $(i)$ `summary`, a matrix with the forecasts and their associated prediction intervals; and $(ii)$ the objects `LS`, `ES`, `AE`, `APE`, `SE`, `Width` and `CR` provided that `out.of.sample=TRUE`.

### Simulating from a TAR model

To illustrate the full flexibility of the simulation framework, we next describe the complete syntax of the `simtar()` function and present several alternative simulation settings.

```
> args(simtar)
function(n, k=2, ars=ars(), Intercept=TRUE, trend=c("none","linear","quadratic"),
         nseason=NULL, parms, delay=0, thresholds=NULL, t.series=NULL,
         ex.series=NULL, dist=c("Gaussian","Student-t","Hyperbolic","Laplace",
         "Slash","Contaminated normal","Skew-Student-t","Skew-normal"),
         skewness=NULL, extra=NULL, setar=NULL, Verbose=TRUE)
```

The function `simtar()` simulates multivariate time series whose behavior follows a pre-specified TAR model. As shown in Table 2, the routine is sufficiently flexible to also accommodate special cases such as SETAR and VAR models. In addition, it includes an optional logical argument that controls whether a summary description of the simulated TAR model is printed.

| | Argument | Type | Description |
|---|---|---|---|
| $T$ | `n` | Integer | Length of the time series |
| $k$ | `k` | Integer | Dimension of the output series |
| $l$ | | | |
| $\boldsymbol{p}=(p_1,\ldots,p_l)$ $\boldsymbol{q}=(q_1,\ldots,q_l)$ $\boldsymbol{d}=(d_1,\ldots,d_l)$ | `ars` | List | Same as the argument `ars` of `mtar()` |
| $h$ | `delay` | Non-negative integer | Delay parameter |
| $\boldsymbol{c}=(c_1,\ldots,c_{l-1})$ | `thresholds` | Vector of size $l-1$ | Threshold values for the input series |
| | `setar` | Integer in $\{1,\ldots,k\}$ | Same as the argument `setar` of `mtar()` |
| $z_t$ | `t.series` | Matrix of size $T^*\times 1$ † | Threshold series |
| $\boldsymbol{x}_t$ | `ex.series` | Matrix of size $T^*\times r$ † | Exogenous series |
| $\boldsymbol{\epsilon}_t \overset{\text{ind}}{\sim}$ | `dist` | Character | Same as the argument `dist` of `mtar()` |
| $\boldsymbol{\lambda}=(\lambda_1,\ldots,\lambda_k)^\top$ | `skewness` | Vector of size $k$ | Skewness parameter |
| $\nu$ | `extra` | Numeric | Extra parameter value |
| | `Intercept` | Logical | Same as the argument `Intercept` of `mtar()` |
| $\boldsymbol{\phi}_0^{(j)}$, $j=1,\ldots,l$ | `trend` | Character | Same as the argument `trend` of `mtar()` |
| | `nseason` | Integer | Same as the argument `nseason` of `mtar()` |
| $\boldsymbol{\phi}_0^{(j)},\boldsymbol{\phi}_1^{(j)},\ldots,\boldsymbol{\phi}_{p_j}^{(j)}$, $\boldsymbol{\beta}_1^{(j)},\ldots,\boldsymbol{\beta}_{q_j}^{(j)},\boldsymbol{\delta}_1^{(j)}$, $\ldots,\boldsymbol{\delta}_{d_j}^{(j)}$ and $\boldsymbol{\Sigma}_j$ for $j=1,\ldots,l$ | `parms` | List composed of $l$ lists ‡ | Location and scale parameter values |

† $T^*=T+\max\{p_1,\ldots,p_l,q_1,\ldots,q_l,d_1,\ldots,d_l,h\}$

‡ The $j$-th list, named `j`, is composed of 2 matrices. The first one, named `location`, stores the location parameters, that is, $(\boldsymbol{\phi}_0^{(j)},\boldsymbol{\phi}_1^{(j)},\ldots,\boldsymbol{\phi}_{p_j}^{(j)},\boldsymbol{\beta}_1^{(j)},\ldots,\boldsymbol{\beta}_{q_j}^{(j)},\boldsymbol{\delta}_1^{(j)},\ldots,\boldsymbol{\delta}_{d_j}^{(j)})^\top$. The second matrix, named `scale`, stores the scale parameters, that is, $\boldsymbol{\Sigma}_j$.

**Table 2:** Arguments of the `simtar()` function.

For example, the following code simulates a bivariate time series of size 2000 whose behavior follows a VAR(2) model with a linear time trend, four seasonal periods and a multivariate Laplace distribution describing the noise process. In addition, the elements of the matrix of location parameters

(i.e., $(\boldsymbol{\phi}_0, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2)^\top$) are randomly generated from $(1-2W) \times V$, where $W \sim$ Bernoulli$(0.5)$ and $V \sim$ Beta$(4, 16)$ are independent random variables, whereas the scale parameter (i.e., $\boldsymbol{\Sigma}$) is set as a diagonal matrix with elements randomly generated from the Exponential$(1)$.

```
> set.seed(0220)
> n <- 2000
> k <- 2
> myars <- ars(nregim=1, p=2)
> Intercept <- TRUE
> trend <- "linear"
> nseason <- 4
> dist <- "Laplace"
> deterministic <- Intercept + switch(trend,"linear"=1,"quadratic"=2,"none"=0) +
+                              ifelse(is.null(nseason),0,nseason-1)
> parms <- list()
> parms[[1]] <- list()
> np <- deterministic + myars$p*k
> parms[[1]]$location <- ifelse(runif(np*k)<=0.5,1,-1)*rbeta(np*k,shape1=4,shape2=16)
> parms[[1]]$location <- matrix(parms[[1]]$location,np,k)
> parms[[1]]$scale    <- rexp(k,rate=1)*diag(k)
>
> myVAR <- simtar(n=n, k=k, ars=myars, dist=dist, Intercept=Intercept, trend=trend,
+                 nseason=nseason, parms=parms, Verbose=TRUE)

Sample size          : 2000  time points
Output Series        : Y1 | Y2
Error Distribution   : Laplace
Number of regimes    : 1
Deterministics       : Intercept + a linear time trend + 4 seasonal periods
Autoregressive orders: 2 in each regime

Regime  1 :

Autoregressive coefficients
    Intercept    linear   season.2  season.3   season.4       phi_1                phi_2
Y1  0.10794066 -0.2298236 -0.2158606 0.2096798  0.1300396 | -0.4496798 -0.1437272 | 0.23212132
Y2 -0.06531457  0.1238318 -0.3082912 0.2128472 -0.3215392 | -0.2693481  0.1936919 | 0.06750432

Y1 0.06180898 |
Y2 0.20084720 |

Scale parameter
         Y1        Y2
Y1 2.856245 0.0000000
Y2 0.000000 0.6909536

> str(myVAR)
'data.frame':   2002 obs. of  3 variables:
 $ Y1    : num  -0.652 1.943 -14.48 5.515 -12.308 ...
 $ Y2    : num  0.694 -0.424 -4.698 4.514 -4.043 ...
 $ Regime: num  NA NA 1 1 1 1 1 1 1 ...
```

The model can be fitted to the simulated time series using the following instructions.

```
> fit <- mtar( ~ Y1 + Y2, data=myVAR, ars=myars, Intercept=Intercept, trend=trend,
+                nseason=nseason, dist=dist, n.burnin=2000, n.sim=3000, n.thin=2)
```

Similarly, the following code simulates a bivariate series $\{\boldsymbol{Y}_t\}_{t\geq 1}$, where $\boldsymbol{Y}_t = (Y_{1,t}, Y_{2,t})^\top$, according to a SETAR model with the following characteristics: $(i)$ there are 2 regimes; $(ii)$ the location parameter of the conditional distribution of $\boldsymbol{Y}_t$ is $\boldsymbol{\phi}_0^{(1)} + \boldsymbol{\phi}_1^{(1)} y_{t-1}$ in Regime 1, and $\boldsymbol{\phi}_0^{(2)} + \boldsymbol{\phi}_1^{(2)} y_{t-1} + \boldsymbol{\phi}_2^{(2)} y_{t-2}$ in Regime 2; $(iii)$ the threshold time series corresponds to the second component of $\boldsymbol{Y}_t$, that is, $Y_{2,t}$; $(iv)$ the threshold value is 0.3 and the delay parameter $h$ is 1, so that $\boldsymbol{Y}_t$ is in Regime 1 if $Y_{2,t-1} \leq 0.3$, and it is in Regime 2 if $Y_{2,t-1} > 0.3$; $(v)$ the noise process distribution is Student-$t$ with 4 degrees of freedom, that is, $\boldsymbol{\epsilon}_t \overset{\text{ind}}{\sim}$ Student-$t(\boldsymbol{0}, \boldsymbol{I}, 4)$, so that its mean and variance-covariance matrix are finite, which implies that both the conditional mean and the conditional variance-covariance matrix of $\boldsymbol{Y}_t$ are also finite; and

$(vi)$ the length of the series is 2000. The values of the location parameters of the first (i.e., $(\boldsymbol{\phi}_0^{(1)}, \boldsymbol{\phi}_1^{(1)})^\top$) and second (i.e., $(\boldsymbol{\phi}_0^{(2)}, \boldsymbol{\phi}_1^{(2)}, \boldsymbol{\phi}_2^{(2)})^\top$) regimes are stored in matrices named `parms[[1]]$location` and `parms[[2]]$location`, respectively. Similarly, the values of the scale parameters of the first (i.e., $\boldsymbol{\Sigma}_1$) and second (i.e., $\boldsymbol{\Sigma}_2$) regimes are stored in matrices named `parms[[1]]$scale` and `parms[[2]]$scale`, respectively.

```
> set.seed(0220)
> n <- 2000
> k <- 2
> myars <- ars(nregim=2, p=c(1,2))
> setar <- 2
> thresholds <- 0.3
> delay <- 1
> Intercept <- TRUE
> dist <- "Student-t"
> extra <- 4
> parms <- list()
> for(j in 1:myars$nregim){
+     np <- Intercept + myars$p[j]*k
+     parms[[j]] <- list()
+   parms[[j]]$location <- ifelse(runif(np*k)<=0.5,1,-1)*rbeta(np*k,shape1=4,shape2=16)
+     parms[[j]]$location <- matrix(parms[[j]]$location,np,k)
+     parms[[j]]$scale    <- rexp(k,rate=1)*diag(k)
> }
> mySETAR <- simtar(n=n, k=k, ars=myars, Intercept=Intercept, setar=setar, parms=parms,
+             thresholds=thresholds, delay=delay, dist=dist, extra=extra, Verbose=TRUE)

Sample size          : 2000  time points
Output Series        : Y1 | Y2
Threshold Series     : Y2 with a delay equal to 1
Error Distribution   : Student-t(4)
Number of regimes    : 2
Deterministics       : Intercept
Autoregressive orders: 1, 2

Thresholds
Regime 1 (-Inf,0.3]
Regime 2  (0.3,Inf)

Regime  1 :
Autoregressive coefficients
   Intercept        phi_1
Y1 0.3391083 | -0.1423164 -0.1146139 |
Y2 0.2180902 |  0.1641546 -0.1079407 |

Scale parameter
         Y1        Y2
Y1 3.774202 0.0000000
Y2 0.000000 0.3872779

Regime  2 :
Autoregressive coefficients
    Intercept        phi_1                  phi_2
Y1 -0.1898829 | -0.2227718  0.2326686 | 0.1879023 0.09084321 |
Y2 -0.1567592 |  0.3272486 -0.1928849 | 0.2885574 0.29326418 |

Scale parameter
          Y1        Y2
Y1 0.6175893 0.0000000
Y2 0.0000000 0.7249159

> str(mySETAR)
'data.frame':   2002 obs. of  3 variables:
 $ Y1    : num  0.505 0.307 1.911 -1.652 0.468 ...
```

```
 $ Y2    : num  -2.357 0.188 0.452 -1.218 -1.021 ...
 $ Regime: num  NA NA 1 2 1 1 1 1 1 1 ...
```

The simulated time series is stored in a `data-frame` named `mySETAR`. The SETAR($\boldsymbol{p} = (1,2)$) Student-$t$ model can be fitted to the simulated time series using the following instructions.

```
> fit <- mtar( ~ Y1 + Y2, data=mySETAR, ars=myars, setar=setar, dist=dist,
+              n.burnin=2000, n.sim=3000, n.thin=2)
```

## 4 River flows in Iceland

The data consist of two daily river flow series from Iceland, originally reported by Tong (1990), and are available in the `iceland.rf` dataset included in **mtarm**. The corresponding variables are summarized in Table 3. While Tong (1990) described the geographical and meteorological characteristics of the rivers and analyzed each series separately, Tsay (1998) later adopted a multivariate perspective by jointly modeling both river flow series. The response vector $\boldsymbol{Y}_t = (Y_{1,t}, Y_{2,t})^\top$ consists of the daily flows of the Jökulsá Eystri $(Y_{1,t})$ and Vatnsdalsá $(Y_{2,t})$ rivers, measured in cubic meters per second $(m^3/s)$, over the period from January 1, 1972, to December 31, 1974, yielding a total of 1096 observations. The explanatory variables include daily precipitation $(X_t)$, measured in millimeters $(mm)$, and daily temperature $(Z_t)$, measured in degrees Celsius (°C), both recorded at the Hveravellir meteorological station. Precipitation values correspond to the total rainfall accumulated during the preceding 24 hours and recorded at 9:00 A.M. Given the subarctic climate of the region, river flow dynamics are expected to be strongly influenced by temperature-dependent processes such as snow accumulation and snowmelt. Consequently, temperature $(Z_t)$ is a natural choice for the threshold variable in a multivariate TAR model, allowing the relationship between river flows and meteorological conditions to vary across distinct thermal regimes. To assess the benefits of incorporating such nonlinearities, standard VAR models are also considered as linear benchmark specifications. Model comparison is based on both in-sample fit criteria and out-of-sample predictive performance, thereby providing a comprehensive evaluation of the competing models.

| Column | Role | Description |
|---|---|---|
| `Date` | Labels for time points | Dates when measurements were taken |
| `Jokulsa` | First component of the output series | Jökulsá Eystri river flow, in $m^3/s$ |
| `Vatnsdalsa` | Second component of the output series | Vatnsdalsá river flow, in $m^3/s$ |
| `Precipitation` | Exogenous series | Rainfall, in $mm$ |
| `Temperature` | Threshold series | Temperature, in degrees Celsius (°C) |

**Table 3:** Description of the variables in the `iceland.rf` dataset included in the **mtarm** package.

Similar to Tsay (1998), the observed series are analyzed here using a TAR$(2; p = (15,15), q = (4,4), d = (2,2))$ model, that is,

$$\boldsymbol{Y}_t = \sum_{j=1}^{2} I(Z_{t-h} \in (c_{j-1}, c_j]) \Big( \boldsymbol{\phi}_0^{(j)} + \sum_{i=1}^{15} \boldsymbol{\phi}_i^{(j)} \boldsymbol{Y}_{t-i} + \sum_{i=1}^{4} \boldsymbol{\beta}_i^{(j)} X_{t-i} + \sum_{i=1}^{2} \boldsymbol{\delta}_i^{(j)} Z_{t-i} + \boldsymbol{\epsilon}_{t_j} \Big).$$

### 4.1 Model estimation

The following code fits a TAR$(2; \boldsymbol{p} = (15,15), \boldsymbol{q} = (4,4), \boldsymbol{d} = (2,2))$ model to the output series under four alternative assumptions for the noise process distribution: Gaussian, Student-$t$, skew-normal, and skew-$t$. In each specification, the relevant lags of the output series $\{\boldsymbol{Y}_t\}_{t\geq 1}$, the exogenous series $\{X_t\}_{t\geq 1}$, and the threshold series $\{Z_t\}_{t\geq 1}$ are selected using the Stochastic Search Variable Selection (SSVS) procedure. Furthermore, the last 10 observations of the sample period (from December 22, 1974, to December 31, 1974) are excluded from the estimation stage, as they are subsequently used to evaluate out-of-sample predictive performance.

```
> data(iceland.rf)
> str(iceland.rf)
'data.frame':   1096 obs. of  5 variables:
 $ Vatnsdalsa   : num  16.1 19.2 14.5 11 13.6 12.5 10.5 10.1 9.68 9.02 ...
 $ Jokulsa      : num  30.2 29 28.4 27.8 27.8 27.8 27.8 27.8 27.8 27.3 ...
 $ Precipitation: num  8.1 4.4 7 0 0 0 1.9 1.2 0 0.1 ...
 $ Temperature  : num  0.9 1.6 0.1 0.6 2 0.8 1.4 1.3 2.2 0.1 ...
```

```
 $ Date         : Date, format: "1972-01-01" "1972-01-02" "1972-01-03" ...
>
> set.seed(0220)
> models <- mtar_grid(~ Jokulsa + Vatnsdalsa | Temperature | Precipitation, data=iceland.rf,
+                     subset={Date<="1974-12-21"}, row.names=Date, dist=c("Gaussian",
+               "Student-t","Skew-normal","Skew-Student-t"), nregim.min=2, nregim.max=2,
+                 p.min=15, p.max=15, q.min=4, q.max=4, d.min=2, d.max=2, n.burnin=5000,
+                     n.sim=4000, n.thin=2, ssvs=TRUE, plan_strategy="multisession")
>
> models

Sample size          : 1071 time points (1972-01-16 to 1974-12-21)
Output Series        : Jokulsa    |    Vatnsdalsa
Threshold Series (TS): Temperature
Exogenous Series (ES): Precipitation
Error Distribution   : Gaussian, Skew-normal, Skew-Student-t, Student-t
Number of regimes    : 2
Deterministics       : Intercept
Autoregressive order : 15
Maximum lag for ES   : 4
Maximum lag for TS   : 2
```

### 4.2 Model selection

For each fitted model, both in-sample and out-of-sample predictive performance measures are computed. The in-sample measures include the DIC and WAIC, whereas the out-of-sample measures consist of the average log-score, the average Energy Score, the empirical coverage rate, and the average width of the 95% prediction intervals.

```
> DICs <- DIC(models)
> WAICs <- WAIC(models)
>
> round(cbind(DICs,WAICs),2)
                             DIC     WAIC
Gaussian.2.15.4.2       10263.25 10489.02
Skew-normal.2.15.4.2     9530.98  9747.58
Skew-Student-t.2.15.4.2  7660.34  7687.53
Student-t.2.15.4.2       7627.35  7724.44

> future.obs <- subset(iceland.rf, Date>"1974-12-21")
> set.seed(0220)
> oos <- out_of_sample(models, credible=0.95, newdata=future.obs,
+                      n.ahead=nrow(future.obs), FUN=mean)
>
> round(oos[,c(1,2,9:12)], 2)
             Log.Score Energy.Score Jokulsa.Width Vatnsdalsa.Width Jokulsa.CR Vatnsdalsa.CR
Gaussian.2.15.4.2       3.02        1.69         9.63           6.08         1            1
Skew-normal.2.15.4.2    2.56        1.15         6.46           3.91         1            1
Skew-Student-t.2.15.4.2 3.96        1.95        11.28           9.09         1            1
Student-t.2.15.4.2      2.61        1.70        10.33           8.29         1            1
```

The competing models exhibit some variation in their rankings across the evaluation criteria. According to the DIC and WAIC, the specification with skew-Student-$t$/Student-$t$ innovations provides the best overall fit, with the Student-$t$/skew-Student-$t$ model performing nearly as well. In contrast, both the average log-score and the average Energy Score favor the specification with skew-normal innovations, while the Student-$t$ model consistently ranks second. The comparison based on the 95% prediction intervals is less informative because all models achieve complete empirical coverage. Nevertheless, the skew-normal specification produces substantially narrower prediction intervals on average, indicating greater predictive sharpness without compromising calibration. Overall, the results reveal a trade-off between in-sample fit and out-of-sample predictive performance. Although the skew-Student-$t$ and Student-$t$ specifications are preferred according to the information criteria, the skew-normal specification yields superior predictive accuracy together with more precise prediction intervals. Given the emphasis on forecasting performance, the skew-normal specification is selected for the subsequent analysis.

### 4.3 Overview of the chosen model

The preferred model is summarized below.

```
> summary(models[["Skew-normal.2.15.4.2"]])

Sample size          : 1071 time points (1972-01-16 to 1974-12-21)
Output Series (OS)   : Jokulsa    |    Vatnsdalsa
Threshold Series (TS): Temperature with a estimated delay equal to 0
Exogenous Series (ES): Precipitation
Error Distribution   : Skew-normal
Number of regimes    : 2
Deterministics       : Intercept
Autoregressive orders: 15 in each regime
Maximum lags for ES  : 4 in each regime
Maximum lags for TS  : 2 in each regime

Thresholds (Mean, HDI.Lower, HDI.Upper)
Regime 1 (-Inf,-0.45131] (-Inf,-0.4994] (-Inf,-0.40566]
Regime 2  (-0.45131,Inf)  (-0.4994,Inf)  (-0.40566,Inf)


Regime1:
    OS.lag(1)  OS.lag(2)  OS.lag(3)  OS.lag(4)  OS.lag(5)  OS.lag(6)  OS.lag(7) OS.lag(8)
SSVS         1          1          1          0          0          0          0         0
    OS.lag(9) OS.lag(10) OS.lag(11) OS.lag(12) OS.lag(13) OS.lag(14) OS.lag(15) ES.lag(1)
            0          0          1          0          0          0          0         0
     ES.lag(2)  ES.lag(3)  ES.lag(4)  TS.lag(1)  TS.lag(2)
          0.02          0          0          0          0

Autoregressive coefficients
                   Mean  2(1-PD) HDI.Lower HDI.Upper    Mean  2(1-PD) HDI.Lower HDI.Upper
(Intercept)        7.82970 0.00001  7.05110  8.60872 |  2.16764 0.00001  1.85504  2.53065
Jokulsa.lag( 1)    0.51454 0.00001  0.43375  0.59311 | -0.05485 0.00001 -0.08455 -0.02651
Vatnsdalsa.lag( 1)  0.41706 0.00001  0.29846  0.53476 |  0.87652 0.00001  0.81374  0.93883
Jokulsa.lag( 2)    0.00961 0.82000 -0.06125  0.08595 |  0.02573 0.13150 -0.00644  0.06288
Vatnsdalsa.lag( 2) -0.20241 0.00200 -0.33230 -0.06691 | -0.20115 0.00001 -0.26561 -0.13553
Jokulsa.lag( 3)    0.12441 0.00001  0.07372  0.17434 |  0.00598 0.63100 -0.01854  0.03100
Vatnsdalsa.lag( 3) -0.08044 0.07800 -0.17010  0.01040 |  0.09105 0.00100  0.04060  0.14201
Jokulsa.lag(11)    0.05987 0.00001  0.03537  0.08581 |  0.00537 0.42750 -0.00744  0.01843
Vatnsdalsa.lag(11) -0.04732 0.01400 -0.08295 -0.01176 |  0.05106 0.00001  0.02942  0.06887

Scale parameter (Mean, HDI.Lower, HDI.Upper)
           Jokulsa Vatnsdalsa      Jokulsa Vatnsdalsa      Jokulsa Vatnsdalsa
Jokulsa    1.29030    0.15385    . 0.98590    0.07924    . 1.57325    0.23292
Vatnsdalsa 0.15385    0.20602    . 0.07924    0.14868    . 0.23292    0.26663

Regime2:
    OS.lag(1)  OS.lag(2)  OS.lag(3)  OS.lag(4)  OS.lag(5)  OS.lag(6)  OS.lag(7) OS.lag(8)
SSVS         1          1        0.3        0.7          0          0          0         0
    OS.lag(9) OS.lag(10) OS.lag(11) OS.lag(12) OS.lag(13) OS.lag(14) OS.lag(15) ES.lag(1)
            0          0          0          0          0          0          0      0.01
     ES.lag(2)  ES.lag(3)  ES.lag(4)  TS.lag(1)  TS.lag(2)
            0          0          0          1          1

Autoregressive coefficients
                   Mean  2(1-PD) HDI.Lower HDI.Upper    Mean  2(1-PD) HDI.Lower HDI.Upper
(Intercept)        3.61496 0.00001  1.74559  5.40196 |  2.43909 0.00001  1.84246  3.04396
Jokulsa.lag( 1)    1.04322 0.00001  0.95892  1.13108 |  0.00106 0.94150 -0.02486  0.02917
Vatnsdalsa.lag( 1)  0.79057 0.00001  0.50879  1.05482 |  1.21472 0.00001  1.11489  1.32856
Jokulsa.lag( 2)   -0.22844 0.00001 -0.38460 -0.10183 | -0.01011 0.53900 -0.04316  0.02164
Vatnsdalsa.lag( 2) -0.99689 0.00001 -1.42281 -0.58657 | -0.53671 0.00001 -0.76564 -0.37173
Jokulsa.lag( 4)   -0.02972 0.22750 -0.06799  0.02291 | -0.02065 0.02300 -0.03485 -0.00411
Vatnsdalsa.lag( 4)  0.52088 0.00001  0.36035  0.72024 |  0.26715 0.00001  0.20760  0.32753
Temperature.lag(1)  1.71978 0.00001  1.34312  2.08671 |  0.11227 0.07350 -0.00177  0.24110
```

```
Temperature.lag(2) -0.87721 0.00001 -1.22428 -0.57624 | -0.12042 0.02200 -0.22141 -0.01141

Scale parameter (Mean, HDI.Lower, HDI.Upper)
           Jokulsa Vatnsdalsa       Jokulsa Vatnsdalsa       Jokulsa Vatnsdalsa
Jokulsa    60.1243    5.01920    . 53.21215    3.37068    . 66.85740    6.81176
Vatnsdalsa  5.0192    6.34542    .  3.37068    5.62933    .  6.81176    7.14503

Skewness parameter
                        Mean    2(1-PD)  HDI.Lower HDI.Upper
lambda1              -1.13231   0.00150  -1.48391  -0.75087
lambda2              -0.86829   0.00001  -0.99011  -0.73948
```

The posterior mean estimates of the delay and threshold parameters for the selected model are 0 and $-0.45$, respectively. These estimates suggest that the dynamics of the daily river flows depend on contemporaneous temperature values and on whether the temperature lies below or above the threshold of $-0.45\,^{\circ}$C. Accordingly, Regime 1 corresponds to days with temperatures not exceeding $-0.45\,^{\circ}$C, whereas Regime 2 corresponds to days with temperatures above this level. The observations are distributed almost evenly across the two regimes, with 470 observations assigned to Regime 1 and 601 to Regime 2. Examination of the posterior means of the variable-selection indicators $\boldsymbol{\zeta}_1$ and $\boldsymbol{\zeta}_2$ reveals notable differences in the dynamic structure across regimes. In Regime 1, only the first, second and eleventh lags of $\{\boldsymbol{Y}_t\}_{t\geq 1}$ are selected, indicating that the current river flows are primarily explained by their recent past values. In contrast, Regime 2 exhibits a richer dependence structure: in addition to the first, second and fourth lags of $\{\boldsymbol{Y}_t\}_{t\geq 1}$, the first and second lags of the temperature series $\{Z_t\}_{t\geq 1}$ are also selected. Furthermore, the posterior means and corresponding 95% credible intervals of the skewness parameters $\lambda_1$ and $\lambda_2$ provide evidence of negative skewness in the marginal conditional distributions of both river-flow series, thereby supporting the use of a skewed innovation distribution. Overall, while river flows in the colder regime are primarily driven by their own persistence, the warmer regime reflects a more complex interplay between hydrological and meteorological factors, together with distributional asymmetries in the flow process.

### 4.4 Forecasting

The following code allows us to obtain point and by-interval forecasts of the output time series for the 10 time points excluded from the estimation stage.

```
> set.seed(0220)
> out <- predict(models[["Skew-normal.2.15.4.2"]], newdata=future.obs, credible=0.95,
+                n.ahead=nrow(future.obs), row.names=Date)
> round(cbind(out[["summary"]][,1:3], Jokulsa=future.obs[,"Jokulsa"]),2)

           Jokulsa.Median Jokulsa.Lower Jokulsa.Upper Jokulsa
1974-12-22          24.73         21.94         27.21    24.6
1974-12-23          24.88         21.78         27.83    24.6
1974-12-24          25.03         21.85         28.10    24.6
1974-12-25          25.10         21.55         28.25    24.6
1974-12-26          25.13         21.89         28.62    24.6
1974-12-27          25.17         21.78         28.48    24.6
1974-12-28          25.19         21.74         28.36    24.6
1974-12-29          25.24         21.85         28.61    24.6
1974-12-30          25.22         21.73         28.57    24.6
1974-12-31          25.25         21.86         28.54    25.7
>
> round(cbind(out[["summary"]][,4:6], Vatnsdalsa=future.obs[,"Vatnsdalsa"]),2)

           Vatnsdalsa.Median Vatnsdalsa.Lower Vatnsdalsa.Upper Vatnsdalsa
1974-12-22              4.87             3.45             6.09       5.16
1974-12-23              5.07             3.16             6.67       5.16
1974-12-24              5.15             3.10             6.99       5.16
1974-12-25              5.16             3.09             7.22       5.16
1974-12-26              5.27             3.12             7.28       5.16
1974-12-27              5.33             3.16             7.36       5.16
1974-12-28              5.37             3.21             7.45       5.16
1974-12-29              5.44             3.31             7.61       5.16
1974-12-30              5.46             3.32             7.48       5.34
1974-12-31              5.48             3.40             7.55       5.34
```

## Conclusions

We have introduced **mtarm**, an R package for Bayesian estimation, inference, simulation, and forecasting in multivariate Threshold Autoregressive (TAR) models, including multivariate SETAR and VAR specifications as special cases. The package supports a broad range of innovation distributions beyond the Gaussian assumption, including Student-$t$, slash, symmetric hyperbolic, Laplace, contaminated normal, skew-normal, and skew-$t$ distributions. It also provides tools for posterior predictive forecasting, model simulation, and model evaluation through a variety of in-sample and out-of-sample predictive criteria, including DIC, WAIC, AE, APE, SE, log-score, and Energy Score. Model parameters, with the exception of the number of regimes, are estimated within a Bayesian framework using an MCMC algorithm. The resulting posterior samples can be readily analyzed using functions available in **mtarm** as well as diagnostic and summary tools from **coda**. By combining multivariate threshold structures, Bayesian inference, flexible innovation distributions, and a unified set of procedures for estimation, model comparison, diagnostics, and forecasting, **mtarm** extends the range of tools available for nonlinear time-series analysis in R. The package's capabilities were illustrated through two empirical applications involving multivariate hydrological time series. These examples demonstrate how threshold effects and flexible innovation distributions can provide a richer characterization of complex dynamic processes and, in some cases, lead to improved predictive performance relative to more restrictive model specifications.

## Acknowledgements

The author would like to thank the Associate Editor and the anonymous reviewers for their valuable comments and suggestions. Their careful assessment of the manuscript and constructive feedback substantially improved both the paper and the **mtarm** package.

## 1 Appendix: Noise distribution

- **Slash distribution**. If $\boldsymbol{Y} \sim \text{Slash}_k(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu)$ then its probability density function is given by (Wang and Genton, 2006; Ghayour Moradi et al., 2017)

$$f_Y(\boldsymbol{y}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu) = \begin{cases} \dfrac{\nu}{(2\pi)^{\frac{k}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}(\nu+k)} & \text{if} \quad \boldsymbol{y} = \boldsymbol{\mu} \\ \dfrac{\frac{\nu}{2}\gamma(\frac{k+\nu}{2}, \frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{y}-\boldsymbol{\mu}))}{(2\pi)^{\frac{k}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}\left[\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})\right]^{\frac{k+\nu}{2}}} & \text{if} \quad \boldsymbol{y} \neq \boldsymbol{\mu}, \end{cases}$$

  where $\nu > 0$ and $\gamma(a, b) = \int_0^b t^{a-1}\exp(-t)dt$ represents the lower incomplete gamma function. The multivariate Slash distribution can be expressed as a scale mixture of a multivariate normal distribution with an independent Beta mixing variable, which induces an elliptically contoured density with substantially heavier tails than the Gaussian case. While the distribution remains approximately normal around its centre, the mixing mechanism generates pronounced tail thickness and very high kurtosis, occasionally approaching tail behaviour comparable to that of the multivariate Cauchy distribution. This combination of local Gaussian behaviour and extreme tail flexibility allows the multivariate Slash distribution to accommodate infrequent but severe joint shocks, making it particularly appealing as an error specification in models exhibiting impulsive dynamics or sporadic extreme events. Empirical and Bayesian studies have shown that, in such settings, Slash-type innovations can provide greater robustness than both Gaussian and Student-$t$ errors, especially when extreme observations occur with low frequency but large magnitude.

- **Contaminated Gaussian distribution**. If $\boldsymbol{Y} \sim \text{CG}_k(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu_1, \nu_2)$ then its probability density function is given by (Tukey, 1960; Punzo and Tortora, 2021)

$$f_Y(\boldsymbol{y}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu_1, \nu_2) = \nu_1 \frac{\nu_2^{\frac{k}{2}}}{(2\pi)^{\frac{k}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} \exp\left(-\frac{\nu_2}{2}(\boldsymbol{y}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})\right) + (1-\nu_1)\frac{1}{(2\pi)^{\frac{k}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})\right),$$

where $\nu_1, \nu_2 \in (0,1)$. The contaminated multivariate normal distribution is an elliptically contoured extension of the multivariate normal obtained by mixing a "good" component, $\text{Normal}_k(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with a small proportion of "bad" observations drawn from a more dispersed component, $\text{Normal}_k(\boldsymbol{\mu}, \frac{1}{\nu_2}\boldsymbol{\Sigma})$, where $\nu_2 \in (0,1)$. Under this parametrization, $1-\nu_1$ for $\nu_1 \in (0,1)$, represents the approximate proportion of clean data, $\nu_1$ the proportion of contaminated observations, and $1/\nu_2$ the degree of variance inflation in the contaminated component (Tukey, 1960; Punzo and Tortora, 2021). Intuitively, the model behaves almost like a multivariate normal for the bulk of the data, while automatically down-weighting suspicious observations through the inflated-variance component. This makes it a convenient error specification in time-series settings where one anticipates occasional mild outliers rather than extremely heavy-tailed behaviour, and where robust inference is desired without departing dramatically from Gaussian assumptions (Wang et al., 2009; Abanto-Valle et al., 2011, 2012; Punzo and Tortora, 2021).

- **Symmetric hyperbolic distribution**. If $\boldsymbol{Y} \sim \text{SH}_k(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu)$ then its distribution is a special case of the multivariate symmetric generalized hyperbolic distribution described in McNeil et al. (2015, example 6.8), and its probability density function is given by

$$f_Y(\boldsymbol{y}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu) = \frac{\nu^{\frac{k}{2}} K_{\frac{2-k}{2}}\left(\nu\sqrt{1+(\boldsymbol{y}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})}\right)\left(\sqrt{1+(\boldsymbol{y}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(y-\mu)}\right)^{\frac{2-k}{2}}}{K_1(\nu)(2\pi)^{\frac{k}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}},$$

where $\nu > 0$ and $K_a(b) = \int_0^\infty x^{a-1}\exp\left(-\frac{b}{2}(x+x^{-1})\right)dx$ is the modified Bessel function of third-order and index $a$ (see, for example, Abramowitz and Stegun (1965, section 9.6)). Under this construction, the random vector $\boldsymbol{Y}$ belongs to the (generalized) hyperbolic family originally introduced by Barndorff-Nielsen (1977), which arises as a normal mean–variance mixture with a generalized inverse Gaussian mixing variable (McNeil et al., 2015, Chapter 6). This representation yields an elliptically contoured distribution whose log-density exhibits approximately hyperbolic decay in the tails. The shape parameter $\nu > 0$ provides direct control over excess kurtosis relative to the multivariate normal, allowing for heavier tails while preserving a unimodal, bell-shaped density (Barndorff-Nielsen, 1977; Barndorff-Nielsen and Blæsild, 1983). In practical terms, the symmetric hyperbolic distribution behaves similarly to a multivariate Gaussian near the centre of the distribution but assigns substantially higher probability to moderate-to-large joint deviations. This added flexibility has been shown to produce markedly improved fits for financial returns and other economic time series characterized by pronounced kurtosis and tail risk (Eberlein and Keller, 1995; Bibby and Sørensen, 2003).

- **Laplace distribution**. If $\boldsymbol{Y} \sim \text{Laplace}_k(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ then its probability density function is given by (Kotz et al., 2001, page 234)

$$f_Y(\boldsymbol{y}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{2^{k+1}\pi^{\frac{k}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}}\left(\sqrt{(\boldsymbol{y}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})}\right)^{\frac{2-k}{2}} K_{\frac{2-k}{2}}\left(\sqrt{(\boldsymbol{y}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})}\right), \qquad \boldsymbol{y} \neq \boldsymbol{\mu}.$$

The multivariate Laplace distribution admits a convenient representation as a normal variance mixture, obtained by scaling a multivariate Gaussian vector by an independent exponential random variable (Andrews and Mallows, 1974; Eltoft et al., 2006). This construction yields an elliptically contoured distribution with finite second moments and a covariance matrix proportional to the scale matrix $\boldsymbol{\Sigma}$, while exhibiting substantially heavier tails than the multivariate normal distribution (Eltoft et al., 2006, Sec. II.A). When standardized to unit variance, the symmetric Laplace law is characterized by a sharper central peak and pronounced leptokurtosis relative to the Gaussian case, reflecting its greater concentration of probability mass both near the mean and in the tails (Kotz et al., 2001, Ch. 2, Sec. 2.1.3.4). From a time-series perspective, these features make Laplace innovations particularly attractive in robust modeling frameworks, as they naturally accommodate occasional jumps or double-exponential–type disturbances. Such specifications have been shown to provide improved empirical performance in ARMA and related models when the data exhibit impulsive behavior or outliers (Damsleth and El-Shaarawi, 1989; Kotz et al., 2001).

- **Skew-normal distribution**. If $\boldsymbol{Y} \sim \text{SN}_k(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\lambda})$ then its probability density function is given by (Sahu et al., 2003, page 133)

$$f_Y(\boldsymbol{y}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\lambda}) = 2^k \frac{\exp\left(-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu})^\top(\boldsymbol{\Sigma}+\boldsymbol{D}_{(\lambda)}^2)^{-1}(\boldsymbol{y}-\boldsymbol{\mu})\right)}{(2\pi)^{\frac{k}{2}}|\boldsymbol{\Sigma}+\boldsymbol{D}_{(\lambda)}^2|^{\frac{1}{2}}}\Pr(\boldsymbol{V}>0),$$

where $\boldsymbol{V} \sim \text{Normal}_k\left(\boldsymbol{D}_{(\lambda)}(\boldsymbol{\Sigma}+\boldsymbol{D}_{(\lambda)}^2)^{-1}(\boldsymbol{y}-\boldsymbol{\mu}), \boldsymbol{I}_k - \boldsymbol{D}_{(\lambda)}(\boldsymbol{\Sigma}+\boldsymbol{D}_{(\lambda)}^2)^{-1}\boldsymbol{D}_{(\lambda)}\right)$, with $\boldsymbol{D}_{(\lambda)} = \text{diag}(\boldsymbol{\lambda})$ and

$\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_k)^\top$. In this parametrization, the shape vector $\boldsymbol{\lambda}$ governs both the direction and magnitude of skewness in the distribution. When $\boldsymbol{\lambda} = \mathbf{0}$, one obtains $\boldsymbol{D}_{(\lambda)} = \mathbf{0}$ and $\Pr(\boldsymbol{V} > 0) = 2^{-k}$, so that the model reduces to the multivariate normal distribution. From a constructive perspective, the multivariate skew-normal distribution can be interpreted as starting from an elliptically contoured Gaussian kernel with covariance matrix $\boldsymbol{\Sigma}$ and then tilting its probability mass along the direction induced by $\boldsymbol{\lambda}$ (Azzalini, 2013). This transformation preserves light, Gaussian-type tail behaviour while introducing asymmetry, thereby increasing the likelihood of moderate and large realizations on one side of the distribution without inducing heavy tails. Such flexibility is particularly appealing in time-series applications where residuals exhibit mild but statistically significant departures from symmetry, yet do not display extreme kurtosis, as in autoregressive or ARMA models with skew-normal innovations (Pourahmadi, 2007; Azzalini, 2013).

## 2 Appendix: Hyperparameters

**Practical note on hyperparameter specification.** The hyperparameters controlling the threshold and delay search spaces can influence both estimation stability and computational performance. In particular, the percentile bounds $\alpha_0$ and $\alpha_1$ determine the range of admissible threshold values. By default, $\alpha_0 = 0$ and $\alpha_1 = 1$, allowing the threshold parameter to be explored over the full range of observed values of the threshold variable. While this choice maximizes flexibility, it may occasionally produce extreme regimes containing relatively few observations, especially in small samples or when the threshold variable exhibits outlying values. In such situations, users may consider restricting the threshold search space by selecting more conservative percentile bounds. Similarly, the delay bounds $h_{\min}$ and $h_{\max}$ determine the set of candidate delay values considered during estimation. The default values, $h_{\min} = 0$ and $h_{\max} = 3$, provide a moderate search range while preserving a sufficiently large effective sample size. Increasing $h_{\max}$ enlarges the search space and may improve the ability to detect longer delay effects, but it also increases computational cost and reduces the number of observations available for estimation. Consequently, whenever substantive knowledge or exploratory analyses suggest a plausible range of delays, users may benefit from restricting the search accordingly. More generally, as with the choice of the number of regimes, broader search spaces increase model flexibility but may also lead to higher computational costs and greater uncertainty in parameter estimation.

## References


C. Abanto-Valle, H. Migon, and V. Lachos. Stochastic volatility in mean models with scale mixtures of normal distributions and correlated errors: A bayesian approach. *Journal of Statistical Planning and Inference*, 141(5):1875–1887, 2011. ISSN 0378-3758. doi: https://doi.org/10.1016/j.jspi.2010.11.039. URL https://www.sciencedirect.com/science/article/pii/S0378375810005392. [p195]

C. A. Abanto-Valle, D. Bandyopadhyay, V. H. Lachos, and I. Enriquez. Robust bayesian analysis of heavy-tailed stochastic volatility models using scale mixtures of normal distributions. *Computational Statistics & Data Analysis*, 54(12):2883–2898, 2010. doi: 10.1016/j.csda.2009.06.011. [p170]

C. A. Abanto-Valle, H. S. Migon, and V. H. Lachos. Stochastic volatility in mean models with heavy-tailed distributions. *Brazilian Journal of Probability and Statistics*, 26(4):402–422, 2012. doi: 10.1214/11-BJPS169. [p170, 195]

C. A. Abanto-Valle, V. H. Lachos, and D. K. Dey. Bayesian estimation of a skew-student-*t* stochastic volatility model. *Methodology and Computing in Applied Probability*, 17(3):721–738, 2015. doi: 10.1007/s11009-013-9389-9. [p170]

M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables*. Applied mathematics series. Dover Publications, 1965. ISBN 9780486612720. URL https://books.google.com.co/books?id=MtU8uP7XMvoC. [p195]

D. F. Andrews and C. L. Mallows. Scale mixtures of normal distributions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(1):99–102, 1974. doi: 10.1111/j.2517-6161.1974.tb00989.x. [p195]

A. Azzalini. *The Skew-Normal and Related Families*. Institute of Mathematical Statistics Monographs. Cambridge University Press, 2013. [p196]

O. E. Barndorff-Nielsen. Exponentially decreasing distributions for the logarithm of particle size. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 353(1674):401–419, 1977. doi: 10.1098/rspa.1977.0041. [p195]

O. E. Barndorff-Nielsen and P. Blæsild. Hyperbolic distributions and ramifications: Contributions to theory and application. In C. Taillie, G. P. Patil, and B. A. Baldessari, editors, *Statistical Distributions in Scientific Work*, volume 4, pages 19–44. Reidel, Dordrecht, 1983. [p195]

H. Bengtsson. *future: Unified Parallel and Distributed Processing in R for Everyone*, 2024a. URL https://CRAN.R-project.org/package=future. R package version 1.33.2. [p177]

H. Bengtsson. *future.apply: Apply Functionals to Futures*, 2024b. URL https://CRAN.R-project.org/package=future.apply. R package version 1.11.0. [p177]

B. M. Bibby and M. Sørensen. Hyperbolic processes in finance. In S. T. Rachev, editor, *Handbook of Heavy Tailed Distributions in Finance*, pages 211–248. Elsevier, Amsterdam, 2003. [p195]

S. A. Calderón and F. H. Nieto. Bayesian analysis of multivariate threshold autoregressive models with missing data. *Communications in Statistics - Theory and Methods*, 46(1):296–318, 2017. [p172, 175, 182]

C. Chen, M. So, and F.-C. Liu. A review of threshold time series models in finance. *Statistics and Its Interface*, 4(2):167–181, 2011. ISSN 1938-7997. doi: 10.4310/SII.2011.v4.n2.a12. [p167]

C. W. S. Chen, E. M. H. Lin, F. C. Liu, and R. Gerlach. *BAYSTAR: On Bayesian Analysis of Threshold Autoregressive Models*, 2022. URL https://CRAN.R-project.org/package=BAYSTAR. R package version 0.2-10. [p168]

E. Damsleth and A. H. El-Shaarawi. Arma models with double-exponentially distributed noise. *Journal of the Royal Statistical Society: Series B (Methodological)*, 51(1):61–69, 1989. doi: 10.1111/j.2517-6161.1989.tb01748.x. [p195]

P. K. Dunn and G. K. Smyth. Randomized quantile residuals. *Journal of Computational and Graphical Statistics*, 5(3):236–244, 1996. doi: 10.2307/1390802. [p176]

E. Eberlein and U. Keller. Hyperbolic distributions in finance. *Bernoulli*, 1(3):281–299, 1995. doi: 10.3150/bj/1193667819. [p195]

T. Eltoft, T. Kim, and T.-W. Lee. On the multivariate laplace distribution. *IEEE Signal Processing Letters*, 13(5):300–303, 2006. doi: 10.1109/LSP.2006.870353. [p195]

K.-T. Fang, S. Kotz, and K. NG. *Symmetric Multivariate and Related Distributions*. CRC Press, 2018. [p170]

E. I. George and R. E. McCulloch. Variable selection via gibbs sampling. *Journal of the American Statistical Association*, 88(423):881–889, 1993. ISSN 01621459, 1537274X. [p169]

E. I. George and R. E. McCulloch. *Stochastic search variable selection*, pages 203–214. Chapman & Hall/CRC Interdisciplinary Statistics. CRC Press, 1995. ISBN 9781482214970. [p169]

E. I. George and R. E. McCulloch. Approaches for bayesian variable selection. *Statistica Sinica*, 7(2): 339–373, 1997. ISSN 1017-0405. [p169]

J. Geweke. Evaluating the Accuracy of Sampling-Based Approaches to the Calculation of Posterior Moments. In *Bayesian Statistics 4: Proceedings of the Fourth Valencia International Meeting, Dedicated to the memory of Morris H. DeGroot, 1931–1989*. Oxford University Press, 08 1992. [p182]

Z. Ghayour Moradi, M. Arashi, O. Arslan, and A. Iranmanesh. A new family of multivariate slash distributions. *Communications in Statistics - Theory and Methods*, 46:3264–3275, 2017. [p194]

S. Giannerini and G. Goracci. *tseriesTARMA: Analysis of Nonlinear Time Series Through Threshold Autoregressive Moving Average Models*, 2024. URL https://CRAN.R-project.org/package=tseriesTARMA. R package version 0.5-1. [p167]

S. Giannerini, G. Goracci, and A. Rahbek. The validity of bootstrap testing for threshold autoregression. *Journal of Econometrics*, 239(1):105379, 2024. ISSN 0304-4076. doi: https://doi.org/10.1016/j.jeconom.2023.01.004. URL https://www.sciencedirect.com/science/article/pii/S0304407623000040. Climate Econometrics. [p167]

T. Gneiting, L. I. Stanberry, E. P. Grimit, L. Held, and N. A. Johnson. Assessing probabilistic forecasts of multivariate quantities, with an application to ensemble predictions of surface winds. *TEST*, 17 (2):211–235, 2008. doi: 10.1007/s11749-008-0114-x. [p177]

I. J. Good. Rational decisions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 14(1): 107–114, 1952. [p177]

G. Goracci, S. Giannerini, C. K.-S. Chan, and T. Howell. Testing for threshold effects in the tarma framework. *Statistica Sinica*, 33(3):1879–1901, 2023. ISSN 1996-8507. doi: doi:10.5705/ss.202021.0120. URL https://www3.stat.sinica.edu.tw/statistica/j33n3/J33N305/J33N305.html. [p167]

E. P. Grimit, T. Gneiting, V. J. Berrocal, and N. A. Johnson. The continuous ranked probability score for circular variables and its application to mesoscale forecast ensemble verification. *Quarterly Journal of the Royal Meteorological Society*, 132(621C):2925–2942, 2006. doi: 10.1256/qj.05.235. [p177]

G. Grynkiv and L. Stentoft. Stationary threshold vector autoregressive models. *Journal of Risk and Financial Management*, 11(3):45, 2018. doi: 10.3390/jrfm11030045. [p169]

A. Gupta and D. Nagar. *Matrix Variate Distributions*. Monographs and Surveys in Pure and Applied Mathematics. Taylor & Francis, 1999. ISBN 9781584880462. URL https://books.google.com.co/books?id=PQOYnT7P1loC. [p171]

B. E. Hansen. Threshold autoregression in economics. *Statistics and Its Interface*, 4(2):123–127, 2011. [p167]

S. Karlsson, S. Mazur, and H. Nguyen. Vector autoregression models with skewness and heavy tails. *Journal of Economic Dynamics and Control*, 146:104605, 2023. doi: 10.1016/j.jedc.2022.104605. [p170]

S. Kotz and S. Nadarajah. *Multivariate T-Distributions and Their Applications*. Cambridge University Press, 2004. [p170]

S. Kotz, T. J. Kozubowski, and K. Podgórski. *The Laplace Distribution and Generalizations: A Revisit with Applications to Communications, Economics, Engineering, and Finance*. Birkhäuser Basel, 2001. [p195]

K. L. Lange, R. J. A. Little, and J. M. G. Taylor. Robust statistical modeling using the t distribution. *Journal of the American Statistical Association*, 84(408):881–896, 1989. ISSN 01621459, 1537274X. URL http://www.jstor.org/stable/2290063. [p170]

J. Liu, W. Li, and C. Li. On a threshold autoregression with conditional heteroscedastic variances. *Journal of Statistical Planning and Inference*, 62(2):279–300, 1997. ISSN 0378-3758. doi: https://doi.org/10.1016/S0378-3758(96)00196-6. URL https://www.sciencedirect.com/science/article/pii/S0378375896001966. [p167]

X. Liu, R. Chen, and R. Tsay. Nts: An r package for nonlinear time series analysis. *The R Journal*, 12(2): 1–18, 2020. doi: 10.32614/rj-2021-016. [p167]

D. Makowski, M. Ben-Shachar, S. A. Chen, and D. Lüdecke. Indices of effect existence and significance in the bayesian framework. *Frontiers in Psychology*, 10, 2019a. [p181]

D. Makowski, M. Ben-Shachar, and D. Lüdecke. bayestestr: Describing effects and their uncertainty, existence and significance within the bayesian framework. *Journal of Open Source Software*, 4(40): 1541, 2019b. [p181]

J. E. Matheson and R. L. Winkler. Scoring rules for continuous probability distributions. *Management Science*, 22(10):1087–1096, 1976. doi: 10.1287/mnsc.22.10.1087. [p177]

A. McNeil, R. Frey, and P. Embrechts. *Quantitative Risk Management, Concepts, Techniques and Tools*. Princeton University Press, 2015. [p169, 170, 195]

M. Niglio and C. D. Vitale. Threshold vector arma models. *Communications in Statistics - Theory and Methods*, 44(14):2911–2923, 2015. doi: 10.1080/03610926.2013.814785. URL https://doi.org/10.1080/03610926.2013.814785. [p167]

M. Plummer, , N. Best, K. Cowles, and K. Vines. Coda: Convergence diagnosis and output analysis for mcmc. *R News*, 6(1):7–11, 2006. URL https://journal.r-project.org/archive/. [p182]

S. M. Potter. A nonlinear approach to US GNP. *Journal of Applied Econometrics*, 10(2):109–125, 1995. doi: 10.1002/jae.3950100203. [p169]

M. Pourahmadi. Skew-normal arma models with nonlinear heteroscedastic predictors. *Communications in Statistics – Theory and Methods*, 36(9):1803–1819, 2007. doi: 10.1080/03610920601126274. [p196]

A. Punzo and C. Tortora. Multiple scaled contaminated normal distribution and its application in clustering. *Statistical Modelling*, 21:332—-358, 2021. [p194, 195]

S. K. Sahu, D. K. Dey, and M. D. Branco. A new class of multivariate skew distributions with applications to bayesian regression models. *The Canadian Journal of Statistics*, 31(2):129–150, 2003. ISSN 03195724. [p170, 195]

D. Spiegelhalter, N. Best, B. Carlin, and A. Van Der Linde. Bayesian measures of model complexity and fit. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 64(4):583–639, 2002. [p176]

D. Spiegelhalter, N. Best, B. Carlin, and A. Van Der Linde. The Deviance Information Criterion: 12 Years on. *Journal of the Royal Statistical Society Series B: (Statistical Methodology)*, 76(3):485–493, 04 2014. [p176]

M. Stigler. *tsDyn: Nonlinear Time Series Models with Regime Switching*, 2018. URL https://CRAN.R-project.org/package=tsDyn. R package version 11.0. [p167]

H. Tong. On a threshold model. *Proceedings of the Symposium on the Statistical Analysis of Time Series*, pages 573–588, 1978. [p167]

H. Tong. *Threshold Models in Non-linear Time Series Analysis*, volume 21 of *Lecture Notes in Statistics*. Springer, New York, 1983. [p167]

H. Tong. *Non-linear Time Series: A Dynamical System Approach*. Oxford Statistical Science Series. Oxford University Press, Oxford, UK, 1990. ISBN 0198523009. [p167, 169, 190]

H. Tong. Threshold models in time series analysis—30 years on. *Statistics and Its Interface*, 4(2):107–118, 2011. ISSN 1938-7997. doi: 10.4310/SII.2011.v4.n2.a1. [p167]

H. Tong. Threshold models in time series analysis—some reflections. *Journal of Econometrics*, 189 (2):485–491, 2015. ISSN 0304-4076. doi: https://doi.org/10.1016/j.jeconom.2015.03.039. URL https://www.sciencedirect.com/science/article/pii/S0304407615001177. Frontiers in Time Series and Financial Econometrics. [p167]

R. S. Tsay. Testing and modeling multivariate threshold models. *Journal of the American Statistical Association*, 93(443):1188–1202, 1998. doi: 10.1080/01621459.1998.10473779. [p167, 169, 190]

J. W. Tukey. A survey of sampling from contaminated distributions. 1960. [p194, 195]

L. Vanegas, S. Calderón V, and L. Rondón. Bayesian estimation of a multivariate tar model when the noise process distribution belongs to the class of gaussian variance mixtures. *International Journal of Forecasting*, 2025. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2025.08.001. URL https://www.sciencedirect.com/science/article/pii/S016920702500069X. [p167, 168, 172, 174, 185]

J. Wang and M. G. Genton. The multivariate skew-slash distribution. *Journal of Statistical Planning and Inference*, 136:209 – 220, 2006. [p194]

Q. Wang, L. A. Stefanski, M. G. Genton, and D. D. Boos. Robust time series analysis via measurement error modeling. *Statistica Sinica*, 19(3):1263–1280, 2009. [p195]

S. Watanabe. Asymptotic equivalence of bayes cross validation and widely applicable information criterion in singular learning theory. *The Journal of Machine Learning Research*, 11:3571–3594, 2010. [p176]

A. Zeileis and Y. Croissant. Extended model formulas in R: Multiple parts and multiple responses. *Journal of Statistical Software*, 34(1):1–13, 2010. doi: 10.18637/jss.v034.i01. [p174]

*Luis Hernando Vanegas*
*Departamento de Estadística, Universidad Nacional de Colombia*
*Ciudad Universitaria, Bogotá*
*Colombia*
lhvanegasp@unal.edu.co

*Sergio Alejandro Calderón*
*Departamento de Estadística, Universidad Nacional de Colombia*
*Ciudad Universitaria, Bogotá*
*Colombia*
sacalderonv@unal.edu.co

*Luz Marina Rondón*
*Departamento de Estadística, Universidad Nacional de Colombia*
*Ciudad Universitaria, Bogotá*
*Colombia*
lmrondonp@unal.edu.co

# goSorensen: A Bioconductor R-Package to Detect Equivalence Between Gene Lists

*by Pablo Flores, Jordi Ocaña, Alex Sánchez, and Miquel Salicrú*

**Abstract** goSorensen is a software package designed to implement a statistical method for the functional comparison of gene lists based on the joint enrichment of Gene Ontology terms and a measure derived from the Sorensen-Dice index, which assesses the functional dissimilarity between the compared gene lists. The present paper provides an overview of the underlying method, followed by a description of the package's design and internal structure developed to efficiently manage the extensive data flow generated by the method. Finally, an application using real data is presented to illustrate the package's functionality, accompanied by a complementary Shiny-based dashboard that enables dynamic visualisation of the input and output data involved in the analysis.

## 1 Introduction

Since the introduction of microarrays (Schena et al., 1995; Nguyen et al., 2002), high-throughput omics technologies have advanced substantially, leading to the development of more advanced technologies such as RNA-seq (Chu and Corey, 2012; Van den Berge et al., 2019), single-cell sequencing (Jovic et al., 2022), and other recent ones, which have transformed modern biology and medicine (Lee, 2023; Negi et al., 2022). These high-throughput experiments allow us to study the behaviour of hundreds or thousands of genes (or proteins, metabolites, etc., often referred to as "features" in a general way) under different experimental conditions, resulting in lists of these features (e.g., a list of differentially expressed genes in a health condition) with critical information to solve relevant problems, such as the detection of bio-markers, drug development, molecular sub-classification of cancer, clinical applications, personalised medicine, among many others (Climente-González et al., 2017; Cieślik and Chinnaiyan, 2018; Pedersen and Kanigan, 2016; Jain, 2004).

Although obtaining feature lists is an important first step, a comprehensive study should also include a functional analysis to elucidate the biological significance of these data. External biological knowledge databases, such as Gene Ontology GO (Ashburner et al., 2000; Consortium et al., 2023), Kyoto Encyclopedia of Genes and Genomes KEGG (Kanehisa and Goto, 2000; Kanehisa et al., 2025), Reactome (Jassal et al., 2020), etc., serve as foundational resources for this purpose, enabling researchers to associate features in the lists with biological concepts or pathways and thereby gain meaningful insights into underlying mechanisms.

In particular, the **goSorensen** R package focuses on the functional comparison of gene lists using the Gene Ontology (GO) as a framework for representing biological knowledge. The GO is a structured, standardised representation of biological knowledge in which biological concepts, known as GO terms, are connected through formally defined relationships. It is designed to be species-agnostic, allowing gene products, such as the proteins or functional RNAs encoded by genes, from different organisms to be annotated using a common vocabulary and facilitating the comparison and integration of functional information across biological databases.

The GO is organised into three aspects, or sub-ontologies: Molecular Function (MF), Cellular Component (CC), and Biological Process (BP). Molecular Function terms represent molecular-level activities performed by gene products, such as catalytic or binding activities. Cellular Component terms capture the cellular location where a molecular function takes place, including cellular structures, membrane-enclosed compartments, and protein-containing complexes. Biological Process terms represent larger biological processes or programs accomplished by the concerted action of multiple molecular activities. Thus, the three GO aspects provide complementary descriptions of the activities performed by gene products, the cellular locations in which these activities occur, and the broader biological processes in which they participate.

Within each aspect, GO terms are organised hierarchically. The GO is structured as a graph in which terms are represented as nodes and the relationships between them as edges. Child terms are generally more specialised than their parent terms, although a term may have more than one parent. Consequently, the GO hierarchy is not a simple tree but a directed acyclic graph (DAG). Terms representing more general concepts connected to a given term are referred to as its ancestors, whereas more specialised terms below it in the hierarchy are its descendants.

Each of the three GO aspects is represented by a separate root term. GO defines several types of relationships between terms; in this work, particular attention is given to the *is a* relation, which represents subclass relationships and defines the basic hierarchical structure used by the method. Since a GO term may have more than one parent, several paths can connect a term to the corresponding

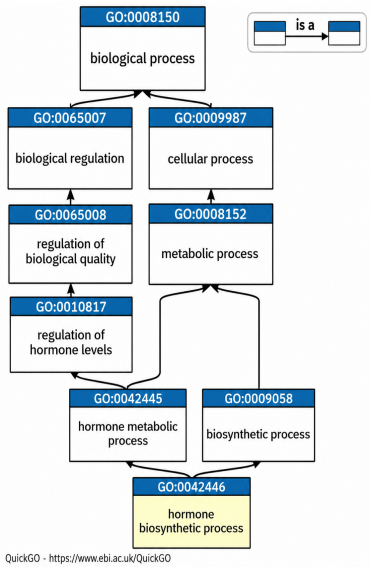



**Figure 1:** An example of GO terms in the BP ontology and their relationship to their ancestor terms.

root.

Throughout this work, we define the *GO level* of a term as the length of the shortest path composed exclusively of *is a* relationships connecting that term to the root of its corresponding GO aspect. Under this operational definition, larger GO levels generally correspond to more specific biological concepts.

For example, Figure 1 illustrates the multiple inheritance of the GO term GO:0042446 in the Biological Process ontology through *is a* relationships with its ancestors. The shortest *is a* path connecting this term to the root has length 4; accordingly, the term is assigned to GO level 4.

Gene products are associated with GO terms through *annotations*, which link a gene product to a biological concept represented in the ontology. Given a gene list obtained from an experiment, enrichment analysis can be used to identify GO terms that are represented more frequently than expected under an appropriate reference model (Hosack et al., 2003). These enriched terms provide a functional summary of the biological characteristics represented in the gene list and constitute the basic information used by **goSorensen** to compare gene lists from a functional perspective.

When the goal is to identify similarities across two or more gene lists, it is typically more informative to compare the biological concepts associated with the genes (annotations) rather than the genes themselves as simple elements. Some methods developed for this purpose include the Semantic Similarity (Pesquita, 2017) or the PANTHER web tool (Mi et al., 2017). These approaches often use comparison metrics, such as the Resnik, Lin, or Jaccard indices, to quantify similarity based on the overlap of annotations between lists. However, these methods generally overlook that the gene lists represent a sample derived from an experiment (e.g., from RNA-Seq), which may vary across repetitions or samplings. So, studying the sampling distribution associated with these metrics can provide a valuable complement, allowing one to assess whether an observed biological similarity is statistically significant or simply the result of chance.

One of the few methods introduced in the scientific literature to address these statistical aspects to provide significant functional comparison results is **goProfiles**. For each gene list, this method establishes a vector of annotation frequencies across all GO terms at a given GO level, referred to as an annotation profile. A distance metric is then defined between annotation profiles, and its sampling distribution is evaluated to develop tests for significant functional similarity. While **goProfiles** represents a robust statistical framework, it also has certain limitations. In particular, including GO terms with low or zero frequency, or assigning equal weight to both enriched and non-enriched terms, may introduce unnecessary noise into the distance metric.

To address these issues, we have developed a method for comparing feature lists, available in Flores (2025) and Flores et al. (2022), which we hereafter refer to as the *goSorensen method*. The fundamental guiding principle underlying this method is that two lists can be considered biologically similar if there is a significant prevalence of shared enriched GO terms in both lists compared to non-shared enriched GO terms. We assess the prevalence of this proportion using a dissimilarity measure derived from the Sorensen-Dice index (Sorensen, 1948), whose sample distribution was studied and implemented, allowing for more general conclusions regarding the biological similarity between lists. The **goSorensen** R package (Flores and Ocana, 2026), implements this novel method

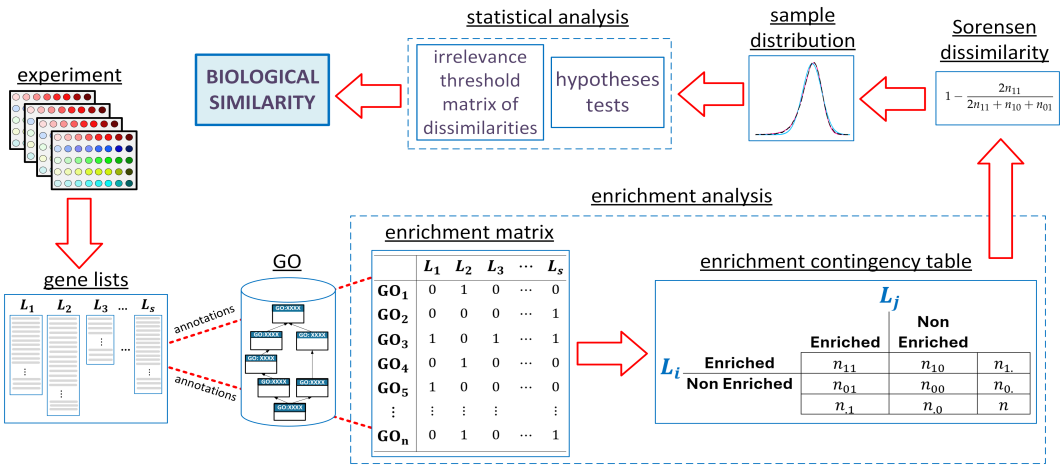



**Figure 2:** Sequential flow of the goSorensen method to detect biological similarity between feature lists

within the R/Bioconductor ecosystem. Its design integrates enrichment analysis, statistical testing, and visualisation in a unified workflow, ensuring full compatibility with widely used packages such as **clusterProfiler** (Wu et al., 2021), **goProfiles** (Sanchez et al., 2026), and different databases of functional annotation. By formalising the entire analytical process into a coherent, object-oriented structure and offering extensive documentation and vignettes, **goSorensen** enables researchers to perform statistically grounded and reproducible functional comparisons of gene lists efficiently.

This article provides a comprehensive overview of how the **goSorensen** package manages the data flow (genes, GO annotations and enrichment analysis, statistical analysis, etc.) involved in the *goSorensen method* to compare gene lists, providing the user with an easy-to-understand and use environment. The following section briefly explains the theoretical framework of the *goSorensen method*. The third section describes the general design of the package, elucidating its structure and leading advantages. In section four, we present an application of the package using real gene lists, and finally, section five is devoted to the relevant conclusions.

## 2 The goSorensen method

Figure 2 illustrates an overview of the sequential steps involved in the *goSorensen method*, which are detailed in Flores et al. (2022) and Flores (2025). As discussed above, the first step in comparing the $s$ gene lists derived from a high-throughput experiment involves annotating each element of these lists in the $n$ GO terms of a specific ontology (BP, CC, or MF) and level for performing an analysis to detect the enriched and non-enriched GO terms by each of the lists. The enrichment matrix compiles the results of this enrichment analysis. In Figure 2 the enriched terms of the matrix are shown with a number 1 and the non-enriched terms with a 0 but, in fact, in package **goSorensen** this is a logical matrix, of `TRUE/FALSE` values.

The $2 \times 2$ enrichment contingency table, also depicted in Figure 2, summarises the joint and the marginal enrichment patterns of any two specific lists from the enrichment matrix, where:

- $n_{11}$ denotes the number of GO terms enriched in both lists, representing the joint enrichment.
- $n_{10}$ denotes the number of GO terms enriched exclusively in the first list $L_i$, and non-enriched in the second list $L_j$.
- $n_{01}$ represents the number of GO terms enriched only in the second list $L_j$, and non-enriched in the first list $L_i$.
- $n_{00}$ represents the number of GO terms not enriched in either of the two gene lists.

The next step is to assess the degree of prevalence of the joint enrichment ($n_{11}$) over the marginal enrichment ($n_{01} + n_{10}$). For this purpose, we use the following measure, based on the Sorensen-Dice index, which we refer to as the Sorensen dissimilarity $\widehat{d}_S$:

$$\widehat{d}_S = 1 - \frac{2n_{11}}{2n_{11} + n_{10} + n_{01}},$$

or, in terms of relative frequencies:

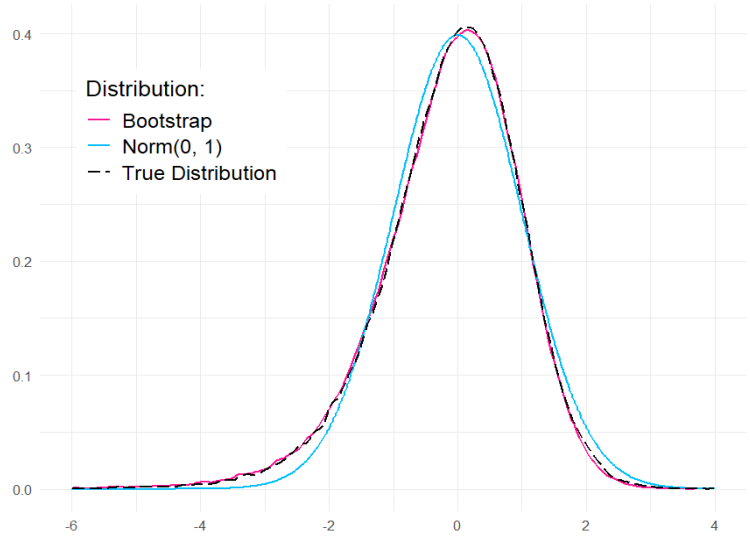



**Figure 3:** The true distribution (approximated) of the studentized statistic $Z_S$ and its fitting degree with its bootstrap estimation and the normal distribution $\text{Norm}(0,1)$, for a low enrichment level $(p_{11} = 0.125, p_{01} = 0.05, p_{10} = 0.05)$.

$$\widehat{d}_S = 1 - \frac{2\widehat{p}_{11}}{2\widehat{p}_{11} + \widehat{p}_{10} + \widehat{p}_{01}}, \qquad \text{with } \widehat{p}_{ij} = n_{ij}/n.$$

Considering that the frequencies $(n_{11}, n_{01}, n_{10}, n_{00})$ come from a multinomial distribution with parameters $(n, (p_{11}, p_{01}, p_{10}, p_{00}))$, where $p_{11}$ denotes the joint enrichment probability and $(p_{01}, p_{10})$ denotes the marginal enrichment probabilities, it is possible to prove that the sample distribution of $\widehat{d}_S$ can be approximated by a normal distribution with mean given by the theoretical dissimilarity $d_S$, and variance given by $\widehat{\sigma}_S^2$:

$$\widehat{d}_S \approx \text{Norm}\left(d_S, \frac{\widehat{\sigma}_s}{\sqrt{n}}\right), \qquad \text{when } n \to \infty.$$

being

$$d_S = 1 - \frac{2p_{11}}{2p_{11} + p_{10} + p_{01}} \quad \text{and} \quad \widehat{\sigma}_S^2 = \frac{4\widehat{p}_{11}\left(\widehat{p}_{10} + \widehat{p}_{01}\right)\left(\widehat{p}_{11} + \widehat{p}_{10} + \widehat{p}_{01}\right)}{\left(2\widehat{p}_{11} + \widehat{p}_{10} + \widehat{p}_{01}\right)^4}$$

So, using a significance level $\alpha$, one can obtain the following upper bound for a confidence interval of $d_S$:

$$\widehat{d}_S - z_\alpha \frac{\widehat{\sigma}_S}{\sqrt{n}},$$

where $z_\alpha$ is the $\alpha$ quantile associated with the studentised statistic $Z_S = \dfrac{\widehat{d}_S - d_S}{\widehat{\sigma}_S/\sqrt{n}}$, which asymptotically follows a standard normal distribution $\text{Norm}(0,1)$:

$$Z_S = \frac{\widehat{d}_S - d_S}{\widehat{\sigma}_S/\sqrt{n}} \approx \text{Norm}(0,1), \qquad \text{when } n \to \infty.$$

While the standard normal distribution adequately fits the true distribution of the studentized statistic $Z_S$, its accuracy decreases at low enrichment levels, which results in an increased type I error probability (TIEP) during inference processes. Figure 3 illustrates the results of a simulation process, which demonstrates that for low enrichment levels, the distribution derived from the bootstrap methodology fits the true distribution of $Z_S$ better than the theoretical normal distribution $\text{Norm}(0,1)$.

Both approximations (normal and bootstrap) to the true sample distribution play a crucial role in the sequential flow illustrated in Figure 2. They provide the basis for developing the statistical analysis to assess biological similarities between gene lists. These statistical analysis include:

- A test for the equivalence hypotheses:

$$H_0 : d_S \geq d_0$$
$$H_1 : d_S < d_0$$

  to establish the irrelevance of biological dissimilarities, up to a predefined equivalence limit $d_0$. This test can be extended to the equivalence comparison of $s > 2$ lists.
- A new index of dissimilarity between lists, which is based on the irrelevance-threshold determining whether the lists are significantly equivalent. This dissimilarity index can be employed in subsequent analyses, such as clustering or multidimensional scaling (MDS), where the grouping of biologically similar gene lists is grounded in the theory of equivalence testing.

See (Flores et al., 2022) and (Flores, 2025) for more details on these statistical methods, as well as for other issues such as criteria for setting equivalence limits or the desirability of performing equivalence tests (Wellek, 2002) versus the typical (and inadvisable, in our opinion) difference testing approach $H_0 : d_S = 0$ versus $H_1 : d_S > 0$.

## 3 goSorensen: A Bioconductor R-package

**goSorensen** is the informatics tool designed to implement the *goSorensen method*. This software package is hosted on Bioconductor, an open-source project based on the R programming language, specifically designed for bioinformatics research (Gentleman et al., 2004). This platform provides **goSorensen** with multiple advantages, including ease of visualisation, access, community support, documentation, reproducibility, and, most importantly, the ability to integrate with dependency packages essential for performing calculations that are part of the *goSorensen method*. Key dependency packages include software packages such as **clusterProfiler** (Wu et al., 2021; Yu et al., 2012) and **goProfiles** (Sanchez et al., 2026) for developing the enrichment analysis at different GO levels, and the `GO.db` annotation package (Carlson, 2026a), which provides extensive information on ontologies, terms, and annotations of the Gene Ontology, which is constantly updated to reflect changes and expansions to this biological knowledge database. Furthermore, Bioconductor hosts numerous genomic databases annotation packages that enable **goSorensen** to expand its analysis to several species or organisms, such as `org.Hs.eg.db` for the Homo sapiens (Carlson, 2026c), `org.Mm.eg.db` for the Mus musculus (Carlson, 2026d), `org.Dm.eg.db` for the Drosophila melanogaster (Carlson, 2026b), and others. As new annotation resources become available in Bioconductor, **goSorensen** can be easily extended to additional organisms, ensuring scalability and broad applicability.

### 3.1 Design and structure of the goSorensen package.

Managing the extensive data flow associated with the *goSorensen method* is a considerable challenge. This data flow mainly includes the gene lists, which often contain hundreds or even thousands of elements; the annotations of each of these elements in each of the thousands of GO terms located in a specific ontology and level; the enrichment analysis for these GO terms; and the results of the statistical analyses, such as hypothesis tests, irrelevance-threshold matrices of dissimilarities, relevant statistics, and plots. Figure 2 summarises this data flow, from obtaining gene lists to the enrichment and statistical analyses that ultimately assess biological similarity. Each step in this process is represented in **goSorensen** by specific data structures: Objects of class "list" correspond to the gene lists used as inputs; objects of class "matrix" and "tableList" represent the enrichment analyses; and objects of class "equivSDhtestList" and "distList" capture the statistical analysis and final results.

The components of the **goSorensen** package are structured to handle this large data flow efficiently, ensuring consistency between intermediate objects and facilitating integration with the accompanying documentation and vignettes. Figure 4 illustrates this organisation, showing the most relevant classes, functions, and documentation elements that define the design of **goSorensen**. To support their use, the package includes several example objects representing instances of these classes. These examples, accessible through the standard R function `data` (e.g., `data("allOncoGeneLists")`), are used throughout the vignettes and help files to demonstrate the package workflow. Below, we describe the most relevant classes, functions and documentation elements depicted in Figure 4:

The standard R class "list" is used to represent a collection of several gene lists (e.g., the gene lists obtained by different laboratories that have carried out a similar experiment). Each one of the components of this "list" is an object of class "character", containing a vector of gene identifiers[1]. This

[1]At this point, it may be important to reiterate the distinction we make between the term (commonly used in bioinformatics) gene list, represented in **goSorensen** by an object of class "character", and the class "list" , which is used here to designate several gene lists.

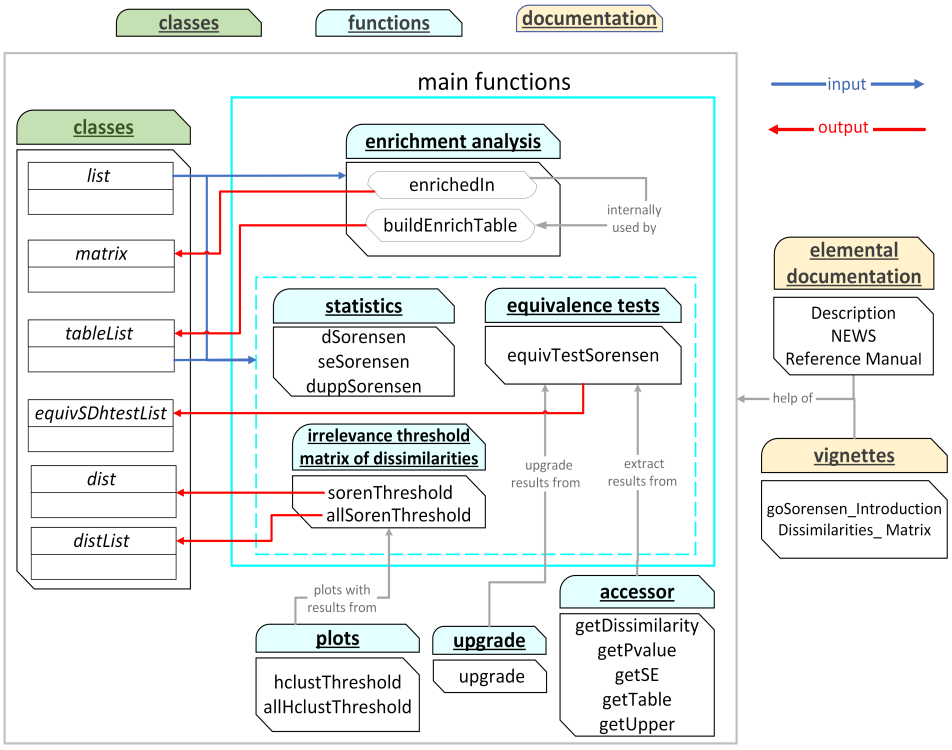



**Figure 4:** Organisational structure of the main classes, functions, and documentation elements in the goSorensen package, designed to manage the extensive data flow associated with the goSorensen method.

structure provides a flexible and intuitive way to manage multiple gene lists as input to the *goSorensen method*. The object `allOncoGeneLists` exemplifies this data organisation and serves as a reference dataset for demonstration purposes:

```
data("allOncoGeneLists")
class(allOncoGeneLists)

#> [1] "list"

length(allOncoGeneLists)

#> [1] 7

names(allOncoGeneLists)

#> [1] "atlas"         "cangenes"      "cis"           "miscellaneous"
#> [5] "sanger"        "Vogelstein"    "waldman"

class(allOncoGeneLists$atlas)

#> [1] "character"

# First ENTREZ gene identifiers in gene list atlas:
head(allOncoGeneLists$atlas)

#> [1] "11186" "27086" "239"   "7764"  "5934"  "246"
```

In **goSorensen**, the R class "matrix" is used to store the results obtained from the enrichment analysis. Each row corresponds to a GO term, and each column represents one of the analysed gene lists, where the logical values indicate whether the GO term is enriched (`TRUE`) or not (`FALSE`). This representation offers a simple and efficient way to summarise enrichment results across several gene lists, making it easier to apply the subsequent statistical steps of the *goSorensen method*. As an example, the object `enrichedInBP4` illustrates this structure for the ontology BP at the GO level 4.

```
data("enrichedInBP4")

class(enrichedInBP4)

#> [1] "matrix" "array"

dim(enrichedInBP4)

#> [1] 473   7

# Only those GO terms enriched in almost one gene list are included as rows of
# enrichedInBP4. The true number of terms in this ontology and level is:
attr(enrichedInBP4, "nTerms")

#> [1] 3381

colnames(enrichedInBP4)

#> [1] "atlas"         "cangenes"      "cis"           "miscellaneous"
#> [5] "sanger"        "Vogelstein"    "waldman"

# First GO terms represented in enrichedInBP4:
head(rownames(enrichedInBP4))

#> [1] "GO:0001649" "GO:0030278" "GO:0030279" "GO:0030282" "GO:0045778"
#> [6] "GO:0048755"
```

The class "tableList" is specific to **goSorensen** and corresponds to the typical output of the function `buildEnrichTable`. It is an inheritance specialisation of "list", which stores the enrichment contingency tables obtained from all pairwise comparisons between gene lists. Each element of an object "tableList" represents one of the analysed lists and contains the contingency tables associated with its comparisons to the other lists. Although this nested structure may appear complex at first, it offers a practical way to organise the large number of pairwise enrichment results produced by the method in a clear and consistent format. The object `cont_all_BP4` illustrates this structure for the BP ontology at GO level 4:

```
data("cont_all_BP4")
class(cont_all_BP4)

#> [1] "tableList" "list"

class(cont_all_BP4$sanger)

#> [1] "list"

class(cont_all_BP4$sanger$atlas)

#> [1] "table"
```

The class "equivSDhtestList" is also specific to **goSorensen** and corresponds to the typical output of the function `equivTestSorensen`. It extends the S3 class structure to store all hypothesis tests performed between pairs of gene lists for a given ontology and GO level. An object of this class is a list whose elements correspond to the analysed gene lists, and within each element are the results of the equivalence tests comparing that list with all others. Each of these inner elements is an object of class "equivSDhtest", which contains the detailed results of a single pairwise test, including the test statistic, Sorensen dissimilarity, standard error, confidence interval, and p-value. The object `eqTest_all_BP4` illustrates this structure for the BP ontology at GO level 4:

```
data("eqTest_all_BP4")
class(eqTest_all_BP4)

#> [1] "equivSDhtestList" "list"
```

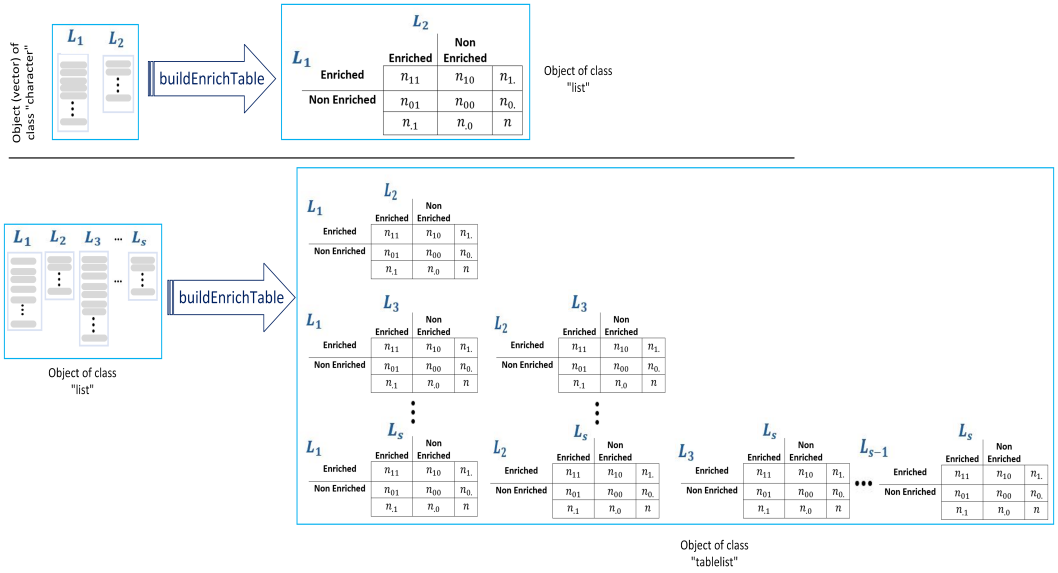



**Figure 5:** buildEnrichTable: a generic function to compute the enrichment contingency table from two or more gene lists.

```
class(eqTest_all_BP4$sanger$atlas)
```

```
#> [1] "equivSDhtest" "htest"
```

The class "distList" is also specific to **goSorensen** and corresponds to the output of the function `allSorenThreshold`. It extends the R class "list" to store the irrelevance-threshold matrices of dissimilarities obtained for different GO levels and ontologies. Each element of a "distList" object represents a particular GO ontology and level, and contains an object of class "dist" with the corresponding irrelevance-threshold matrix of dissimilarities. This structure provides an organised and consistent way to compare functional similarity between gene lists across multiple ontologies and levels. The object `allDissMatrx` illustrates this class, encompassing the three ontologies (BP, CC, and MF) and GO levels 3 through 10:

```
data("allDissMatrx")
class(allDissMatrx)
```

```
#> [1] "distList" "list"
```

```
class(allDissMatrx$BP$`level 4`)
```

```
#> [1] "dist"
```

The **goSorensen** ***functions*** are the core components of the package. Figure 4 depicts these functions highlighted in turquoise, where the functions enclosed in the box outlined in turquoise with solid borders comprise the main functions of the package since they are directly related to the sequential flow of the *goSorensen method* depicted in Figure 2. Among these main functions are those devoted to the enrichment analysis, which exclusively accept objects of class "list" as input. Like most functions of **goSorensen**, these are implemented as generic functions under the S3 object-oriented programming paradigm, providing a flexible yet consistent framework for defining outputs, ensuring that results from the different stages of the method can be stored, accessed, and extended in a coherent and standardised way. For example, Figure 5 illustrates how the function `buildEnrichTable` generates different outputs based on the type of input provided by the user. If the input is a pair of vectors of class "character", the function returns an object of class "table" containing the enrichment contingency table between these two compared gene lists. Whereas, if the input is an object of class "list" with two or more gene lists, the function returns an object of class "tableList" containing all the possible enrichment contingency tables between the compared gene lists.

The remaining main functions of **goSorensen** complete the last step (statistical analysis) in the sequential flow depicted in Figure 2 to detect biological similarity through the *goSorensen method*.

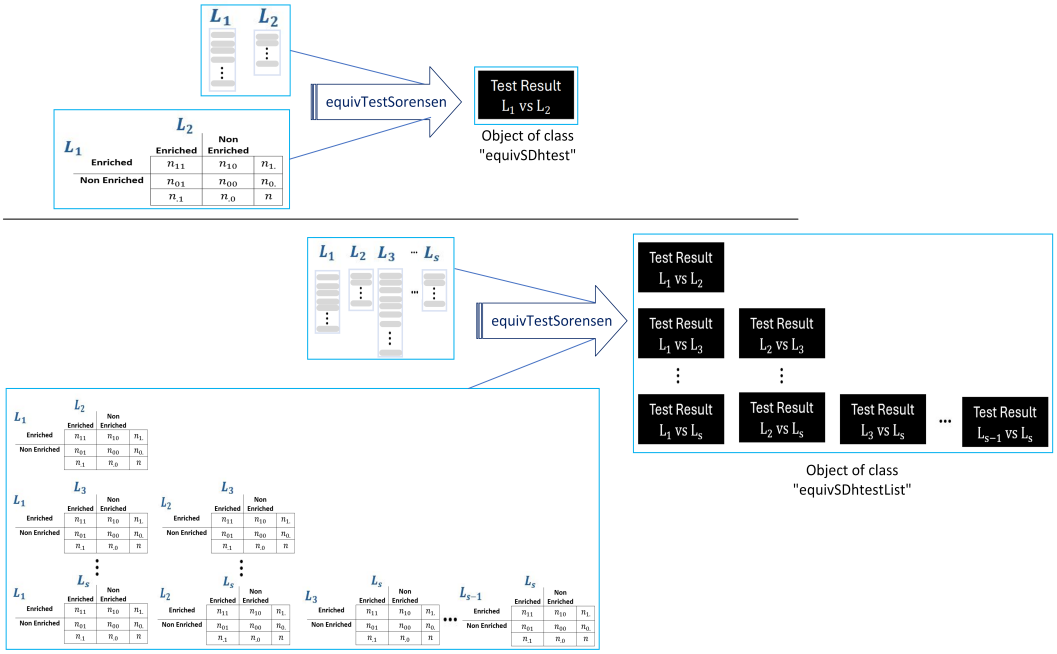



**Figure 6:** equivTestSorensen: a generic function to compute the outputs of the equivalence hypothesis tests to detect biological similarity between two or more gene lists.

These functions can generate their outputs based on either the normal or the bootstrap sample distribution of the Sorensen dissimilarity, depending on the user's choice. As illustrated in Figure 4, these functions (highlighted in the turquoise dashed box) encompass three main tasks: estimating relevant statistics, performing the results for the equivalence hypothesis test to prove functional similarity, and computing the irrelevance-threshold matrix of dissimilarities. These generic functions accept as input either objects of class "list" (raw data, i.e., gene lists) or objects of class "tableList" (enrichment contingency tables). While both inputs yield the same outputs, the computational efficiency is significantly enhanced when the user provides an object of class "tableList" directly, as the enrichment analysis step has been previously performed. Figure 6 illustrates this behaviour using the function `equivTestSorensen`, as an example. This figure shows that an object of class "equivSDhtest", which contains the results of an equivalence test between two gene lists, can be generated using either two "character" vectors representing the gene lists or a "matrix" object containing the corresponding enrichment contingency table as input. Moreover, when comparing more than two gene lists, the same function, `equivTestSorensen`, accepts either an object of class "list" containing all the gene lists to be compared or an object of class "tableList" comprising all the possible contingency tables between the lists as input. Regardless of the input, the output is an object of class `equivSDhtestList`, which includes the results of all pairwise hypothesis tests performed between the compared gene lists.

In addition to the main functions already described, **goSorensen** includes two extended functions designed to operate across multiple GO ontologies and levels. The function `allBuildEnrichTable` generalises `buildEnrichTable` to generate enrichment contingency tables simultaneously for several GO levels and ontologies, producing objects of class "allTableList." Similarly, the function `allEquivTestSorensen` extends `equivTestSorensen` to perform equivalence tests over multiple ontologies and levels, generating objects of class "AllEquivSDhtest". These functions ensure that the package can scale the *goSorensen method* to more complex analyses, maintaining consistency and reproducibility across the full GO structure.

Finally, **goSorensen** includes a set of complementary functions designed to facilitate the management and interpretation of results. Accessor functions allow users to extract specific components from the outputs of equivalence tests, such as sample dissimilarities, p-values, standard errors, contingency tables, or confidence interval limits. These functions are particularly useful when handling large-scale analyses involving numerous pairwise tests, as they enable efficient retrieval and post-processing of relevant statistics, including adjustments for multiple testing. In addition, an upgrade function that enables users to update existing test results when modifying key parameters, such as the significance level, irrelevance threshold, or sampling distribution. This approach improves computational efficiency by recalculating only the affected components rather than repeating the full analysis. The package also provides plotting functions that generate objects to plot dendrograms to visualise hierarchical clustering of the irrelevance-threshold matrices of similarities.

According to Figure 4, the last component of the **goSorensen** package is its ***documentation*** files. In addition to the standard documentation that every R package provides, such as help pages, descriptions, illustrative examples, and a Reference Manual, **goSorensen** includes two detailed vignettes that serve as comprehensive user guides. The first vignette, 'An Introduction to goSorensen R-Package', offers a comprehensive overview of the package's structure, its main classes and functions. It explains, through reproducible examples, how to perform the enrichment and equivalence analyses and how to interpret their results. The second vignette, 'Working with the Irrelevance-threshold Matrix of Dissimilarities', focuses on the statistical analyses that can be carried out using the dissimilarity matrices produced by the package, showing how these can be explored to study biological similarity across gene lists and GO levels. Together, these documents provide a clear and practical introduction to the package, bridging methodological concepts with applied examples and supporting reproducibility and transparency in its use.

## 4 An example application

All examples in this section were generated using Bioconductor 3.23, the current Bioconductor release at the time of revision. Running the examples with a different Bioconductor release may yield different numerical results because the underlying annotation resources, including GO.db and org.Hs.eg.db, evolve between releases. A reproducibility analysis comparing Bioconductor releases 3.21, 3.22 and 3.23, together with the corresponding code and computational environments, is available at `https://github.com/ASPresearch/goSorensen-Bioc-Reproducibility`.

The objective of this section is to illustrate the application of the main functions of **goSorensen** and, consequently, how to obtain the different classes of objects described in the previous section. For this purpose, the package includes the object `allOncoGeneLists`, a "list" containing seven gene lists derived from a compilation of data carried out by the Bushman Lab on the human species (`https://www.med.upenn.edu/bushmanlab/resources.html`). These lists represent sets of transcribed genes with functional evidence of expression at different stages of cancer progression.

Before applying the **goSorensen** functions to the data contained in `allOncoGeneLists`, it is necessary to define a vector representing the gene universe available in the annotation databases for the species under study. In the case of Homo sapiens, this vector can be obtained from the annotation package `org.Hs.eg.db` as follows:

```
library(org.Hs.eg.db)
humanEntrezIDs <- keys(org.Hs.eg.db, keytype = "ENTREZID")
```

The object `humanEntrezIDs` is a vector of class "character" containing the ENTREZ identifiers that define the gene universe for the human species. The ENTREZ ID is a system of gene identifiers recognised by most biological databases, including GO, and will be used as the reference identifier throughout this section. However, the gene universe can also be represented using other identifier systems, such as ENSEMBL IDs, GENE SYMBOL IDs, or NCBI IDs.

### 4.1 Enrichment analysis

#### Enrichment matrix

The first step of the analysis is to compute the enrichment matrix. For example, for the ontology BP, at GO level 4, this can be done as follows:

```
enrichedInBP4 <- enrichedIn(allOncoGeneLists,
                            geneUniverse = humanEntrezIDs,
                            orgPackg = "org.Hs.eg.db",
                            onto = "BP", GOLevel = 4)
```

By default, this matrix includes only GO terms enriched in at least one list. Setting `onlyEnriched = FALSE` (the default is `TRUE`) returns the full matrix, and the attribute `nTerms` indicates the total number of GO terms evaluated. Since the full matrix is mostly composed of non-enriched terms, using the reduced version is generally more practical.

Analysing a large number of gene lists, which would involve repeated enrichment analyses, could potentially create a computational bottleneck. In this case, it might be advisable to use the `enrichedIn` argument `parallel = TRUE`. This is certainly not the case in the present example – parallelisation, particularly on Windows, can result in a certain amount of computational overhead, especially during

initialisation. All functions whose execution ultimately depends on `enrichedIn` have this argument, which is set to `FALSE` by default.

### Enrichment contingency table

Based on the total number of GO terms annotated in the selected ontology and level, the enrichment contingency tables are obtained as the cross-frequency between pairs of lists from the enrichment matrix. For two specific lists (e.g., atlas and sanger) and a given ontology-level combination (e.g., BP, level 4), the function `buildEnrichTable` internally computes the corresponding enrichment matrix to generate the contingency table between both lists. If the argument `storeEnrichedIn` is set to `TRUE` (value by default), the enrichment matrix is accessible in the attribute `enriched`. The code routine to obtain this example contingency table is as follows:

```
cont_atlas.sanger_BP4 <- buildEnrichTable(allOncoGeneLists$atlas,
                                          allOncoGeneLists$sanger,
                                          listNames = c("atlas", "sanger"),
                                          geneUniverse = humanEntrezIDs,
                                          orgPackg = "org.Hs.eg.db",
                                          onto = "BP", GOLevel = 4)
cont_atlas.sanger_BP4

#>                  Enriched in sanger
#> Enriched in atlas TRUE FALSE
#>             TRUE   204   186
#>             FALSE   27  2964
```

To obtain all pairwise tables at once, one can run:

```
cont_all_BP4 <- buildEnrichTable(allOncoGeneLists,
                                 geneUniverse = humanEntrezIDs,
                                 orgPackg = "org.Hs.eg.db",
                                 onto = "BP", GOLevel = 4)
```

and for several ontologies and GO levels:

```
allContTabs <- allBuildEnrichTable(allOncoGeneLists,
                                   geneUniverse = humanEntrezIDs,
                                   orgPackg = "org.Hs.eg.db",
                                   ontos = c("BP", "CC", "MF"),
                                   GOLevels = 3:10)
```

## 4.2 Statistical analysis

### Hypothesis tests

The next step is to evaluate functional similarity through equivalence testing on the Sorensen dissimilarity ($H_0 : d_S \geq d_0$ Vs. $H_1 : d_S < d_0$). The irrelevance threshold $d_0$ defines the maximum dissimilarity at which two lists are still considered functionally equivalent. Following the approach of Flores et al. (2022), $d_0$ can be derived from a ratio $\rho = 2p_{11}/(p_{01} + p_{10})$ of joint versus marginal enrichment probability, which can be expressed as:

$$d_0 = \frac{1}{1+\rho}.$$

Following conventional criteria of robustness in statistical model building, which accept variations of up to 20% as practically irrelevant (Box, 1979), and in line with the standard bioequivalence range of [80% ˜ 125%] adopted by international regulatory guidelines (FDA, 1992; EMA, 2010), Flores et al. (2022) propose a ratio $\rho = 1/0.8 = 1.25$. This value represents the upper bound of acceptable deviation and corresponds to an equivalence limit of $d_0 = 0.4444$ in the context of the *goSorensen method*.

For illustration:

```
eqTest_atlas.sanger_BP4 <- equivTestSorensen(allOncoGeneLists$atlas,
                                             allOncoGeneLists$sanger,
                                             listNames = c("atlas", "sanger"),
                                             geneUniverse = humanEntrezIDs,
                                             orgPackg = "org.Hs.eg.db",
                                             onto = "BP", GOLevel = 4,
                                             d0 = 0.4444, conf.level = 0.95)

eqTest_atlas.sanger_BP4

#>
#>  Normal asymptotic test for 2x2 contingency tables based on the
#>  Sorensen-Dice dissimilarity
#>
#> data:  cont_atlas.sanger_BP4
#> (d - d0) / se = -4.5899, p-value = 2.217e-06
#> alternative hypothesis: true equivalence limit d0 is less than 0.4444444
#> 95 percent confidence interval:
#>  0.0000000 0.3793505
#> sample estimates:
#> Sorensen dissimilarity
#>              0.3429952
#> attr(,"se")
#> standard error
#>     0.02210248
```

In this example, the test does not reject the null hypothesis of non-equivalence at $\alpha = 0.05$. Therefore, there is insufficient statistical evidence to conclude that the atlas and sanger lists are functionally equivalent under the threshold $d0 = 0.4444$, for the BP ontology at GO level 4.

Similarly, for a specific ontology and GO level, when all pairwise tests are needed, one can use:

```
eqTest_all_BP4 <- equivTestSorensen(allOncoGeneLists,
                                    geneUniverse = humanEntrezIDs,
                                    orgPackg = "org.Hs.eg.db",
                                    onto = "BP", GOLevel = 4,
                                    d0 = 0.4444, conf.level = 0.95)
```

If contingency tables are pre-computed (e.g., `cont_all_BP4`), the same output can be generated more efficiently:

```
eqTest_all_BP4 <- equivTestSorensen(cont_all_BP4,
                                    d0 = 0.4444, conf.level = 0.95)
```

Across multiple ontologies and GO levels, the function `allEquivTestSorensen` generalises this procedure. Users can also apply bootstrap inference by setting `boot = TRUE`, or update existing results with the `upgrade` function without rerunning the full analysis.

See the vignette An Introduction to goSorensen R-Package for more details about the use of the functions described above.

### Irrelevance-threshold matrix of dissimilarities

Each element of this matrix represents the irrelevance-threshold dissimilarity for the corresponding pair of gene lists; equivalence is statistically supported for irrelevance thresholds larger than this value. For the ontology BP at GO level 4:

```
dissMatrx_BP4 <- sorenThreshold(allOncoGeneLists,
                                geneUniverse = humanEntrezIDs,
                                orgPackg = "org.Hs.eg.db",
                                onto = "BP", GOLevel = 4,
                                trace = FALSE)
dissMatrx_BP4
```

```
#>                  atlas       cis miscellaneous    sanger Vogelstein
#> cis           0.7135865
#> miscellaneous 0.5667299 0.6286086
#> sanger        0.4606465 0.6589836     0.3384844
#> Vogelstein    0.3461926 0.6896145     0.5109424 0.2489031
#> waldman       0.3051649 0.6742598     0.2756020 0.4153017  0.3744204
```

Using pre-computed tables (e.g., `cont_all_BP4`) makes this step faster. Multiple ontologies and levels can be processed simultaneously using:

```
allDissMatrx <- allSorenThreshold(allOncoGeneLists,
                                  geneUniverse = humanEntrezIDs,
                                  orgPackg = "org.Hs.eg.db",
                                  ontos = c("BP", "CC", "MF"),
                                  GOLevels = 3:10)
```

These matrices are obtained using the normal approach; if one wants to use the bootstrap approach instead, the argument `boot = TRUE`, which is set to `FALSE` by default, can be set.

Hierarchical clustering of these matrices can be performed directly with the `hclustThreshold` function, which extends the standard `hclust` framework to objects generated by **goSorensen**. For "distList" objects containing matrices across several ontologies and GO levels, the function `allHclustThreshold` generalises this procedure. The resulting clustering objects can be visualised using the standard `plot` method. In contrast, MDS can be performed directly using standard R functions, since the methodological contribution of **goSorensen** lies in the construction of the irrelevance-threshold matrix of dissimilarities rather than in introducing a new multidimensional scaling procedure. Worked examples for both visualisations, including all the R code required to reproduce the dendrogram and MDS biplot, are provided in the vignette *Working with the Irrelevance-threshold Matrix of Dissimilarities*.

Although the irrelevance-threshold matrix of dissimilarities, which can be conveniently visualised through dendrograms or MDS biplots, provides an alternative way of assessing statistical equivalence between gene lists from that provided by p-values, both results are available to users and can be examined together. For any irrelevance-threshold matrix computed by **goSorensen**, such as `dissMatrx_BP4`, the enrichment contingency tables used in its computation are retained in the `all2x2Tables` attribute. These tables can therefore be recovered and used to perform the corresponding equivalence tests without repeating the enrichment analysis. The accessor functions provided by **goSorensen** can then be used to retrieve the p-values, as well as other outputs of the equivalence tests. For example, the pairwise p-values associated with `dissMatrx_BP4` can be obtained as follows:

```
# Extract the contingency tables
contTable <- attr(dissMatrx_BP4, "all2x2Tables")

# Get the p-values
pvals <- getPvalue(equivTestSorensen(contTable))
```

Figure 7 presents the two-dimensional MDS biplot and the dendrogram for the matrix contained in the object `dissMatrx_BP4`. Since these dissimilarities have an inferential basis, the MDS plot and dendrogram provide useful exploratory representations of patterns of functional similarity. However, proximity or clustering in these visualisations should not by itself be interpreted as evidence of pairwise statistical equivalence; the corresponding equivalence tests should be considered for inferential conclusions. For example, we can see that the lists *waldman*, *vogelstein*, *sanger*, and *miscellaneous* are minimally distanced from each other, which suggests a group of functionally similar lists; the corresponding pairwise equivalence tests can then be examined to assess the statistical evidence supporting these relationships.

The abscissa and ordinate axes of the MDS - biplot represent the two most representative dimensions that capture the variability from the original space. The characterisation of these dimensions constitutes an interesting component of the statistical analysis, as it allows us to detect which GO terms are associated with the formation of these dimensions. In other words, a characterisation of the biplot dimensions enables us to detect which biological concepts contribute to the patterns of functional similarity represented in the biplot.

The main idea behind the characterisation of the MDS - Biplot dimensions is that a GO term with as different and stable enrichment as possible in the lists located at the extremes of dimensions helps explain the formation of clusters. Table 1 presents the results obtained by applying this characterisation to the object `dissMatrx_BP4`. Owing to space limitations, only a few GO terms are shown. The complete

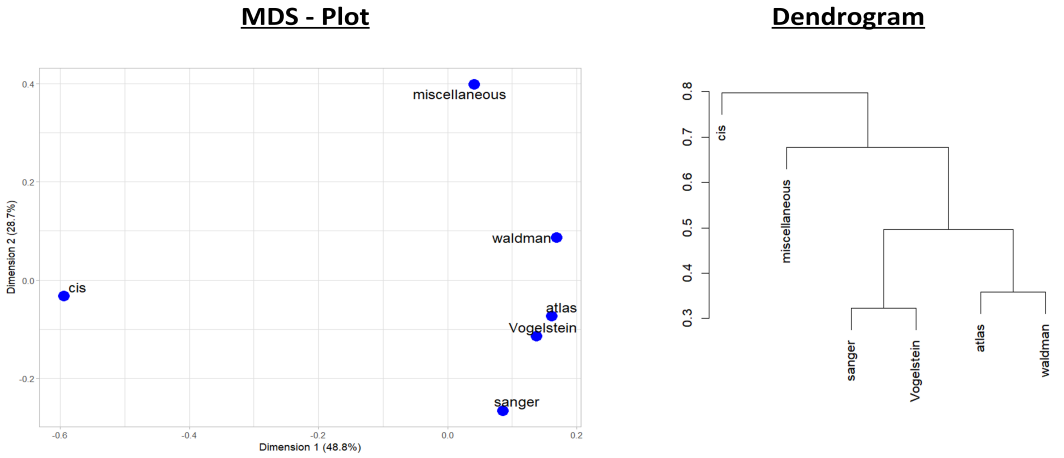



**Figure 7:** MDS – biplot and dendrogram representing the clustered lists resulting from the irrelevance-threshold matrix of dissimilarities generated for the lists included in allOncoGeneLists, for the ontology BP and the GO level 4.

| Dimension 1 | |
|---|---|
| **GO ID's** | **Description** |
| GO:0061138 | morphogenesis of a branching epithelium |
| GO:0030168 | platelet activation |
| GO:0007548 | sex differentiation |
| GO:0045137 | development of primary sexual characteristics |
| GO:0048608 | reproductive structure development |
| GO:0001666 | response to hypoxia |
| GO:0006979 | response to oxidative stress |
| GO:0009611 | response to wounding |
| ⋮ | |
| GO:0104004 | cellular response to environmental stimulus |
| GO:0099072 | regulation of postsynaptic membrane neurotransmitter receptor levels |

| Dimension 2 | |
|---|---|
| **GO ID's** | **Description** |
| GO:0060688 | regulation of morphogenesis of a branching structure |
| GO:0002263 | cell activation involved in immune response |
| GO:0042118 | endothelial cell activation |
| GO:0002260 | lymphocyte homeostasis |
| GO:0001819 | positive regulation of cytokine production |
| GO:0032613 | interleukin-10 production |
| GO:0032633 | interleukin-4 production |
| GO:0071604 | transforming growth factor beta production |
| ⋮ | |
| GO:0061982 | meiosis I cell cycle process |
| GO:1900182 | positive regulation of protein localization to nucleus |

**Table 1:** GO terms associated with the formation of clusters of functionally similar gene lists contained in `allOncoGeneLists`, in the ontology BP and GO level 4.

set of results can be explored in the HTML version of this article, which includes an interactive table displaying all GO terms.

For specific details of the irrelevance-threshold matrix of dissimilarities, its clustering and visualisation, and the identification of the most relevant GO terms explaining biological similarity through the characterisation of MDS biplot dimensions, readers are referred to Chapter 5 of Flores (2025) and to the vignette Working with the Irrelevance-threshold Matrix of Dissimilarities.

### 4.3 Interactive visualisation dashboard

Since it is not feasible to observe all the results depicted in this section, we have created an interactive Shiny application to help users explore all of them. This dashboard, included as supplementary material, is designed exclusively for visualisation purposes and is not intended to serve as an additional analysis tool for **goSorensen**. It is accessible directly at `https://pablof1988.shinyapps.io/app_allOnco/`, and the source code with a brief guide to the application is available from the GitHub

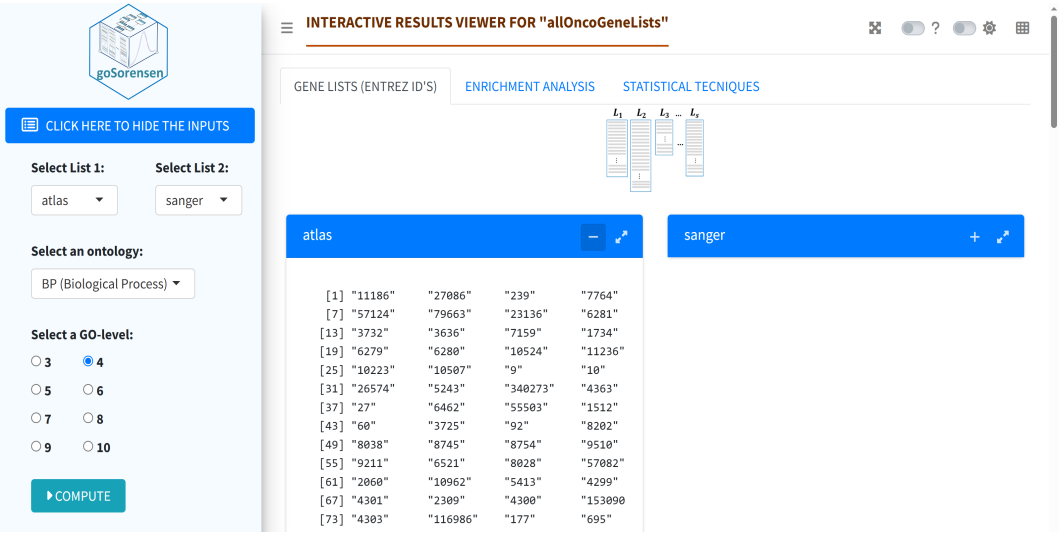



**Figure 8:** Screenshot of the first tab in the Shiny app. An interactive dashboard to explore the results of applying the goSorensen functions to the gene lists contained in the object allOncoGeneLists.

repository located at https://github.com/pablof1988/allOncoGeneLists_viewer

Figure 8 shows a screenshot of the first tab in the Shiny application. The application is organised into three main tabs following the sequential steps of the *goSorensen method*:

- **GENE LISTS** displays the ENTREZ identifiers of the selected lists.
- **ENRICHMENT ANALYSIS** shows the enrichment matrix for all lists and the contingency table for the selected pair of lists, both adjusted to the chosen ontology and GO level.
- **STATISTICAL ANALYSIS** presents in two subtabs the results of the equivalence hypothesis tests and the irrelevance-threshold matrix of dissimilarities, including visualisations such as dendrograms and MDS biplots. Users can dynamically modify key parameters (e.g., irrelevance limit $d_0$, confidence level, or sampling distribution) to update the visual outputs.

Additionally, the app includes a tool to identify the GO terms contributing to the biological similarity observed in each MDS dimension.

## 5 Conclusions and discussions

This article introduces **goSorensen**, an R package developed to implement the *goSorensen method* for assessing biological similarity between gene lists. The package integrates enrichment analysis, equivalence testing, and visualisation, allowing the entire process to be carried out in a coherent and reproducible way, promoting flexible workflows and facilitating integration with other Bioconductor packages. Built under the object-oriented programming paradigm, **goSorensen** allows the user to manipulate and combine objects of different classes within its functions. The statistical calculations used by **goSorensen** to determine significant equivalence between lists are mainly based on a normal approximation to the sampling distribution of the Sorensen dissimilarity, which quantifies their functional similarity. Although this approximation provides an efficient and straightforward approach for inference, its accuracy may decrease at low enrichment levels, where a bootstrap alternative—also implemented in the package—offers more reliable results but at the expense of a higher computational cost.

Since the **goSorensen** package is a direct implementation of the *goSorensen method*, future work is closely linked to improving, optimising, and scaling this approach. A first potential improvement is to develop an analysis procedure that does not depend on selecting a specific GO level. Because GO terms within the same level differ in their degree of specificity, difficulties may arise when functional similarity is inferred from overly general or uninformative concepts. In some cases, a term may fail to appear enriched at a general level even though enrichment is evident at more specific levels—or conversely, enrichment may only be observed for broader ancestral terms. Developing a procedure that integrates information across levels would therefore provide a more consistent representation of biological similarity, reducing the influence of arbitrary level selection.

A limitation of the current implementation arises from the components of the Sorensen dissimilarity. Enriched GO terms—either joint or marginal—carry an inherent level of uncertainty associated with

their detection. As is well known, the assessment of GO term enrichment involves a certain degree of arbitrariness since it depends on criteria established by the researcher's perception, such as cut-off points. In addition, the identification of enrichment is itself the outcome of a hypothesis test, which depends on earlier decisions, such as the choice of significance levels or the method adopted to address multiple testing. Developing adaptive approaches that explicitly model the uncertainty derived from the enrichment stage could control a possible overall inflation of type I error.

Another important challenge is related to the management of large volumes of information. When the analysis involves a great number of genes or lists, the resulting amount of data can substantially increase computational demand. In some situations, the required resources may exceed the capacity of standard computing equipment, limiting the practical applicability of the method. Future developments should therefore consider strategies to reduce the computational burden.

Since *goSorensen* relies on external databases to provide biological meaning, another direction for future work is to explore the behaviour of the method using additional resources beyond the Gene Ontology, for example, the Disease Ontology, the Drug Ontology, or even databases based on biological pathways, such as REACTOME or KEGG. Furthermore, the conceptual foundation underlying *goSorensen* can be generalised to other similarity measures beyond the Sorensen–Dice index, such as the Jaccard coefficient or other metrics to assess joint enrichment overlap. In addition, the framework can be extended to other omics domains, such as proteomics, metabolomics, or transcriptomics, where assessing the equivalence between feature lists is essential for interpreting the biological knowledge associated with more complex studies.

In summary, *goSorensen* offers a statistically grounded and reproducible framework for evaluating functional equivalence between gene lists within R and Bioconductor. Its continued development will aim to refine the underlying method, enhance computational efficiency, and expand its applicability to other biological domains. In this way, goSorensen will continue to contribute to the integration of rigorous statistical methodology with practical tools for interpreting complex biological data.

## Acknowledgements

This work was supported by the Agencia Estatal de Investigación (AEI), grant PID2023-148013OB-C22, and by the Instituto de Salud Carlos III through CIBERFES (CB16/10/00269).

## References

M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, et al. Gene ontology: tool for the unification of biology. *Nature genetics*, 25(1): 25–29, 2000. URL https://doi.org/10.1038/75556. [p201]

G. E. Box. Robustness in the strategy of scientific model building. In *Robustness in statistics*, pages 201–236. Elsevier, 1979. URL https://www.sciencedirect.com/science/article/abs/pii/B9780124381506500182. [p211]

M. Carlson. *GO.db: A set of annotation maps describing the entire Gene Ontology*, 2026a. URL https://www.bioconductor.org/packages/release/data/annotation/html/GO.db.html. R package version 3.23.1. [p205]

M. Carlson. *org.Dm.eg.db: Genome wide annotation for Fly*, 2026b. URL https://bioconductor.org/packages/release/data/annotation/html/org.Dm.eg.db.html. R package version 3.23.1. [p205]

M. Carlson. *org.Hs.eg.db: Genome wide annotation for Human*, 2026c. URL https://bioconductor.org/packages/release/data/annotation/html/org.Hs.eg.db.html. R package version 3.23.1. [p205]

M. Carlson. *org.Mm.eg.db: Genome wide annotation for Mouse*, 2026d. URL https://bioconductor.org/packages/release/data/annotation/html/org.Mm.eg.db.html. R package version 3.23.1. [p205]

Y. Chu and D. R. Corey. Rna sequencing: platform selection, experimental design, and data interpretation. *Nucleic acid therapeutics*, 22(4):271–274, 2012. URL https://doi.org/10.1089/nat.2012.0367. [p201]

M. Cieślik and A. M. Chinnaiyan. Cancer transcriptome profiling at the juncture of clinical translation. *Nature Reviews Genetics*, 19(2):93–109, 2018. URL https://doi.org/10.1038/nrg.2017.96. [p201]

H. Climente-González, E. Porta-Pardo, A. Godzik, and E. Eyras. The functional impact of alternative splicing in cancer. *Cell reports*, 20(9):2215–2226, 2017. URL https://doi.org/10.1016/j.celrep.2017.08.012. [p201]

G. O. Consortium et al. The gene ontology knowledgebase in 2023. *Genetics*, 224(1), 2023. URL https://doi.org/10.1093/genetics/iyad031. [p201]

EMA. Guideline on the investigation of bioequivalence. Technical report, European Medicines Agency, Committee for Medicinal Products for Human Use CMHP, London, 2010. URL https://www.ema.europa.eu/en/documents/scientific-guideline/guideline-investigation-bioequivalence-rev1_en.pdf. [p211]

FDA. Guidance on statistical procedures for bioequivalence using a standard two-treatment crossover design. Technical report, US Food and drug administration, Division of Bioequivalence, Office of Generic Drugs, Centre for Drug Evaluation and Research, Rockville MD, 1992. URL https://www.fda.gov/media/70958/download. [p211]

P. Flores. *Functional Similarity Between Feature Lists Based on the Joint Enrichment of the Gene Ontology Terms*. PhD thesis, Universitat Politecnica de Catalunya, Barcelona, España, 2025. URL https://upcommons.upc.edu/entities/publication/66a72ffc-f214-403a-8cc4-64fa7a1fb673. [p202, 203, 205, 214]

P. Flores and J. Ocana. *goSorensen: Statistical inference based on the Sorensen-Dice dissimilarity and the Gene Ontology (GO)*, 2026. URL https://bioconductor.org/packages/goSorensen. R package version 1.14.0. [p202]

P. Flores, M. Salicrú, A. Sánchez-Pla, and J. Ocaña. An equivalence test between features lists, based on the sorensen–dice index and the joint frequencies of go term enrichment. *BMC bioinformatics*, 23 (1):1–21, 2022. URL https://doi.org/10.1186/s12859-022-04739-2. [p202, 203, 205, 211]

R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, et al. Bioconductor: open software development for computational biology and bioinformatics. *Genome biology*, 5:1–16, 2004. URL https://doi.org/10.1186/gb-2004-5-10-r80. [p205]

D. A. Hosack, J. Dennis, Glynn, B. T. Sherman, H. C. Lane, and R. A. Lempicki. Identifying biological themes within lists of genes with EASE. *Genome Biology*, 4(10):R70, 2003. doi: 10.1186/gb-2003-4-10-r70. [p202]

K. K. Jain. Applications of biochips: from diagnostics to personalized medicine. *Current opinion in drug discovery & development*, 7(3):285–289, 2004. URL https://europepmc.org/article/med/15216931. [p201]

B. Jassal, L. Matthews, G. Viteri, C. Gong, P. Lorente, A. Fabregat, K. Sidiropoulos, J. Cook, M. Gillespie, R. Haw, et al. The reactome pathway knowledgebase. *Nucleic Acids Research*, 48(D1):D498–D503, 2020. URL https://doi.org/10.1093/nar/gkz1031. [p201]

D. Jovic, X. Liang, H. Zeng, L. Lin, F. Xu, and Y. Luo. Single-cell rna sequencing technologies and applications: A brief overview. *Clinical and translational medicine*, 12(3):e694, 2022. URL https://doi.org/10.1002/ctm2.694. [p201]

M. Kanehisa and S. Goto. Kegg: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, 28 (1):27–30, 2000. URL https://doi.org/10.1093/nar/28.1.27. [p201]

M. Kanehisa, M. Furumichi, Y. Sato, Y. Matsuura, and M. Ishiguro-Watanabe. Kegg: biological systems database as a model of the real world. *Nucleic Acids Research*, 53:D672–D677, 2025. URL https://doi.org/10.1093/nar/gkaa970. [p201]

J.-Y. Lee. The principles and applications of high-throughput sequencing technologies. *Development & Reproduction*, 27(1):9, 2023. URL https://doi.org/10.1007/s00438-023-01938-8. [p201]

H. Mi, X. Huang, A. Muruganujan, H. Tang, C. Mills, D. Kang, and P. D. Thomas. Panther version 11: expanded annotation data from gene ontology and reactome pathways, and data analysis tool enhancements. *Nucleic acids research*, 45(D1):D183–D189, 2017. URL https://doi.org/10.1093/nar/gkw1138. [p202]

A. Negi, A. Shukla, A. Jaiswar, J. Shrinet, and R. Jasrotia. Applications and challenges of microarray and rna-sequencing. *Bioinformatics*, pages 91–103, 2022. URL https://doi.org/10.1016/B978-0-323-89775-4.00016-X. [p201]

D. V. Nguyen, A. Bulak Arpat, N. Wang, and R. J. Carroll. Dna microarray experiments: biological and technological aspects. *Biometrics*, 58(4):701–717, 2002. URL https://doi.org/10.1111/j.0006-341X.2002.00701.x. [p201]

G. Pedersen and T. Kanigan. Clinical rna sequencing in oncology: where are we? *Personalized Medicine*, 13(3):209–213, 2016. URL https://doi.org/10.2217/pme-2016-0010. [p201]

C. Pesquita. Semantic similarity in the gene ontology. In *The gene ontology handbook*, pages 161–173. Humana Press, New York, NY, 2017. URL https://library.oapen.org/bitstream/handle/20.500.12657/27802/1/1002203.pdf#page=163. [p202]

A. Sanchez, J. Ocana, and M. Salicru. *goProfiles: goProfiles: an R package for the statistical analysis of functional profiles*, 2026. URL https://bioconductor.org/packages/goProfiles. R package version 1.74.0. [p203, 205]

M. Schena, D. Shalon, R. W. Davis, and P. O. Brown. Quantitative monitoring of gene expression patterns with a complementary dna microarray. *Science*, 270(5235):467–470, 1995. URL https://doi.org/10.1126/science.270.5235.467. [p201]

T. Sorensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on danish commons. *Biologiske skrifter*, 5:1–34, 1948. [p202]

K. Van den Berge, K. M. Hembach, C. Soneson, S. Tiberi, L. Clement, M. I. Love, R. Patro, and M. D. Robinson. Rna sequencing data: hitchhiker's guide to expression analysis. *Annual Review of Biomedical Data Science*, 2:139–173, 2019. URL https://doi.org/10.1146/annurev-biodatasci-072018-021255. [p201]

S. Wellek. *Testing statistical hypotheses of equivalence*. Chapman and Hall/CRC, 2002. [p205]

T. Wu, E. Hu, S. Xu, M. Chen, P. Guo, Z. Dai, T. Feng, L. Zhou, W. Tang, L. Zhan, xiaochong Fu, S. Liu, X. Bo, and G. Yu. clusterprofiler 4.0: A universal enrichment tool for interpreting omics data. *The Innovation*, 2(3):100141, 2021. URL https://doi.org/10.1016/j.xinn.2021.100141. [p203, 205]

G. Yu, L.-G. Wang, Y. Han, and Q.-Y. He. clusterprofiler: an r package for comparing biological themes among gene clusters. *OMICS: A Journal of Integrative Biology*, 16(5):284–287, 2012. URL https://doi.org/10.1089/omi.2011.0118. [p205]

*Pablo Flores*
*Escuela Superior Politécnica de Chimborazo (ESPOCH)*
*Facultad de Ciencias*
*Riobamba, Ecuador*
*ORCID: 0000-0002-7156-8547*
p_flores@espoch.edu.ec

*Jordi Ocaña*
*Universitat de Barcelona*
*Department of Genetics, Microbiology and Statistics*
*Barcelona, Spain*
*ORCID: 0000-0002-4736-6996*
jocana@ateneu.ub.edu

*Alex Sánchez*
*Universitat de Barcelona; Centro de Investigación Biomédica en Red de Fragilidad y Envejecimiento Saludable (CIBERFES)*
*Department of Genetics, Microbiology and Statistics*
*Barcelona, Spain*
*Instituto de Salud Carlos III*
*Madrid, Spain*
*ORCID: 0000-0002-8673-7737*
asanchez@ub.edu

*Miquel Salicrú*
*Universitat de Barcelona*
*Department of Genetics, Microbiology and Statistics*
*Barcelona, Spain*
*ORCID: 0000-0001-9644-5626*
msalicru@ub.edu

# MSTest: An R Package for Testing Markov Switching Models

*by Gabriel Rodriguez-Rondon and Jean-Marie Dufour*

**Abstract** We present the R package MSTest, which implements hypothesis testing procedures to determine the number of regimes in Markov switching models. These models have wide-ranging applications in economics, finance, and many other fields. MSTest provides several testing frameworks, including Monte Carlo likelihood ratio tests, moment-based tests, parameter stability tests, and classical likelihood ratio procedures. In addition, the package offers tools for simulating and estimating univariate and multivariate Markov switching and hidden Markov models using either the expectation–maximization algorithm or maximum likelihood estimation. The functionality of the package is demonstrated through simulation-based examples.

## 1 Introduction

Markov switching models were introduced by Goldfeld and Quandt (1973) but became an active area of research in economics after Hamilton (1989) proposed modeling the first difference of U.S. GNP as a nonlinear stationary process, where the nonlinearity arises from discrete shifts in the process.

These models have since been applied in a wide range of macroeconomic and financial settings. For example, they have been used to identify business cycles and provide probabilistic statements about the state of the economy (Chauvet, 1998; Chauvet and Hamilton, 2006; Chauvet et al., 2002; Diebold and Rudebusch, 1996; Hamilton, 1989; Kim and Nelson, 1999; Qin and Qu, 2026), to model stock market volatility using Markov switching ARCH, GARCH, and stochastic volatility models (Hamilton, 1994; Gray, 1996; Klaassen, 2002; Haas et al., 2004; Pelletier, 2006; So et al., 1998), to model interest rate dynamics (Cai, 1994; Garcia and Perron, 1996), to study state-dependent impulse response functions (Sims and Zha, 2006; Caggiano et al., 2017), to identify structural shocks in SVAR models (Lanne et al., 2010; Herwartz and Lütkepohl, 2014; Lütkepohl et al., 2021), and more recently to improve measures of core inflation by allowing for multiple inflation regimes (Rodriguez-Rondon, 2024; Ahn and Luciani, 2026; Le Bihan et al., 2024). Hamilton (2016) provides a detailed survey of regime switching models in macroeconomics.

Outside of macroeconomic and financial applications, these models have also been applied in climate change research (Golosov et al., 2014; Dietz and Stern, 2015), environmental and energy economics (Cevik et al., 2021; Charfeddine, 2017), and industrial organization (Aguirregabiria and Mira, 2007; Sweeting, 2013). Additionally, there is a related class of models—the hidden Markov model—which has been widely used in computational molecular biology (Krogh et al., 1994; Baldi et al., 1994), handwriting and speech recognition (Rabiner and Juang, 1986; Nag et al., 1986; Rabiner and Juang, 1993; Jelinek, 1997), computer vision and pattern recognition (Bunke and Caelli, 2001), and other machine learning applications.

Given their empirical relevance, determining the number of regimes needed to capture these nonlinearities is a key step in specifying a Markov switching model, since it is not determined endogenously during estimation. However, the asymptotic results underlying conventional testing procedures do not apply in this setting, because the required regularity conditions are violated. This has motivated a range of alternative testing procedures in the literature.

A range of procedures have been proposed for testing the null hypothesis of a linear model against a two-regime alternative (Hansen, 1992, 1996a; Garcia, 1998; Cho and White, 2007; Marmer, 2008; Carrasco et al., 2014; Kasahara et al., 2014; Dufour and Luger, 2017; Qu and Zhuo, 2021). Testing the null of an $M$-regime model against an $M + m$-regime alternative for $M \geq 1$ and $m = 1$ has been considered by Kasahara and Shimotsu (2018), who show that the parametric bootstrap test can be asymptotically valid under restrictions on the parameter space and for univariate models with fixed or predetermined regressors. Qu and Zhuo (2021) present similar bootstrap results for $M = m = 1$ for a broader class of univariate models, though still under constrained parameter spaces. More recently, Rodriguez-Rondon and Dufour (2026) propose Monte Carlo likelihood ratio tests that accommodate both $M \geq 1$ and $m \geq 1$ and extend to multivariate settings, which had not been addressed previously. Their procedures are the most general available and deal transparently with the violations of regularity conditions that arise in this setting. Importantly, they do not require parametric restrictions, normality of the errors, or stationarity of the underlying process, since they do not rely on the existence of an asymptotic distribution. As a result, they apply in a wider range of settings than the parametric bootstrap, including cases where the bootstrap's asymptotic validity has not been established. The maximized Monte Carlo version of the test further controls size in finite samples, which is particularly

relevant for macroeconomic applications with quarterly data that are often short, and is robust to the identification problems that are common in Markov switching models.

Because carrying out these tests is not always straightforward, and conducting more than one can quickly become tedious, we introduce **MSTest**, an R package designed to test the null hypothesis of $M$ regimes against the alternative of $M + m$ regimes for both univariate and multivariate models. By making these testing procedures readily available to a broad audience, the package allows users to compare different tests and determine the appropriate number of regimes for economic research and policy-relevant applications.

The package also allows users to simulate and estimate univariate and multivariate Markov switching models, as well as hidden Markov models, with estimation available through the expectation–maximization (EM) algorithm or maximum likelihood estimation (MLE). For computational efficiency, **MSTest** relies on **Rcpp** (Eddelbuettel and Balamuta, 2018) and **RcppArmadillo** (Eddelbuettel et al., 2026) and provides parallel computing options, which are especially valuable given the computational burden of testing in the presence of nuisance parameters. The package implements the methodologies of Rodriguez-Rondon and Dufour (2026), Dufour and Luger (2017), Carrasco et al. (2014), and Hansen (1992). The parametric bootstrap discussed by Qu and Zhuo (2021) and Kasahara and Shimotsu (2018) is not implemented explicitly, but it can be carried out using specific settings in combination with the local Monte Carlo likelihood ratio test of Rodriguez-Rondon and Dufour (2026).

The paper first introduces the Markov switching and hidden Markov models, then the testing procedures the package implements and how they address the regularity and identification problems specific to this setting. It then turns to the main focus of the article: the **MSTest** package itself—its principal functions, a typical simulation, estimation, and testing workflow, and practical guidance that complements the CRAN documentation. A brief summary concludes.

## 2 Markov switching models

The **MSTest** package considers Markov switching models in which only the mean and variance are governed by the Markov process $S_t$. We also consider the hidden Markov model, a special case with no autoregressive coefficients. In both cases, exogenous explanatory variables may be included.

The first-order Markov process $S_t$ that governs changes in the parameters of the Markov switching model is unobserved and evolves according to a first-order ergodic Markov chain with an $(M \times M)$ transition probability matrix given by

$$\mathbf{P} = \begin{bmatrix} p_{11} & \cdots & p_{M1} \\ \vdots & \ddots & \vdots \\ p_{1M} & \cdots & p_{MM} \end{bmatrix}$$

Here, $p_{ij} = P(S_t = j \mid S_{t-1} = i)$ is the probability that state $i$ is followed by state $j$, $M$ is the number of regimes, and $S_t \in \{1, \ldots, M\}$. The columns of $\mathbf{P}$ sum to one (i.e., $\sum_{j=1}^{M} p_{ij} = 1$ for all $i$). We can obtain the ergodic probabilities using

$$\boldsymbol{\pi} = (\mathbf{A}'\mathbf{A})^{-1}\mathbf{A}'\mathbf{e}_{M+1} \quad \& \quad \mathbf{A} = \begin{bmatrix} \mathbf{I}_M - \mathbf{P} \\ \mathbf{1}' \end{bmatrix}$$

where $\mathbf{e}_{M+1}$ is the $(M+1)$th column of $\mathbf{I}_{M+1}$. These ergodic probabilities tell us, on average, in the long run, the proportion of time the process $S_t$ spends in each regime.

A Markov switching model can be expressed as

$$y_t = \mu_{S_t} + \sum_{k=1}^{p} \phi_k (y_{t-k} - \mu_{S_{t-k}}) + Z_t \beta_z + \sigma_{S_t} \epsilon_t \tag{1}$$

where, in a univariate setting, $y_t$ is a scalar, $Z_t$ is a $(1 \times q_z)$ vector of exogenous variables whose coefficients do not depend on the latent Markov process $S_t$, and $\epsilon_t$ represents the error process, which, for example, may be distributed as a $\mathcal{N}(0, 1)$. The error term is multiplied by the standard deviation $\sigma_{S_t}$, which may either depend on the Markov process or remain constant throughout (i.e., $\sigma$).

This model is labeled `MSARmdl` in **MSTest** when exogenous regressors $Z_t$ are excluded, and `MSARXmdl` when included. Other error distributions, such as the Student-$t$, may be added in future versions. Currently, **MSTest** simulates only normally distributed errors, so we focus on this setup.

For the Markov switching model in equation (1) with $M$ regimes, the sample log likelihood

conditional on the first $p$ observations of $y_t$ is given by

$$L_T(\theta) = \log f(y_1^T | y_{-p+1}^0; \theta) = \sum_{t=1}^{T} \log f(y_t | \mathcal{Y}_{t-1}; \theta) \tag{2}$$

where $\mathcal{Y}_{t-1} = \sigma\text{-field}\{\ldots, Z_{t-1}, y_{t-2}, Z_t, y_{t-1}\}$ and $\theta = (\mu_1, \ldots, \mu_M, \beta, \sigma_1, \ldots, \sigma_M, vec(\mathbf{P}))$ and $vec(\cdot)$ is the vectorization operator, which stacks the columns of a matrix to form a column vector.

The conditional density $f(y_t \mid \mathcal{Y}_{t-1}; \theta)$ is obtained by summing the regime-specific densities over the possible configurations of the current and previous $p$ regimes, weighted by their conditional probabilities. The explicit expression can be found in (Hamilton, 1989, 1994; Rodriguez-Rondon and Dufour, 2026).

Krolzig (1997) generalized the univariate model to the Markov switching vector autoregressive (MS-VAR) model, which in the **MSTest** package can be expressed as

$$\boldsymbol{y}_t = \boldsymbol{\mu}_{S_t} + \boldsymbol{\Phi}_1(\boldsymbol{y}_{t-1} - \boldsymbol{\mu}_{S_{t-1}}) + \cdots + \boldsymbol{\Phi}_p(\boldsymbol{y}_{t-p} - \boldsymbol{\mu}_{S_{t-p}}) + Z_t\boldsymbol{\beta} + \boldsymbol{\Sigma}_{S_t}^{1/2}\boldsymbol{\epsilon}_t \tag{3}$$

where $\boldsymbol{y}_t = [y_{1,t}, \ldots, y_{q,t}]'$, $\boldsymbol{\mu}_{S_t} = [\mu_{1,S_t}, \ldots, \mu_{q,S_t}]'$, $\boldsymbol{\epsilon}_t = [\epsilon_{1,t}, \ldots, \epsilon_{q,t}]'$, $\boldsymbol{\Phi}_k$ is a $(q \times q)$ matrix containing the autoregressive parameters at lag $k$, $\boldsymbol{\beta}$ is now a $(q_z \times q)$ matrix, and $\boldsymbol{\Sigma}_{S_t} = \boldsymbol{\Sigma}_{S_t}^{1/2}(\boldsymbol{\Sigma}_{S_t}^{1/2})'$ is the $(q \times q)$ regime-dependent covariance matrix. As in the univariate setting, the **MSTest** package includes versions with (`MSVARXmdl`) and without (`MSVARmdl`) exogenous regressors. More sophisticated versions of the MS-VAR model, together with their likelihood functions, are described in Krolzig (1997).

Hidden Markov models (HMMs) are a special case of the Markov switching model defined above. Because they need not be applied to time series data, they typically exclude lags of the endogenous variable $y_t$. Removing them from (3) yields

$$\boldsymbol{y}_t = \boldsymbol{\mu}_{S_t} + Z_t\boldsymbol{\beta} + \sigma_{S_t}\epsilon_t \tag{4}$$

When $q = 1$, we recover a univariate HMM from (1). This version and its multivariate counterpart are the HMMs considered in the package **MSTest** and are labeled as `HMmdl`.

As An et al. (2013) note, dependence on past observations allows richer interactions between $y_t$ and $S_t$, capturing more complex relationships between economic and financial variables. Including past observations is common in economic time series, often to control for stochastic trends, which may explain why Markov switching models are more popular than basic HMMs in this literature.

Markov switching models are typically estimated by the expectation–maximization (EM) algorithm (Dempster et al., 1977), Bayesian methods, or the Kalman filter in a state-space representation. They can also be estimated by maximum likelihood (MLE), which is commonly done, but only when the number of regimes $M$ is small, since $S_t$ is latent and the likelihood can exhibit several modes of equal height and other irregularities that make optimization harder as $M$ grows.

In **MSTest**, the estimation functions described below use either the EM algorithm (`control = list(method = "EM")`) or MLE (`control = list(method = "MLE")`). Estimates can often be improved by using the EM results as starting values for the Newton-type (MLE) optimizer, a two-step procedure common in empirical work.

We omit the details of EM and MLE, since our focus is the **MSTest** package. Detailed treatments are given in (Hamilton, 1990, 1994), and, for MS-VAR models, in Krolzig (1997).

## 3 Hypothesis testing for number of regimes

The number of regimes in a Markov switching model must be specified by the researcher, since it is not determined endogenously during estimation. Testing for it is non-standard, however, because the regularity conditions underlying conventional procedures fail.

The general hypothesis of interest is

$$H_0 : M = M_0 \quad \text{against} \quad H_1 : M = M_0 + m$$

where, in principle, both $M_0, m \geq 1$, though most classical procedures only handle the simplest case, $H_0 : M = 1$ against $H_1 : M = 2$. Below we describe the main procedures, focusing on those in **MSTest**. The Monte Carlo LR procedures of Rodriguez-Rondon and Dufour (2026) are the most general, accommodating $M_0 \geq 1$, $m \geq 1$, non-Gaussian errors, non-stationary data, and multivariate models, though at a higher computational cost that can make other methods attractive in the simpler cases they were designed for. For each, we also note how it fits within the literature and handles identification failures and violations of regularity conditions.

Evidence on the size and power of the individual tests is available in the references given for each procedure below. In addition, Rodriguez-Rondon and Dufour (2026) provide a direct comparison of six of the seven tests implemented in **MSTest**, the LMC-LRT, MMC-LRT, the local and maximized moment-based tests of Dufour and Luger (2017), and the supTS and expTS tests (all but the Hansen test), under a common set of data-generating processes and show that all tests control size. Even in the simple $M_0 = m = 1$ case, the LMC-LRT is the most powerful in the large majority of settings. For example, it has the highest power when only the mean switches, a case in which the moment-based and CHP tests can have substantially lower power, especially when the autoregressive persistence is high. All tests tend to improve when the variance switches, but even then the LMC-LRT typically leads, motivating its use even in simple settings, despite its greater computational cost.

### 3.1 Likelihood ratio–type tests

Hansen (1992) was the first to propose a testing procedure for Markov switching models when $M_0 = m = 1$, and we begin with it, as it is available in **MSTest** (`HLRTest()`; see Table 4). Hansen discusses several issues that complicate the likelihood ratio approach in this setting. First, the likelihood is usually assumed locally quadratic near the null and the global optimum, but because some parameters are unidentified under the null, it is instead flat in those directions. Such unidentified nuisance parameters have been studied by (Davies, 1977, 1987; Andrews and Ploberger, 1994; Dufour, 2006). Second, the score, usually assumed positive, can be identically 0 at the restricted maximum likelihood estimator of the linear (null) model. Third, some parameters, such as the transition probabilities, may take the boundary values 0 or 1, giving rise to the parameter boundary problem (Andrews, 1999, 2001). In addition, the likelihood surface may have multiple local optima, so the null need not lie in the same region as the global optimum.

To address these issues, Hansen (1992) models the likelihood function as an empirical process of the unknown parameters and applies empirical process theory to obtain a bound on the asymptotic distribution of a standardized likelihood ratio statistic. The nuisance parameters, which include the second-regime parameters $(\mu_2, \sigma_2)$ and the transition probabilities $(p_{11}, p_{22})$, are not identified under the null, so the statistic is evaluated over a grid of values for these parameters and optimized accordingly. Because the underlying empirical process may be serially correlated for some parameter values, Hansen (1996a) proposes a correction that is applied when computing the asymptotic distribution. This correction is incorporated in the implementation in **MSTest**.

This procedure has two main drawbacks. First, it provides only a bound for the standardized likelihood ratio statistic, so it can be conservative, and the critical values reported by **MSTest** are those of the process $Q$ rather than of the statistic itself. Second, it optimizes over the nuisance parameters by grid search, which becomes computationally intensive as more parameters are allowed to switch. We nonetheless include it because it is a common benchmark in the literature.

Several likelihood ratio–based procedures have followed Hansen (1992). Garcia (1998) reduce the dimensionality of the nuisance parameter space but rely on asymptotic assumptions known to be problematic in Markov switching settings (Hansen, 1996b; Andrews and Ploberger, 1994), and Cho and White (2007) develop a quasi-likelihood ratio test that accounts for parameters on the boundary but may not properly handle the time dependence induced by autoregressive dynamics (Carter and Steigerwald, 2012; Cho and White, 2011); for these reasons neither is included in **MSTest**. More recently, Qu and Zhuo (2021) and Kasahara and Shimotsu (2018) establish the asymptotic validity of parametric bootstrap procedures for likelihood ratio tests, the former for a broad class of univariate models and the latter for testing $M_0 \geq 1$ against $M_0 + 1$ regimes under fixed or predetermined regressors. While these parametric bootstrap procedures are not implemented directly in **MSTest**, they can be replicated using the local Monte Carlo likelihood ratio test of Rodriguez-Rondon and Dufour (2026) described below, which applies in even more general settings.

Rodriguez-Rondon and Dufour (2026) propose the maximized and local Monte Carlo likelihood ratio tests (MMC-LRT and LMC-LRT), the most general procedures currently available (implemented as `MMCLRTest()` and `LMCLRTest()`; see Table 4). They apply when both $M_0 \geq 1$ and $m \geq 1$, remain valid under non-stationarity, non-Gaussian errors, and parameters on the boundary, and are the only tests available for multivariate Markov switching models, including the MS-VAR and multivariate hidden Markov models discussed above. The MMC-LRT additionally controls size in finite samples and is robust to identification problems, which is especially relevant for macroeconomic applications.

Here, we provide a brief overview of the MMC-LRT and LMC-LRT procedures. A formal treatment is given in Rodriguez-Rondon and Dufour (2026). Both procedures are based on the likelihood ratio statistic $LR_T = 2[\bar{L}_T(H_1) - \bar{L}_T(H_0)]$, where $\bar{L}_T(H_1)$ and $\bar{L}_T(H_0)$ are the maximized log-likelihoods under the alternative and the null, with the log-likelihood as defined above. Under a null of $M = 1$, the parameter vector $\theta_0 = (\mu, \sigma^2, \phi_1, \ldots, \phi_p)'$ collects the nuisance parameters, since the null distribution of $LR_T$ depends on $\theta_0$. Because the model is parametric, for any $\theta_0$ we can simulate $N$ i.i.d. replications

$LR_T^{(1)}(\theta_0), \ldots, LR_T^{(N)}(\theta_0)$ of the statistic, and, provided the observed statistic $LR_T^{(0)}$ and the simulated statistics are exchangeable for some $\theta_0 \in \bar{\Omega}_0$, the Monte Carlo $p$-value is

$$\hat{p}_N[x \mid \theta_0] = \frac{N + 1 - R_{LR}[LR_T^{(0)}; N]}{N + 1} \tag{5}$$

where $R_{LR}[LR_T^{(0)}; N] = \sum_{i=1}^{N} I[LR_T^{(0)} \geq LR_T^{(i)}(\theta_0)]$ and $I(C) := 1$ if condition $C$ holds, and $I(C) = 0$ otherwise. As can be seen, $R_{LR}[LR_T^{(0)}; N]$ simply computes the rank of the test statistic from the observed data within the generated series $LR(N, \theta_0)$. As shown in Rodriguez-Rondon and Dufour (2026), a critical region for this test statistic with level $\alpha$ is given by $\sup_{\theta_0 \in \bar{\Omega}_0} \hat{p}_N[LR_T^{(0)} \mid \theta_0] \leq \alpha$. More precisely, if $(N+1)\alpha$ is an integer, then

$$\mathbb{P}\left[\sup\{\hat{p}_N[LR_T^{(0)} \mid \theta_0] : \theta_0 \in \bar{\Omega}_0\} \leq \alpha\right] \leq \alpha$$

under the null hypothesis, so it is a valid test with level $\alpha$ for $H_0$. Because this result does not depend on the sample size $T$, the test is also valid in finite samples.

This maximized Monte Carlo procedure requires searching for the maximum Monte Carlo $p$-value over the nuisance space $\bar{\Omega}_0$. Because this space can be large, following Dufour (2006) the maximization is carried out over a consistent set $C_T$ built around a consistent estimate $\hat{\theta}_0$. **MSTest** lets users define $C_T$ from a confidence interval around $\hat{\theta}_0$ ($C_T^{CI}$), from a fixed-radius set ($C_T^{\epsilon}$), or from their union.

As discussed in Dufour (2006) and Rodriguez-Rondon and Dufour (2026), the solution to this optimization problem need not be unique, meaning that the maximum Monte Carlo $p$-value may correspond to multiple parameter vectors. For this reason, derivative-free numerical optimization methods are recommended to locate the maximum Monte Carlo $p$-value over the nuisance parameter space. **MSTest** allows users to employ the generalized simulated annealing algorithm (**GenSA**), the genetic algorithm (**GA**), and particle swarm optimization (**pso**); see (Xiang et al., 2013; Zambrano-Bigiarini et al., 2013; Scrucca, 2013; Dufour, 2006; Dufour and Neves, 2019).

Choosing $C_T$ to be the singleton $\{\hat{\theta}_0\}$ yields the local Monte Carlo test (LMC-LRT), a finite-sample analogue of the parametric bootstrap that requires only a consistent estimate $\hat{\theta}_0$ and, unlike the bootstrap, does not require $N \to \infty$. As shown in Rodriguez-Rondon and Dufour (2026), both the LMC-LRT and the MMC-LRT remain valid even when no asymptotic distribution exists, and the parametric bootstrap procedures of Qu and Zhuo (2021) and Kasahara and Shimotsu (2018) can be reproduced as special cases by constraining the nuisance space accordingly.

### 3.2 Moment-based tests

Dufour and Luger (2017) propose an alternative approach to testing Markov switching models that avoids the statistical issues associated with likelihood ratio–type tests, is computationally less demanding, and perfectly controls the size of the test through the Monte Carlo testing methods developed in Dufour (2006) (implemented as `DLMCTest()` and `DLMMCTest()`; see Table 4). However, their method is restricted to the case where $M_0 = m = 1$.

The moment-based test of Dufour and Luger (2017) is based on computing moments of the least squares residuals from autoregressive models estimated under the null hypothesis of $M_0 = 1$. More specifically, they focus on the mean, variance, skewness, and excess kurtosis of the least squares residuals. These moments are calculated as

$$M(\hat{\epsilon}) = \frac{|m_2 - m_1|}{\sqrt{s_1^2 + s_2^2}}, \quad V(\hat{\epsilon}) = \frac{\vartheta_2(\hat{\epsilon})}{\vartheta_1(\hat{\epsilon})}, \quad S(\hat{\epsilon}) = \left|\frac{\sum_{t=1}^{T} \hat{\epsilon}_t^3}{T(\hat{\sigma}^2)^{3/2}}\right|, \quad K(\hat{\epsilon}) = \left|\frac{\sum_{t=1}^{T} \hat{\epsilon}_t^4}{T(\hat{\sigma}^2)^2} - 3\right|$$

where the truncated means $m_1, m_2$, the truncated variances $s_1^2, s_2^2$, the variance components $\vartheta_1, \vartheta_2$, and $\hat{\sigma}^2 = T^{-1}\sum_{t=1}^{T} \hat{\epsilon}_t^2$ are computed from the negative and positive parts of the least squares residuals, as defined in Dufour and Luger (2017).

The testing procedure computes a test statistic for each moment, obtains the corresponding individual $p$-values, and combines them using one of two standard methods for aggregating independent tests: a minimum-based combination (Tippett, 1931; Wilkinson, 1951), or a product-based combination (Fisher, 1932; Pearson, 1933) (see Dufour et al., 2004, 2015, for further discussion). The Monte Carlo $p$-value of the combined test statistic is then obtained from its rank among the values simulated under $\eta \sim N(0, I_T)$, in the same way as for the likelihood ratio statistic above.

Because the transition probabilities, mean, and variance need not be treated as nuisance parameters—only the parameters on explanatory variables may be unidentified under the null—the test avoids much of the identification difficulty that arises in Hansen (1992), Garcia (1998), and Carrasco et al.

(2014), and it extends easily to a maximized Monte Carlo framework. Although it is limited to comparing linear models against two-regime alternatives only, it is the least computationally intensive procedure available and can be computed in seconds, even in its maximized Monte Carlo version.

### 3.3 Optimal test for regime switching

Carrasco et al. (2014) propose a test for parameter constancy in random coefficient and Markov switching models that extends the information matrix test of White (1982). Like the moment-based test of Dufour and Luger (2017), it applies only to the case where $M_0 = m = 1$ and only requires estimation under the null, which is attractive in Markov switching settings where estimating the alternative is computationally demanding because of nonlinearity and multiple local optima. In contrast, likelihood ratio–based procedures such as those of Hansen (1992), Garcia (1998), Cho and White (2007), Qu and Zhuo (2021), Kasahara and Shimotsu (2018), and Rodriguez-Rondon and Dufour (2026) require estimation under both hypotheses.

Although Carrasco et al. (2014) show that their test is asymptotically locally equivalent to the likelihood ratio test, simulation evidence in (Rodriguez-Rondon and Dufour, 2026; Qu and Zhuo, 2021) suggests that likelihood ratio–based approaches can be more powerful in some settings, such as when only the mean switches and the autoregressive persistence is high. The procedure also involves bootstrapping and optimization over nuisance parameters, which can make it more computationally intensive than the moment-based test of Dufour and Luger (2017) or the Monte Carlo likelihood ratio tests of Rodriguez-Rondon and Dufour (2026), particularly when the variance switches.

Carrasco et al. (2014) test the null of parameter constancy, $\theta_t = \theta_0$, against the alternative $\theta_t = \theta_0 + \eta_t$, where the switching variable $\eta_t$ is unobservable, stationary, and may depend on nuisance parameters $\beta$. Their statistic uses the second derivatives of the log-likelihood and the outer products of the scores, as in the information matrix test, together with an extra term that captures the serial dependence of the time-varying coefficients, so it depends on the latent process $\eta_t$ only through its second-order properties. To handle the nuisance parameters, Carrasco et al. (2014) follow Davies (1987) and set $\eta_t = chS_t$, where $c$ is the amplitude of the change, $h$ its direction, and $S_t$ a bounded, mean-zero AR(1) process, collecting the nuisance parameters in $\beta = (c^2, h', \rho')$. From the second-order properties of the score they obtain the supremum-type statistic

$$\text{supTS} = \sup_{\{h,\rho:||h||=1,\underline{\rho}<\rho<\bar{\rho}\}} \frac{1}{2}(max(0, \frac{\Gamma_T^*}{\sqrt{\hat{\epsilon^*}'\hat{\epsilon^*}}}))^2$$

where $\Gamma_T^*$ and $\hat{\epsilon}^*$ are normalized quantities that do not depend on $c^2$, defined in Carrasco et al. (2014). The test bootstraps over the distribution of the nuisance parameters, for which **MSTest** uses a uniform prior. Because $c^2$ need not be bounded above, they also suggest an exponential-type statistic following Andrews and Ploberger (1994),

$$expTS = \int_{\{\underline{\rho}\leq\rho\leq\bar{\rho},||h||<1\}} \Psi(h,\rho)d\rho dh$$

where $\Psi(h,\rho)$ is defined in Carrasco et al. (2014).

The tests of Carrasco et al. (2014) have been widely used in empirical applications (Hamilton, 2005; Warne and Vredin, 2006; Kahn and Rich, 2007; Morley and Piger, 2012; Dufrénot et al., 2011), in testing MS-GARCH models (Hu and Shin, 2008), and as benchmarks (Dufour and Luger, 2017; Qu and Zhuo, 2021; Rodriguez-Rondon and Dufour, 2026). Given their broad use and optimality properties, they are included in **MSTest** (`CHPTest()`; see Table 4).

## 4 The R package MSTest

The **MSTest** package is designed to conduct hypothesis tests for the number of regimes in Markov switching models. Because many of these procedures require estimating restricted and unrestricted models and simulating under the null, the package also provides tools for simulating and estimating Markov switching models. These features primarily support the testing procedures and are not the main focus of the package.

What distinguishes **MSTest** is its implementation of testing procedures that remain valid under violations of standard regularity conditions, together with its use of **Rcpp** and parallel computing, which make it feasible to handle nuisance parameters and Monte Carlo–based procedures in practice. Although other software exists for simulating and estimating Markov switching models, this combination of generality, computational efficiency, and modern hypothesis tests sets **MSTest** apart.

| Label | Series | Source | Sample span |
|---|---|---|---|
| `hamilton84GNP` | Real GNP | Hamilton (1989) | 1951Q2–1984Q4 |
| `chp10GNP` | Real GNP | Carrasco et al. (2014) | 1951Q2–2010Q4 |
| `USGNP` | Real GNP | Rodriguez-Rondon and Dufour (2026) | 1947Q2–2024Q2 |
| `USRGDP` | Real GDP | Rodriguez-Rondon and Dufour (2026) | 1947Q2–2024Q2 |

**Table 1:** U.S. real GNP and GDP data sets included in **MSTest**

| Function | Description |
|---|---|
| `simuNorm` | Normally distributed process, optionally with exogenous regressors. |
| `simuAR` | Autoregressive process, AR($p$). |
| `simuARX` | AR($p$) process with exogenous regressors. |
| `simuVAR` | Vector autoregressive process, VAR($p$). |
| `simuVARX` | VAR($p$) process with exogenous regressors. |
| `simuMSAR` | Markov switching AR($p$) process. |
| `simuMSARX` | Markov switching AR($p$) process with exogenous regressors. |
| `simuMSVAR` | Markov switching VAR($p$) process. |
| `simuMSVARX` | Markov switching VAR($p$) process with exogenous regressors. |
| `simuHMM` | Hidden Markov model (HMM), optionally with exogenous regressors. |

**Table 2:** Simulation functions available in **MSTest**

### 4.1 Data sets

The **MSTest** package includes three samples of U.S. real GNP and one of U.S. real GDP, all readily accessible once the package is loaded. These series have been used by Hansen (1992), Carrasco et al. (2014), Dufour and Luger (2017), and Rodriguez-Rondon and Dufour (2026), among others, to test for the number of regimes in Markov switching models. Table 1 reports the label, series, source, and sample span of each.

These data sets can be loaded using the following commands once **MSTest** has been attached:

```
data("hamilton84GNP", package = "MSTest")
data("chp10GNP", package = "MSTest")
data("USGNP", package = "MSTest")
data("USRGDP", package = "MSTest")
```

Each data set has three columns: a `Date` column of class `Date` (constructed with `as.Date()`), the level of real GNP or GDP (`GNP` or `RGDP`), and the corresponding growth rate (`GNP_gr` or `RGDP_gr`).

### 4.2 Simulation

Simulation plays a central role in **MSTest**, as many of the testing procedures rely on Monte Carlo methods to obtain valid critical values in the presence of nuisance parameters and identification failures. The simulation functions are used primarily by the hypothesis testing procedures that build sample or asymptotic null distributions, such as `LMCLRTest`, `MMCLRTest`, `DLMCTest`, `DLMMCTest`, and `CHPTest`. They may also be useful to users developing new estimation or testing procedures for Markov switching models who wish to assess performance through controlled experiments.

Table 2 lists the simulation functions provided by **MSTest** and the classes of processes they generate. Each function takes as input a `List` specifying the data-generating process (DGP). We present illustrative examples below, and a complete description of each function's arguments is available in the package documentation on CRAN.

As an illustration, consider the `simuNorm` function, which generates a univariate or multivariate Gaussian process. The required input is a `List` containing the sample size (`n`), the number of series (`q`) (where `q=1` indicates a univariate process and `q>1` a multivariate one), a ($q \times 1$) vector of means, and a ($q \times q$) covariance matrix. Users may optionally specify a `burnin` period to discard initial observations, which defaults to zero for `simuNorm` but is larger for autoregressive processes to reduce sensitivity to initialization, or supply a custom matrix of innovations via the `eps` argument. The code below applies these functions to simulate several linear and regime-switching processes, including multivariate normal, AR, VAR, Markov switching AR and VAR, and hidden Markov models.

```
mdl_norm <- list(n     = 500,
```

```
                q     = 2,
                mu    = c(5, -2),
                sigma = rbind(c(5.0, 1.5),
                              c(1.5, 1.0)))
simu_norm <- simuNorm(mdl_norm)

mdl_ar <- list(n     = 500,
               mu    = 5,
               sigma = 1,
               phi   = c(0.75))
simu_ar <- simuAR(mdl_ar)

mdl_var <- list(n     = 500,
                p     = 1,
                q     = 2,
                mu    = c(5, -2),
                sigma = rbind(c(5.0, 1.5),
                              c(1.5, 1.0)),
                phi   = rbind(c(0.50, 0.30),
                              c(0.20, 0.70)))
simu_var <- simuVAR(mdl_var)

mdl_hmm <- list(n     = 500,
                q     = 2,
                mu    = rbind(c(5, -2),
                              c(10, 2)),
                sigma = list(rbind(c(5.0, 1.5),
                                   c(1.5, 1.0)),
                             rbind(c(7.0, 3.0),
                                   c(3.0, 2.0))),
                k     = 2,
                P     = rbind(c(0.95, 0.10),
                              c(0.05, 0.90)))
simu_hmm <- simuHMM(mdl_hmm)

mdl_ms <- list(n     = 500,
               mu    = c(5,10),
               sigma = c(1,1),
               phi   = c(0.75),
               k     = 2,
               P     = rbind(c(0.95, 0.10),
                             c(0.05, 0.90)))
simu_msar <- simuMSAR(mdl_ms)

mdl_msvar <- list(n     = 500,
                  p     = 1,
                  q     = 2,
                  mu    = rbind(c(5, -2),
                                c(10, 2)),
                  sigma = list(rbind(c(5.0, 1.5),
                                     c(1.5, 1.0)),
                               rbind(c(7.0, 3.0),
                                     c(3.0, 2.0))),
                  phi   = rbind(c(0.50, 0.30),
                                c(0.20, 0.70)),
                  k     = 2,
                  P     = rbind(c(0.95, 0.10),
                                c(0.05, 0.90)))
simu_msvar <- simuMSVAR(mdl_msvar)
```

Figure 1 shows the simulated processes. Regime switching is clearly visible in the Markov switching specifications, most notably when comparing the autoregressive process (middle left) with its Markov switching counterpart (middle right), and similar differences arise between the VAR and MS-VAR processes.

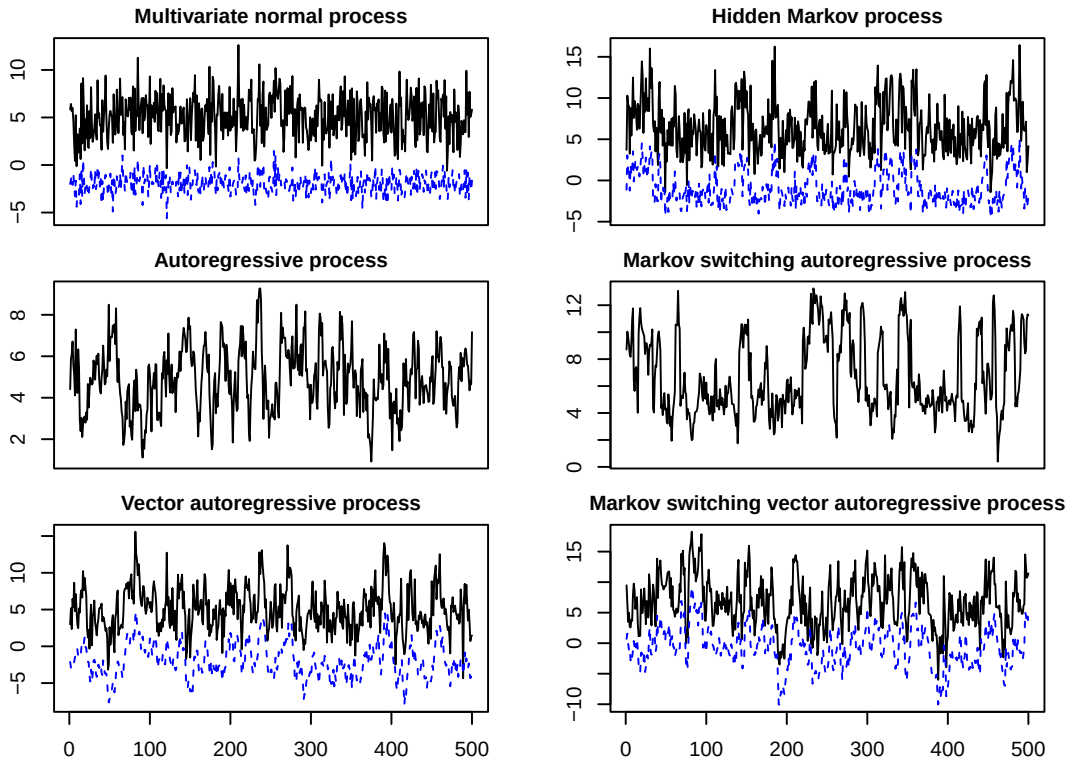



**Figure 1:** Simulated linear (left column) and Markov switching (right column) processes. Abrupt shifts between regimes are visible in the Markov switching processes.

### 4.3 Model estimation

**MSTest** can estimate the ten model classes listed in Table 3, together with their **MSTest** labels and specifications. Although `Nmdl` and `HMmdl` use multivariate notation, both automatically detect a univariate setting when given a $(T \times 1)$ vector. Labels containing an "X" allow exogenous regressors, though these can also be included in `Nmdl` and `HMmdl` despite their names.

The simulation functions from the previous subsection generated a multivariate hidden Markov process ($q = 2$ series), a Markov switching autoregressive process ($p = 1$ lag), and a Markov switching vector autoregressive process ($p = 1$ lag, $q = 2$ series). Each returns a list with the simulated data, the true latent state variable $S_t$, and other DGP elements. These series serve as inputs to the corresponding estimation functions below, for the hidden Markov model and the two Markov switching models.

```
control <- list(msmu   = TRUE,
                msvar  = TRUE,
                method = "EM",
                use_diff_init = 30)
```

| Model | Label | Equation |
|---|---|---|
| N($\boldsymbol{\mu} + \boldsymbol{x}_t\boldsymbol{\beta}, \boldsymbol{\Sigma}$) | `Nmdl` | $\boldsymbol{y}_t = \boldsymbol{\mu} + \boldsymbol{x}_t\boldsymbol{\beta} + \boldsymbol{\Sigma}^{1/2}\boldsymbol{\epsilon}_t$ |
| AR($p$) | `ARmdl` | $y_t = \mu + \sum_{k=1}^{p} \phi_k(y_{t-k} - \mu) + \sigma\epsilon_t$ |
| ARX($p$) | `ARXmdl` | $y_t = \mu + \sum_{k=1}^{p} \phi_k(y_{t-k} - \mu) + \boldsymbol{x}_t\boldsymbol{\beta} + \sigma\epsilon_t$ |
| VAR($p$) | `VARmdl` | $\boldsymbol{y}_t = \boldsymbol{\mu} + \sum_{k=1}^{p} (\boldsymbol{y}_{t-k} - \boldsymbol{\mu})\boldsymbol{\Phi}_k + \boldsymbol{\Sigma}^{1/2}\boldsymbol{\epsilon}_t$ |
| VARX($p$) | `VARXmdl` | $\boldsymbol{y}_t = \boldsymbol{\mu} + \sum_{k=1}^{p} (\boldsymbol{y}_{t-k} - \boldsymbol{\mu})\boldsymbol{\Phi}_k + \boldsymbol{x}_t\boldsymbol{\beta} + \boldsymbol{\Sigma}^{1/2}\boldsymbol{\epsilon}_t$ |
| MS-AR($p$) | `MSARmdl` | $y_t = \mu_{S_t} + \sum_{k=1}^{p} \phi_k(y_{t-k} - \mu_{S_{t-k}}) + \sigma_{S_t}\epsilon_t$ |
| MS-ARX($p$) | `MSARXmdl` | $y_t = \mu_{S_t} + \sum_{k=1}^{p} \phi_k(y_{t-k} - \mu_{S_{t-k}}) + \boldsymbol{x}_t\boldsymbol{\beta} + \sigma_{S_t}\epsilon_t$ |
| MS-VAR($p$) | `MSVARmdl` | $\boldsymbol{y}_t = \boldsymbol{\mu}_{S_t} + \sum_{k=1}^{p} \boldsymbol{\Phi}_k(\boldsymbol{y}_{t-k} - \boldsymbol{\mu}_{S_{t-k}}) + \boldsymbol{\Sigma}_{S_t}^{1/2}\boldsymbol{\epsilon}_t$ |
| MS-VARX($p$) | `MSVARXmdl` | $\boldsymbol{y}_t = \boldsymbol{\mu}_{S_t} + \sum_{k=1}^{p} \boldsymbol{\Phi}_k(\boldsymbol{y}_{t-k} - \boldsymbol{\mu}_{S_{t-k}}) + \boldsymbol{x}_t\boldsymbol{\beta} + \boldsymbol{\Sigma}_{S_t}^{1/2}\boldsymbol{\epsilon}_t$ |
| HMM | `HMmdl` | $\boldsymbol{y}_t = \boldsymbol{\mu}_{S_t} + \boldsymbol{x}_t\boldsymbol{\beta} + \boldsymbol{\Sigma}_{S_t}^{1/2}\boldsymbol{\epsilon}_t$ |

**Table 3:** Models and their specifications available in **MSTest**.

```
mdl_est_hmm <- HMmdl(simu_hmm[["y"]], k = 2, control = control)
summary(mdl_est_hmm)

#> Hidden Markov Model
#>               coef     s.e.
#> mu_1,1    5.151200 0.119030
#> mu_2,1   -2.007400 0.053269
#> mu_1,2    9.628800 0.236840
#> mu_2,2    1.821400 0.117870
#> sig_11,1  4.973200 0.375510
#> sig_12,1  1.388800 0.140140
#> sig_22,1  0.962750 0.075585
#> sig_11,2  7.384600 0.909040
#> sig_12,2  3.006800 0.422390
#> sig_22,2  1.738000 0.227480
#> p_11      0.961300 0.010523
#> p_12      0.038702 0.010523
#> p_21      0.099480 0.026777
#> p_22      0.900520 0.026777
#>
#> log-likelihood =  -1796.469
#> AIC =  3620.939
#> BIC =  3679.943
#>
#> Residuals:
#>        Min       1Q     Median      3Q    Max
#> Y1 -6.5756 -1.68620 -0.0424050 1.72610 6.8022
#> Y2 -2.9919 -0.75449  0.0072404 0.75641 3.2839

control <- list(msmu   = TRUE,
                msvar  = FALSE,
                method = "EM",
                use_diff_init = 30)
mdl_est_msar <- MSARmdl(simu_msar[["y"]], p = 1, k = 2, control = control)
summary(mdl_est_msar)

#> Markov Switching Autoregressive Model
#>            coef     s.e.
#> mu_1   4.994000 0.160730
#> mu_2  10.041000 0.186710
#> phi_1  0.715960 0.031774
#> sig    0.895540 0.057329
#> p_11   0.953170 0.011205
#> p_12   0.046832 0.011205
#> p_21   0.092460 0.022379
#> p_22   0.907540 0.022379
#>
#> log-likelihood =  -797.9881
#> AIC =  1611.976
#> BIC =  1645.677
#>
#> Residuals:
#>       Min       1Q    Median      3Q    Max
#> Y1 -3.723 -0.62857 -0.021467 0.61808 2.8027

control <- list(msmu   = TRUE,
                msvar  = TRUE,
                method = "EM",
                use_diff_init = 30)
mdl_est_msvar <- MSVARmdl(simu_msvar[["y"]], p = 1, k = 2, control = control)
summary(mdl_est_msvar)

#> Markov Switching Vector Autoregressive Model
#>               coef     s.e.
```

```
#> mu_1,1    4.897400 0.543750
#> mu_2,1   -2.099200 0.509290
#> mu_1,2   10.290000 0.581890
#> mu_2,2    2.141000 0.547620
#> phi_1,11  0.508240 0.063077
#> phi_1,12  0.287540 0.075495
#> phi_1,21  0.204040 0.031217
#> phi_1,22  0.687860 0.037718
#> sig_11,1  5.561300 0.460000
#> sig_12,1  1.911700 0.193760
#> sig_22,1  1.204800 0.103680
#> sig_11,2  7.440600 0.830910
#> sig_12,2  3.343700 0.412780
#> sig_22,2  2.115700 0.239960
#> p_11      0.927340 0.015909
#> p_12      0.072658 0.015909
#> p_21      0.136240 0.030757
#> p_22      0.863760 0.030757
#>
#> log-likelihood =  -1868.071
#> AIC =  3772.142
#> BIC =  3847.969
#>
#> Residuals:
#>        Min       1Q     Median      3Q    Max
#> Y1 -9.8948 -1.63520  0.0415660 1.59780 7.6673
#> Y2 -3.7363 -0.81008 -0.0036109 0.83766 3.2878
```

In all three examples, `method = "EM"` in the `control` list estimates the models by the expectation–maximization (EM) algorithm, while `method = "MLE"` instead performs direct maximum likelihood estimation. The arguments `msmu` and `msvar` control whether the mean and variance switch across regimes, and setting either to `FALSE` holds the corresponding parameter constant across regimes. For the Markov switching AR example we set `msvar = FALSE` because its data-generating process holds the variance constant across regimes.

The option `use_diff_init = 30` runs the optimizer thirty times from different initial values, retaining the highest-log-likelihood solution as the primary output while storing all runs in the `trace` element. All estimation functions return `S3` objects with dedicated `print()` and `summary()` methods. The `summary()` method reports parameter estimates, log-likelihood values, information criteria (AIC and BIC), and diagnostics such as residual quantiles, which in this controlled setting can be compared directly with the true DGP values.

The `summary()` method also reports asymptotic standard errors for the estimated parameters, computed from the inverse of the observed information matrix (the numerically approximated Hessian of the log-likelihood) evaluated at the estimate. Large within-regime standard errors can serve as an informal indication that a model with fewer regimes may be adequate, which in turn motivates the formal testing procedures that are the focus of **MSTest**. These standard errors are valid under the usual regularity conditions for a given, correctly specified $M$-regime model. They should be interpreted with caution near the boundary of the parameter space (for example, transition probabilities close to 0 or 1) or under weak regime separation, where the normal approximation may be poor, and **MSTest** returns `NA` in such cases. They concern inference within a fixed $M$-regime model and are distinct from the non-standard problem of testing the number of regimes, which is what the test procedures described below are designed to address.

Figure 2 displays the simulated series (black and blue), the true latent regime states $S_t$ (red, dashed), and the estimated smoothed regime probabilities from the fitted models (green, solid). The smoothed probabilities track the true regime changes closely across all three models, and particularly well for the hidden Markov and Markov switching autoregressive models. Regime classification is somewhat less precise over certain intervals in the Markov switching VAR case, reflecting the greater complexity of multivariate models, which can typically be mitigated by a larger sample size or more initial values.

### 4.4 Hypothesis testing

This section describes the hypothesis testing functions available in **MSTest**. Designed for ease of use, the package in most cases requires only the series $y_t$ (and $Z_t$ when applicable) and the number of regimes to be tested.

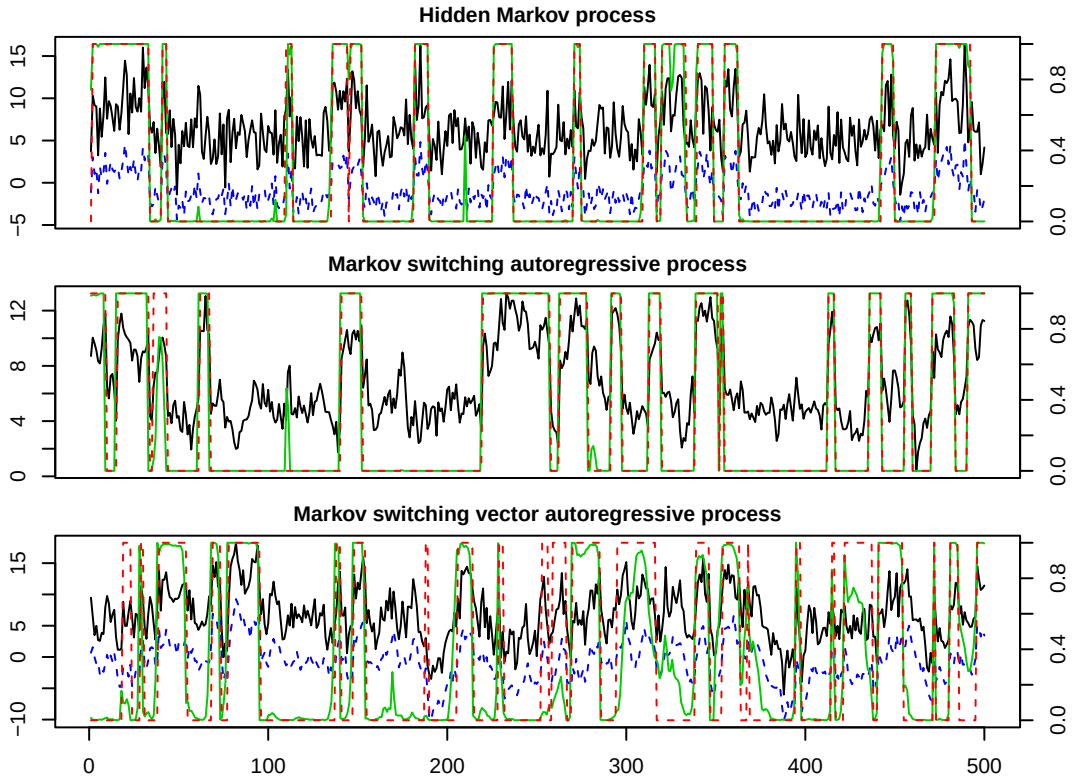



**Figure 2:** Simulated processes (black and blue), true regimes (red), and smoothed probabilities (green) from the estimated models. The smoothed probabilities closely track the true regime paths.

| Function | Description | $H_0$: $M_0$ vs. $H_1$: $M_0+m$ | Multiv. | Ident. robust | Non-stat. | Cost |
|---|---|---|---|---|---|---|
| `LMCLRTest` | Local Monte Carlo LRT | $M_0$ & $m \geq 1$ | ✓ | ✓ | ✓ | High |
| `MMCLRTest` | Maximized Monte Carlo LRT | $M_0$ & $m \geq 1$ | ✓ | ✓ | ✓ | High |
| `DLMCTest` | Local MC moment-based test | $M_0 = m = 1$ | – | ✓ | ✓* | Low |
| `DLMMCTest` | Maximized MC moment-based test | $M_0 = m = 1$ | – | ✓ | ✓* | Low |
| `CHPTest` | Optimal parameter-constancy test | $M_0 = m = 1$ | – | – | – | Med. |
| `HLRTest` | LRT empirical bound | $M_0 = m = 1$ | – | – | – | High |

**Table 4:** Hypothesis tests available in **MSTest**. Multiv.: multivariate models; Ident. robust: valid under identification failure without relying on an asymptotic distribution; Non-stat.: valid under non-stationarity.

Table 4 summarizes the hypothesis testing functions currently implemented in **MSTest** and reports, for each, the number of regimes it can test, its applicability to multivariate models, its identification robustness, its validity under non-stationarity, and its relative computational cost, as discussed above for each procedure. The `DLMCTest` and `DLMMCTest` procedures were not originally designed for non-stationary data, but Rodriguez-Rondon and Dufour (2026) provide simulation evidence that they control size and retain power under a random walk, which is why they carry an asterisk (*) in Table 4. Each returns the test statistic, its $p$-value, and, when applicable, critical values and parameter estimates under the null and alternative hypotheses.

The reported critical values warrant clarification. For `HLRTest()`, they are those of the process $Q$ introduced in Hansen (1992), which bounds the likelihood ratio statistic from above rather than providing critical values for the LR statistic itself. For `LMCLRTest()`, `MMCLRTest()`, `DLMCTest()`, and `DLMMCTest`, they are empirical quantiles of the simulated null distribution. Throughout the package, "critical values" therefore denotes, in a generic sense, rejection thresholds derived from the relevant reference distribution.

### Monte Carlo likelihood ratio test

The local Monte Carlo likelihood ratio test (LMC-LRT) is implemented via `LMCLRTest()` (Table 4). Because it simulates the null distribution by estimating both the restricted and unrestricted models, the

estimation options accepted by the model functions can be passed to the test through `mdl_h0_control` and `mdl_h1_control`. The same structure applies to the maximized Monte Carlo likelihood ratio test discussed below. These control lists specify whether the mean and variance switch across regimes, the estimation method, the number of initial values used in optimization, and related options. A separate number of initial values for the null distribution simulation can be set via `use_diff_init_sim`, which defaults to the value used on the observed data (in `mdl_h0_control` and `mdl_h1_control`) and is generally recommended. Beyond these estimation controls, `LMCLRTest()` also requires the number of lags `p`, the number of regimes under the null `k0`, and under the alternative `k1`.

```
lmc_control = list(N = 19,
                   mdl_h0_control = list(const  = TRUE,
                                         getSE  = FALSE),
                   mdl_h1_control = list(msmu   = TRUE,
                                         msvar  = TRUE,
                                         getSE  = FALSE,
                                         method = "EM",
                                         use_diff_init = 1))
lmclrt <- LMCLRTest(simu_msvar[["y"]], p = 1, k0 = 1 , k1 = 2, control = lmc_control)
summary(lmclrt)

#> Restricted Model
#>              coef
#> mu_1      6.77680
#> mu_2     -0.62311
#> phi_1,11  0.55850
#> phi_1,12  0.23027
#> phi_1,21  0.18764
#> phi_1,22  0.63895
#> sig_11    8.22430
#> sig_12    3.98700
#> sig_22    2.91830
#>
#> log-likelihood =  -1938.175
#> AIC =  3894.349
#> BIC =  3932.263
#>
#> Unrestricted Model
#>              coef
#> mu_1,1    6.66590
#> mu_2,1   -0.55236
#> mu_1,2    7.63580
#> mu_2,2   -0.35457
#> phi_1,11  0.60883
#> phi_1,12  0.22478
#> phi_1,21  0.22281
#> phi_1,22  0.64515
#> sig_11,1  6.25930
#> sig_12,1  2.18750
#> sig_22,1  1.33840
#> sig_11,2 14.00300
#> sig_12,2  9.91160
#> sig_22,2  8.27010
#> p_11      0.75303
#> p_12      0.24697
#> p_21      0.83510
#> p_22      0.16490
#>
#> log-likelihood =  -1901.817
#> AIC =  3839.633
#> BIC =  3915.46
#>
#> Rodriguez-Rondon & Dufour (2026) Local Monte Carlo Likelihood Ratio Test
#>          LRT_0  0.90%  0.95%  0.99% p-value
#> LMC_LRT 72.716 27.244 30.894 47.549    0.05
```

We illustrate it on the Markov switching vector autoregressive model with $p = 1$ lag simulated in the previous subsection, testing $H_0 : M = 1$ against $H_1 : M = 2$. Since the data were generated from a two-regime model, rejection is expected. The output reports the likelihood ratio statistic, Monte Carlo critical values, and the Monte Carlo $p$-value, and here the null of a single regime is rejected. Because the $p$-value is computed from a finite number of simulations, set here to `N = 19`, it is discrete and can only take values on the grid $\{1/(N+1), 2/(N+1), \ldots, N/(N+1)\}$. With $N = 19$ the smallest attainable value is $1/(19+1) = 0.05$, so the reported $p$-value of 0.05 is the strongest possible evidence against the null at this simulation size. As discussed in (Dufour, 2006; Rodriguez-Rondon and Dufour, 2026), larger `N` can improve power but need not be large, since these methods do not approximate asymptotic critical values. Consistent with this, Dufour et al. (2004) show that using more than 100 replications (including the observed statistic) has little effect on power, so $N = 99$ is typically recommended. We use `N = 19` purely for illustration.

As noted earlier, `LMCLRTest()` can also replicate the parametric bootstrap procedures (Qu and Zhuo, 2021; Kasahara and Shimotsu, 2018). Setting `mdl_h1_control = list(method = "MLE")` enables direct maximum likelihood estimation and parameter constraints via `mle_theta_low` and `mle_theta_upp`, which allow constraining the parameters away from the boundary of the parameter space, as required for asymptotic validity (Kasahara and Shimotsu (2018) additionally constrain the variances). Because the bootstrap approximates asymptotic critical values, a larger `N` is typical, though lower values remain reasonable (Qu and Zhuo (2021) use $N = 199$ and Kasahara and Shimotsu (2018) use $N = 299$) given the cost of repeatedly estimating Markov switching models.

The maximized Monte Carlo likelihood ratio test (MMC-LRT) is implemented via `MMCLRTest()`. It shares the LMC-LRT estimation controls for the restricted and unrestricted models and likewise requires `p`, `k0`, and `k1`, adding several options specific to the maximization step over the nuisance parameter space. The constant `eps` defines the consistent set over which the maximization is performed. Setting `CI_union = TRUE` constructs $C_T^* = C_T^{CI} \cup C_T^{\epsilon}$, while `eps = 0` with `CI_union = TRUE` restricts the search to $C_T^{CI}$. The optimization algorithm is selected via `type`. By default the search stops when a Monte Carlo $p$-value of 1 is reached (the maximum), or, by setting `threshold_stop`, once the test fails to reject, that is, when the $p$-value exceeds the significance level $\alpha$.

```
mmc_control = list(N = 19,
                   eps = 0.3,
                   threshold_stop = 0.05 + 1e-6,
                   type = "GenSA",
                   CI_union = FALSE,
                   silence = TRUE,
                   mdl_h0_control = list(const  = TRUE,
                                         getSE  = FALSE),
                   mdl_h1_control = list(msmu   = TRUE,
                                         msvar  = TRUE,
                                         getSE  = FALSE,
                                         method = "EM"),
                   maxit  = 100)
mmclrt <- MMCLRTest(simu_norm[["y"]], p = 0, k0 = 1 , k1 = 2, control = mmc_control)
summary(mmclrt)

#> Restricted Model
#>           coef
#> mu_1    5.0774
#> mu_2   -1.9382
#> sig_11  4.9946
#> sig_12  1.5268
#> sig_22  1.0552
#>
#> log-likelihood =  -1688.442
#> AIC =  3386.884
#> BIC =  3407.957
#>
#> Unrestricted Model
#>                 coef
#> mu_1,1    5.33240000
#> mu_2,1   -1.77730000
#> mu_1,2    4.80550000
#> mu_2,2   -2.10980000
#> sig_11,1  5.85090000
```

```
#> sig_12,1  1.79770000
#> sig_22,1  1.03890000
#> sig_11,2  3.93790000
#> sig_12,2  1.14750000
#> sig_22,2  1.01540000
#> p_11      0.06294900
#> p_12      0.93705000
#> p_21      0.99983000
#> p_22      0.00016602
#>
#> log-likelihood =  -1684.86
#> AIC =  3397.72
#> BIC =  3456.724
#>
#> Rodriguez-Rondon & Dufour (2026) Maximized Monte Carlo Likelihood Ratio Test
#>          LRT_0 p-value
#> MMC_LRT 7.1641     0.7
```

We illustrate the MMC-LRT on the linear bivariate model with $M = 1$ regime simulated from a normal distribution in the previous subsection, testing $H_0 : M = 1$ against $H_1 : M = 2$ with `p = 0` since the series has no autoregressive dynamics. Because the data are single-regime, we expect no rejection. We set `eps = 0.3`, `CI_union = FALSE`, and use simulated annealing via `type = "GenSA"`, with `threshold_stop = 0.05 + 1e-6` so the search stops once the null fails to be rejected. Parallel computation can be enabled by setting `workers` to the number of cores to use (for example, `workers = 8`), distributing the null distribution simulation across them. Here the MMC-LRT fails to reject the null, as indicated by the reported Monte Carlo $p$-value, whose resolution again depends on the number of simulations `N`. Even so, the MMC-LRT remains the most general likelihood ratio–based test currently available for Markov switching models, maximizing over the nuisance parameters while retaining finite-sample validity even when identification fails.

### Moment-based tests

The Monte Carlo moment-based test proposed by Dufour and Luger (2017) is implemented in **MSTest** via the functions `DLMCTest()` (local version) and `DLMMCTest()` (maximized version). As with the other testing procedures, the user supplies the series $y_t$ and the number of autoregressive lags $p$, and `N` sets the number of Monte Carlo replications used to compute the test (default $N = 99$). The procedure additionally approximates the distribution of the $p$-value of each moment-based statistic through an auxiliary simulation, with the number of replications set by `simdist_N` (default 10,000).

Below we illustrate the moment-based local Monte Carlo test on the previously simulated Markov switching autoregressive process. The test is computationally inexpensive and completes almost instantaneously. Because we set `getSE = TRUE`, the `summary()` method reports the estimated restricted model, here an AR(1) with autoregressive coefficient $\phi_1$, together with standard errors for its coefficients, followed by the test statistics for each of the four moments (mean, variance, skewness, and excess kurtosis), the combined statistic $F(\varepsilon)$, the corresponding critical values (quantiles of the simulated distribution), and the Monte Carlo $p$-values. The value of $\phi_1$ reported in the test block equals the restricted-model estimate above, because the local test simulates the null distribution at this consistent point estimate. As expected, the null hypothesis of linearity is clearly rejected, since the data were generated from a Markov switching process with $M = 2$ regimes.

```
lmc_control = list(N = 99,
                   simdist_N = 10000,
                   getSE = TRUE)
lmcmoment <- DLMCTest(simu_msar[["y"]], p = 1, control = lmc_control)

summary(lmcmoment)

#> Restricted Model
#>          coef     s.e.
#> mu    6.76700 0.450970
#> phi_1 0.84667 0.023987
#> sig   2.38540 0.151020
#>
#> log-likelihood =  -924.9604
```

```
#> AIC =  1855.921
#> BIC =  1868.559
#>
#> Dufour & Luger (2017) Moment-Based Local Monte Carlo Test
#>            phi_1 M(eps) V(eps)  S(eps) K(eps) F(eps)   0.90%   0.95%   0.99%
#> LMC_min  0.84667 1.3431 16.942 0.37429  3.473      1 0.96333 0.97859 0.99455
#> LMC_prod 0.84667 1.3431 16.942 0.37429  3.473      1 0.99822 0.99932 0.99977
#>          p-value
#> LMC_min     0.01
#> LMC_prod    0.01
```

This computational efficiency carries over to the maximized moment-based test, implemented via `DLMMCTest()`, which maximizes the Monte Carlo $p$-value over the nuisance parameter space while remaining fast in practice. The optimization algorithm is selected via `optim_type` (the analogue of `type` in `MMCLRTest()`), and, as with `MMCLRTest()`, `eps` and `CI_union` define the consistent set over which the maximization is performed. Many of the other optimization options available there also apply here.

```
mmc_control <- list(N = 99,
                    getSE = TRUE,
                    eps = 0,
                    CI_union = TRUE,
                    optim_type = "GenSA",
                    silence = TRUE,
                    threshold_stop = 0.05 + 1e-6,
                    maxit = 100)
mmcmoment <- DLMMCTest(simu_msar[["y"]], p = 1, control = mmc_control)

summary(mmcmoment)
```

```
#> Restricted Model
#>          coef     s.e.
#> mu    6.76700 0.450970
#> phi_1 0.84667 0.023987
#> sig   2.38540 0.151020
#>
#> log-likelihood =  -924.9604
#> AIC =  1855.921
#> BIC =  1868.559
#>
#> Dufour & Luger (2017) Moment-Based Maximized Monte Carlo Test
#>            phi_1 M(eps) V(eps)  S(eps) K(eps) F(eps) p-value
#> MMC_min  0.84667 1.3431 16.942 0.37429  3.473      1    0.01
#> MMC_prod 0.84667 1.3431 16.942 0.37429  3.473      1    0.01
```

In the example above, we again apply the test to the simulated Markov switching autoregressive process. We set the stopping threshold to $0.05 + 1\text{e-}6$, so that the optimization terminates once the null hypothesis fails to be rejected, although this does not occur in this case. We also set `eps = 0`, restricting the search to the confidence interval–based consistent set, as in Dufour and Luger (2017). The test block also reports the value of $\phi_1$ that maximizes the Monte Carlo $p$-value over this set, which here coincides with the point estimate from the restricted model, although in general the two need not be equal. Once again, the null hypothesis of a linear model is rejected, in line with the results obtained from the other testing procedures.

### Parameter stability test

The parameter stability test proposed by Carrasco et al. (2014) is implemented in **MSTest** via the function `CHPTest()`. As with the other testing procedures, the user must provide the series $y_t$ and the number of autoregressive lags $p$. Here the parameter $N$ specifies the number of bootstrap replications used to obtain critical values and is set to 3000 by default, following Carrasco et al. (2014). The parameter $\rho_b$ bounds the autocorrelation nuisance parameter $\rho$ and is set to 0.7 by default, again as in their study, so that the grid search is performed over $\rho \in [-\rho_b, \rho_b] = [-0.7, 0.7]$.

```
chp_control = list(N = 1000,
```

```
                  rho_b = 0.7,
                  msvar = FALSE)
pstabilitytest <- CHPTest(simu_ar[["y"]], p = 1, control = chp_control)
summary(pstabilitytest)

#> Restricted Model
#>          coef     s.e.
#> mu    4.91830 0.182520
#> phi_1 0.74989 0.029756
#> sig   1.03960 0.065817
#>
#> log-likelihood =  -717.7421
#> AIC =  1441.484
#> BIC =  1454.122
#>
#> Carrasco, Hu, & Ploberger (2014) Parameter Stability Test
#>
#> - Switch in Mean only
#>       test-stat  0.90%  0.95%  0.99% p-value
#> supTS   0.37868 1.8429 2.5673 4.1247   0.574
#> expTS   0.77573 1.3717 1.8293 3.4564   0.541
```

Above, we apply the parameter stability test to the linear autoregressive process simulated earlier. We set `N = 1000` to keep the example fast and `msvar = FALSE`, so that the variance is held constant and the test focuses on potential instability in the mean. The `summary()` method reports the parameter estimates of the restricted model, which is the only model that needs to be estimated for this test, along with the results for both the supTS and expTS versions of the test statistic. As expected, the test fails to reject the null hypothesis of parameter stability, consistent with the fact that the data were generated from a linear autoregressive model without regime switching.

### Stochastic likelihood ratio test

The stochastic likelihood ratio test proposed by Hansen (1992) is implemented in **MSTest** via the function `HLRTest()`. As with `DLMCTest()`, `DLMMCTest()`, and `CHPTest()`, the user need only provide the series $y_t$ and the number of autoregressive lags $p$, since this test likewise assesses only the null hypothesis of linearity, that is, a single regime, in an autoregressive model.

This procedure performs a grid search over the nuisance parameters of the unrestricted model, so most options control the construction of that grid. The argument `gridsize` sets the number of grid points, and `mugrid_from` and `mugrid_by` give the starting value and step size for the mean grid, with analogous arguments for the variance grid. For each point on this grid, the test estimates the restricted model over a bounded region defined by `theta_null_low` and `theta_null_upp`. Setting `msvar = TRUE` additionally allows regime-dependent variances, but this enlarges the nuisance parameter space and can substantially raise the computational cost.

```
hlrt_control  <- list(msvar          = FALSE,
                      gridsize       = 20,
                      mugrid_from    = 0,
                      mugrid_by      = 1,
                      theta_null_low = c(0,-0.99,0.01),
                      theta_null_upp = c(20,0.99,20))
hlrt <- HLRTest(simu_msar[["y"]], p = 1, control = hlrt_control)
summary(hlrt)

#> Restricted Model
#>          coef     s.e.
#> mu    6.76700 0.450970
#> phi_1 0.84667 0.023987
#> sig   2.38540 0.151020
#>
#> log-likelihood =  -924.9604
#> AIC =  1855.921
#> BIC =  1868.559
#>
```

```
#> Hansen (1992) Likelihood Ratio Bound Test -  Switch in Mean only
#>       test-stat 0.90 % 0.95 % 0.99 % p-value
#> M = 0     6.972 2.6900 2.9574 3.4479       0
#> M = 1     6.972 2.7176 2.9612 3.4493       0
#> M = 2     6.972 2.7856 3.0960 3.5397       0
#> M = 3     6.972 2.8679 3.1516 3.6352       0
#> M = 4     6.972 2.8962 3.2477 3.8034       0
```

Above, we illustrate the stochastic likelihood ratio test using the previously simulated Markov switching autoregressive process. The `summary()` method reports parameter estimates for the restricted model, which is the only model that needs to be estimated for this test, together with the standardized likelihood ratio statistic and its Monte Carlo critical values and $p$-values for each bandwidth used in the heteroskedasticity-robust procedure of Hansen (1996a). Across all bandwidths, the null hypothesis of linearity is strongly rejected, with Monte Carlo $p$-values effectively equal to zero, consistent with the data having been generated from a Markov switching process with two regimes.

## 5 Conclusion

The importance of testing the number of regimes in Markov switching models has motivated a substantial body of research, reflecting the statistical and computational challenges inherent in this problem. Seminal contributions such as Hansen (1992), Carrasco et al. (2014), and Dufour and Luger (2017) focus on testing the null hypothesis of a single regime (i.e., a linear model) against the alternative of two regimes. More recently, Rodriguez-Rondon and Dufour (2026) introduced a class of Monte Carlo likelihood ratio tests that allow one to test a null of $M_0$ regimes against an alternative of $M_0 + m$ regimes, for any $M_0 \geq 1$ and $m \geq 1$ and for a broad class of models, including multivariate and non-stationary ones, substantially expanding the scope of feasible hypothesis testing.

The **MSTest** package makes these procedures readily available by implementing the methods of these four studies within a unified, user-friendly framework, and it additionally provides tools for simulating and estimating a wide range of Markov switching and hidden Markov models to support the testing procedures. This paper has reviewed the underlying methodology and illustrated how **MSTest** is used in practice to conduct inference on the number of regimes. By lowering the computational and implementation barriers to regime testing and offering multiple complementary procedures with differing assumptions and robustness properties, the package lets users select methods well suited to their empirical setting and make informed decisions about model specification.

## Computational details

All results in this paper were obtained using R 4.4.0 (R Core Team, 2024) and version 0.1.9 of the package **MSTest**, available from the Comprehensive R Archive Network (CRAN). For computational efficiency, **MSTest** depends on **Rcpp** 1.1.1 (Eddelbuettel and Balamuta, 2018) and **RcppArmadillo** 15.2.4.1 (Eddelbuettel et al., 2026), and it uses **GenSA** 1.1.14.1 (Xiang et al., 2013), **GA** 3.2.4 (Scrucca, 2013), and **pso** 1.0.4 (Bendtsen, 2022) for numerical optimization.

All tables and figures can be reproduced with the file `article.R`, distributed with the **MSTest** package (`inst/examples/article.R`) and also available from the package's public GitHub repository. Because the Monte Carlo procedures are simulation-based, exact reproduction requires the random seed set in `article.R`, though results can vary slightly across hardware, BLAS/LAPACK implementations, and operating systems. On Darwin 25.6.0 (arm64), the full script runs in approximately three minutes of elapsed time, which varies with hardware and parallelization settings.

## References

V. Aguirregabiria and P. Mira. Sequential estimation of dynamic discrete games. *Econometrica*, 75(1): 1–53, 2007. doi: 10.1111/j.1468-0262.2007.00731.x. [p219]

H. J. Ahn and M. Luciani. Common and idiosyncratic inflation. *Journal of Applied Econometrics*, 41(2): 156–168, 2026. doi: 10.1002/jae.70023. [p219]

Y. An, Y. Hu, and M. Shum. Identifiability and inference of hidden Markov models. Technical report, Johns Hopkins University, 2013. [p221]

D. W. Andrews. Estimation when a parameter is on a boundary. *Econometrica*, 67(6):1341–1383, 1999. doi: 10.1111/1468-0262.00082. [p222]

D. W. Andrews. Testing when a parameter is on the boundary of the maintained hypothesis. *Econometrica*, 69(3):683–734, 2001. doi: 10.1111/1468-0262.00210. [p222]

D. W. Andrews and W. Ploberger. Optimal tests when a nuisance parameter is present only under the alternative. *Econometrica*, 62(6):1383–1414, 1994. doi: 10.2307/2951753. [p222, 224]

P. Baldi, Y. Chauvin, T. Hunkapiller, and M. A. McClure. Hidden Markov models of biological primary sequence information. *Proceedings of the National Academy of Sciences*, 91(3):1059–1063, 1994. doi: 10.1073/pnas.91.3.1059. [p219]

C. Bendtsen. *pso: particle swarm optimization*, 2022. URL https://CRAN.R-project.org/package=pso. R package version 1.0.4. [p236]

H. Bunke and T. M. Caelli, editors. *Hidden Markov models: applications in computer vision*. Series in Machine Perception and Artificial Intelligence. World Scientific, 2001. doi: 10.1142/4648. [p219]

G. Caggiano, E. Castelnuovo, and J. M. Figueres. Economic policy uncertainty and unemployment in the United States: a nonlinear approach. *Economics Letters*, 151:31–34, 2017. doi: 10.1016/j.econlet.2016.12.002. [p219]

J. Cai. A Markov model of switching-regime ARCH. *Journal of Business & Economic Statistics*, 12(3):309–316, 1994. doi: 10.1080/07350015.1994.10524546. [p219]

M. Carrasco, L. Hu, and W. Ploberger. Optimal test for Markov switching parameters. *Econometrica*, 82(2):765–784, 2014. doi: 10.3982/ECTA8609. [p219, 220, 223, 224, 225, 234, 236]

A. V. Carter and D. G. Steigerwald. Testing for regime switching: a comment. *Econometrica*, 80(4):1809–1812, 2012. doi: 10.3982/ECTA9622. [p222]

E. I. Cevik, D. Ç. Yıldırım, and S. Dibooglu. Renewable and non-renewable energy consumption and economic growth in the US: A Markov-switching VAR analysis. *Energy & Environment*, 32(3):519–541, 2021. doi: 10.1177/0958305X20944035. [p219]

L. Charfeddine. The impact of energy consumption and economic development on ecological footprint and CO2 emissions: evidence from a Markov switching equilibrium correction model. *Energy Economics*, 65:355–374, 2017. doi: 10.1016/j.eneco.2017.05.009. [p219]

M. Chauvet. An econometric characterization of business cycle dynamics with factor structure and regime switching. *International Economic Review*, 39(4):969–996, 1998. doi: 10.2307/2527348. [p219]

M. Chauvet and J. D. Hamilton. Dating business cycle turning points. In C. Milas, P. A. Rothman, D. van Dijk, and D. E. Wildasin, editors, *Nonlinear Time Series Analysis of Business Cycles*, volume 276 of *Contributions to Economic Analysis*, pages 1–54. Emerald Group Publishing Limited, 2006. doi: 10.1016/S0573-8555(05)76001-6. [p219]

M. Chauvet, C. Juhn, and S. Potter. Markov switching in disaggregate unemployment rates. *Empirical Economics*, 27:205–232, 2002. doi: 10.1007/s001810100101. [p219]

J.-S. Cho and H. White. Testing for regime switching. *Econometrica*, 75(6):1671–1720, 2007. doi: 10.1111/j.1468-0262.2007.00809.x. [p219, 222, 224]

J.-S. Cho and H. White. Testing for regime switching: rejoinder. Unpublished Manuscript, University of California, San Diego, 2011. [p222]

R. B. Davies. Hypothesis testing when a nuisance parameter is present only under the alternative. *Biometrika*, 64(2):247–254, 1977. doi: 10.2307/2335690. [p222]

R. B. Davies. Hypothesis testing when a nuisance parameter is present only under the alternatives. *Biometrika*, 74(1):33–43, 1987. doi: 10.2307/2336019. [p222, 224]

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. doi: 10.1111/j.2517-6161.1977.tb01600.x. [p221]

F. X. Diebold and G. D. Rudebusch. Measuring business cycles: A modern perspective. *The Review of Economics and Statistics*, 78(1):67–77, 1996. doi: 10.2307/2109848. [p219]

S. Dietz and N. Stern. Endogenous growth, convexity of damage and climate risk: How Nordhaus' framework supports deep cuts in carbon emissions. *The Economic Journal*, 125(583):574–620, 2015. doi: 10.1111/ecoj.12188. [p219]

J.-M. Dufour. Monte Carlo tests with nuisance parameters: A general approach to finite-sample inference and nonstandard asymptotics. *Journal of Econometrics*, 133(2):443–477, 2006. doi: 10.1016/j.jeconom.2005.06.007. [p222, 223, 232]

J.-M. Dufour and R. Luger. Identification-robust moment-based tests for Markov switching in autoregressive models. *Econometric Reviews*, 36(6-9):713–727, 2017. doi: 10.1080/07474938.2017.1307548. [p219, 220, 222, 223, 224, 225, 233, 234, 236]

J.-M. Dufour and J. Neves. Finite-sample inference and nonstandard asymptotics with Monte Carlo tests and R. In H. D. Vinod and C. R. Rao, editors, *Conceptual Econometrics Using R*, volume 41 of *Handbook of Statistics*, pages 3–31. North-Holland, 2019. doi: 10.1016/bs.host.2019.05.001. [p223]

J.-M. Dufour, L. Khalaf, J.-T. Bernard, and I. Genest. Simulation-based finite-sample tests for heteroskedasticity and ARCH effects. *Journal of Econometrics*, 122(2):317–347, 2004. doi: 10.1016/j.jeconom.2003.10.024. [p223, 232]

J.-M. Dufour, L. Khalaf, and M. Voia. Finite-sample resampling-based combined hypothesis tests, with applications to serial correlation and predictability. *Communications in Statistics-Simulation and Computation*, 44(9):2329–2347, 2015. doi: 10.1080/03610918.2013.858164. [p223]

G. Dufrénot, V. Mignon, and A. Péguin-Feissolle. The effects of the subprime crisis on the Latin American financial markets: An empirical assessment. *Economic Modelling*, 28(5):2342–2357, 2011. doi: 10.1016/j.econmod.2011.04.012. [p224]

D. Eddelbuettel and J. J. Balamuta. Extending R with C++: a brief introduction to Rcpp. *The American Statistician*, 72(1):28–36, 2018. doi: 10.1080/00031305.2017.1375990. [p220, 236]

D. Eddelbuettel, R. Francois, D. Bates, B. Ni, and C. Sanderson. *RcppArmadillo: Rcpp integration for the Armadillo templated linear algebra library*, 2026. URL https://CRAN.R-project.org/package=RcppArmadillo. R package version 15.2.4.1. [p220, 236]

R. A. Fisher. *Statistical methods for research workers*. Oliver and Boyd, Edinburgh, 4th edition, 1932. [p223]

R. Garcia. Asymptotic null distribution of the likelihood ratio test in Markov switching models. *International Economic Review*, 39(3):763–788, 1998. doi: 10.2307/2527399. [p219, 222, 223, 224]

R. Garcia and P. Perron. An analysis of the real interest rate under regime shifts. *Review of Economics and Statistics*, 78(1):111–125, 1996. doi: 10.2307/2109851. [p219]

S. M. Goldfeld and R. E. Quandt. A Markov model for switching regressions. *Journal of Econometrics*, 1 (1):3–15, 1973. doi: 10.1016/0304-4076(73)90002-X. [p219]

M. Golosov, J. Hassler, P. Krusell, and A. Tsyvinski. Optimal taxes on fossil fuel in general equilibrium. *Econometrica*, 82(1):41–88, 2014. doi: 10.3982/ECTA10217. [p219]

S. F. Gray. Modeling the conditional distribution of interest rates as a regime-switching process. *Journal of Financial Economics*, 42(1):27–62, 1996. doi: 10.1016/0304-405X(96)00875-6. [p219]

M. Haas, S. Mittnik, and M. S. Paolella. A new approach to Markov-switching GARCH models. *Journal of Financial Econometrics*, 2(4):493–530, 2004. doi: 10.1093/jjfinec/nbh020. [p219]

J. D. Hamilton. A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica*, 57(2):357–384, 1989. doi: 10.2307/1912559. [p219, 221, 225]

J. D. Hamilton. Analysis of time series subject to changes in regime. *Journal of Econometrics*, 45(1-2): 39–70, 1990. doi: 10.1016/0304-4076(90)90093-9. [p221]

J. D. Hamilton. *Time series analysis*. Princeton University Press, Princeton, NJ, 1994. [p219, 221]

J. D. Hamilton. What's real about the business cycle? *Federal Reserve Bank of St. Louis Review*, 87(4): 435–452, 2005. doi: 10.20955/r.87.435-452. [p224]

J. D. Hamilton. Macroeconomic regimes and regime shifts. In J. B. Taylor and H. Uhlig, editors, *Handbook of Macroeconomics*, volume 2, pages 163–201. Elsevier, 2016. doi: 10.1016/bs.hesmac.2016.03.004. [p219]

B. E. Hansen. The likelihood ratio test under nonstandard conditions: testing the Markov switching model of GNP. *Journal of Applied Econometrics*, 7(S1):S61–S82, 1992. doi: 10.1002/jae.3950070506. [p219, 220, 222, 223, 224, 225, 230, 235, 236]

B. E. Hansen. Erratum: The likelihood ratio test under nonstandard conditions: testing the Markov switching model of GNP. *Journal of Applied Econometrics*, 11(2):195–198, 1996a. doi: 10.1002/(SICI)1099-1255(199603)11:2<195::AID-JAE375>3.0.CO;2-2. [p219, 222, 236]

B. E. Hansen. Inference when a nuisance parameter is not identified under the null hypothesis. *Econometrica*, 64(2):413–430, 1996b. doi: 10.2307/2171789. [p222]

H. Herwartz and H. Lütkepohl. Structural vector autoregressions with Markov switching: Combining conventional with statistical identification of shocks. *Journal of Econometrics*, 183(1):104–116, 2014. doi: 10.1016/j.jeconom.2014.06.012. [p219]

L. Hu and Y. Shin. Optimal test for Markov switching GARCH models. *Studies in Nonlinear Dynamics & Econometrics*, 12(3), 2008. doi: 10.2202/1558-3708.1528. [p224]

F. Jelinek. *Statistical methods for speech recognition*. MIT Press, Cambridge, MA, 1997. [p219]

J. A. Kahn and R. W. Rich. Tracking the new economy: Using growth theory to detect changes in trend productivity. *Journal of Monetary Economics*, 54(6):1670–1701, 2007. doi: 10.1016/j.jmoneco.2006.07.008. [p224]

H. Kasahara and K. Shimotsu. Testing the number of regimes in Markov regime switching models. arXiv preprint arXiv:1801.06862, 2018. [p219, 220, 222, 223, 224, 232]

H. Kasahara, T. Okimoto, and K. Shimotsu. Modified quasi-likelihood ratio test for regime switching. *The Japanese Economic Review*, 65(1):25–41, 2014. doi: 10.1111/jere.12027. [p219]

C.-J. Kim and C. R. Nelson. Has the U.S. economy become more stable? A Bayesian approach based on a Markov-switching model of the business cycle. *Review of Economics and Statistics*, 81(4):608–616, 1999. doi: 10.1162/003465399558472. [p219]

F. Klaassen. Improving GARCH volatility forecasts with regime-switching GARCH. *Empirical Economics*, 27:363–394, 2002. doi: 10.1007/s001810100100. [p219]

A. Krogh, I. S. Mian, and D. Haussler. A hidden Markov model that finds genes in E. coli DNA. *Nucleic Acids Research*, 22(22):4768–4778, 1994. doi: 10.1093/nar/22.22.4768. [p219]

H.-M. Krolzig. *Markov-switching vector autoregressions: Modelling, statistical inference, and application to business cycle analysis*, volume 454 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin, Heidelberg, 1997. doi: 10.1007/978-3-642-51684-9. [p221]

M. Lanne, H. Lütkepohl, and K. Maciejowska. Structural vector autoregressions with Markov switching. *Journal of Economic Dynamics and Control*, 34(2):121–131, 2010. doi: 10.1016/j.jedc.2009.08.002. [p219]

H. Le Bihan, D. Leiva-León, and M. Pacce. Underlying inflation and asymmetric risks. *Review of Economics and Statistics*, pages 1–45, 2024. doi: 10.1162/rest_a_01522. [p219]

H. Lütkepohl, M. Meitz, A. Netšunajev, and P. Saikkonen. Testing identification via heteroskedasticity in structural vector autoregressive models. *The Econometrics Journal*, 24(1):1–22, 2021. doi: 10.1093/ectj/utaa008. [p219]

V. Marmer. Testing the null hypothesis of no regime switching with an application to GDP growth rates. *Empirical Economics*, 35:101–122, 2008. doi: 10.1007/s00181-007-0145-2. [p219]

J. Morley and J. Piger. The asymmetric business cycle. *Review of Economics and Statistics*, 94(1):208–221, 2012. doi: 10.1162/REST_a_00169. [p224]

R. Nag, K. Wong, and F. Fallside. Script recognition using hidden Markov models. In *ICASSP'86. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 11, pages 2071–2074. IEEE, 1986. doi: 10.1109/ICASSP.1986.1168951. [p219]

K. Pearson. On a method of determining whether a sample of size n supposed to have been drawn from a parent population having a known probability integral has probably been drawn at random. *Biometrika*, 25(3-4):379–410, 1933. doi: 10.1093/biomet/25.3-4.379. [p223]

D. Pelletier. Regime switching for dynamic correlations. *Journal of Econometrics*, 131(1-2):445–473, 2006. doi: 10.1016/j.jeconom.2005.01.013. [p219]

A. Qin and Z. Qu. Regime-switching models for disaggregated data. arXiv preprint arXiv:2606.08398, 2026. [p219]

Z. Qu and F. Zhuo. Likelihood ratio-based tests for Markov regime switching. *The Review of Economic Studies*, 88(2):937–968, 2021. doi: 10.1093/restud/rdaa035. [p219, 220, 222, 223, 224, 232]

R Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2024. URL https://www.R-project.org/. [p236]

L. Rabiner and B. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, 3(1):4–16, 1986. doi: 10.1109/MASSP.1986.1165342. [p219]

L. R. Rabiner and B. H. Juang. *Fundamentals of speech recognition*. Prentice Hall, Englewood Cliffs, NJ, 1993. [p219]

G. Rodriguez-Rondon. Underlying core inflation with multiple regimes. arXiv preprint arXiv:2411.12845, 2024. [p219]

G. Rodriguez-Rondon and J.-M. Dufour. Monte Carlo likelihood-ratio tests for Markov switching models. Staff Working Paper 2026-23, Bank of Canada, 2026. URL https://doi.org/10.34989/swp-2026-23. [p219, 220, 221, 222, 223, 224, 225, 230, 232, 236]

L. Scrucca. GA: a package for genetic algorithms in R. *Journal of Statistical Software*, 53(4):1–37, 2013. doi: 10.18637/jss.v053.i04. [p223, 236]

C. A. Sims and T. Zha. Were there regime switches in U.S. monetary policy? *American Economic Review*, 96(1):54–81, 2006. doi: 10.1257/000282806776157678. [p219]

M. K. P. So, K. Lam, and W. K. Li. A stochastic volatility model with Markov switching. *Journal of Business & Economic Statistics*, 16(2):244–253, 1998. doi: 10.1080/07350015.1998.10524758. [p219]

A. Sweeting. Dynamic product positioning in differentiated product markets: The effect of fees for musical performance rights on the commercial radio industry. *Econometrica*, 81(5):1763–1803, 2013. doi: 10.3982/ECTA7473. [p219]

L. H. C. Tippett. *The methods of statistics: an introduction mainly for workers in the biological sciences*. Williams & Norgate, London, 1931. [p223]

A. Warne and A. Vredin. Unemployment and inflation regimes. *Studies in Nonlinear Dynamics & Econometrics*, 10(2), 2006. doi: 10.2202/1558-3708.1280. [p224]

H. White. Maximum likelihood estimation of misspecified models. *Econometrica*, 50(1):1–25, 1982. doi: 10.2307/1912526. [p224]

B. Wilkinson. A statistical consideration in psychological research. *Psychological Bulletin*, 48(2):156–158, 1951. doi: 10.1037/h0059111. [p223]

Y. Xiang, S. Gubian, B. Suomela, and J. Hoeng. Generalized simulated annealing for global optimization: the GenSA package. *The R Journal*, 5(1):13–28, 2013. doi: 10.32614/RJ-2013-002. [p223, 236]

M. Zambrano-Bigiarini, M. Clerc, and R. Rojas. Standard particle swarm optimisation 2011 at CEC-2013: A baseline for future PSO improvements. In *2013 IEEE Congress on Evolutionary Computation*, pages 2337–2344. IEEE, 2013. doi: 10.1109/CEC.2013.6557848. [p223]

*Gabriel Rodriguez-Rondon*
*Bank of Canada*
*Canadian Economic Analysis Department*
*Ottawa, ON, Canada*
https://grodriguezrondon.com
*ORCID: 0009-0005-3769-9921*
grodriguezrondon@bankofcanada.ca

*Jean-Marie Dufour*
*McGill University*

*Department of Economics*
*Montreal, QC, Canada*
http://www.jeanmariedufour.com
*ORCID: 0000-0002-7731-2278*
jean-marie.dufour@mcgill.ca

# admix: An R Package for Estimation, Test and Clustering in Admixture Models

*by Xavier Milhaud, Denys Pommeret, Yahia Salhi, and Pierre Vandekerkhove*

**Abstract** The R package admix, available on CRAN, implements a wide variety of functions dedicated to estimation, tests and clustering of admixture models, also known as contamination models. Such models are two-component mixture distributions having only one single known component. They are particularly useful when a known random phenomenon is contaminated by an unknown/unexpected random effect. We propose in this setup to estimate semiparametrically the unknown quantities of the model, and compare or cluster the unknown sources of randomness across different samples without making any parametric assumptions about them. The admix package appears thus as a powerful tool for applications where latent variables induce hidden heterogeneity, often revealed through multimodality in data distributions.

## 1 Introduction

The general framework is the semiparametric two-component mixture model, with cumulative distribution function (cdf)

$$L(x) = (1-p)G(x) + pF(x), \quad x \in \mathbb{R}, \tag{1}$$

where $G$ is a known cdf and where the unknown parameters are the mixture proportion $p \in ]0,1[$ and the cdf $F$ which is not supposed to belong to any parametric family. The above model (1), so-called **contamination** or **admixture** model, has been widely investigated in the last decades, see for instance Bordes and Vandekerkhove (2010), Cai and Jin (2010) or Celisse and Robin (2010) among others. Such a modelling arises in various applications, in particular during crisis times where some populations are impacted when others are not concerned yet. In genomics, studying the genetic make-up of populations has been the subject of extensive attention. The admixture behaviour is of paramount importance in comparing the populations with known ancestors and exhibits linkage relationships between them (Loh et al., 2013).

As an illustration, McLachlan et al. (2006) proposed a method for detecting differentially expressed genes in breast cancer tissues, from women carrying hereditary BRCA1 or BRCA2 mutations (Hedenfalk et al., 2001). Their approach is based on a two-component mixture model with one known component. More specifically, a $z$-score type statistic is built for each gene, distributed according to a mixture model where the gene is either normally expressed or not. When it is normally expressed, its $z$-score is expected to follow a (known) standard Gaussian distribution. Otherwise, the $z$-score is modeled by a trash (unknown) distribution. Figure 1 shows the histogram of the 3,170 observed $z$-scores, along with its decomposition into two density components. Parameter estimation in Pommeret and Vandekerkhove (2019) resulted in a proportion $p = 0.65$ for the known $\mathcal{N}(0,1)$ component, while the second component follows a Gaussian distribution with mean 1.5 and variance 0.98. In this context, it may be useful to identify the gene family of not normally expressed genes, and then classify them by gene family type. This is one illustration of what the admix package allows one to do.

More generally, such a problem arises when a known random phenomenon is contaminated by an unknown random effect. The interest in such models is twofold: i) understand how and to which extent the original population distribution $G$ has been contaminated or distorted by the extra component distribution $F$, ii) compare this impact over various populations. In that latter case, the data of interest is hence made of $K$ independent and identically distributed (i.i.d.) samples $X_1 = (X_{1,1}, \ldots, X_{1,n_1}), \ldots, X_K = (X_{K,1}, \ldots, X_{K,n_K})$ with cdfs:

$$L_i(x) = (1-p_i)\, G_i(x) + p_i\, F_i(x), \quad x \in \mathbb{R}, \quad i = 1, \ldots, K, \tag{2}$$

where $p_1, \ldots, p_K$ are the unknown mixture proportions and $F_1, \ldots, F_K$ are unknown cdf components. Hereafter, we consider situations where the $G_i$'s and $F_i$'s distributions are: (i) absolutely continuous with respect to the Lebesgue measure, supported over $\mathbb{R}$, $\mathbb{R}^+$ or intervals of $\mathbb{R}$; (ii) finite discrete or $\mathbb{N}$-discrete distributions such as Binomial or Poisson; (iii) a mixture of a discrete and an absolutely continuous distribution. All our implementations still work in such frameworks.

The goal of the package **admix** is to provide consistent numerical solutions for the following tasks:

- Provide robust estimations of the unknown parameters, *i.e.* $(p_i, F_i)$ for $i = 1, \ldots, K$;
- Test the equality of the $F_i$'s involved in a sub-$k$-sample for $2 \leq k \leq K$;
- Detect potential clusters (samples sharing the same unknown cdf) inside the $K$-sample.

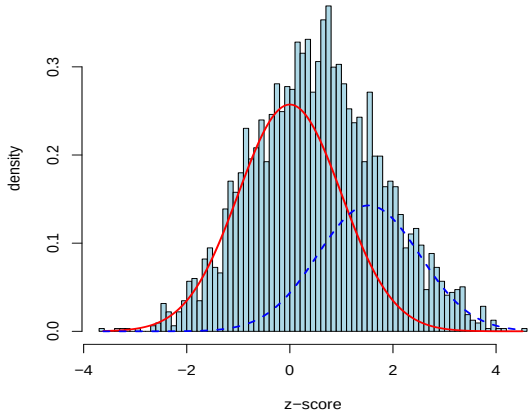



**Figure 1:** Histogram of $z$-scores in the breast cancer study, where the known component (in red) is $\mathcal{N}(0,1)$, and the unknown one is estimated as $\mathcal{N}(1.5, 0.98)$ with proportion $p = 0.35$.

As might be expected given recent advances, and contrary to the parametric mixture model framework, there are very few implementations dealing with semiparametric mixture models. As far as we are aware, the **admix** package is the first to deal with admixture models in such a general setting, followed by the packages **MixSemiRob** (Kang et al., 2023) and **mixmodel**[1] (Patra and Sen, 2016). Other recent packages are **admixr** (Petr, 2020) and **admixturegraph** (Leppälä et al., 2017), but they are only targeted towards genomic studies and strongly connected to **ADMIXTOOLS 2**[2]. We also mention **mclust** (Scrucca et al., 2023) and **ContaminatedMixt** (Punzo et al., 2018), despite they focus on model-based clustering and classification relying on Gaussian mixtures only. More generally, numerous R packages dealing with mixture distributions have already been proposed, see for instance the most famous of them, **mixtools** (Benaglia et al., 2009). The CRAN Task View *Cluster Analysis and Finite Mixture Models* provides a more exhaustive list of them. However, these packages mostly rely on parametric assumptions made on the mixture component distributions, which is too restrictive in the context of admixture models where no knowledge about the contamination effect is *a priori* accessible. This brief review highlights the need to develop new tools in this area.

The rest of the paper is organized as follows. Section 2 describes the theoretical properties of the estimation and testing strategies implemented in our package. Section 3 gives a synthetic overview of the package, focusing on core functions and their structure. Then, Section 4 provides some examples on how to use the package in practice.

## 2 Theory of implemented statistical methodologies

The **admix** package (Milhaud, 2026) provides several ways to perform estimation and tests in admixture models, all of them coming from publications in high-ranking statistical journals. Hereafter, we review the main ideas on which the package is grounded. We present first the different estimation methods available, and talk about associated testing strategies. Finally, we discuss the clustering scheme that has been proposed in the specific $K$-sample setup to create groups of samples similarly contaminated. To be self-contained, the paper provides models, assumptions and key ideas underlying each method. However, for the sake of conciseness, we briefly detail and comment these methods. The reader can find further details, especially about identifiability and consistency, in the cited references.

### 2.1 Estimation of the mixture proportion

Inside the available $K$ samples, let us consider $X_i = (X_{i,1}, \ldots, X_{i,n_i})$ the $i$-th sample, for $i = 1, \ldots, K$, made of $n_i$ independent and identically distributed random variables with common cdf $L_i$:

$$L_i(x) = (1 - p_i)\, G_i(x) + p_i\, F_i(x), \qquad x \in \mathbb{R}, \tag{3}$$

where $G_i$ is a known cdf when the mixture proportion $p_i$ and the extra cdf $F_i$ are unknown.

[1]Not on CRAN, but available at https://github.com/rohitpatra/mixmodel.
[2]For further information, see https://uqrmaie1.github.io/admixtools/index.html.

The estimation step first seeks a robust estimation of $p_i$, followed by an estimation of $F_i$ based on plugged-in inversion formula involving the previous $p_i$ estimation and the empirical cdf of the dataset $X_i$. In the sequel, we denote respectively by $\ell_i$, $f_i$ and $g_i$ the probability density function (pdf) associated with $L_i$, $F_i$ and $G_i$. Three estimation procedures of $p_i$ can be used:

- The *PS* estimator (Patra and Sen, 2016), which works under tail conditions about $F_i$ and $G_i$, and does not require any specific shape constraint thanks to a relevant definition of identifiability in the case of semiparametric admixture models. This method provides a consistent estimator of $p_i$, which fails to be asymptotically normal (see Theorem 4 in Patra and Sen, 2016).
- The *BVdk* estimator (Bordes et al., 2006a) satisfies a joint functional central limit theorem (see also Bordes and Vandekerkhove, 2010), in the context where $f_i$ and $g_i$ are supposed to be symmetric densities with respect to some location parameters. The procedure provides $\sqrt{n}$-consistent estimators of both $p_i$ and the location parameter, along with the unknown cdf.
- The *IBM* estimator (Milhaud et al., 2024a), developed in a testing perspective, provides $\sqrt{n}$-consistent estimators in the two-sample case when the unknown components in both samples share the same distribution (null hypothesis). When this condition is not satisfied, the asymptotic behavior of the IBM estimator along with the associated test statistic is also explicitly derived.

### 2.1.1 The PS estimator

In contrast with previous works about semiparametric mixture estimation, see Bordes et al. (2006b), Bordes et al. (2006a) or Butucea et al. (2017), all based on symmetry type assumption about the unknown component, Patra and Sen (2016) proposed to relax this restrictive shape constraint by noticing that model (3) could also be rewritten as

$$L_i(x) = (1 - [p_i + \lambda])G_i(x) + [p_i + \lambda]\left(\frac{p_i}{p_i + \lambda}F_i(x) + \frac{\lambda}{p_i + \lambda}G_i(x)\right), \qquad x \in \mathbb{R},$$

with $0 \leq \lambda \leq 1 - p_i$, and where $(p_i F_i(x) + \lambda G_i(x))/(p_i + \lambda)$ could be thought as the nonparametric component (denote further $\pi_i = p_i + \lambda$). This writing shows a clear lack of identifiability since $p_i$ is not uniquely defined. The authors propose to solve this lack of identifiability by considering the following definition of the *correct* parameter

$$p_{0,i} := \inf\left\{\pi_i \in (0,1] \,:\, \frac{L_i - (1-\pi_i)G_i}{\pi_i} \text{ is a true cdf}\right\}, \tag{4}$$

and estimate this parameter by

$$\hat{p}_i^{c_n} = \inf\left\{\pi_i \in (0,1) \,:\, \pi_i d_n\big(F_i(\cdot, \pi_i, \hat{L}_{i,n}), \check{F}_{i,n}(\cdot, \pi_i)\big) \leq \frac{c_n}{\sqrt{n_i}}\right\},$$

where the tuning parameter $c_n \to +\infty$ not too fast, $d_n$ is the empirical $L^2(\mathbb{L}_{i,n})$ distance, $\check{F}_{i,n}(\cdot, \pi_i)$ is the closest cdf from $F_i(\cdot, \pi_i, \hat{L}_{i,n})$ in the $L^2(\mathbb{L}_{i,n})$ sense obtained by isotonic regression and the pool-adjacent-violators algorithm. The authors prove the consistency of the estimator $\hat{p}_i^{c_n}$ and provide its convergence rate (see Theorem 4 in Patra and Sen, 2016). However, there is no central limit theorem for this estimator, and cross-validations are often required to improve the estimation of the unknown correct mixture proportion $p_{0,i}$ defined in (4).

### 2.1.2 The BVdk estimator

As already mentioned, the *BVdk* estimator relies on a symmetry assumption regarding the unknown component $F_i$ about some location parameter $\mu_i$, see Bordes and Vandekerkhove (2010; and previous works by Bordes et al., 2006a). Suppose that we observe an i.i.d. sample $X_i = (X_{i,1}, \ldots, X_{i,n_i})$ with cdf $L_i$ defined by

$$L_i(x) = (1 - p_i)\, G_i(x) + p_i\, F_i(x - \mu_i), \qquad x \in \mathbb{R}, \tag{5}$$

where the unknown Euclidean parameters to be estimated are $p_i \in ]0,1[$ and $\mu_i \in \mathbb{R}$. Under some identifiability assumptions, the following inversion formula with parameters $(p_i, \mu_i)$ holds:

$$F_i(x) = \frac{1}{p_i}\left(L_i(x + \mu_i) - (1 - p_i)G_i(x + \mu_i)\right), \qquad x \in \mathbb{R}.$$

In addition, according to the symmetry assumption, one knows that $F_i(x) = 1 - F_i(-x)$ for all $x \in \mathbb{R}$. Keeping this in mind, we can introduce the two following functions

$$H_1(x; p_i, \mu_i, L_i) = \frac{1}{p_i} L_i(x + \mu_i) - \frac{1 - p_i}{p_i} G_i(x + \mu_i),$$

and

$$H_2(x; p_i, \mu_i, L_i) = 1 - \frac{1}{p_i} L_i(-x + \mu_i) + \frac{1 - p_i}{p_i} G_i(-x + \mu_i).$$

Using the symmetry of $F_i$ and denoting $(p_0, \mu_0)$ the true (unknown) Euclidean parameters, we have

$$H(x; p_{0,i}, \mu_{0,i}, L_i) = H_1(x; p_{0,i}, \mu_{0,i}, L_i) - H_2(x; p_{0,i}, \mu_{0,i}, L_i) = 0, \qquad \text{for all } x \in \mathbb{R}.$$

In this setup, one expects that for all $(p_i, \mu_i) \neq (p_{0,i}, \mu_{0,i})$ an *ad hoc* $\mathbb{R}$-weighted quadratic norm applied to the function $H$ will be strictly positive. This leads to consider the *discrepancy measure*, also called *contrast function*, given by

$$d(p_i, \mu_i) = \int_{\mathbb{R}} H^2(x; p_i, \mu_i, L_i)\, dL_i(x).$$

The final step consists then in estimating the Euclidean parameters, based on an adequate empirical version $d_{n_i}$ of the contrast function $d$, as follows:

$$(\hat{p}_i, \hat{\mu}_i) = \arg \min_{(p_i, \mu_i) \in \Theta} d_{n_i}(p_i, \mu_i), \text{ where } d_{n_i}(p_i, \mu_i) = \int_{\mathbb{R}} H^2(x; p_i, \mu_i, \tilde{L}_i)\, d\hat{L}_i(x), \tag{6}$$

where $\hat{L}_i$ and $\tilde{L}_i$ denote respectively the empirical and the kernel-smoothed version of the cdf $L_i$ and $\Theta$ is a compact set of $]0, 1[ \times \mathbb{R}$. Identifiability along with kernel conditions are established in Bordes and Vandekerkhove (2010) to prove the asymptotic normality of the vector $(\hat{\mu}_i, \hat{p}_i, \hat{F}_i(\cdot))$, see Theorem 3.2 of the paper. The authors also show how to estimate consistently the variance-covariance matrix involved in their functional central limit theorem. This matrix is key to get a correct insight about the accuracy of the obtained estimates, and fundamental for future derivation of statistical tests.

### 2.1.3 The IBM estimator

Consider the two-sample case involving two separate i.i.d. samples $X_i = (X_{i,l})_{1 \leq l \leq n_i}$ and $X_j = (X_{j,l})_{1 \leq l \leq n_j}$ distributed respectively according to the contamination model (3). Under the assumption $H_0 : F_i = F_j$, Milhaud et al. (2024a) propose, under very minimal conditions, a method to $\sqrt{n}$-consistently estimate, with a fully derived functional central limit theorem, the proportion parameters $p_i$ and $p_j$, along with the unknown cdf $(F_i = F_j)$. The idea is to get benefit from the two separate samples at disposal (sharing the same unknown component under $H_0$) to test whether the equality of their unknown contaminating distributions $F_i$ and $F_j$ is plausible or not. To get their theoretical result, the authors first consider an inversion step which allows them to define two parametric function families $\mathcal{F}_k$ $(k = i, j)$ such that

$$\mathcal{F}_k = \{F_k(x, p_k) := (L_k(x) - (1 - p_k)G_k(x))/p_k,\ p_k \in \Theta_k,\ x \in \mathbb{R}\}, \tag{7}$$

with $\Theta_k = [\delta_{1,k}, \delta_{2,k}]$ where $0 < \delta_{1,k} < 1 < \delta_{2,k} < +\infty$.

They propose to call *IBM-method* the semiparametric estimation strategy based on this 'Inversion' step followed by the 'Best Matching' step between the $\mathcal{F}_i$ and $\mathcal{F}_j$ families, to look at how close they can possibly be. In this view, they introduce the best-matching step by considering the discrepancy measure

$$d(\theta) = \int_{\mathbb{R}} \left(F_i(x, p_i) - F_j(x, p_j)\right)^2 dU(x), \tag{8}$$

where $\theta = (p_i, p_j) \in \Theta_i \times \Theta_j$, $U$ is a finite integrating measure over $\mathbb{R}$ that allows one to weigh more or less some domains of the observed distribution (in practice $U$ is a Uniform distribution over the observations support). The best matching is reached when $d(\theta) = 0$, which condition is achieved if and only if $\theta = \theta_0 = (p_{0,i}, p_{0,j})$ the true value of the parameter under identifiability conditions, see condition (I) in Section 3 of Milhaud et al. (2024a).

The final step consists then in estimating the Euclidean parameters, based on an adequate empirical version $d_{n_i,n_j}$ of the contrast function $d$, as follows:

$$(\hat{p}_i, \hat{p}_j) = \arg \min_{(p_i, p_j) \in \Theta} d_{n_i,n_j}(p_i, p_j), \tag{9}$$

where

$$\hat{d}_{n_i,n_j}(p_i,p_j) = \int_{\mathbb{R}} \left(\tilde{F}_i(x,p_i) - \tilde{F}_j(x,p_j)\right)^2 dU(x),$$

and $\tilde{F}_i$ (respectively $\tilde{F}_j$) belongs to the parametric family defined in (7) (replacing $L_i$ by its empirical estimator $\hat{L}_i$). The asymptotic properties of the above estimator are detailed under $H_0$ and $H_1$ in Milhaud et al. (2024a), see Theorems 1-2.

### 2.2 Estimation of the decontaminated cumulative distribution function

Once the weight $p_i$ of the unknown component distribution $F_i$ has been consistently estimated, it is pretty straightforward to estimate the unknown cdf $F_i$ itself by using the inversion formula

$$\hat{F}_i(x) = \frac{\hat{L}_i(x) - (1-\hat{p}_i)G_i(x)}{\hat{p}_i}, \tag{10}$$

where the estimators $\hat{L}_i$ and $\hat{p}_i$ respectively stand for the empirical cumulative distribution function of the observations and the estimated mixture proportion. The estimator $\hat{F}_i(x)$ is the so-called *decontaminated* version of the unknown cdf of the admixture model.

### 2.3 Testing strategies

Consider $K$ separate admixture samples drawn from (3). Two questions then naturally arise:

*i)* Do the $F_i$'s belong to one given parametric family?

*ii)* Are some $F_i$'s, involved in the $K$-sample, possibly the same?

In the sequel, these two questions will be simply referred to as the *i)* and *ii)* testing problem. We now introduce the methods we implemented in the **admix** package to answer these questions.

#### 2.3.1 Polynomial expansions (1-sample and 2-sample cases)

In Pommeret and Vandekerkhove (2019), the authors consider admixture models in the 1-sample case having a probability density function (pdf) with respect to the Lebesgue measure:

$$\ell_i(x) = (1-p)g_i(x) + pf_i(x), \quad x \in \mathbb{R}, \tag{11}$$

and are interested in testing if the unknown pdf $f_i$, associated with $F_i$ in model (3), belongs to a given parametric family (in practice $i = 1$ since there is only one sample available). The hypotheses of the test, answering the above *i)* testing problem in the absolutely continuous case, are expressed as follows:

$$H_0 : f_i \in \mathcal{F} = \{f_\xi; \xi \in \Lambda\} \quad \text{against} \quad H_1 : f_i \notin \mathcal{F}, \tag{12}$$

where $f_\xi$ is a pdf parametrized by an Euclidean parameter $\xi$ belonging to a parametric space $\Lambda$. The authors focus on the so-called *Gaussianity test*, meaning that $f_i$ is tested to be a Gaussian density. The main idea is to use the expansions of the densities $\ell_i$, $f_i$, and $g_i$ in a dense basis $\{Q_r; r \in \mathbb{N}\}$ (e.g. the Hermite polynomials in the Gaussian case), assuming that $\ell_i$ is square integrable:

$$f_i(x) = \sum_{r\in\mathbb{N}} a_r Q_r(x). \tag{13}$$

The test statistics have the form

$$T_R = \sum_{r\le R} \frac{(\widehat{a}_r - a_r(\widehat{\xi}))^2}{\widehat{\mathrm{Var}}(\widehat{a}_r - a_r(\widehat{\xi}))},$$

which compare pairwise the $a_r$'s coefficients estimated under a relaxed set of conditions versus under the $H_0$ assumption, *i.e.* where each $\widehat{a}_r$ is an empirical estimator of $a_r$ obtained using the inversion formula (10), and $a_r(\widehat{\xi})$ is the estimated coefficient under the null $H_0$ based on the *PS* or *BVdk* estimator. Theoretical guarantees of this testing strategy remain valid if the plugged estimators are $\sqrt{n}$-consistent, which is not the case with the *PS* estimator. However, in practice and from our experiments, both estimators provide satisfactory results in terms of test efficiency.

A penalized rule is used to select the number $R$ of components to be integrated into the test statistic. Theoretically it is assumed that $R$ is lower than $R_n = o(\log(n))$, but in practice $R_n$ is set to 3 by default (which can easily be modified by the user through arguments inside the appropriate function) as this

value seems to provide satisfactory results in a wide range of situations. Writing

$$S_n = \min\left(\arg\max_{1\leq R\leq R_n}\left(n^{\lambda-1}T_R - R\log(n)\right)\right),$$

with $0 < \lambda < 1/2$, Pommeret and Vandekerkhove (2019) show that under the null $H_0$ the penalized test statistic $T_{S_n}$ tends to a chi-square distribution with one degree of freedom, denoted further $\chi^2_1$, see Theorem 2 and Corollary 3 in Pommeret and Vandekerkhove (2019).

An extension of this test to the 2-sample case, considered as a first brick to answer the above *ii)* testing problem, has been proposed later in Milhaud et al. (2022). In their paper, the authors develop a testing methodology suited for the comparison of two symmetric unknown component densities $f_i$ and $f_j$ ($i \neq j$, $1 \leq i, j \leq K$), in line with the hypothesis testing problem

$$H_0 : F_i = F_j \quad \text{against} \quad F_i \neq F_j. \tag{14}$$

Here, $i$ and $j$ represent any pair of populations among the $K$ samples. As in the 1-sample case, they propose a semiparametric penalized $\chi^2$-type testing procedure allowing a pairwise comparison of the unknown components. The test procedure consists in expanding the two unknown densities $f_i$ and $f_j$ in an orthogonal polynomial basis, and then comparing, with an ad hoc method, their coefficients up to a parsimonious rank $R$ selected according to a data-driven technique.

Basically, denoting $\mathcal{Q} = \{Q_r;\, r \in \mathbb{N}\}$ an orthonormal basis of the $L^2(\nu)$ space, where $\nu$ is a common reference measure for the densities expressed in (11), the densities of the model can be rewritten as

$$\ell_k(x) = \sum_{r\geq 0} \ell_{k,r} Q_r(x), \quad \text{with } \ell_{k,r} = \int_{\mathbb{R}} Q_r(x)\ell_k(x)\,\nu(dx), \quad k = i, j. \tag{15}$$

The densities $f_k$ and $g_k$ ($k = i, j$) can be decomposed in the same way, and from (3) and (15), we deduce that

$$\ell_{k,r} = p_k\, f_{k,r} + (1 - p_k)\, g_{k,r}.$$

Finally, testing $H_0 :\ f_i = f_j$ in this set-up is equivalent to the following statistical test:

$$H_0 :\ a_{i,r} \ = \ a_{j,r}, \quad r = 1, 2, \ldots, \tag{16}$$

where

$$a_{k,r} = \frac{\ell_{k,r} - (1 - p_k)g_{k,r}}{p_k}, \quad k = i, j. \tag{17}$$

Given that $g_i$ and $g_j$ are fully known densities (and that their respective coefficients $g_{i,r}$ and $g_{j,r}$ in the polynomial basis expansions are therefore exactly known), it remains to estimate the expansion coefficients of $\ell_i$, $\ell_j$ along with unknown proportions $p_i$ and $p_j$ in (17). To build the test statistic, the idea is then to explore pairwise the difference of such coefficients up to some order $R_n$, and incorporate these differences into a test statistic. Similarly to Pommeret and Vandekerkhove (2019), the test statistics have the following expression

$$T_R = \sum_{r\leq R} \frac{(\widehat{a}_{i,r} - \widehat{a}_{j,r})^2}{\widehat{\mathrm{Var}}(\widehat{a}_{i,r} - \widehat{a}_{j,r})},$$

where each $\widehat{a}_{i,r}$ is an estimator of $a_{i,r}$ obtained by combining empirical estimators with the estimators of $p_i$. A penalized rule is used to select the number of components $S_{n_i,n_j}$ in the test statistic:

$$S_{n_i,n_j} = \min\left(\arg\max_{1\leq R\leq R_n}\left(\left(\frac{n_i n_j}{n_i + n_j}\right)^{\lambda-1} T_R - R\log\left(\frac{n_i n_j}{n_i + n_j}\right)\right)\right),$$

with $0 < \lambda < 1/2$.

To get valid asymptotic results, it is assumed that $R_n = o(\log(n_i n_j/(n_i + n_j)))$. The parameter $\lambda$ can be fixed arbitrarily close to 0.5 to increase the power of the test, or close to 0 to get a more conservative test. Under the null hypothesis, Milhaud et al. (2022) prove that the penalized test statistic $T_{S_{n_i,n_j}}$ tends again towards a chi-square distribution $\chi^2_1$, see Proposition 1 and Corollary 4.

In theory, the latter test statistic can be used to perform the test under the constraint that the plugged estimators $\hat{p}_i$ and $\hat{p}_j$ are $\sqrt{n}$-consistent (thus leading to use the *BVdk* estimator, explaining why the test is originally designed to test the equality of symmetric densities). However, in practice, this testing strategy gives good results in full generality, when plugging in the *PS* estimator (and thus not requiring that $f_i$ and $f_j$ are symmetric densities, see the identifiability definition (4)).

### 2.3.2 Inner convergence regime of the IBM approach ($k$-sample, $k \geq 2$)

First consider the two-sample case and the testing problem (14). For the sake of simplicity, we denote $m = \min(n_i, n_j)$, with the assumption that there exist $\kappa \in \mathbb{Q}$ such that $n_i = \kappa n_j$. The IBM statistic proposed in Milhaud et al. (2024a) is based on the statistic $\mathbf{T}_m = m d_m(\hat{\theta})$, where $d_m = d_{n_i,n_j}$ is an empirical version of $d$ in (8), see expressions (8) and (9).

The authors prove under minimal cross-identifiability conditions (see Section 3 of Milhaud et al., 2024a) that $\hat{\theta} = (\hat{p}_i, \hat{p}_j)$ converges in probability towards the true value of the parameter $\theta_0 = (p_{0,i}, p_{0,j})$ under $H_0$ and $\theta^c$ the minimizer of the contrast function $d(\cdot)$ under $H_1$, with $\theta^c \neq \theta_0$ (Theorem 1). In addition the authors establish the following basic asymptotic behavior (Theorem 2):

$$\begin{aligned} &\mathbf{T}_m = \mathbf{U}_m^0 \xrightarrow{\mathcal{L}} Z(\theta_0, L_i, L_j), && \text{under } H_0, \\ &\mathbf{T}_m = \mathbf{U}_m^1 + \mathbf{V}_m^1, \text{with } \mathbf{U}_m^1 \xrightarrow{\mathcal{L}} Z(\theta^c, L_i, L_j) \text{ and } \mathbf{V}_m^1 \xrightarrow{P} +\infty, && \text{under } H_1, \end{aligned}$$

where the random variables $Z(\theta_0, L_i, L_j)$ and $Z(\theta^c, L_i, L_j)$, corresponding to a parameterized closed form stochastic integral, could be consistently sampled (and thus tabulated) under both $H_0$ or $H_1$ by generating $Z(\hat{\theta}, \hat{L}_i, \hat{L}_j)$-type random variables. These last limiting random variables are strictly connected to the *inner convergence* phenomenon arising either under $H_0$ or $H_1$. Finally, by considering an empirical sample-based $(1-\alpha)$-quantile of the stochastic integral $Z(\hat{\theta}, \hat{L}_i, \hat{L}_j)$, denoted $\hat{q}_{1-\alpha}$, one retains the following $H_0$-rejection rule:

$$\mathbf{T}_m \geq \hat{q}_{1-\alpha} \quad \Rightarrow \quad H_0 \text{ is rejected.} \tag{18}$$

This decision rule expresses the following principle: if the test statistic $\mathbf{T}_m$ is too remote from the *Inner ConVergence* (ICV) regime we could legitimately suspect a difference between $F_i$ and $F_j$, and reject $H_0$.

In Milhaud et al. (2024b), the latter two-sample test is extended to the case where one has to deal with $k$ ($k \geq 3$) separate samples. Write $m = \min_{1 \leq i \leq k}(n_i)$, and suppose that the $n_i$'s are fixed multiples of $m$. The null hypothesis becomes

$$H_0 : F_1 = \cdots = F_k \quad \text{against} \quad \exists i \neq j \text{ s.t. } F_i \neq F_j. \tag{19}$$

Introduce the following set of pairs of indices:

$$\mathcal{S}(k) = \{(i,j) \in \mathbb{N}^2 \mid 1 \leq i < j \leq k\}.$$

Clearly $\mathcal{S}(k)$ contains $c(k) = k(k-1)/2$ elements that can be lexicographically ordered as follows: denote $(i,j) < (i',j')$ if $i < i'$, or if $i = i'$ and $j < j'$, and denote by $r_k[(i,j)]$ the associated rank of $(i,j)$ in the set $S(k)$. This ordering is then used to sum the test statistics over all the pairs of populations. For $(i,j) \in \mathcal{S}(k)$, denote by $T_{i,j}$ the two-sample test statistic (denoted $\mathbf{T}_m$ in the previous two-sample case) to compare populations $i$ and $j$. Then consider the sequence

$$\begin{aligned} U_1 &= T_{1,2} \\ U_2 &= T_{1,2} + T_{1,3} \\ &\vdots \\ U_{c(k)} &= T_{1,2} + \cdots + T_{k-1,k}, \end{aligned}$$

or equivalently

$$U_r = \sum_{(i,j) \in \mathcal{S}(k) : r_k[(i,j)] \leq r} T_{i,j}. \tag{20}$$

Clearly, since the $T_{i,j}$ are positive, one has with probability 1 that $U_1 \leq \cdots \leq U_{c(k)}$. By construction $U_1$ compares the first two populations $(1,2)$, $U_2$ compares simultaneously the first two pairs of populations $(1,2)$ and $(1,3)$, and more generally $U_r$ compares simultaneously the $r$ first pairs of populations. The authors then propose a penalized rule to select $r$ which allows one to select which $r$ pairs will be compared. Basically, this rule is given by

$$S_m = \min\left( \arg \max_{1 \leq r \leq c(k)} \{U_r - r\, m^\varepsilon\} \right), \tag{21}$$

with $0 < \varepsilon < 1$, and $m$ stands for the minimal size $n_i (i = 1, \ldots, K)$ among all samples. The choice of $\varepsilon$ is important for small and moderate sample sizes. A value close to 1 favours a lower value of the test statistic, a lower rejection rate, while a value close to 0 clearly empowers the test.

Each statistic $U_r$ is penalized by $m^\epsilon$ and by the number $r$ of pairs of populations in it (scale factor), according to the standard parsimony principle introduced by Schwarz (1978). In this sense, the sensitive rank $S_m$ will select automatically the rank associated with the most significant group of $T_{i,j}$'s statistics incorporated cumulatively in $U_{S_m}$. The final test statistic is given by $U_{S_m}$. Under the null $H_0$, each difference tends to zero at a rate faster than the penalty term: it is thus expected that only the first statistic $U_1 = T_{1,2}$ will be kept, which enables one to use the theoretical results obtained in the two-sample case, with the same $H_0$-rejection rule, see Theorems 5–7 in Milhaud et al. (2024b).

### 2.4 Clustering

In Section 5 of Milhaud et al. (2024b), the authors propose to adapt the $k$-sample test procedure to obtain a data-driven method to cluster $K$ populations into $N$ subgroups characterized by a common unknown component distribution. This way, one can detect groups that were affected by the same contamination phenomenon. Here, the number $N$ of clusters is not assumed at the beginning of the procedure, but is automatically deduced at the end of a run (which differs from most existing clustering strategies).

To build the first cluster the authors consider the two closest populations in terms of the statistics $T_{i,j}$, $i \neq j \in \{1, \ldots, K\}$. Two populations are thus proposed to be merged to create the first group $\mathcal{G}_1$. The authors test their equality according to the testing procedure introduced in the two-sample test related to the inner convergence (icv) regime, to validate the construction of such a cluster. It is then decided to add populations to the group until the test rejects equality, based on the $k$-sample test. Once this first cluster $\mathcal{G}_1$ is fully identified: close the cluster, remove clustered samples from the initial collection of samples, and create a new cluster $\mathcal{G}_2$. Then look for still unclustered neighbours from the last studied sample that led to rejection of $H_0$ (remaining pool). Again a search for the biggest collection of samples, among the remaining pool, that is tested to share a common unknown component starts. This creates the second cluster. One can iterate this procedure several times until every sample is associated with a cluster, see Algorithm 3 in Milhaud et al. (2024b).

Technical details such as the tuning of hyperparameters involved in the clustering algorithm are available in Milhaud et al. (2024b), Section 4 and Algorithms 1-2. It is proved that the detected number of clusters is asymptotically close to the actual unknown number of clusters (always greater when not the same, see Proposition 9), and their simulation study shows that running the algorithm a few times and averaging the clustering results to answer the question about the composition of clusters leads to an extremely reliable conclusion.

## 3 Package overview

### 3.1 Package structure and usage

The R package **admix** is available under the General Public License (GPL $\geq$ 3) from CRAN at `https://CRAN.R-project.org/package=admix`. You can install and load it using:

```
install.packages("admix")
library(admix)
```

It is developed in an object-oriented design, using the standard S3 paradigm. For the sake of computational speed, some parts of the code have been written using the C++ language. Parallel computation can be used on multicore PCs and computer clusters. The proposed package provides a set of core routines for simulation, visualization and analysis (estimation, test and clustering) of datasets drawn from a contamination (aka admixture) model. Several object classes have been created to simplify the understanding and manipulation of data and results. The list of object classes is available in Table 1.

**Table 1:** List of main object classes available in the admix package.

| Name | Description |
|---|---|
| twoComp_mixt | Simulation of two-component mixtures with given parameter values. |
| admix_model | Admixture model, with the known component distribution and parameters. |
| decontaminated_density | Decontaminated density of the model, once estimated the unknown weight. |
| admix_estim | Estimator of the unknown component weight and related information. |
| htest | Result of the test for the unknown component(s) of the admixture model(s). |
| admix_cluster | Result of the clustering process among K admixture models. |

The three last classes in Table 1 deal with objects created from estimation, test or clustering applied to admixture models. They are detailed further in Section 3.2. Objects of class `decontaminated_density` allow one to compute the value of the decontaminated density at any point of the observed support distribution. In the `twoComp_mixt` class (specifically designed for simulations), the information about the generating model (i.e. the distributions and parameters of the two components of the mixture) is available, in addition to the simulated data itself. Finally, the class `admix_model` stores information about the known component distribution and parameter(s) of the considered admixture model. Classical methods like `print()` and `summary()` (and `plot()` when appropriate) have been implemented for all these objects, in order to provide a user-friendly interface for the results. Numerous extractors were also defined, for instance to retrieve the simulated two-component mixture data and the estimated mixing parameter(s) (respectively using `get_mixture_data()` and `get_mixing_weights()`).

### 3.2 Description of the main functions

There are three main functions for the user, corresponding to the three objectives of the package. Indeed, functions `admix_estim()`, `admix_test()` and `admix_cluster()` are the main workhorses of the **admix** package. We now detail these methods, knowing that they call unexported (but documented) subroutines depending on the specification of their arguments. Each of these three functions leads to creation of a new object, respectively of class `admix_estim`, `htest`, and `admix_cluster`.

#### 3.2.1 Estimation of the mixture proportion

The generic function of the package dedicated to the estimation of the unknown component weight $p_i$ of the admixture model is the method `admix_estim()`. Several samples can be considered (in a $K$-sample handling perspective) inside the estimation process, leading to as many estimates as the number of samples provided. The list of arguments of this function is detailed in Table 2, or obtained by typing `?admix_estim` in the R console (see also the documentation of the package).

**Table 2:** Arguments of the function admix_estim().

| Argument | Description |
|---|---|
| samples | List of the K samples to be studied, all following admixture distributions. |
| admixMod | List of the K admixture models: distributions and parameters for the K known components. |
| est_method | Estimation technique ('PS', 'BVdk' or 'IBM'). |
| ... | Optional specific arguments related to estim_PS(), estim_BVdk() or estim_IBM(). |

The use of the function is deliberately simple, with a very limited number of arguments at first sight. The user must at least include the sample(s) (observations), the considered admixture model(s) (distribution and parameters of the known component), and the chosen estimation method (see Section 2.1). Additional arguments can be specified through the argument `...` to further control the estimation process, depending on which estimation method is selected. These arguments specific to the selected estimation method are listed in the documentation of appropriate unexported subroutines (see `?estim_PS`, `?estim_BVdk`, `?estim_IBM`). Procedure `admix_estim()` fits the model (2) and returns an object of class `admix_estim`, composed of several attributes related to the number of samples under study, the admixture model(s) involved, the population size(s), the chosen estimation method and the estimated unknown weight(s) (as well as the estimated location parameter(s) if *BVdk* estimation was selected). Depending on the chosen estimation technique, a class inheritance system was defined to facilitate, among other things, the display of results. A message reminds the user of the underlying assumptions required for consistent use of the estimation approach in the case of *BVdk* or *IBM* estimators. If one suspects that these assumptions are violated, it is safer to choose the *PS* estimator.

#### 3.2.2 Equality test

The generic function to perform the statistical tests is `admix_test()`. Table 3 provides the list of its arguments, where the first two are shared with `admix_estim()`. This function creates an object of class `htest`, the standard object class storing results of statistical hypothesis tests. The latter object is a list containing attributes related to the distribution of the test statistic under both the null and alternative hypotheses, as well as the $p$-value of the test and some informative sample estimates.

In the same spirit as in `admix_estim()`, the argument `test_method` allows one to choose between two very different testing strategies (see Section 2.3). Statistical tests can be handled either using polynomial expansions of the densities involved in the mixture distributions (by setting `test_method="poly"`,

**Table 3:** Arguments of the function admix_test().

| Argument | Description |
|---|---|
| samples | List of the K samples to be studied, all following admixture distributions. |
| admixMod | List of the K admixture models: distributions and parameters for the K known components. |
| test_method | Testing strategy: either polynomial basis expansion ('poly') or inner convergence ('icv'). |
| conf_level | The confidence level of the statistical test, default to 95%. |
| ... | Optional arguments related to gaussianity_test(), orthobasis_test(), IBM_k_samples_test(). |

calling the unexported subroutines `gaussianity_test()` in the 1-sample case, or `orthobasis_test()` in the 2-sample case), or thanks to the inner convergence property following the *IBM* approach (calling the unexported function `IBM_k_samples_test()`, by setting `test_method="icv"`). In the latter case, intermediary computations use the number of simulations (through argument `n_sim_tab`, see `?IBM_k_samples_test`) involved in the tabulation of the quantile of interest, see Section 2.3. The user can also speed up the computations using the arguments `parallel` and `n_cpu` in place of argument `...`, allowing parallel computations specifying the number of cpu used. The argument `conf_level` is easily understandable and stands for $(1-\alpha)$, where $\alpha$ refers to the level of the test (also called the first-type error, *i.e.* rejecting the null hypothesis when it is actually true). A message also reminds the user of the underlying assumptions required for the consistent use of tests based on polynomial expansion of the probability density function. If these assumptions are not met in the $k$-sample case (with $k \geq 2$), it is better to use the statistical test based on the *inner convergence regime*, setting `test_method="icv"`.

### 3.2.3 Clustering

The function `admix_cluster()` works using the clustering algorithm described in Section 2.4, based on the $k$-sample test. It creates an object of class `admix_cluster` from the list of inputs given in Table 4 , with attributes given as a list containing twelve elements: 1) the number of samples studied; 2) the sample sizes; 3) the admixture models considered; 4) the number of detected clusters; 5) the list of $p$-values for each performed ($k$-sample) statistical test; 6) the cluster affiliation for each population; 7) the chosen confidence level for the $k$-sample tests; 8) the composition of clusters; 9) the size of the clusters; 10) the estimated weights of the unknown component distributions within each cluster; 11) the matrix of pairwise discrepancies across samples; and 12) the tabulated distributions used for the $k$-sample tests (see Section 2.3).

**Table 4:** Arguments of the function admix_cluster().

| Argument | Description |
|---|---|
| samples | List of the K samples to be studied, all following admixture distributions. |
| admixMod | List of the K admixture models: distributions and parameters for the K known components. |
| conf_level | The confidence level of the underlying k-sample tests, default to 95%. |
| tune_penalty | Boolean (default TRUE) to choose between classical or optimized penalty. |
| tabul_dist | Useful for comparisons of detected clusters at different confidence levels. |
| echo | Boolean (default TRUE) to display the remaining computation time of the clustering process. |
| ... | Optional specific arguments related to IBM_k_samples_test(), used in the clustering process. |

Given that this function relies on $k$-sample tests, its arguments are partly similar to the inputs of `admix_test()`. All of them were already discussed in the previous section, except `tabul_dist` and `tune_penalty`. The latter allows one to optimize the penalty of the $k$-sample tests, working automatically on choosing the best tuning parameters involved in the procedure, see Section 4 in (Milhaud et al., 2024b) for further details. Optimized penalty is useful for low/mid-sized samples or unbalanced sizes among the provided samples, to detect alternatives to the null hypothesis. It is thus rather recommended to use it, despite it increases the computation time. By contrast, `tabul_dist` enables one to save running time in that it can store the tabulated distributions involved in the $k$-sample tests related to a previous call of `admix_cluster()`. This way, the sensitivity of the $k$-sample tests, and thus the robustness of the cluster selection, can be explored under various confidence levels.

## 3.3 Computation time

We now offer a brief overview of the computational cost of such implementations using Gaussian mixtures. Our focus is on the three main functions and how the computation time evolves depending on either the sample size or the number of samples under study. Of course, many other frameworks

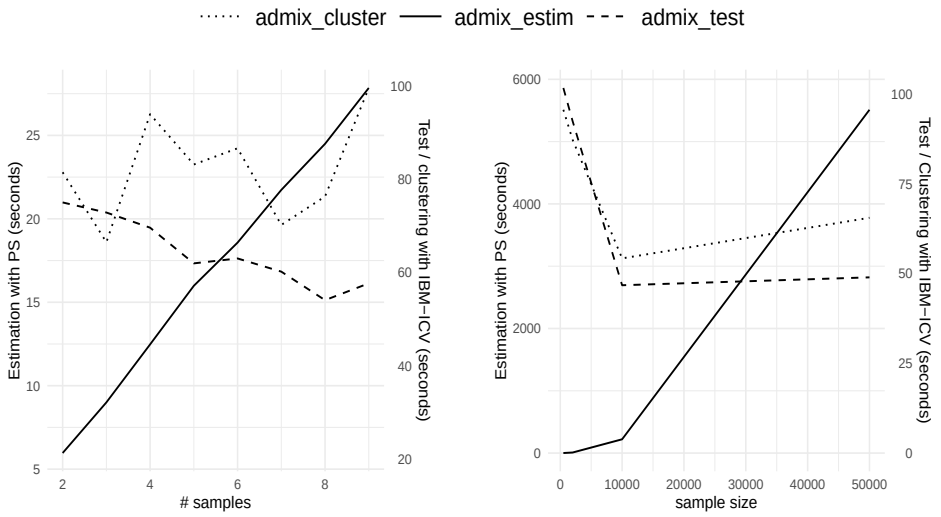



**Figure 2:** Computation time of the three main functions with respect to the number of samples under study (left panel, $n = 2000$ fixed) and the sample size (right panel, $K = 3$ fixed).

could be considered. However, as far as we know, the relationship between computation cost and these two quantities remains similar whatever the choice of the distributions involved in the mixture models. Figure 2 shows computation time as a function of the number of samples $K$ studied for a fixed sample size ($n_i = 2000$; $i = 1, ..., K$) on the left panel, and computation time as a function of sample size (with $K = 3$ fixed) on the right panel. For estimation, we only consider the Patra–Sen method as it is the most general. Testing and clustering are performed under the null hypothesis (similar unknown components), involving the *IBM* estimation method: we consider the *icv* approach, as the latter can be used for any distribution support. Examining the results, we conclude that the time required to estimate with Patra-Sen the unknown quantities of the contamination models increases linearly with both the sample size and the number of samples under study. Conversely, testing and clustering appear costly for the lowest sample size ($n = 500$) or lowest number of samples ($K = 2$), then decrease sharply and finally tend to stabilise. In fact, the calibration of the quantile against which the test statistic must be compared is the longest step of the procedure, which is done at the beginning of the testing or clustering tasks (after estimating the unknown proportions). Firstly, increasing the sample size enables faster and more robust estimations to be obtained, which strongly reduce the *IBM* optimisation process time (this can also be observed when looking at the number of iterations in the optimization run). Secondly, the calibrated quantile remains fixed when new samples are added for testing or clustering, which explains why the testing and clustering times do not increase in such situations. The boolean argument `echo` can be used to provide more information about the remaining time when performing the clustering task.

## 4 Practical use of the package

To make the functionalities of the package as clear as possible, this section has been divided into subsections according to the number of samples the user has to deal with, *i.e.* 1, 2 or $K$, with $K > 2$. We also detail which are the available options depending on the symmetry assumption made (or not) about the unknown component density.

### 4.1 The 1-sample case: estimation and test

The parameters involved in our simulations for the 1-sample case are stored in Table 5. Values to be estimated are the proportions in the third column, as well as unknown distributions in the fifth one.

They refer to two different cases: either the unknown component density is symmetric (`sim1`) or not (`sim2`). We also make the support of the distributions change, to show that available methods enable one to deal with such contexts. More precisely, we consider supports on $\mathbb{R}$ and $\mathbb{R}^+$.

**Table 5:** Simulation parameters in the 1-sample case, with Gaussian-Exponential mixtures.

| | $n_i$ | $p_i$ | $G_i$ | $F_i$ | Support |
|---|---|---|---|---|---|
| sim1 | 700 | 0.44 | $\mathcal{N}(0,1)$ | $\mathcal{N}(-2,1)$ | $\mathbb{R}$ |
| sim2 | 1000 | 0.23 | $\mathcal{E}(3)$ | $\mathcal{E}(1/3)$ | $\mathbb{R}^+$ |

### 4.1.1 Estimation of the mixture proportion assuming a symmetric unknown density

In the one-sample case and assuming a symmetric unknown density, both *BVdk* and *PS* estimators can be used. In terms of implementation, the user has to set the argument `est_method` to the value "BVdk" or "PS" in `admix_estim()`, which calls the appropriate subroutine. If "BVdk" is chosen (which is the case below), the function returns the two estimated Euclidean unknown parameters, *i.e.* the estimated unknown weight as well as the estimated unknown location parameter.

For illustration purposes and thanks to the method `twoComp_mixt()`, we first simulate a two-component Gaussian mixture where the known component distribution $G$ in model (1) is a standard Gaussian distribution, see Table 5. The created object contains the simulated mixture data, as well as the information regarding the generating model. This can be useful when defining the admixture model under consideration in such a simulation framework, through the `admix_model()` method.

```
# Simulate data: 'f' represents the unknown distribution, and 'g' the known one.
list.comp <- list(f = "norm", g = "norm")
list.param <- list(f = list("mean"=-2, "sd"=1), g = list("mean"=0, "sd"=1))
set.seed(1)
mix1 <- twoComp_mixt(n = 700, weight = 0.44, comp.dist = list(list.comp$f, list.comp$g),
                     comp.param = list(list.param$f, list.param$g))
print(mix1)

#>
#> Call:twoComp_mixt(n = 700, weight = 0.44, comp.dist = list(list.comp$f,
#>     list.comp$g), comp.param = list(list.param$f, list.param$g))
#>
#> Number of observations: 700
#>
#> Simulated data (first 5 obs.):
#>  1.473881 0.6772685 -1.620037 -2.192798 1.577892
#> Simulated observations coming from the 1st component (first 5 obs.):
#>  -1.620037 -2.192798 -1.403766 -3.173577 -2.155643
#> Simulated observations coming from the 2nd component (first 5 obs.):
#>  1.473881 0.6772685 1.577892 -0.1952588 -2.592328

# Get simulated data, define the admixture model (only 1 known component in real-life)
sim1 <- get_mixture_data(mix1)
admixMod1 <- admix_model(knownComp_dist = "norm",
                         knownComp_param = list("mean" = 0, "sd" = 1))
# Performs estimation with Bordes and Vandekerkhove estimator:
est <- admix_estim(samples = list(sim1), admixMod = list(admixMod1),
                   est_method = 'BVdk', compute_var = TRUE)
print(est)

#>
#> Call:
#> admix_estim(samples = list(sim1), admixMod = list(admixMod1),
#>     est_method = "BVdk", compute_var = TRUE)
#>
#> Method: BVdk  -  Number of samples: 1
#>
#>  sample size mix_weight location var_mix_weight var_location
#>    sim1  700      0.431    -2.10        0.00128      0.01302
```

The obtained estimates are close to the true values, see Table 5. Moreover, one also gets the information about the variance of the estimators in that estimation approach.

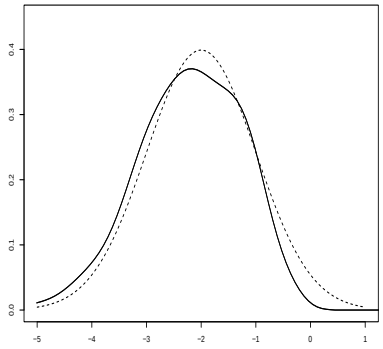


**Figure 3:** Theoretical gaussian density $\mathcal{N}(-2,1)$ (dashed) and decontaminated density (plain).

### 4.1.2 Estimation of the mixture proportion without the symmetric constraint

In this framework, the only available estimator is the *PS* estimator. The estimation of the unknown proportion is performed using `admix_estim()`, setting `est_method = "PS"`. To go further into the analysis, one can use specific arguments in place of the `...` argument, which are available by typing `?estim_PS`. Notably, parameters related to the cross-validation procedure (supposed to make the estimation more robust) can be specified. We now consider the second dataset `sim2` in Table 5.

```
list.comp <- list(f = "exp", g = "exp")
list.param <- list(f = list("rate" = 1/3), g = list("rate" = 3))
mix2 <- twoComp_mixt(n = 1000, weight = 0.23, comp.dist = list(list.comp$f, list.comp$g),
                     comp.param = list(list.param$f, list.param$g))
sim2 <- get_mixture_data(mix2)
admixMod2 <- admix_model(knownComp_dist = mix2$comp.dist[[2]],
                         knownComp_param = mix2$comp.param[[2]])
admix_estim(samples = list(sim2), admixMod = list(admixMod2), est_method = 'PS')

#>
#> Call:
#> admix_estim(samples = list(sim2), admixMod = list(admixMod2),
#>     est_method = "PS")
#>
#> Method: PS  -  Number of samples: 1
#>
#>  sample size mix_weight
#>    sim2 1000      0.208
```

Again, the mixing weight has been correctly estimated. Note that the asymptotic variance of the estimator is explicitly known with the *BVdk* estimator, which is not the case here.

### 4.1.3 Estimation of the decontaminated density

Once the estimations of the mixing weight are performed, it is then possible to estimate the decontaminated density as detailed in Section 2.2. To illustrate this, we focus on the first admixture model considered in Table 5, with dataset `sim1`. The following code enables one to get the decontaminated density of the unknown component in admixture `sim1`, and visualize it in Figure 3.

```
est_weight <- get_mixing_weights(est)
# Build the object related to the decontaminated density:
dec.f <- decontaminated_density(sample1 = sim1, estim.p = est_weight,
                                admixMod = admixMod1)
par(mar = c(3,3,0.5,1.5), cex = 0.7)
plot(x = dec.f, xlim = c(-5,1), ylim = c(0,0.45), main = "")
lines(x = seq(from = -5, to = 1, by = 0.01),
      y = dnorm(seq(from=-5, to=1, by=0.01), mean=-2, sd=1), lty=2)
```

As expected, the decontaminated density looks like a standardized Gaussian centered at -2.

#### 4.1.4 Statistical test

In the 1-sample case, the implemented test is a *gaussianity test*, called by `admix_test()` setting argument `test_method` to "poly". In addition to the classical inputs of `admix_test()`, the user can specify through argument `...` other additional arguments of the unexported subroutine `gaussianity_test()` (see `?gaussianity_test`). These arguments are connected to the underlying theory, namely the order to which the expansion of the density is performed and the penalization exponent which appears in the penalization rule (see the parameter $\lambda$ in Section 2.3). It is thus possible and recommended to play with these arguments, whose default values have been fixed in order to give satisfactory results in most of the situations (without any prior knowledge about the result of the test, the penalty exponent is set to 0.25).

```
admix_test(samples = list(sim1), admixMod = list(admixMod1),
           test_method = 'poly', conf_level = 0.95)

#>
#>  Gaussianity test for the unknown component distribution
#>
#> data:  samples[[1]]
#> T = 0.10176, expansion order S = 1, p-value = 0.7497
#> alternative hypothesis: Unknown component of the contamination
#>                         model is not normally distributed
#> sample estimates:
#>     Weight   Location   Variance
#>  0.4934484 -1.9588427  1.3772585
```

Here, the results show that the null hypothesis cannot be rejected, with a p-value about 0.75. This is the expected output since the data `sim1` were generated according to a two-component Gaussian mixture, see Table 5.

### 4.2 The 2-sample case: estimation and test

In contrast with the 1-sample case, the 2-sample setting gives the opportunity to use the *IBM* approach, whether for estimation or testing purposes. Table 6 stores the simulation parameters used in the framework of continuous random variables, with symmetric unknown component densities or not. Examples with discrete random variables, only available through the *IBM* approach, are provided further in the current section.

**Table 6:** Parameters in the two-sample case (K=2), with Gaussian and Exponential distributions.

| | $n_i$ | $p_i$ | $G_i$ | $F_i$ | Support |
|---|---|---|---|---|---|
| sim3 | 1300 | 0.75 | $\mathcal{E}(1/2)$ | $\mathcal{N}(-3,1)$ | $\mathbb{R}$ |
| sim4 | 1500 | 0.47 | $\mathcal{E}(1/4)$ | $\mathcal{N}(-2.2,1)$ | $\mathbb{R}$ |
| sim5 | 1850 | 0.31 | $\mathcal{N}(-3,1)$ | $\mathcal{E}(1)$ | $\mathbb{R}$ |
| sim6 | 2350 | 0.15 | $\mathcal{N}(5,1)$ | $\mathcal{E}(1)$ | $\mathbb{R}$ |

#### 4.2.1 Estimation of the mixture proportion assuming a symmetric unknown density

Assuming a symmetric density for the unknown component in the two-sample case does not change anything from Section 4.1, except that one duplicates the estimation technique over each observed sample and gets the list of estimated unknown parameters. Following this principle, the implementation is very simple in that the new sample is added into the list of samples through the first argument of the function, when everything else remains identical. We start by simulating the first two models in Table 6, leading to samples `sim3` and `sim4`:

```
list.comp <- list(f3 = "norm", g3 = "exp", f4 = "norm", g4 = "exp")
list.param <- list(f3 = list("mean"=-3, "sd"=1), g3 = list("rate"=1/2),
                   f4 = list("mean"=-2.2, "sd"=1), g4 = list("rate"=1/4))
mix3 <- twoComp_mixt(n = 1300, weight = 0.75, comp.dist = list(list.comp$f3, list.comp$g3),
                     comp.param = list(list.param$f3, list.param$g3))
```

```
mix4 <- twoComp_mixt(n = 1500, weight = 0.47, comp.dist = list(list.comp$f4, list.comp$g4),
                     comp.param = list(list.param$f4, list.param$g4))
sim3 <- get_mixture_data(mix3) ; sim4 <- get_mixture_data(mix4)
admixMod3 <- admix_model(knownComp_dist = mix3$comp.dist[[2]],
                         knownComp_param = mix3$comp.param[[2]])
admixMod4 <- admix_model(knownComp_dist = mix4$comp.dist[[2]],
                         knownComp_param = mix4$comp.param[[2]])
admix_estim(samples = list(sim3, sim4), admixMod = list(admixMod3, admixMod4),
            est_method = 'BVdk', compute_var = FALSE)

#>
#> Call:
#> admix_estim(samples = list(sim3, sim4), admixMod = list(admixMod3,
#>     admixMod4), est_method = "BVdk", compute_var = FALSE)
#>
#> Method: BVdk  -  Number of samples: 2
#>
#>  sample size mix_weight location
#>    sim3 1300      0.737    -2.97
#>    sim4 1500      0.487    -2.13
```

We can see that the estimators are quite close to the true unknown values (see Table 6), respectively $(p_3, \mu_3) = (0.75, -3)$ and $(p_4, \mu_4) = (0.47, -2.2)$.

### 4.2.2 Estimation of the mixture proportion without the symmetric constraint

Here, there are two possibilities: either duplicate the procedure described in Section 4.1 using the *PS* estimator, or consider the *IBM* approach. In the first case, simply add the new sample to the list of existing samples where unknown proportions have to be estimated. We now work with samples `sim5` and `sim6` in Table 6:

```
list.comp <- list(f5 = "exp", g5 = "norm", f6 = "exp", g6 = "norm")
list.param <- list(f5 = list("rate"=1), g5 = list("mean"=-3, "sd"=1),
                   f6 = list("rate"=1), g6 = list("mean"=5, "sd"=1))
mix5 <- twoComp_mixt(n = 1850, weight = 0.31, comp.dist = list(list.comp$f5, list.comp$g5),
                     comp.param = list(list.param$f5, list.param$g5))
mix6 <- twoComp_mixt(n = 2350, weight = 0.15, comp.dist = list(list.comp$f6, list.comp$g6),
                     comp.param = list(list.param$f6, list.param$g6))
sim5 <- get_mixture_data(mix5) ; sim6 <- get_mixture_data(mix6)
admixMod5 <- admix_model(knownComp_dist = mix5$comp.dist[[2]],
                         knownComp_param = mix5$comp.param[[2]])
admixMod6 <- admix_model(knownComp_dist = mix6$comp.dist[[2]],
                         knownComp_param = mix6$comp.param[[2]])
admix_estim(samples = list(sim5,sim6), admixMod = list(admixMod5,admixMod6),
            est_method = 'PS', method = "fixed")

#>
#> Call:
#> admix_estim(samples = list(sim5, sim6), admixMod = list(admixMod5,
#>     admixMod6), est_method = "PS", method = "fixed")
#>
#> Method: PS  -  Number of samples: 2
#>
#>  sample size mix_weight
#>    sim5 1850      0.315
#>    sim6 2350      0.147
```

Once again, the parameters are consistently estimated (0.315 versus 0.31, and 0.147 versus 0.15).

The alternative, which consists in using the *IBM* estimator, gives consistent results if the unknown components of the two admixture models have the same distribution (and provided that the two known components have different distributions). If the two known components are identical, the estimated ratio of weights is consistent towards the actual ratio of weights (instead of the weights themselves), see the Supplementary Material in Milhaud et al. (2024a). Given our parameters for

simulating `sim5` and `sim6`, we have two different known components and two similar unknown components, which should thus lead to consistent estimators. *However, the user must keep in mind that obtained estimators using IBM are most of the time inconsistent, since there is no reason for the two unknown components to share the same distribution*. To perform the estimation, simply call `admix_estim()` with argument `est_method` set to "IBM", which calls in practice the unexported subroutine `estim_IBM()` that estimates the unknown proportions of the contaminating components.

```
est <- admix_estim(samples = list(sim5, sim6), admixMod = list(admixMod5, admixMod6),
                   est_method = 'IBM', compute_var = TRUE)
print(est)
```

```
#>
#> Call:
#> admix_estim(samples = list(sim5, sim6), admixMod = list(admixMod5,
#>     admixMod6), est_method = "IBM", compute_var = TRUE)
#>
#> Method: IBM  -  Pairwise estimation
#>
#>          pair size_1st size_2nd mix_weight_1st var_1st mix_weight_2nd var_2nd
#>  sim5 vs sim6     1850     2350          0.318 0.00014          0.149   7e-05
```

We can check whether using this estimator in the general case leads to unreliable estimates, due to different unknown component distributions:

```
admix_estim(samples = list(sim3,sim4), admixMod = list(admixMod3,admixMod4),
            est_method = 'IBM', compute_var = FALSE)
```

```
#>
#> Call:
#> admix_estim(samples = list(sim3, sim4), admixMod = list(admixMod3,
#>     admixMod4), est_method = "IBM", compute_var = FALSE)
#>
#> Method: IBM  -  Pairwise estimation
#>
#>          pair size_1st size_2nd mix_weight_1st var_1st mix_weight_2nd var_2nd
#>  sim3 vs sim4     1300     1500          1.356      NA          0.720      NA
```

Here, the estimated weight of the unknown component distribution in the first sample is unrealistic, making it clear that the estimate is unreliable. On the other hand, it is possible to observe a realistic weight estimate while in this configuration. The user should therefore *never* choose the *IBM* estimation method without performing the test for equality of unknown component distributions before. In full generality, it is safer to use the *PS* estimator for robust estimation purposes.

### 4.2.3 Estimate the decontaminated densities

In the two-sample case, both decontaminated densities can be plotted on the same graph, which is useful to get a visual comparison. To this aim, it is recommended to use the `plot()` method designed for objects of class `decontaminated_density`. In Figure 4, we compare the decontaminated densities obtained from admixture models `sim5` and `sim6`.

Clearly, these decontaminated densities look similar (which is expected as they are both exponentially distributed with mean 1, see Table 6). The black curve is based on more observations associated with the unknown component ($0.31 * 1850$), leading to a more accurate estimate. Due to the inversion formula (10), the imperfect estimated proportion and the negative mean of the Gaussian known component, the support of the estimated decontaminated density function is on the real line. The bumps appearing on the red curve may come from the inversion formula applied to some kernel density estimator of the observations that can sometimes enhance some peaks depending on the bandwidth choice. To balance this feeling we also provide on the right side of Figure 4 the decontaminated cdf which sticks overall pretty well to the true unknown cdf. We now show how to formally test their equality.

### 4.2.4 Statistical test

Testing for equality between the two unknown component distributions can be made through two approaches: comparison of expansion coefficients of the corresponding densities in some orthonormal

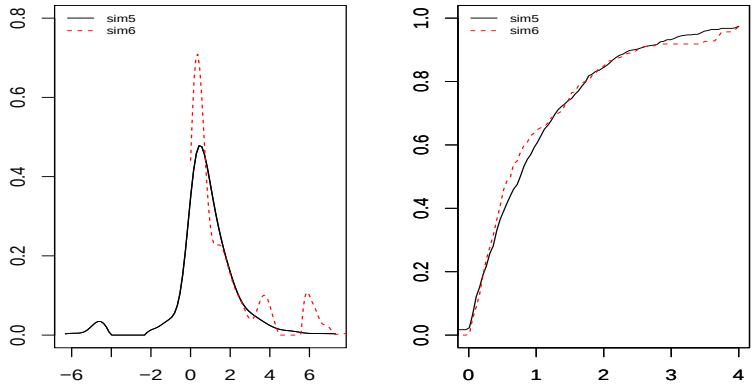



**Figure 4:** Decontaminated densities and cdfs obtained from samples sim5 (plain) and sim6 (dashed).

basis, or using the inner convergence property (see Section 2.3). The function dedicated to performing this statistical test is `admix_test()`. Depending on the specification of its arguments, either the unexported subroutines `orthobasis_test()` or `IBM_k_samples_test()` is called. We first show how to perform the test based on polynomial expansions of the densities, setting argument `test_method` to the value "poly" and testing the equality of unknown components in `sim3` and `sim4` models (see Table 6).

```
admix_test(samples = list(sim3,sim4), admixMod = list(admixMod3,admixMod4),
           test_method = "poly", conf_level = 0.95, support = "Real")

#>
#>  Equality test of unknown distributions with polynomial expansions of
#>  pdfs
#>
#> data:  samples
#> T = 5.1288, expansion order S = 1, p-value = 0.02353
#> alternative hypothesis: Distributions of unknown components involved
#>                         in the contamination models are different
```

The tests provide consistent results, in that `sim3` and `sim4` were admixture models with close (but different) unknown component distributions. The test is performed with default values for all arguments, including hidden arguments of the internal subroutine `orthobasis_test()` which is called. In practice, we advise using this test only if the initial sample sizes are large enough. Indeed, the procedure splits the initial samples into subsamples to decorrelate the various estimators involved in the test strategy, which leads to lowering the size of the data considered for each estimator plugged into the test statistic. In addition, we recommend avoiding using this testing strategy in the general case, since conditions that guarantee the asymptotic properties of this test are usually not met. Indeed, on the one hand the symmetry of the unknown component density is sometimes hard to figure out, which should prevent using the *BVdk* estimator (through argument `est_method`). On the other hand, using the *PS* estimator plugged into the test statistic is in theory not allowed, since the latter estimator is not square-root $\sqrt{n}$-consistent (see Section 2.3).

An alternative to test the equality of the two unknown components is therefore to use the *i*nner *con*vergence property (*icv*), which remains valid whatever the context. In this case, computations can be accelerated thanks to the package **parallel** that allows for parallel computing. One of the most important arguments for the reliability of the test is `n_sim_tab` (with default value set to 100), which controls the number of simulated realizations of the stochastic integral involved in the tabulation of the quantile to be tested against (see Section 2.3). The higher this argument is, the more reliable the test should be. Not surprisingly, increasing `n_sim_tab` causes a much longer computation time. In terms of implementation, setting the argument `test_method` to "icv" in the generic function `admix_test()` calls the unexported subroutine `IBM_k_samples_test()` (see `?IBM_k_samples_test` for the list of potential specific arguments in place of `...` in `admix_test()`). We first test the equality of unknown component distributions between `sim3` and `sim4`, and then between `sim5` and `sim6`.

```
admix_test(samples = list(sim3,sim4), admixMod = list(admixMod3,admixMod4),
           test_method = "icv", conf_level = 0.95, n_sim_tab = 50,
           parallel = TRUE, n_cpu = 8)
```

```
#>
#>  Equality test of unknown distributions using Inner ConVergence regime
#>
#> data:  samples
#> T = 2.2896, p-value = 1e-12
#> alternative hypothesis: Distributions of unknown components involved
#>                         in the contamination models are different

admix_test(samples = list(sim5,sim6), admixMod = list(admixMod5,admixMod6),
           test_method = "icv", conf_level = 0.95, n_sim_tab = 50,
           parallel = TRUE, n_cpu = 8)

#>
#>  Equality test of unknown distributions using Inner ConVergence regime
#>
#> data:  samples
#> T = 1.0315, p-value = 0.9762
#> alternative hypothesis: Distributions of unknown components involved
#>                         in the contamination models are different
```

Again, the results of the tests are consistent with the nature of the samples (see Table 6). In particular, the first equality test between `sim3` and `sim4` leads to unreliable estimates of the weights, causing the automatic rejection of the null hypothesis. Now, one can introduce some examples based on other types of support, since the *IBM* approach as well as the *ICV* property are suitable for every kind of observation. More precisely, the next section considers either continuous or discrete distributions.

### 4.2.5 Estimations and tests on other supports with IBM

In Table 7, we store the information about simulation parameters for distributions supported on discrete or bounded supports. This way, we show that the package covers a large panel of potential applications. For conciseness, we do not explore here a wide range of situations, but the user is encouraged to play with different distributions and settings (equality or not of the unknown component distributions).

**Table 7:** Parameters in the 2-sample case, with other support types (involving Binomial, Poisson, Negative Binomial, Uniform and Beta distributions).

| | $n_i$ | $p_i$ | $G_i$ | $F_i$ | Support |
|---|---|---|---|---|---|
| sim7 | 550 | 0.40 | $\mathcal{B}(10, 0.4)$ | $\mathcal{P}(2)$ | $\mathbb{N}$ |
| sim8 | 900 | 0.25 | $\mathcal{NB}(10, 0.9)$ | $\mathcal{P}(2)$ | $\mathbb{N}$ |
| sim9 | 600 | 0.33 | $\mathcal{U}([0,1])$ | $\mathcal{B}eta(1.2, 5)$ | $[0,1]$ |
| sim10 | 900 | 0.18 | $\mathcal{U}([0,2])$ | $\mathcal{B}eta(1.2, 5)$ | $[0,2]$ |

Firstly, let us perform some estimations and tests on $\mathbb{N}$-discrete support, considering the first two models in Table 7 leading to samples `sim7` and `sim8`.

```
list.comp <- list(f7 = "pois", g7 = "binom", f8 = "pois", g8 = "nbinom")
list.param <- list(f7 = list("lambda"=2), g7 = list("size"=10, "prob"=0.4),
                   f8 = list("lambda"=2), g8 = list("size"=10, "prob"=0.9))
mix7 <- twoComp_mixt(n = 550, weight = 0.4, comp.dist = list(list.comp$f7, list.comp$g7),
                     comp.param = list(list.param$f7, list.param$g7))
mix8 <- twoComp_mixt(n = 900, weight = 0.25, comp.dist = list(list.comp$f8, list.comp$g8),
                     comp.param = list(list.param$f8, list.param$g8))
sim7 <- get_mixture_data(mix7) ; sim8 <- get_mixture_data(mix8)
admixMod7 <- admix_model(knownComp_dist = mix7$comp.dist[[2]],
                         knownComp_param = mix7$comp.param[[2]])
admixMod8 <- admix_model(knownComp_dist = mix8$comp.dist[[2]],
                         knownComp_param = mix8$comp.param[[2]])
admix_test(samples = list(sim7,sim8), admixMod = list(admixMod7,admixMod8),
           test_method = 'icv', n_sim_tab = 50, parallel = T, n_cpu = 8)
```

```
#>
#>  Equality test of unknown distributions using Inner ConVergence regime
#>
#> data:  samples
#> T = 0.29595, p-value = 0.98
#> alternative hypothesis: Distributions of unknown components involved
#>                         in the contamination models are different
```

The test did not reject the equality between `sim7` and `sim8`, which is in line with reality. Knowing that the null hypothesis has not been rejected, we now estimate the proportions with the *IBM* method:

```
admix_estim(samples = list(sim7,sim8), admixMod = list(admixMod7,admixMod8),
            est_method = "IBM")
```

```
#>
#> Call:
#> admix_estim(samples = list(sim7, sim8), admixMod = list(admixMod7,
#>     admixMod8), est_method = "IBM")
#>
#> Method: IBM  -  Pairwise estimation
#>
#>          pair size_1st size_2nd mix_weight_1st var_1st mix_weight_2nd var_2nd
#>  sim7 vs sim8      550      900          0.419      NA          0.270      NA
```

Secondly we are interested in finite bounded support, see the third and fourth models (`sim9` and `sim10`) in Table 7:

```
list.comp <- list(f9 = "beta", g9 = "unif", f10 = "beta", g10 = "unif")
list.param <- list(f9= list("shape1"=1.2, "shape2"=5, "ncp"=0), g9= list("min"=0, "max"=1),
              f10= list("shape1"=1.2, "shape2"=5, "ncp"=0), g10= list("min"=0, "max"=2))
mix9 <- twoComp_mixt(n = 600, weight = 0.33, comp.dist = list(list.comp$f9, list.comp$g9),
                     comp.param = list(list.param$f9, list.param$g9))
mix10 <- twoComp_mixt(n = 900, weight = 0.18, comp.dist = list(list.comp$f10, list.comp$g10),
                      comp.param = list(list.param$f10, list.param$g10))
sim9 <- get_mixture_data(mix9) ; sim10 <- get_mixture_data(mix10)
admixMod9 <- admix_model(knownComp_dist = mix9$comp.dist[[2]],
                         knownComp_param = mix9$comp.param[[2]])
admixMod10 <- admix_model(knownComp_dist = mix10$comp.dist[[2]],
                          knownComp_param = mix10$comp.param[[2]])
admix_test(samples=list(sim9,sim10), admixMod=list(admixMod9,admixMod10),
           test_method = 'icv', n_sim_tab = 50, parallel = TRUE, n_cpu = 8)
```

```
#>
#>  Equality test of unknown distributions using Inner ConVergence regime
#>
#> data:  samples
#> T = 1.186, p-value = 1
#> alternative hypothesis: Distributions of unknown components involved
#>                         in the contamination models are different
```

Once again, the results are consistent in that the equality of unknown components is not rejected. We can conclude that whatever the support considered in our examples, the results meet our performance expectations.

### 4.3 The K-sample case: estimation, test and clustering

In the $K$-sample framework, another question arises: do some of the $K$ samples share the same unknown component distribution, creating possible clusters? To proceed to such experiments, Table 8 introduces the simulation parameters in the 3-sample case. Estimations, tests and clustering are performed on either continuous or discrete supports.

**Table 8:** Simulation parameters for estimations and tests in the $K$-sample case.

| | $K$ | $n_i$ | $p_i$ | $G_i$ | $F_i$ | Support |
|---|---|---|---|---|---|---|
| sim1 | | 3500 | 0.70 | $\mathcal{E}(1/4)$ | $\mathcal{N}(1,1)$ | $\mathbb{R}$ |
| sim2 | 3 | 5000 | 0.40 | $\mathcal{E}(1/3)$ | $\mathcal{N}(1,1.3)$ | $\mathbb{R}$ |
| sim3 | | 3000 | 0.55 | $\mathcal{G}(10,1/3)$ | $\mathcal{N}(1,1)$ | $\mathbb{R}$ |
| sim4 | | 3000 | 0.70 | $\mathcal{B}(10,0.4)$ | $\mathcal{P}(12)$ | $\mathbb{N}$ |
| sim5 | 3 | 3500 | 0.50 | $\mathcal{NB}(10,0.9)$ | $\mathcal{P}(12)$ | $\mathbb{N}$ |
| sim6 | | 4000 | 0.30 | $\mathcal{P}(1)$ | $\mathcal{P}(12)$ | $\mathbb{N}$ |

### 4.3.1 Estimation of the unknown quantities

Given that the *IBM* procedure provides accurate estimates of the unknown weights only when the unknown components of the admixture models are tested to be equal, and that the *BVdk* estimator requires shape constraints about the unknown component densities (symmetry), the privileged procedure is the *PS* estimator in full generality. It is thus deployed in each sample, and we now give an example based on three samples ($K = 3$) with simulation parameters stored in Table 8.

```
# Simulation of data (3 samples) with continuous random variables:
comp_c <- list(f1 = "norm", g1 = "exp", f2 = "norm", g2 = "exp",
               f3 = "norm", g3 = "gamma")
param_c <- list(f1 = list("mean"=1,"sd"=1), g1 = list("rate"=1/4),
                f2 = list("mean"=1,"sd"=1.3), g2 = list("rate"=1/3),
                f3 = list("mean"=1,"sd"=1), g3 = list("shape"=10,"scale"=1/3))
mix1 <- twoComp_mixt(n = 3500, weight = 0.7, comp.dist = list(comp_c$f1, comp_c$g1),
                     comp.param = list(param_c$f1, param_c$g1))
mix2 <- twoComp_mixt(n = 5000, weight = 0.4, comp.dist = list(comp_c$f2, comp_c$g2),
                     comp.param = list(param_c$f2, param_c$g2))
mix3 <- twoComp_mixt(n = 3000, weight = 0.55, comp.dist = list(comp_c$f3, comp_c$g3),
                     comp.param = list(param_c$f3, param_c$g3))
sim1 <- get_mixture_data(mix1) ; sim2 <- get_mixture_data(mix2)
sim3 <- get_mixture_data(mix3)
admixMod1 <- admix_model(knownComp_dist = mix1$comp.dist[[2]],
                         knownComp_param = mix1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mix2$comp.dist[[2]],
                         knownComp_param = mix2$comp.param[[2]])
admixMod3 <- admix_model(knownComp_dist = mix3$comp.dist[[2]],
                         knownComp_param = mix3$comp.param[[2]])
# Estimation of unknown mixing weights (Patra and Sen estimator):
admix_estim(samples = list(sim1, sim2, sim3),
            admixMod = list(admixMod1, admixMod2, admixMod3), est_method = 'PS')

#>
#> Call:
#> admix_estim(samples = list(sim1, sim2, sim3), admixMod = list(admixMod1,
#>     admixMod2, admixMod3), est_method = "PS")
#>
#> Method: PS  -  Number of samples: 3
#>
#>  sample size mix_weight
#>    sim1 3500      0.673
#>    sim2 5000      0.368
#>    sim3 3000      0.567

# Simulation of data (3 samples) with discrete random variables:
comp_d <- list(f1 = "pois", g1 = "binom", f2 = "pois",
               g2 = "nbinom", f3 = "pois", g3 = "pois")
param_d <- list(f1 = list("lambda"=12), g1 = list("size"=10, "prob"=0.4),
                f2 = list("lambda"=12), g2 = list("size"=10, "prob"=0.9),
                f3 = list("lambda" = 12), g3 = list("lambda" = 1))
mix4 <- twoComp_mixt(n = 3000, weight = 0.7, comp.dist = list(comp_d$f1, comp_d$g1),
                     comp.param = list(param_d$f1, param_d$g1))
```

```
mix5 <- twoComp_mixt(n = 3500, weight = 0.5, comp.dist = list(comp_d$f2, comp_d$g2),
                     comp.param = list(param_d$f2, param_d$g2))
mix6 <- twoComp_mixt(n = 4000, weight = 0.3, comp.dist = list(comp_d$f3, comp_d$g3),
                     comp.param = list(param_d$f3, param_d$g3))
sim4 <- get_mixture_data(mix4) ; sim5 <- get_mixture_data(mix5)
sim6 <- get_mixture_data(mix6)
admixMod4 <- admix_model(knownComp_dist = mix4$comp.dist[[2]],
                         knownComp_param = mix4$comp.param[[2]])
admixMod5 <- admix_model(knownComp_dist = mix5$comp.dist[[2]],
                         knownComp_param = mix5$comp.param[[2]])
admixMod6 <- admix_model(knownComp_dist = mix6$comp.dist[[2]],
                         knownComp_param = mix6$comp.param[[2]])
# Estimation of unknown mixing weights (Patra and Sen estimator):
admix_estim(samples = list(sim4, sim5, sim6),
            admixMod = list(admixMod4, admixMod5, admixMod6), est_method = 'PS')

#>
#> Call:
#> admix_estim(samples = list(sim4, sim5, sim6), admixMod = list(admixMod4,
#>     admixMod5, admixMod6), est_method = "PS")
#>
#> Method: PS  -  Number of samples: 3
#>
#>  sample size mix_weight
#>    sim4 3000      0.680
#>    sim5 3500      0.484
#>    sim6 4000      0.323
```

Both *BVdk* and *PS* should provide consistent estimates for the first three admixtures since we are in the context of symmetric unknown densities; see Table 8. By contrast, one could not trust the estimates obtained using IBM, since the $F_i$'s do not all have the same distribution. In fact, a pairwise IBM estimation based on comparing the first and third samples should lead to consistent estimates since unknown component distributions are identical in these two samples. However, at this stage, we are not supposed to have this information.

### 4.3.2 Testing

In the $K$-sample framework, with $K > 2$, the unexported subroutine `IBM_k_samples_test()` is systematically called by `admix_test()` (setting argument `test_method` to "icv"). To increase the reliability of the test, the argument `n_sim_tab` plays a key role: the higher it is, the more reliable the conclusion of the test is. If the user can perform parallel computing with multiple cpus, we recommend setting this argument at least to 100 (default value). Moreover, another argument is important in finite sample applications where the sample sizes are unbalanced (or with very limited size): `tune_penalty`. This boolean, set by default to TRUE, allows one to considerably improve the quality of the testing methodology, see (Milhaud et al., 2024b). Here, we test the equality of the three unknown distributions involved in the three models of Table 8, either with continuous or discrete random variables.

```
# Test whether the first three admixtures share the same contaminating phenomenon:
admix_test(samples = list(sim1, sim2, sim3), admixMod = list(admixMod1, admixMod2, admixMod3),
         test_method = "icv", conf_level = 0.95, n_sim_tab = 50, parallel = T, n_cpu = 8)

#>
#>  Equality test of unknown distributions using Inner ConVergence regime
#>
#> data:  samples
#> U = 1.3562, number of terms S = 3, p-value < 2.2e-16
#> alternative hypothesis: Distributions of unknown components involved
#>                         in the contamination models are different

# Same task for the last three admixtures:
admix_test(samples = list(sim4, sim5, sim6), admixMod = list(admixMod4, admixMod5, admixMod6),
         test_method = "icv", conf_level = 0.95, n_sim_tab = 50, parallel = T, n_cpu = 8)
```

```
#>
#>  Equality test of unknown distributions using Inner ConVergence regime
#>
#> data:  samples
#> U = 0.11758, number of terms S = 1, p-value = 0.96
#> alternative hypothesis: Distributions of unknown components involved
#>                         in the contamination models are different
```

The result of these 3-sample tests is in line with what is expected, *i.e.* one rejects the null hypothesis in the continuous case and cannot reject it in the discrete case. Finally, the natural question is now: how can this $k$-sample testing method be used in order to process a model-based clustering of the sample collection?

### 4.3.3 Clustering

The clustering strategy available in the package **admix** is grounded on the $k$-sample testing problem. The generic function of the package dedicated to the clustering of various contaminated samples is the method `admix_cluster()`. In this example, we draw four two-component mixture distributions on $\mathbb{R}^+$ (Exponential-Gamma and Gamma-Gamma distributions), with asymmetric densities. We also draw four two-component multinomial mixtures for the discrete case. The parameters involved in those simulations are available in Table 9.

**Table 9:** Parameters used for simulations (pop1 to pop4 for mixtures of continuous random variables, and pop5 to pop8 in the discrete case), with Gamma, Exponential, and Multinomial distributions.

| | pop1 | pop2 | pop3 | pop4 | pop5 | pop6 | pop7 | pop8 |
|---|---|---|---|---|---|---|---|---|
| $n_i$ | 5600 | 6000 | 4500 | 4000 | 5000 | 3500 | 4000 | 5000 |
| $p_i$ | 0.3 | 0.2 | 0.6 | 0.8 | 0.7 | 0.6 | 0.45 | 0.3 |
| $F_i$ | $\mathcal{G}(16,4)$ | $\mathcal{G}(14,2)$ | $\mathcal{G}(16,4)$ | $\mathcal{G}(14,2)$ | $\mathcal{M}(0.25,0.5,0.25)$ | $\mathcal{M}(0.25,0.5,0.25)$ | $\mathcal{M}(0.1,0.8,0.1)$ | $\mathcal{M}(0.25,0.5,0.25)$ |
| $G_i$ | $\mathcal{E}(1/4)$ | $\mathcal{E}(1/5)$ | $\mathcal{G}(12,0.5)$ | $\mathcal{E}(1/7)$ | $\mathcal{M}(0.8,0.1,0.1)$ | $\mathcal{M}(0.3,0.3,0.4)$ | $\mathcal{M}(0.5,0.3,0.2)$ | $\mathcal{M}(0.1,0.7,0.2)$ |

First, initialize the parameters and draw simulations of the four finite mixture models:

```
# Simulation of Gamma-Exponential / Gamma-Gamma mixtures:
comp <- list(f1 = "gamma", g1 = "exp", f2 = "gamma", g2 = "exp",
             f3 = "gamma", g3 = "gamma", f4 = "gamma", g4 = "exp")
param <- list(f1 = list("shape"=16, "scale"=1/4), g1 = list("rate" = 1/4),
              f2 = list("shape"=14, "scale"=1/2), g2 = list("rate" = 1/5),
              f3=list("shape"=16,"scale"=1/4), g3=list("shape"=12,"scale"=1/2),
              f4 = list("shape"=14, "scale"=1/2), g4 = list("rate" = 1/7))
mix1 <- twoComp_mixt(n = 5600, weight = 0.3, comp.dist = list(comp$f1,comp$g1),
                     comp.param = list(param$f1,param$g1))
mix2 <- twoComp_mixt(n = 6000, weight = 0.2, comp.dist = list(comp$f2,comp$g2),
                     comp.param = list(param$f2,param$g2))
mix3 <- twoComp_mixt(n = 4500, weight = 0.6, comp.dist = list(comp$f3,comp$g3),
                     comp.param = list(param$f3,param$g3))
mix4 <- twoComp_mixt(n = 4000, weight = 0.8, comp.dist = list(comp$f4,comp$g4),
                     comp.param = list(param$f4,param$g4))
# Simulation of samples with mixtures of multinomial random variables:
comp_d <- list(f1 = "multinom", g1 = "multinom", f2 = "multinom", g2 = "multinom",
               f3 = "multinom", g3 = "multinom", f4 = "multinom", g4 = "multinom")
param_d <- list(f1 = list("size" = 1, "prob" = c(0.25,0.5,0.25)),
                g1 = list("size" = 1, "prob" = c(0.8,0.1,0.1)),
                f2 = list("size" = 1, "prob" = c(0.25,0.5,0.25)),
                g2 = list("size" = 1, "prob" = c(0.3,0.3,0.4)),
                f3 = list("size" = 1, "prob" = c(0.7,0.1,0.2)),
                g3 = list("size" = 1, "prob" = c(0.5,0.3,0.2)),
                f4 = list("size" = 1, "prob" = c(0.25,0.5,0.25)),
                g4 = list("size" = 1, "prob" = c(0.1,0.7,0.2)))
mix5 <- twoComp_mixt(n = 12200, weight = 0.55, comp.dist = list(comp_d$f1, comp_d$g1),
                     comp.param = list(param_d$f1, param_d$g1))
mix6 <- twoComp_mixt(n = 11300, weight = 0.75, comp.dist = list(comp_d$f2, comp_d$g2),
                     comp.param = list(param_d$f2, param_d$g2))
mix7 <- twoComp_mixt(n = 15000, weight = 0.45, comp.dist = list(comp_d$f3, comp_d$g3),
```

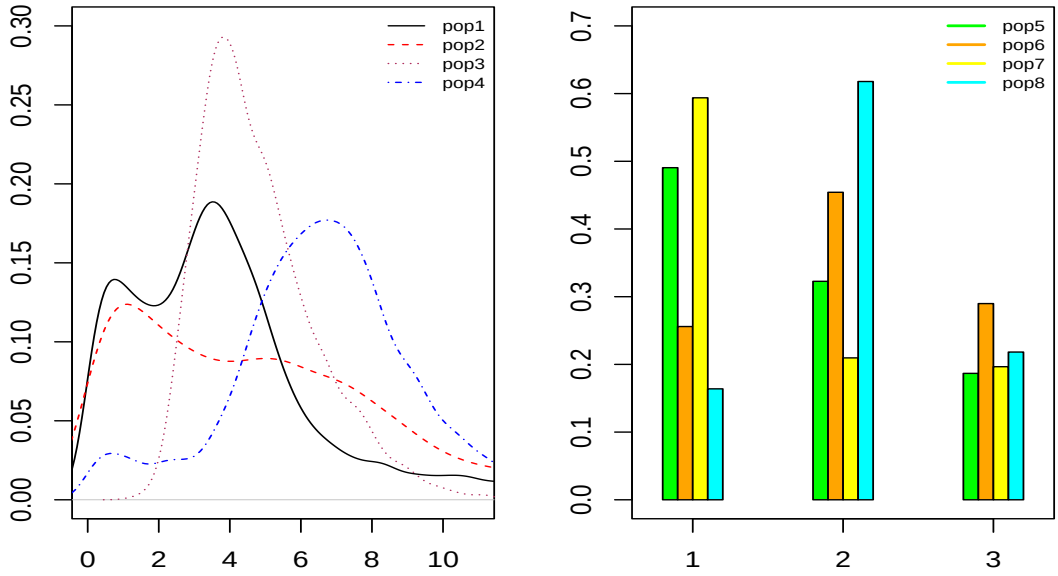



**Figure 5:** Distribution of simulated data (left: continuous case - right: discrete case, see Table 9).

```
                    comp.param = list(param_d$f3, param_d$g3))
mix8 <- twoComp_mixt(n = 17000, weight = 0.42, comp.dist = list(comp_d$f4, comp_d$g4),
                    comp.param = list(param_d$f4, param_d$g4))
```

At this stage, one can observe the distributions displayed in Figure 5.

Can we cluster some of the unknown random sources $F_i$? At first sight and without any other information than the observed distributions and the known $G_i$'s, it seems impossible to decide which populations could share the same unknown components. In particular, the third population, showing an unimodal density, seems very different from the others. However, looking at Table 9, it comes out that the right populations to cluster together (common contamination) are in fact populations 1 and 3, and populations 2 and 4. This shows that it is definitely not trivial to correctly discover the right clusters. We now look for possible clusters among them. The best in this case is to use as many CPUs as possible to speed up the process, since this step is computationally intensive (it requires performing multiple tests):

```
# clustering the continuous contamination models:
admixMod1 <- admix_model(knownComp_dist = mix1$comp.dist[[2]],
                         knownComp_param = mix1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mix2$comp.dist[[2]],
                         knownComp_param = mix2$comp.param[[2]])
admixMod3 <- admix_model(knownComp_dist = mix3$comp.dist[[2]],
                         knownComp_param = mix3$comp.param[[2]])
admixMod4 <- admix_model(knownComp_dist = mix4$comp.dist[[2]],
                         knownComp_param = mix4$comp.param[[2]])
pop1 <- get_mixture_data(mix1) ; pop2 <- get_mixture_data(mix2)
pop3 <- get_mixture_data(mix3) ; pop4 <- get_mixture_data(mix4)
admix_cluster(samples = list(pop1, pop2, pop3, pop4),
              admixMod = list(admixMod1, admixMod2, admixMod3, admixMod4),
              conf_level = 0.95, tune_penalty = TRUE, echo = FALSE,
              n_sim_tab = 100, parallel = TRUE, n_cpu = 8)

#> Call:
#> admix_cluster(samples = list(pop1, pop2, pop3, pop4), admixMod = list(admixMod1,
#>     admixMod2, admixMod3, admixMod4), conf_level = 0.95, tune_penalty = TRUE,
#>     echo = FALSE, n_sim_tab = 100, parallel = TRUE, n_cpu = 8)
#>
#> Number of detected clusters: 2
#> Samples involved in each cluster:
#>   - Cluster #1: pop1, pop3
#>   - Cluster #2: pop2, pop4
```

```
# Clustering with the multinomial admixtures:
pop5 <- get_mixture_data(mix5) ; pop6 <- get_mixture_data(mix6)
pop7 <- get_mixture_data(mix7) ; pop8 <- get_mixture_data(mix8)
admixMod5 <- admix_model(knownComp_dist = mix5$comp.dist[[2]],
                         knownComp_param = mix5$comp.param[[2]])
admixMod6 <- admix_model(knownComp_dist = mix6$comp.dist[[2]],
                         knownComp_param = mix6$comp.param[[2]])
admixMod7 <- admix_model(knownComp_dist = mix7$comp.dist[[2]],
                         knownComp_param = mix7$comp.param[[2]])
admixMod8 <- admix_model(knownComp_dist = mix8$comp.dist[[2]],
                         knownComp_param = mix8$comp.param[[2]])
admix_cluster(samples = list(pop5, pop6, pop7, pop8),
              admixMod = list(admixMod5, admixMod6, admixMod7, admixMod8),
              conf_level = 0.95, tune_penalty = TRUE, echo = FALSE,
              n_sim_tab = 100, parallel = TRUE, n_cpu = 8)

#> Call:
#> admix_cluster(samples = list(pop5, pop6, pop7, pop8), admixMod = list(admixMod5,
#>     admixMod6, admixMod7, admixMod8), conf_level = 0.95, tune_penalty = TRUE,
#>     echo = FALSE, n_sim_tab = 100, parallel = TRUE, n_cpu = 8)
#>
#> Number of detected clusters: 2
#> Samples involved in each cluster:
#>   - Cluster #1: pop5, pop6, pop8
#>   - Cluster #2: pop7
```

As expected, in the case of continuous random variables, the populations 1 and 3 were clustered together, as well as populations 2 and 4. When considering multinomial mixtures (i.e. in the discrete case), two unbalanced clusters were detected, which corresponds to reality.

## 5 Edge cases

In the sequel, we study the behaviour of the estimators of the unknown mixture proportions under limit cases; consisting in (i) low proportion of the unknown component distribution, (ii) mixture components that overlap, and (iii) an admix model whose known component is misspecified. We set a fixed sample size $n = 1000$, and consider 100 Monte Carlo experiments.

### 5.1 Low proportion of the unknown component and component overlap

We study the impact of a low mixing weight for the unknown component distribution, considering Gaussian mixtures whose unknown proportion ranges from $p = 0.02$ to $p = 0.5$. The known component is centred at 2 with standard deviation 0.5, whereas the unknown one is the standard Gaussian $\mathcal{N}(0,1)$. In the left panel of fig.6), we display the distributions of the relative absolute errors $|\hat{p} - p| \,/\, p$ for the three estimators across these settings. The results reveal a clear and expected pattern: the larger the target proportion $p$, the better the estimation quality, regardless of the method used. This is intuitive, as a larger proportion of the unknown component implies more observations available to characterise it. Among the three estimators, *PS* consistently shows the smallest variability and is least affected by boundary effects, always returning realistic estimates within $(0,1)$. By contrast, *BVdk* and *IBM* can produce unrealistic values — particularly at very low proportions such as $p = 0.02$, where *IBM* yields a minimum of -2.83 and *BVdk* a mean of 0.22 driven by extreme values, as confirmed by the descriptive statistics in Table 10. At $p = 0.5$, all three estimators perform comparably well, with *PS* still showing the lowest standard deviation (0.02 versus 0.09 for *BVdk* and 0.03 for *IBM*).

We now examine the effect of component overlap, which is likely to induce weak identifiability. We consider Gaussian mixtures with fixed unknown proportion $p = 0.5$, where the known component is $\mathcal{N}(1,1)$, and the unknown one is $\mathcal{N}(\mu,1)$ with $\mu \in \{5, 0, 0.9\}$, corresponding respectively to weak overlap, strong overlap, and huge overlap situations. Figure 6 (right panel) shows the distributions of absolute errors $|\hat{p} - p|$ across these three regimes.

As expected, increasing overlap significantly degrades estimation quality for all methods. In the weak overlap case, *IBM* and *PS* yield comparable and reliable results, with standard deviations of 0.02, while *BVdk* shows higher variability (SD = 0.24), likely due to the limited sample size. Under strong overlap, all estimators deteriorate, but *PS* remains the most stable (SD = 0.04 versus 0.22 for *BVdk* and 0.06 for *IBM*). In the huge overlap case, both *BVdk* and *IBM* become essentially unreliable —

**Table 10:** Effect of low unknown mixture proportion and component overlap on estimator quality.

| | Unknown component weight | | | | | | | | | Overlap context | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | p=0.02 | | | p=0.10 | | | p=0.50 | | | Weak overlap | | | Strong overlap | | | Huge overlap | | |
| | BVdk | PS | IBM | BVdk | PS | IBM | BVdk | PS | IBM | BVdk | PS | IBM | BVdk | PS | IBM | BVdk | PS | IBM |
| Min. | 0.01 | 0.00 | -2.83 | 0.08 | 0.07 | 0.08 | 0.44 | 0.40 | 0.43 | 0.46 | 0.45 | 0.45 | 0.33 | 0.29 | 0.38 | 0.02 | 0.00 | 0.32 |
| Q1 | 0.01 | 0.02 | 0.04 | 0.11 | 0.09 | 0.10 | 0.50 | 0.44 | 0.48 | 0.50 | 0.48 | 0.49 | 0.43 | 0.32 | 0.47 | 0.10 | 0.03 | 0.73 |
| Med. | 0.02 | 0.03 | 0.06 | 0.14 | 0.10 | 0.11 | 0.52 | 0.46 | 0.50 | 0.52 | 0.49 | 0.50 | 0.54 | 0.37 | 0.51 | 0.99 | 0.04 | 1.16 |
| Mean | 0.22 | 0.04 | -0.01 | 0.13 | 0.10 | 0.11 | 0.56 | 0.46 | 0.50 | 0.68 | 0.49 | 0.50 | 0.61 | 0.36 | 0.52 | 0.65 | 0.05 | 1.85 |
| Q3 | 0.03 | 0.05 | 0.07 | 0.16 | 0.12 | 0.12 | 0.65 | 0.48 | 0.52 | 0.97 | 0.51 | 0.52 | 0.75 | 0.39 | 0.57 | 0.99 | 0.07 | 2.38 |
| Max | 0.99 | 0.11 | 5.00 | 0.23 | 0.19 | 0.14 | 0.78 | 0.52 | 0.56 | 0.99 | 0.52 | 0.54 | 0.99 | 0.48 | 0.65 | 0.99 | 0.12 | 5.00 |
| SD | 0.39 | 0.02 | 0.98 | 0.04 | 0.02 | 0.01 | 0.09 | 0.02 | 0.03 | 0.24 | 0.02 | 0.02 | 0.22 | 0.04 | 0.06 | 0.45 | 0.03 | 1.58 |

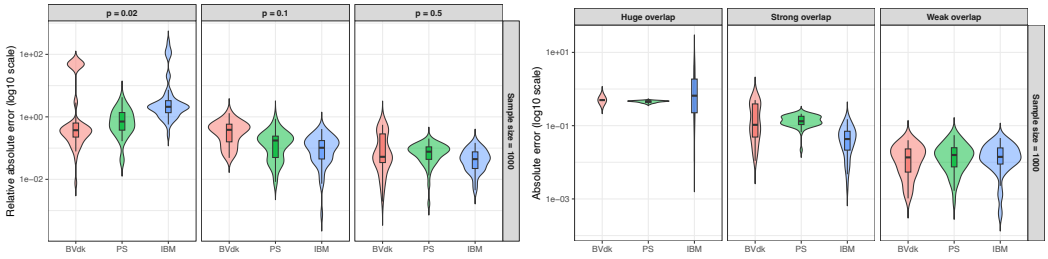



**Figure 6:** Violin plots of estimation errors of $p$: relative absolute error in the case of low proportion of the unknown component (left panel), absolute error in the case of component overlap (right panel).

*BVdk* systematically hits the boundary at 0.99, and *IBM* produces estimates well above 1 — while *PS*, although biased towards low values (median 0.04), at least remains bounded and avoids catastrophic failures. These results are summarised in Table 10. Overall, in cases of heavy overlap, no estimator can be fully trusted, and the practitioner should be aware that identifiability may be compromised.

### 5.2 Misspecification of the known component distribution

We finally investigate the robustness of the estimators when the known component $g$ is incorrectly specified. We simulate a two-component Gaussian mixture with unknown component $\mathcal{N}(0,1)$ and true known component $\mathcal{N}(2,0.5)$. The model is then fitted assuming the known component is $\mathcal{N}(\mu,0.5)$ with $\mu \in \{-2,1,1.9,2\}$, spanning from strong misspecification to a well-specified model. Figure 7 displays the absolute errors $|\hat{p}-p|$ for each configuration, and Table 11 reports the associated descriptive statistics.

The impact of misspecification is severe and monotone: the further $\mu$ is from the true value 2, the larger the estimation errors, regardless of the method. In the well-specified case ($\mu = 2$), all three estimators perform well, with *PS* and *IBM* showing the smallest standard deviations (0.026 each). As misspecification increases, performance degrades sharply. Under moderate misspecification ($\mu = 1$), *IBM* already produces estimates far above 1 (mean 2.490, max 4.895), making it completely unreliable. *BVdk* also suffers, frequently hitting boundary values. *PS* degrades more gracefully, with estimates remaining in a plausible range (mean 0.686, SD 0.020), though still biased. Under strong misspecification ($\mu = -2$), all estimators fail: *IBM* systematically saturates at 5.0, BVdk concentrates near the upper boundary, and even *PS* yields estimates around 0.92 with negligible variance – a sign that it has converged to a wrong fixed point. Among all edge cases considered, misspecification of the known component appears to be the most damaging scenario, underscoring the importance of a careful preliminary specification of $g$ before applying any of the estimation procedures.

**Table 11:** Effect of model misspecification on estimation error related to the proportion $p$, for the three estimators.

| | Well specified | | | Almost well specified | | | Moderate misspecification | | | Strong misspecification | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BVdk | PS | IBM | BVdk | PS | IBM | BVdk | PS | IBM | BVdk | PS | IBM |
| Min. | 0.45 | 0.41 | 0.45 | 0.41 | 0.40 | 0.43 | 0.20 | 0.64 | 0.97 | 0.84 | 0.91 | 1.46 |
| Q1 | 0.50 | 0.44 | 0.48 | 0.44 | 0.42 | 0.46 | 0.22 | 0.67 | 1.99 | 0.89 | 0.92 | 5.00 |
| Med. | 0.53 | 0.46 | 0.50 | 0.45 | 0.43 | 0.47 | 0.99 | 0.69 | 2.24 | 0.90 | 0.92 | 5.00 |
| Mean | 0.56 | 0.46 | 0.50 | 0.50 | 0.44 | 0.48 | 0.74 | 0.69 | 2.49 | 0.91 | 0.92 | 4.66 |
| Q3 | 0.59 | 0.49 | 0.52 | 0.48 | 0.45 | 0.49 | 0.99 | 0.70 | 3.00 | 0.93 | 0.93 | 5.00 |
| Max | 0.80 | 0.52 | 0.57 | 0.78 | 0.49 | 0.52 | 0.99 | 0.73 | 4.90 | 0.99 | 0.93 | 5.00 |
| SD | 0.09 | 0.03 | 0.03 | 0.11 | 0.02 | 0.02 | 0.37 | 0.02 | 0.81 | 0.03 | 0.01 | 1.01 |

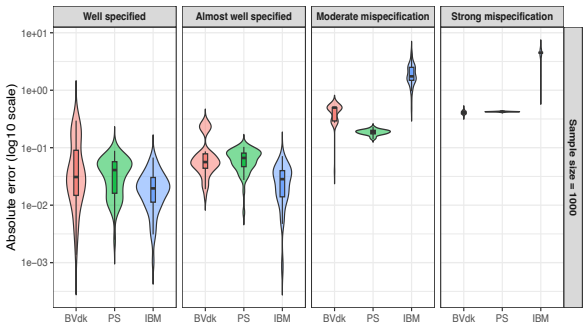



**Figure 7:** Violin plots of the absolute error for the estimation of $p$ for various mispecifications.

## 6 Acknowledgments

We thank the users of the package for their constructive comments and reported bugs. The package is more and more downloaded, shared and used. This work was conducted within the Research Chair DIALog under the aegis of the Risk Foundation, an initiative by CNP Assurances. D. Pommeret, Y. Salhi and P. Vandekerkhove would also like to acknowledge the support received from the Research Chair ACTIONS under the aegis of the Risk Foundation, an initiative by BNP Paribas Cardif and the French Institute of Actuaries.

## References

T. Benaglia, D. Chauveau, D. R. Hunter, and D. Young. mixtools: An r package for analyzing finite mixture models. *Journal of Statistical Software*, 32(6):1–29, 2009. URL https://www.jstatsoft.org/v32/i06/. [p243]

L. Bordes and P. Vandekerkhove. Semiparametric two-component mixture model with a known component: A class of asymptotically normal estimators. *Mathematical Methods of Statistics*, 19:22–41, 2010. [p242, 244, 245]

L. Bordes, C. Delmas, and P. Vandekerkhove. Semiparametric estimation of a two-component mixture model when a component is known. *Scandinavian Journal of Statistics*, 33:733–752, 2006a. [p244]

L. Bordes, S. Mottelet, and P. Vandekerkhove. Semiparametric estimation of a two components mixture model. *The Annals of Statistics*, 34:1204–1232, 2006b. [p244]

C. Butucea, R. N. Tzoumpe, and P. Vandekerkhove. Semiparametric topographical mixture models with symmetric errors. *Bernoulli*, 23:825–862, 2017. [p244]

T. Cai and J. Jin. Optimal rates of convergence for estimating the null density and proportion of nonnull effects in large-scale multiple testing. *The Annals of Statistics*, 38:100–145, 2010. [p242]

A. Celisse and S. Robin. A cross-validation based estimation of the proportion of true null hypotheses. *Journal of Statistical Planning and Inference*, 140:3132–3147, 2010. doi: 10.1016/j.jspi.2010.04.014. [p242]

I. Hedenfalk, D. Duggan, and Y. Chen. Gene-expression profiles in hereditary breast cancer. *N Engl J Med*, 344(8):539–48, 2001. [p242]

S. Kang, X. Shen, W. Yao, S. Xiang, and Y. Ge. *MixSemiRob: Mixture Models: Parametric, Semiparametric, and Robust*, 2023. URL https://CRAN.R-project.org/package=MixSemiRob. R package version 1.1.0. [p243]

K. Leppälä, S. V. Nielsen, and T. Mailund. admixturegraph: an r package for admixture graph manipulation and fitting. *Bioinformatics*, 33:1738–1740, 2017. [p243]

P.-R. Loh, M. Lipson, N. Patterson, P. Moorjani, J. K. Pickrell, D. Reich, and B. Berger. Inferring admixture histories of human populations using linkage disequilibrium. *Genetics*, 193:1233–1254, 2013. [p242]

G. McLachlan, R. Bean, and L. B.-T. Jones. A simple implementation of a normal mixture approach to differential gene expression in multiclass microarrays. *Bioinformatics*, 22(13):1608–1615, 04 2006. ISSN 1367-4803. doi: 10.1093/bioinformatics/btl148. URL https://doi.org/10.1093/bioinformatics/btl148. [p242]

X. Milhaud. *admix: Package Admix for Admixture (aka Contamination) Models*, 2026. URL https://CRAN.R-project.org/package=admix. R package version 2.5.2. [p243]

X. Milhaud, D. Pommeret, Y. Salhi, and P. Vandekerkhove. Semiparametric two-sample mixture components comparison test. *Journal of Statistical Planning and Inference*, 216:135–150, 2022. [p247]

X. Milhaud, D. Pommeret, Y. Salhi, and P. Vandekerkhove. Two-sample contamination model test. *Bernoulli*, 30(1):170–197, 2024a. doi: 10.3150/23-BEJ1593. [p244, 245, 246, 248, 256]

X. Milhaud, D. Pommeret, Y. Salhi, and P. Vandekerkhove. Contamination-source based k-sample clustering. *Journal of Machine Learning Research*, 25:1–32, 2024b. [p248, 249, 251, 262]

R. K. Patra and B. Sen. Estimation of a two-component mixture model with applications to multiple testing. *Journal of the Royal Statistical Society, Series B*, 78:869–893, 2016. [p243, 244]

M. Petr. *admixr: An Interface for Running 'ADMIXTOOLS' Analyses*, 2020. URL https://CRAN.R-project.org/package=admixr. R package version 0.9.1. [p243]

D. Pommeret and P. Vandekerkhove. Semiparametric density testing in the contamination model. *Electronic Journal of Statistics*, 13:4743–4793, 2019. [p242, 246, 247]

A. Punzo, A. Mazza, and P. D. McNicholas. Contaminatedmixt: An r package for fitting parsimonious mixtures of multivariate contaminated normal distributions. *Journal of Statistical Software*, 85(10): 1–25, 2018. doi: 10.18637/jss.v085.i10. [p243]

G. Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461 – 464, 1978. doi: 10.1214/aos/1176344136. URL https://doi.org/10.1214/aos/1176344136. [p249]

L. Scrucca, C. Fraley, T. B. Murphy, and A. E. Raftery. *Model-Based Clustering, Classification, and Density Estimation Using mclust in R*. Chapman and Hall/CRC, 2023. ISBN 978-1032234953. doi: 10.1201/9781003277965. URL https://mclust-org.github.io/book/. [p243]

*Xavier Milhaud*
*Aix-Marseille University*
*Department of Statistics*
*3, Pl. Victor Hugo, 13331, Marseille, France*
https://www.xaviermilhaud.fr/en
*ORCID: 0000-0002-3962-9434*
xavier.milhaud@univ-amu.fr

*Denys Pommeret*
*Aix-Marseille University*
*Department of Statistics*
*3, Pl. Victor Hugo, 13331, Marseille, France*
https://www.i2m.univ-amu.fr/perso/denys.pommeret/start
denys.pommeret@univ-amu.fr

*Yahia Salhi*
*University Lyon 1*
*Institut de Science Financière et d'Assurances*
*50, Avenue Tony Garnier, 69007, Lyon, France*
http://salhi.yahia.free.fr
yahia.salhi@univ-lyon1.fr

*Pierre Vandekerkhove*
*University Gustave Eiffel*
*5, Boulevard Descartes, 77420 Champs-sur-Marne, France*
https://perso.math.u-pem.fr/vandekerkhove.pierre/
pierre.vandekerkhove@univ-eiffel.fr

# VARshrink: An R Package for Shrinkage Estimation of High-Dimensional Vector Autoregressive Models

*by Namgil Lee and Sung-Ho Kim*

**Abstract** Vector autoregressive (VAR) models are widely used to study dynamic relationships in multivariate time series, with applications ranging from economic and financial forecasting to gene network analysis in systems biology and brain connectivity analysis in neuroscience. High-dimensional VAR models involve many parameters, increasing computational costs and making coefficient estimation more challenging. Shrinkage methods address these challenges by regularizing the coefficient estimates and improving their stability. We present the R package VARshrink, which provides ridge, nonparametric, fully Bayesian, and semiparametric shrinkage estimators through a unified interface. As its main design feature, VARshrink returns objects that inherit from classes in the widely used vars package, allowing existing methods for analysis, visualization, and forecasting to be reused. The package also supports model comparison using information criteria adapted to shrinkage estimation. We illustrate the main workflow with reproducible R code and compare the implemented estimators in numerical experiments.

## 1 Introduction

Vector autoregressive (VAR) models describe the joint dynamics of multiple time series through their lagged values. They are widely used for macroeconomic forecasting and policy analysis, financial modeling, the inference of gene regulatory networks from time-course data in systems biology, and the analysis of dynamic brain connectivity in neuroscience (Hamilton, 1994; Koop and Korobilis, 2010; Opgen-Rhein and Strimmer, 2007; Harrison et al., 2003). However, the number of parameters increases rapidly with both the number of variables and the lag order. High-dimensional VAR models can therefore be computationally demanding and difficult to estimate reliably. Shrinkage methods address these challenges by regularizing coefficient estimates, thereby improving estimation stability and potentially forecasting performance.

Several **R** packages provide methods for VAR and related multivariate time-series models. The **MTS** package provides tools for VAR, VARMA, seasonal VARMA, and other multivariate time-series models (Tsay and Wood, 2018), while **vars** supports estimation, inference, forecasting, and diagnostic analysis for VAR, SVAR, and SVEC models (Pfaff and Stigler, 2018). For large systems, **BigVAR** estimates VAR and VARX models using structured penalties (Nicholson et al., 2026), whereas **bigtime** estimates sparse VAR, VARX, and VARMA models using structured lasso penalties (Wilms et al., 2023).

Several packages also support Bayesian VAR analysis. **bvartools** assists in setting up Bayesian inference for VAR and vector error-correction models (Mohr, 2024). **bayesianVARs** implements MCMC estimation of Bayesian VARs with shrinkage priors and stochastic volatility (Gruber et al., 2026), while **bvarsv** implements time-varying parameter VARs with stochastic volatility (Primiceri, 2005; Krueger, 2015). **BVAR** implements hierarchical prior selection for conjugate priors following Giannone et al. (2015; Kuschnig and Vashold, 2021). **bsvarSIGNs** uses Minnesota and dummy-observation priors with estimated hyperparameters in structural VARs identified by sign, zero, and narrative restrictions (Wang and Woźniak, 2025).

In this study, we focus on shrinkage estimation of VAR model parameters. Shrinkage estimation methods have played crucial roles in high-dimensional statistical modeling (Ledoit and Wolf, 2004; Böhm and von Sachs, 2009; Fiecas and Ombao, 2011; Beltrachini et al., 2013). For VAR models, an extensive literature has developed on regularization and shrinkage priors, including Minnesota-type priors and more recent global-local shrinkage priors. Frequentist penalized estimation has also been widely studied. Approaches directly related to this package include a Stein-type nonparametric shrinkage estimation method (Opgen-Rhein and Strimmer, 2007), Bayesian VARs using informative priors (Doan et al., 1984; Litterman, 1986; Bańbura et al., 2010; Koop and Korobilis, 2010), Bayesian VARs using noninformative priors (Sun and Ni, 2004; Ni and Sun, 2005), and a semiparametric Bayesian approach adopting a modified $K$-fold cross-validation (Lee et al., 2016).

Shrinkage methods developed for purposes other than multivariate time series analysis can also be applied to the estimation of VAR parameters. For instance, the function `cov.shrink()` in the package **corpcor** was developed to compute shrinkage estimates of covariances, but it has also been used to estimate VAR coefficients (Opgen-Rhein and Strimmer, 2007; Schäfer et al., 2017). Moreover, VAR models can be reformulated as multivariate regression problems, allowing penalized least squares

methods to be used for shrinkage estimation of VAR parameters. Examples include the functions `lm.gls()` for generalized least squares and `lm.ridge()` for ridge regression in the package **MASS** (Ripley et al., 2018); the function `glmnet()` for Lasso and Elastic-Net regularized generalized linear models in the package **glmnet** (Friedman et al., 2018); the function `linearRidge()` for ridge regression in the package **ridge** (Moritz and Cule, 2018).

While Bayesian approaches have been widely used in the literature, nonparametric and semiparametric approaches have advantages for high-dimensional VARs with several hundred or more time series variables due to their relatively low computational cost (Opgen-Rhein and Strimmer, 2007). In contrast, Bayesian approaches can flexibly impose proper assumptions on multivariate time series data, such as VAR roots near unity and correlations among noise processes (Lee et al., 2016). In this sense, a semiparametric approach serves as a trade-off between nonparametric and parametric approaches (Lee et al., 2016).

The main design contribution of **VARshrink** is to extend the workflow of **vars** from ordinary least squares to shrinkage estimation. Users can apply familiar methods from **vars**, including ARCH-LM tests, causality analysis, diagnostic plots, forecasting, impulse response analysis, and forecast error variance decomposition, directly to **R** objects obtained via shrinkage estimation methods in **VARshrink**. This compatibility also allows shrinkage methods to be introduced within established applied and teaching workflows based on **vars**.

Relative to existing implementations, **VARshrink** brings together ridge regression, a Stein-type nonparametric estimator, fully Bayesian estimation, and semiparametric Bayesian estimation in one interface. Its fully Bayesian method includes both a natural conjugate normal-Wishart prior and a non-conjugate shrinkage-reference prior, with scale-mixture likelihoods available for robust estimation. The package also provides likelihood and information criterion methods that account for the selected noise distribution and the effective number of parameters. These features are intended to complement packages specializing in structured penalties, hierarchical Bayesian prior selection, stochastic volatility, or structural identification.

This paper is organized as follows. In Section 2, we formulate VAR models as a multivariate regression problem, which simplifies the implementation of the package. We also describe the general distributional assumptions for the noise term within a Bayesian framework. Closed-form expressions for the shrinkage estimators implemented in the package are presented to clarify the role of the shrinkage intensity parameters in each method. In addition, we explain how the effective number of parameters is computed for shrinkage estimators. Section 3 introduces the common interface function and provides sample code snippets to demonstrate the usefulness of the package. In Section 4, we present numerical experiments using benchmark and simulated data to compare the performance of the shrinkage estimation methods. Discussion and conclusions are given in Sections 5 and 6.

## 2 Models

### 2.1 Multivariate linear regression formulation

Let $\mathbf{y}_t = (y_{t1}, y_{t2}, \ldots, y_{tK})^\top$ denote a $K \times 1$ vector of endogenous variables. A VAR model of order $p$ can be expressed as

$$\mathbf{y}_t = \mathbf{A}_1 \mathbf{y}_{t-1} + \cdots + \mathbf{A}_p \mathbf{y}_{t-p} + \mathbf{C}\mathbf{d}_t + \boldsymbol{\epsilon}_t, \tag{1}$$

where $\mathbf{d}_t$ is an $L \times 1$ vector of deterministic regressors, $\boldsymbol{\epsilon}_t$ is a $K \times 1$ noise vector, and $\mathbf{A}_1, \ldots, \mathbf{A}_p$ and $\mathbf{C}$ are coefficient matrices (Hamilton, 1994; Tsay, 2005).

The model equation (1) can be expressed in the matrix form of a multivariate regression as

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\Psi} + \mathbf{E} \in \mathbb{R}^{N \times K}, \tag{2}$$

where $\boldsymbol{\Psi} = (\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_p, \mathbf{C})^\top$ is a $(Kp + L) \times K$ matrix of coefficients,

$$\begin{aligned} \mathbf{Y} &= (\mathbf{y}_{p+1}, \mathbf{y}_{p+2}, \ldots, \mathbf{y}_T)^\top \in \mathbb{R}^{N \times K}, \\ \mathbf{X} &= (\mathbf{x}_{p+1}, \mathbf{x}_{p+2}, \ldots, \mathbf{x}_T)^\top \in \mathbb{R}^{N \times (Kp+L)} \end{aligned} \tag{3}$$

are data matrices with $\mathbf{x}_t = (\mathbf{y}_{t-1}^\top, \ldots, \mathbf{y}_{t-p}^\top, \mathbf{d}_t^\top)^\top \in \mathbb{R}^{Kp+L}$, $\mathbf{E} = (\boldsymbol{\epsilon}_{p+1}, \ldots, \boldsymbol{\epsilon}_T)^\top$, and $N = T - p$.

### 2.2 Mathematical expressions for shrinkage estimators

In version 0.5 of the **R** package **VARshrink**, ordinary least squares (OLS) and five shrinkage estimation methods are implemented: multivariate ridge regression (Hoerl and Kennard, 1970; Golub et al.,

1979; Ripley et al., 2018), a Stein-type nonparametric shrinkage method (Schäfer and Strimmer, 2005; Opgen-Rhein and Strimmer, 2007; Schäfer et al., 2017), a full Bayesian shrinkage method (Sun and Ni, 2004; Ni and Sun, 2005), and two semiparametric Bayesian shrinkage methods based on parameterized and K-fold cross-validation, respectively (Lee et al., 2016). OLS is included as a baseline for settings in which the VAR is estimable without shrinkage. For each shrinkage estimation method, the shrinkage estimator of the coefficient matrix $\mathbf{\Psi}$ takes a distinct mathematical form.

### (1) Multivariate ridge regression

The ridge regression estimator, $\widehat{\mathbf{\Psi}}^{\mathrm{R}}(\lambda)$, can be expressed in a closed form as follows (Hoerl and Kennard, 1970):

$$\widehat{\mathbf{\Psi}}^{\mathrm{R}}(\lambda) = \left(\mathbf{X}^\top \mathbf{X} + N\lambda \mathbf{I}\right)^{-1} \mathbf{X}^\top \mathbf{Y}, \tag{4}$$

where $\lambda \geq 0$ is called the regularization parameter or the shrinkage parameter.

### (2) Stein-type nonparametric shrinkage method

Suppose that the VAR model does not include a constant term, that is, a regressor whose value remains constant over time (e.g., $d_t = 1$). Assuming that the data matrices $\mathbf{X}$ and $\mathbf{Y}$ have been mean-corrected, let $\mathbf{S}_{\mathrm{XX}} = (N-1)^{-1}\mathbf{X}^\top\mathbf{X}$, $\mathbf{S}_{\mathrm{XY}} = (N-1)^{-1}\mathbf{X}^\top\mathbf{Y}$, and $\mathbf{S}_{\mathrm{YY}} = (N-1)^{-1}\mathbf{Y}^\top\mathbf{Y}$ denote the sample covariance matrices, $\mathbf{D}_{\mathrm{X}} = \mathrm{diag}(\mathbf{S}_{\mathrm{XX}})$ and $\mathbf{D}_{\mathrm{Y}} = \mathrm{diag}(\mathbf{S}_{\mathrm{YY}})$ denote the diagonal matrices of sample variances, and $\mathbf{R}_{\mathrm{XX}}$, $\mathbf{R}_{\mathrm{XY}}$, and $\mathbf{R}_{\mathrm{YY}}$ denote the sample correlation matrices. The shrinkage estimators of the correlation matrices are defined as:

$$\widehat{\mathbf{R}}_{\mathrm{XX}} = (1-\lambda)\mathbf{R}_{\mathrm{XX}} + \lambda\mathbf{I}, \qquad \widehat{\mathbf{R}}_{\mathrm{XY}} = (1-\lambda)\mathbf{R}_{\mathrm{XY}}, \tag{5}$$

where $0 \leq \lambda \leq 1$ is the shrinkage parameter for the correlations. And the shrinkage estimators of the variance matrices are defined as:

$$\widehat{\mathbf{D}}_{\mathrm{X}} = (1-\lambda_v)\mathbf{D}_{\mathrm{X}} + \lambda_v s_{\mathrm{med}}\mathbf{I}, \;\; \widehat{\mathbf{D}}_{\mathrm{Y}} = (1-\lambda_v)\mathbf{D}_{\mathrm{Y}} + \lambda_v s_{\mathrm{med}}\mathbf{I}, \tag{6}$$

where $0 \leq \lambda_v \leq 1$ is the shrinkage parameter for the variances, and $s_{\mathrm{med}}$ is the median of all sample variances.

The nonparametric shrinkage (NS) estimator of $\mathbf{\Psi}$ can be expressed as follows (Schäfer and Strimmer, 2005; Opgen-Rhein and Strimmer, 2007):

$$\widehat{\mathbf{\Psi}}^{\mathrm{N}}(\lambda, \lambda_v) = \widehat{\mathbf{D}}_{\mathrm{X}}^{-1/2}\widehat{\mathbf{R}}_{\mathrm{XX}}^{-1}\widehat{\mathbf{R}}_{\mathrm{XY}}\widehat{\mathbf{D}}_{\mathrm{Y}}^{1/2}. \tag{7}$$

The shrinkage estimate $\widehat{\mathbf{\Psi}}^{\mathrm{N}}(\lambda, \lambda_v)$ approaches the ordinary least squares estimate, $\widehat{\mathbf{\Psi}}^{\mathrm{OLS}} = \mathbf{S}_{\mathrm{XX}}^{-1}\mathbf{S}_{\mathrm{XY}}$, as $\lambda$ and $\lambda_v$ approach zero.

### (3) Full Bayesian shrinkage method

The full Bayesian shrinkage method supports several likelihood and prior specifications.

**Likelihoods** We assume that the noise vectors are independent and identically distributed from a scale mixture of multivariate normal distributions, including multivariate normal distributions, multivariate t-distributions (Ni and Sun, 2005), and multivariate Laplace distributions (MLD) (Eltoft et al., 2006). By incorporating scale mixture distributions in the model assumption, we can handle outliers appropriately and produce robust estimates (West, 1984). In specific, we consider that a noise vector can be expressed as a product

$$\boldsymbol{\epsilon}_t = \mathbf{z}_t q_t^{-1/2}, \tag{8}$$

where $\mathbf{z}_t \sim \mathrm{N}_K(\mathbf{0}, \mathbf{\Sigma})$ is a $K \times 1$ vector having a multivariate normal distribution with the mean vector of zeros and the covariance matrix of $\mathbf{\Sigma}$, and $q_t \sim \mathrm{Gamma}(\nu/2, \nu/2)$ is a random variable having a gamma distribution with shape $\alpha = \nu/2$ and rate $\beta = \nu/2$. The $\boldsymbol{\epsilon}$ has a multivariate t-distribution with degrees of freedom $\nu$ when $1 < \nu < \infty$, a multivariate normal distribution as $\nu \to \infty$, and a multivariate Cauchy distribution when $\nu = 1$. More detailed discussions can be found in the package vignette and Lee et al. (2016).

**Conjugate prior** Let $\boldsymbol{\psi} = \mathrm{vec}(\mathbf{\Psi}) \in \mathbb{R}^J$ denote the vectorized form of $\mathbf{\Psi} \in \mathbb{R}^{(Kp+L)\times K}$, where $J = K(Kp+L)$. The natural conjugate prior implemented in **VARshrink** is a normal-Wishart prior.

Conditional on $\boldsymbol{\Sigma}$, the prior for $\boldsymbol{\psi}$ is

$$(\boldsymbol{\psi}|\boldsymbol{\Sigma}, \delta) \sim \mathrm{N}_J\left(\mathbf{0}, \delta\boldsymbol{\Sigma} \otimes \mathbf{I}\right), \tag{9}$$

where $\otimes$ represents the Kronecker product of two matrices and $\delta \equiv 1/\lambda > 0$ is a reciprocal of a shrinkage parameter $\lambda > 0$, and an inverse Wishart distribution with a positive definite scale matrix $\mathbf{L}_0$ and degrees of freedom $m_0 > K - 1$ for $\boldsymbol{\Sigma}$ as

$$(\boldsymbol{\Sigma}|\mathbf{L}_0, m_0) \sim \mathrm{InvWishart}\left(\mathbf{L}_0, m_0\right). \tag{10}$$

The prior distributions for $\delta$ and $w = \nu/2$ are given by

$$\begin{aligned} \pi(\delta) &\propto 1, \\ w &\sim \mathrm{Gamma}(a_0, b_0), \end{aligned} \tag{11}$$

for some $a_0, b_0 > 0$.

**VARshrink** uses a Gibbs MCMC method to sample $(\boldsymbol{\psi}, \mathbf{Q}, \boldsymbol{\Sigma}, w, \delta)$ from the conditional posterior distributions; see the package vignette for more details. Let $\mathbf{Q} = \mathrm{diag}(q_{p+1}, \ldots, q_T)$. With $\lambda \equiv 1/\delta$, the mean of the conditional posterior distribution of $\boldsymbol{\psi}$ given $(\mathbf{Q}, \boldsymbol{\Sigma}, w, \delta; \mathbf{Y})$ can be written as

$$\widehat{\boldsymbol{\Psi}}^{\mathrm{F}}_{\mathrm{CJ}}(\lambda) = \left(\mathbf{X}^\top\mathbf{Q}\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^\top\mathbf{Q}\mathbf{Y}. \tag{12}$$

**Non-conjugate priors** The non-conjugate prior adopted in **VARshrink** is the shrinkage-reference prior of Ni and Sun (2005), which is given by

$$\begin{aligned} (\boldsymbol{\psi}|\delta) &\sim \mathrm{N}_J\left(\mathbf{0}, \delta\mathbf{I}\right), \\ \pi(\boldsymbol{\Sigma}) &\propto |\boldsymbol{\Sigma}|^{-1}\prod_{1\le i<j\le K}(\lambda_i - \lambda_j)^{-1}, \\ \pi(\delta) &\propto 1, \\ w &\sim \mathrm{Gamma}(a_0, b_0), \end{aligned} \tag{13}$$

where $\lambda_1 > \lambda_2 > \cdots > \lambda_K$ are eigenvalues of $\boldsymbol{\Sigma}$, $w = \nu/2$, and $a_0, b_0 > 0$.

With $\lambda \equiv 1/\delta$, the mean of the conditional posterior distribution of $\boldsymbol{\psi}$ given $(\mathbf{Q}, \boldsymbol{\Sigma}, w, \delta; \mathbf{Y})$ is given by

$$\widehat{\boldsymbol{\psi}}^{\mathrm{F}}_{\mathrm{NCJ}}(\lambda) = \left(\boldsymbol{\Sigma}^{-1} \otimes \left(\mathbf{X}^\top\mathbf{Q}\mathbf{X}\right) + \lambda\mathbf{I}_J\right)^{-1}\mathrm{vec}\left(\mathbf{X}^\top\mathbf{Q}\mathbf{Y}\boldsymbol{\Sigma}^{-1}\right). \tag{14}$$

#### (4) Semi-parametric shrinkage method

In contrast to the full Bayesian approach, the semiparametric Bayesian approach selects the shrinkage parameter $0 \le \lambda < 1$ through parameterized cross-validation (Lee et al., 2016). Once the $\lambda$ is fixed, the parameters $\boldsymbol{\psi}$, $\boldsymbol{\Sigma}$, and $\mathbf{Q}$ are estimated at the mode of the marginal posterior density $\pi(\boldsymbol{\psi}, \boldsymbol{\Sigma}|\lambda; \mathbf{Y})$. In the case of a non-conjugate prior for $\boldsymbol{\psi}$ (Lee et al., 2016), the mode $\widehat{\boldsymbol{\psi}}^{\mathrm{S}}(\lambda)$ is given by

$$\widehat{\boldsymbol{\psi}}^{\mathrm{S}}(\lambda) = \left[\left(\boldsymbol{\Sigma}^{-1} \otimes \left(\mathbf{X}^\top\mathbf{Q}\mathbf{X}\right)\right) + \frac{(N-1)\lambda}{1-\lambda}\mathbf{I}_J\right]^{-1}\mathrm{vec}\left(\mathbf{X}^\top\mathbf{Q}\mathbf{Y}\boldsymbol{\Sigma}^{-1}\right). \tag{15}$$

The covariance matrix $\boldsymbol{\Sigma}$ is estimated in the same manner.

### 2.3 Estimation of the effective number of parameters

Although all shrinkage estimators are expressed in slightly different forms, they approach the ordinary least squares estimator (or the maximum likelihood estimator) as the shrinkage parameters decrease to zero, and they are shrunk toward zeros as the shrinkage parameters increase. It implies that the effective number of parameters underlying a shrinkage estimator is controlled by the shrinkage parameter values.

The shrinkage estimators presented in the previous section can be expressed in a general form:

$$\widehat{\boldsymbol{\psi}}(\lambda_0) = \left[\left(\boldsymbol{\Sigma}^{-1} \otimes \left(\mathbf{X}^\top\mathbf{Q}\mathbf{X}\right)\right) + \lambda_0\mathbf{I}_J\right]^{-1}\mathrm{vec}\left(\mathbf{X}^\top\mathbf{Q}\mathbf{Y}\boldsymbol{\Sigma}^{-1}\right), \tag{16}$$

where $\lambda_0 > 0$ is the shrinkage parameter. A noise covariance matrix is approximated by its diagonal

elements as $\mathbf{\Sigma} \approx \mathbf{D}_e \equiv \mathrm{diag}(\mathbf{\Sigma}) = \mathrm{diag}(\sigma_{11}, \ldots, \sigma_{KK})$ to simplify the expression for $\widehat{\boldsymbol{\psi}}(\lambda_0)$ as:

$$\widehat{\boldsymbol{\psi}}(\lambda_0) \approx \left[\left(\mathbf{I}_K \otimes \left(\mathbf{X}^\top \mathbf{Q} \mathbf{X}\right)\right) + \lambda_0 \mathbf{D}_e \otimes \mathbf{I}_{J/K}\right]^{-1} \mathrm{vec}\left(\mathbf{X}^\top \mathbf{Q} \mathbf{Y}\right),$$

or equivalently, for each $j = 1, 2, \ldots, K$,

$$\widehat{\boldsymbol{\psi}}_j(\lambda_0) \approx \left(\mathbf{X}^\top \mathbf{Q} \mathbf{X} + \lambda_0 \sigma_{jj} \mathbf{I}_{J/K}\right)^{-1} \mathbf{X}^\top \mathbf{Q} \mathbf{y}_j, \tag{17}$$

where $\widehat{\boldsymbol{\psi}}_j(\lambda_0)$ and $\mathbf{y}_j$ are the $j$th column vectors of $\widehat{\mathbf{\Psi}}(\lambda_0)$ and $\mathbf{Y}$.

From the definition of a hat matrix, $\mathbf{H}(j, \lambda_0)$, we have $\widehat{\mathbf{y}}_j \equiv \mathbf{H}(j, \lambda_0)\mathbf{y}_j = \mathbf{X}\widehat{\boldsymbol{\psi}}_j(\lambda_0)$, and the hat matrix can be expressed as: $\mathbf{H}(j, \lambda_0) \equiv \mathbf{X}\left(\mathbf{X}^\top \mathbf{Q} \mathbf{X} + \lambda_0 \sigma_{jj} \mathbf{I}_{J/K}\right)^{-1} \mathbf{X}^\top \mathbf{Q}$. Let $\mathbf{X}^\top \mathbf{Q} \mathbf{X} = \mathbf{P}\mathbf{L}\mathbf{P}^\top$ denote the eigenvalue decomposition of $\mathbf{X}^\top \mathbf{Q} \mathbf{X}$, where $\mathbf{L} = \mathrm{diag}(l_{11}, l_{22}, \ldots)$ is the diagonal matrix of eigenvalues. The effective number of parameters, $\kappa_{\mathrm{eff}}$, is computed by the trace of the hat matrix as:

$$\kappa_{\mathrm{eff}}(\lambda_0) = \sum_{j=1}^{K} \mathrm{tr}\left(\mathbf{H}(j, \lambda_0)\right) = \sum_{j=1}^{K} \mathrm{tr}\left(\left(\mathbf{L} + \lambda_0 \sigma_{jj} \mathbf{I}_{J/K}\right)^{-1} \mathbf{L}\right) = \sum_{j=1}^{K} \sum_{i=1}^{J/K} \frac{l_{ii}}{l_{ii} + \lambda_0 \sigma_{jj}}. \tag{18}$$

With the estimated value for the effective number of parameters, the AIC and BIC values can be calculated as:

$$\begin{aligned} \mathrm{AIC}(\lambda_0) &= -2\mathrm{logLik}(\lambda_0) + 2\kappa_{\mathrm{eff}}(\lambda_0), \\ \mathrm{BIC}(\lambda_0) &= -2\mathrm{logLik}(\lambda_0) + \kappa_{\mathrm{eff}}(\lambda_0) \log(N), \end{aligned} \tag{19}$$

where logLik represents the log-likelihood. We can use the AIC and BIC values to select an optimal value of the lag order, $p$.

## 3 Overview of the package

The **R** package **VARshrink** performs shrinkage estimation of the VAR model coefficients $\mathbf{\Psi}$ in (2) from multivariate time series data $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T \in \mathbb{R}^K$. It estimates the so-called shrinkage intensity parameter $\lambda > 0$, whose interpretation slightly differs across shrinkage estimation methods. Depending on the method, additional model parameters may also be estimated, such as the $K \times K$ covariance matrix $\mathbf{\Sigma}$ of the noise vector $\boldsymbol{\epsilon}_t \in \mathbb{R}^K$ in (8).

### 3.1 Package architecture

The software architecture of **VARshrink** is illustrated in Figure 1. The main interface function, `VARshrink()`, receives multivariate time series data in a $T \times K$ matrix form, and outputs the estimated parameters as an instance of the `varshrinkest` class. An instance of this class can then be used to produce useful time series analysis results through various class methods and analysis functions. Table 1 summarizes the structure of **VARshrink**, listing the available class methods and analysis functions.

The class `varshrinkest` inherits from the class `varest` in the **vars** package. Consequently, all class methods available for `varest` can also be applied to `varshrinkest`, such as `fevd()`, `Phi()`, and `plot()`. We also note that the classes `varshrinkest`, `varshirf`, and `varshsum` inherit from `varest`, `varirf`, and `varsum`, respectively, in the **vars** package.

The class method `irf.varshrinkest()` retains the interface and identification options of `vars::irf()`. For shrinkage estimators, confidence intervals are obtained via bootstrap procedures that require repeatedly refitting the model. In contrast, when the full Bayesian method is fitted with `store_mcmc = TRUE`, `irf()` can construct Bayesian credible intervals directly from the stored MCMC draws of the VAR coefficients, avoiding the computational cost of bootstrap-based interval estimation.

### 3.2 Main function

We provide a common **R** function interface `VARshrink()` for running the estimation methods, which is defined by

```
VARshrink(y, p = 1, type = c("const", "trend", "both", "none"),
   season = NULL, exogen = NULL, method = c("ridge", "ns",
   "fbayes", "sbayes", "kcv", "ols"), lambda = NULL, lambda_var = NULL,
```

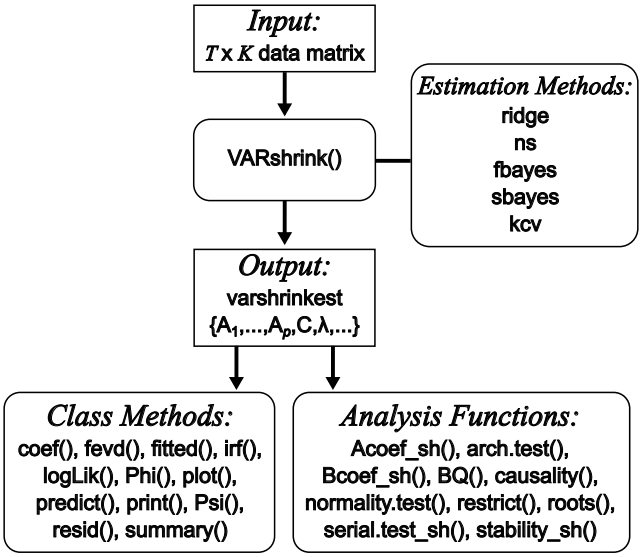



**Figure 1:** Systematic diagram of the VARshrink package.

**Table 1:** Structure of the VARshrink package.

| Function or method | Class | Class methods | Analysis functions for class |
|---|---|---|---|
| VARshrink | varshrinkest, varest | coef, fevd, fitted, irf, logLik, Phi, plot, predict, print, Psi, resid, summary | Acoef_sh, arch.test, Bcoef_sh, BQ, causality, normality.test, restrict, roots, serial.test_sh, stability_sh |
| fevd | varfevd | plot, print | |
| irf | varshirf, varirf | plot, print | |
| predict | varprd | plot, print | fanchart |
| summary | varshsum, varsum | print | |
| arch.test | varcheck | plot, print | |
| normality.test | varcheck | plot, print | |
| serial.test_sh | varcheck | plot, print | |
| stability_sh | varstabil | plot, print | |

```
dof = Inf, ...)
```

The input arguments are described as follows.

- y: A $T \times K$ matrix of endogenous variables
- p: Integer for the lag order
- type: Type of deterministic regressors to include.
  1. "const" - the constant: $\mathbf{d}_t = 1$.
  2. "trend" - the trend: $\mathbf{d}_t = t$.
  3. "both" - both the constant and the trend: $\mathbf{d}_t = (t, 1)^\top$.
  4. "none" - no deterministic regressors.
- season: An integer value of frequency for inclusion of centered seasonal dummy variables.
- exogen: A $T \times L$ matrix of exogenous variables.
- method:
  1. "ridge" - multivariate ridge regression (Hoerl and Kennard, 1970; Golub et al., 1979; Ripley et al., 2018).
  2. "ns" - a Stein-type nonparametric shrinkage method (Schäfer and Strimmer, 2005; Opgen-Rhein and Strimmer, 2007; Schäfer et al., 2017).
  3. "fbayes" - a full Bayesian shrinkage method using noninformative priors (Sun and Ni, 2004; Ni and Sun, 2005).
  4. "sbayes" - a semiparametric Bayesian shrinkage method using parameterized cross-validation (Lee et al., 2016).
  5. "kcv" - a semiparametric Bayesian shrinkage method using K-fold cross-validation (Lee et al., 2016).
  6. "ols" - ordinary least squares estimation, calling `vars::VAR()`.
- lambda, lambda_var: Shrinkage parameter value(s). Use of this parameter is slightly different for each method: the same value does not imply the same shrinkage estimates. See the description in the previous section for the use of shrinkage parameters in each method.
- dof: Degree of freedom of multivariate t-distribution for noise. Valid only for `method = "fbayes"` and `method = "sbayes"`. `dof = Inf` means multivariate normal distribution.
- ...: Additional method-specific options, including `store_mcmc = TRUE` for retaining MCMC draws from the full Bayesian method.

### 3.3 Sample code snippets analysis

Simulated time series data of length $T = 100$ were generated from a VAR model with order $p = 1$ and dimension $K = 2$, assuming multivariate normal noise. The model parameters were set as $\mathbf{A}_1 = 0.5\mathbf{I}_2$, $\mathbf{C} = (0.2, 0.7)^\top$, and $\mathbf{\Sigma} = 0.1^2\mathbf{I}_2$, as follows:

```
set.seed(1000)
myCoef <- list(A = list(matrix(c(0.5, 0, 0, 0.5), 2, 2)), c = c(0.2, 0.7))
myModel <- list(Coef = myCoef, Sigma = diag(0.1^2, 2), dof = Inf)
Y <- simVARmodel(numT = 100, model = myModel, burnin = 10)
resu_estim <- list()
```

The multivariate ridge regression for VAR models can be performed as follows. The output displays all the candidate lambda values along with their corresponding generalized cross-validation (GCV) scores (Golub et al., 1979). The VAR parameters are then estimated using the lambda value that yields the minimum GCV score.

```
resu_estim$`Ridge regression` <-
  VARshrink(Y, p = 1, type = "const", method = "ridge", lambda = NULL)
resu_estim$`Ridge regression`

#>
#> VAR Shrinkage Estimation Results:
#> ================================
#>
```

```
#> Estimated coefficients for equation y1:
#> ========================================
#> Call:
#> y1 = y1.l1 + y2.l1 + const
#>
#>      y1.l1      y2.l1      const
#> 0.61424699 0.07318362 0.05264740
#>
#>
#> Estimated coefficients for equation y2:
#> ========================================
#> Call:
#> y2 = y1.l1 + y2.l1 + const
#>
#>     y1.l1     y2.l1     const
#> 0.1195550 0.3254618 0.9055450
#>
#>
#> lambda: 5e-04 (estimated: TRUE)
#> GCV:  0.0186
```

The `summary()` method reports the estimated coefficient matrices, shrinkage parameters, noise covariance matrix, and method-specific fitting information.

After estimation, the fitted object can be used in a standard VAR workflow for diagnostic checking, forecasting, impulse response analysis, and forecast error variance decomposition. The returned objects have their own `plot()` methods.

```
fit_ridge <- resu_estim$`Ridge regression`

plot(fit_ridge)

diagnostics <- list(
  serial = serial.test_sh(fit_ridge),
  arch = arch.test(fit_ridge),
  normality = normality.test(fit_ridge),
  stability = stability_sh(fit_ridge)
)

plot(predict(fit_ridge, n.ahead = 5), names = "y1")

plot(irf(fit_ridge, impulse = "y1", response = "y2", n.ahead = 10,
         ortho = TRUE, boot = TRUE))

plot(fevd(fit_ridge, n.ahead = 10))
```

The class methods `AIC()` and `BIC()` can be used to compute the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC) as follows:

```
c(AIC = AIC(resu_estim$`Ridge regression`),
  BIC = BIC(resu_estim$`Ridge regression`))

#>       AIC       BIC
#> -366.2358 -351.5082
```

An OLS and the other shrinkage estimation methods ("ns", "fbayes", "sbayes", and "kcv") can be run similarly. For "ns", "sbayes", and "kcv", if the input arguments `lambda` and `lambda_var` are set to `NULL`, the shrinkage parameters $\lambda$ and $\lambda_v$ are determined automatically. For "fbayes", "sbayes", and "kcv", the degree of freedom $\nu$ of the multivariate $t$-distribution for noise can either be set to a fixed value or estimated automatically by specifying the argument `dof`. For "fbayes", "sbayes", and "kcv", `prior_type` can be set to either `"CJ"` or `"NCJ"` corresponding to conjugate and non-conjugate prior distributions, respectively.

```
resu_estim$`Ordinary least squares` <-
```

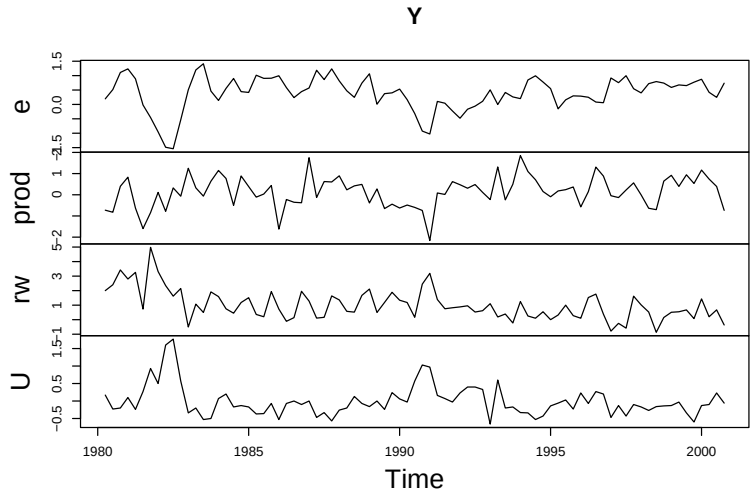



**Figure 2:** Benchmark dataset obtained by differencing the Canada time series from the vars package.

```
  VARshrink(Y, p = 1, type = "const", method = "ols")
resu_estim$`Nonparametric shrinkage` <-
  VARshrink(Y, p = 1, type = "none", method = "ns", lambda = NULL,
            lambda_var = NULL)
resu_estim$`Full Bayes (fixed dof)` <-
  VARshrink(Y, p = 1, type = "const", method = "fbayes", dof = 6,
            prior_type = "NCJ", burnincycle = 1000, mcmccycle = 2000,
            store_mcmc = TRUE)
resu_estim$`Full Bayes (estim dof)` <-
  VARshrink(Y, p = 1, type = "const", method = "fbayes", dof = NULL,
            prior_type = "NCJ", burnincycle = 1000, mcmccycle = 2000,
            store_mcmc = TRUE)
resu_estim$`Semi Bayes (fixed dof)` <-
  VARshrink(Y, p = 1, type = "const", method = "sbayes", dof = 6,
            prior_type = "NCJ", lambda = NULL, lambda_var = NULL, num_folds = 5)
resu_estim$`Semi Bayes (estim dof)` <-
  VARshrink(Y, p = 1, type = "const", method = "sbayes", dof = NULL,
            prior_type = "NCJ", lambda = NULL, lambda_var = NULL, num_folds = 5)
resu_estim$`K-fold CV (fixed dof)` <-
  VARshrink(Y, p = 1, type = "const", method = "kcv", dof = 6,
            prior_type = "NCJ", lambda = NULL, lambda_var = NULL, num_folds = 5)
resu_estim$`K-fold CV (estim dof)` <-
  VARshrink(Y, p = 1, type = "const", method = "kcv", dof = NULL,
            prior_type = "NCJ", lambda = NULL, lambda_var = NULL, num_folds = 5)
```

## 4 Illustrative examples

### 4.1 Benchmark data

The Canada dataset is a benchmark macroeconomic dataset included in the **vars** package. It contains four time series variables: employment (`e`), labor productivity (`prod`), real wage (`rw`), and unemployment rate (`U`), with 84 observations. We differenced the data to remove the trend, resulting in $T = 83$. Figure 2 shows the differenced series.

We illustrate the use of AIC and BIC for comparing VAR models with varying lag order $p = 1, 2, 3$. OLS is included as a baseline, shrinkage parameters were selected automatically for the shrinkage methods, and for "fbayes", "sbayes", and "kcv", the `dof` argument was set to `NULL`. Table 2 shows that, using the NS method, AIC attains its minimum at $p = 3$, whereas BIC attains its minimum at $p = 2$.

The parameters estimated by the NS method with $p = 2$ can be further analyzed using the class methods and functions listed in Table 1. For example, a test for serially correlated errors and an ARCH test can be performed as follows:

```
fitted_Canada <- VARshrink(Y, p = 2, type = "none", method = "ns")
```

**Table 2:** Information criteria (AIC and BIC) for model comparison.

| | AIC.p=1 | BIC.p=1 | AIC.p=2 | BIC.p=2 | AIC.p=3 | BIC.p=3 |
|---|---|---|---|---|---|---|
| Ordinary least squares | 467.6 | 515.7 | 444.2 | 530.4 | 452.3 | 576.2 |
| Ridge regression | 465.8 | 504.6 | 442.9 | 509.3 | 445.3 | 525.9 |
| Nonparametric shrinkage | 454.5 | 478.9 | 426.6 | 471.4 | 422.5 | 485.4 |
| Full Bayes | 457.0 | 497.2 | 444.2 | 508.2 | 445.3 | 523.8 |
| Semi Bayes | 533.2 | 577.5 | 574.1 | 656.1 | 475.0 | 553.4 |
| K-fold CV | 495.7 | 538.2 | 466.2 | 525.4 | 472.9 | 538.1 |

```
serial.test(fitted_Canada)

#>
#>  Portmanteau Test (asymptotic)
#>
#> data:  Residuals of VAR object fitted_Canada
#> Chi-squared = 176.31, df = 228, p-value = 0.9953

arch.test(fitted_Canada)

#>
#>  ARCH (multivariate)
#>
#> data:  Residuals of VAR object fitted_Canada
#> Chi-squared = 508.04, df = 500, p-value = 0.3921
```

We can inspect the summary results to assess the statistical significance of each covariate affecting the unemployment rate through the model coefficients:

```
summary_Canada <- summary(fitted_Canada)
print(summary_Canada$varresult[["U"]])

#>
#> Call:
#> VARshrink(y = Y, p = 2, type = "none", method = "ns")
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -0.84400 -0.16754 -0.00281  0.16971  0.72526
#>
#> Coefficients:
#>         Estimate Std. Error t value Pr(>|t|)
#> e.l1    -0.36506    0.03356 -10.877  < 2e-16 ***
#> prod.l1 -0.13332    0.03502  -3.807 0.000283 ***
#> rw.l1    0.03840    0.02940   1.306 0.195426
#> U.l1     0.07101    0.03193   2.224 0.029098 *
#> e.l2     0.01561    0.03265   0.478 0.633932
#> prod.l2 -0.04137    0.03533  -1.171 0.245312
#> rw.l2    0.09894    0.02930   3.377 0.001157 **
#> U.l2    -0.11933    0.03161  -3.775 0.000314 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.2937 on 76.32037 degrees of freedom
#> Multiple R-squared:  0.5625, Adjusted R-squared:  0.5414
#> F-statistic: 26.67 on 3.679632 and 76.32037 DF,  p-value: 2.873e-13
```

It is worthwhile to examine the correlations among the noise variables to evaluate concurrent relationships between the time series variables within the residuals:

```
print(round(summary_Canada$corres, 3))
```

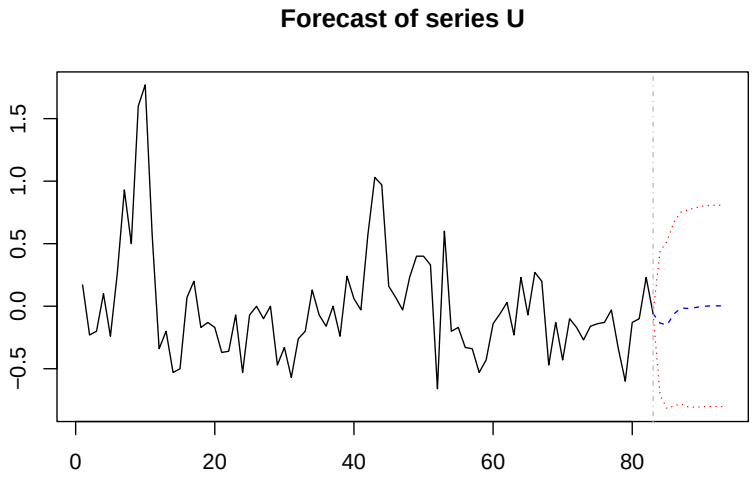


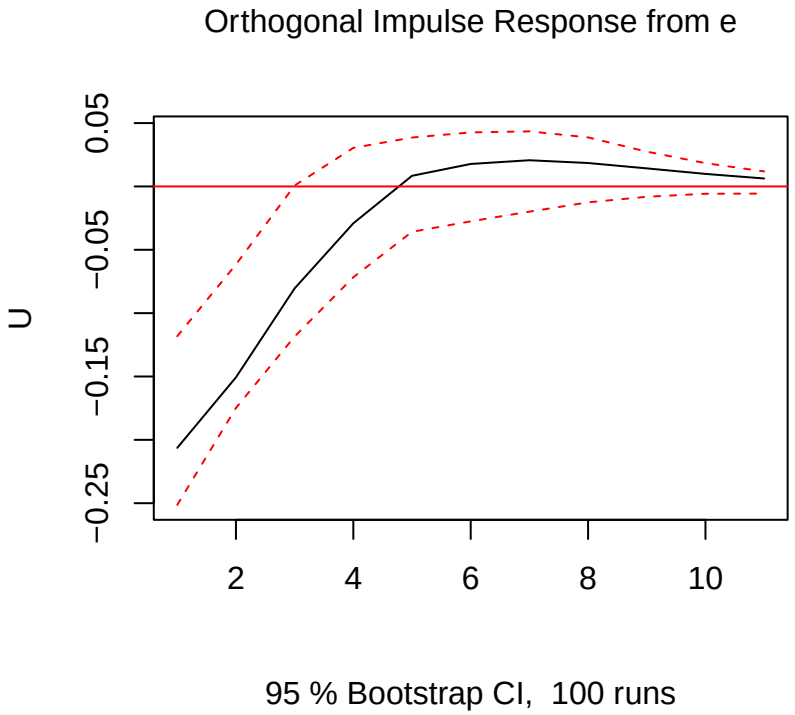

**Figure 3:** 10-step-ahead forecast of the differenced Canada time series using the VAR model estimated via the NS method with lag order $p = 2$.

```
#>            e   prod     rw      U
#> e      1.000 -0.070 -0.039 -0.687
#> prod -0.070  1.000  0.086  0.018
#> rw   -0.039  0.086  1.000  0.207
#> U    -0.687  0.018  0.207  1.000
```

Time series forecasting can be performed, as illustrated in Figure 3.

```
plot(predict(fitted_Canada), names = "U")
```

Impulse responses can be computed from the same fitted object.




### 4.2 Comparative simulation

We generated multivariate time series data of length $T = 20, 40, 80, 160$ from $K = 20$ dimensional VAR models of order $p = 1$. Note that for $T = 20$, the sample size $N = T - p = 19$ is smaller than the dimensionality $K = 20$. The VAR coefficient matrix $\mathbf{A}_1$ was specified with diagonal entries of 0.6 and zeros elsewhere, except for $K$ entries randomly selected from the lower triangular part of the matrix.

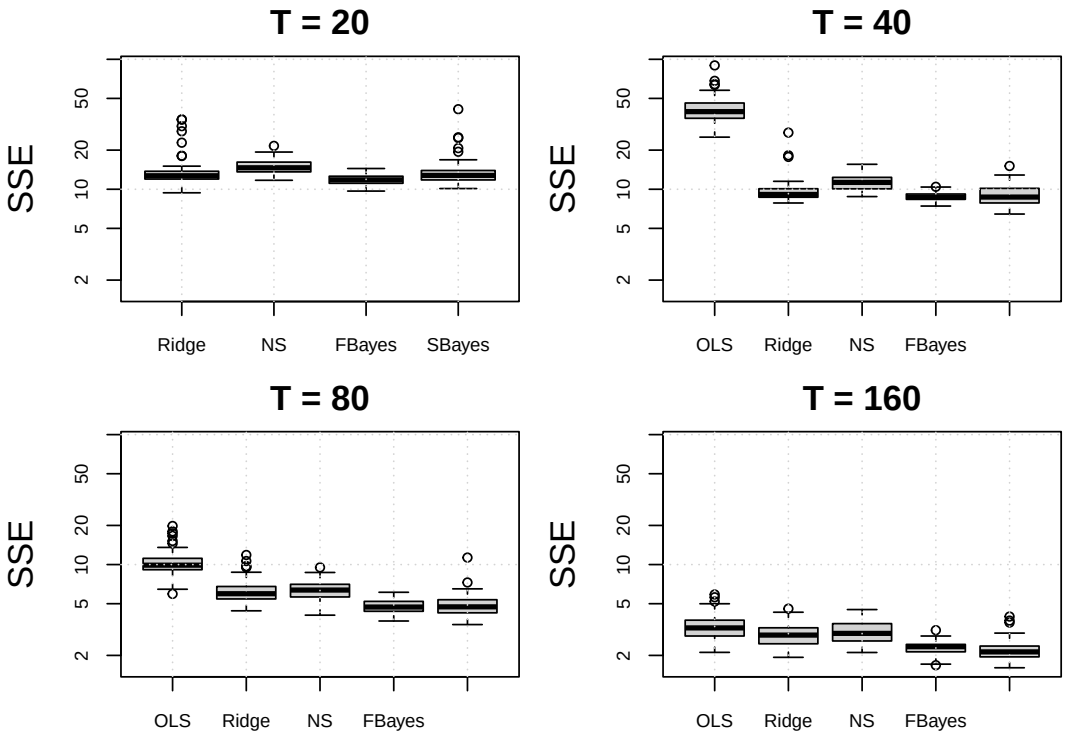



**Figure 4:** Comparison of mean SSEs across the four shrinkage methods and the OLS.

The nonzero off-diagonal entries were drawn uniformly at random from the intervals $[-1, -0.2]$ and $[0.2, 1]$. Since the coefficient matrix is lower triangular, the diagonal entries determine its eigenvalues, which lead to weak stationarity (Hamilton, 1994; Tsay, 2005). The noise $\boldsymbol{\epsilon}_t$ was randomly sampled from the multivariate normal distribution whose covariance matrix has diagonal entries of 1 and off-diagonal entries of 0.5. The experiments were repeated 50 times. We compared the four shrinkage methods and, when $T - p > K$, the OLS baseline. The sum of squared errors (SSEs) of the estimated VAR coefficients was used as a measure of parameter estimation accuracy. The summary of the coefficient SSEs is shown in Figure 4. The figure shows that the full Bayesian and semiparametric Bayesian methods produced accurate estimates with low SSEs, particularly for larger values of $T$, as they correctly incorporated correlations among the noise variables.

Figure 5 presents the out-of-sample forecasting performance of the shrinkage methods and the OLS baseline when estimable. Figure 6 presents the computational costs of the same methods as a function of the sample size. The full Bayesian method required the largest computational cost, indicating that computationally efficient alternative methods are needed for high-dimensional time series analysis.

To assess scalability with respect to the number of VAR coefficients, we also varied the dimension $K$ while holding $T = 160$ and $p = 1$ fixed. The number of autoregressive coefficients is $K^2 p$. As shown in Figure 7, computational costs increase much more rapidly with the number of VAR coefficients than with the sample size.

## 5 Discussion

The **R** ecosystem for multivariate time series has become increasingly specialized, as reflected in the CRAN Time Series Task View (Hyndman and Killick, 2026). Packages such as **vars** and **MTS** provide established tools for VAR estimation, forecasting, and post-estimation analysis. Other packages focus on modern extensions: **BigVAR** and **bigtime** emphasize penalized estimation for large multivariate systems (Nicholson et al., 2026; Wilms et al., 2023), while **BVAR**, **bayesianVARs**, **bvarsv**, and **bvartools** support Bayesian workflows involving hierarchical prior selection, MCMC estimation, and stochastic volatility (Giannone et al., 2015; Kuschnig and Vashold, 2021; Gruber et al., 2026; Primiceri, 2005; Krueger, 2015; Mohr, 2024). For structural Bayesian VARs, **bsvarSIGNs** provides tools for sign, zero, and narrative restrictions (Wang and Woźniak, 2025).

**VARshrink** is designed to complement these packages by bringing an OLS baseline and several shrinkage estimators into a common interface compatible with the widely used **vars** workflow. It currently includes multivariate ridge regression (Hoerl and Kennard, 1970; Golub et al., 1979), a Stein-type nonparametric shrinkage method (Opgen-Rhein and Strimmer, 2007), fully Bayesian methods

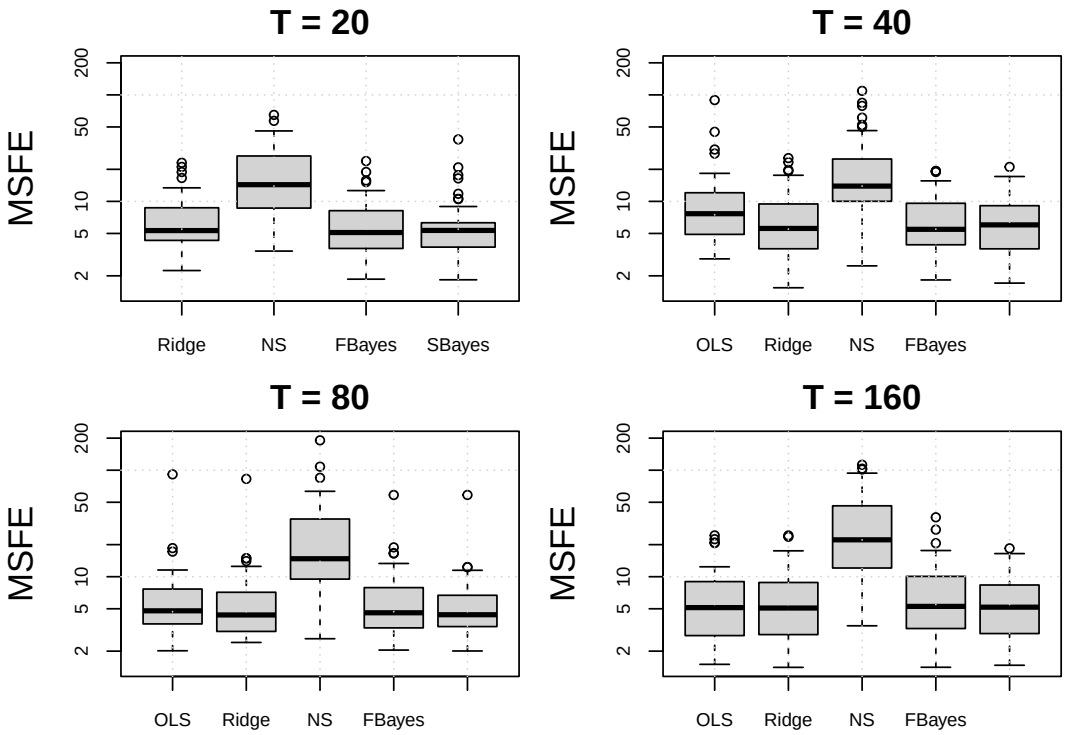



**Figure 5:** Comparison of 10-step out-of-sample MSFEs across the four shrinkage methods and the OLS baseline when estimable.

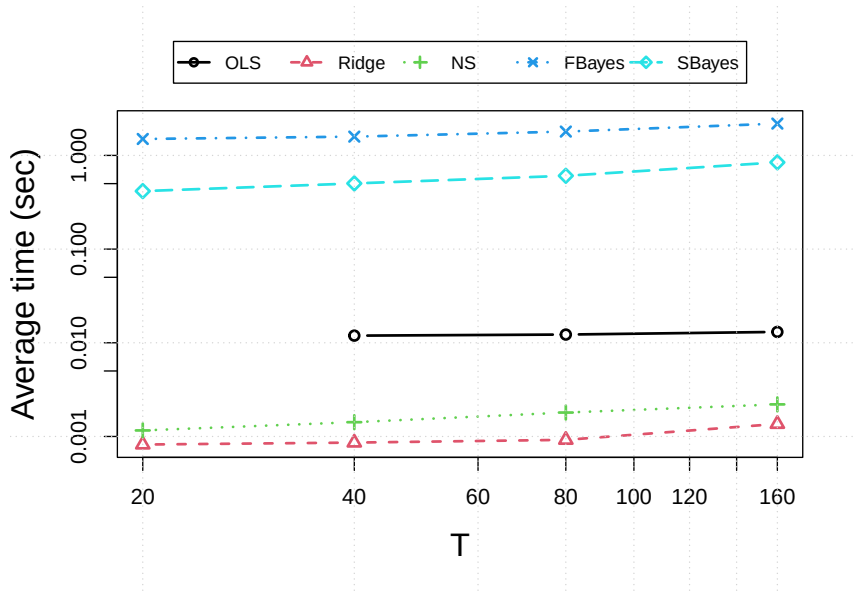



**Figure 6:** Comparison of computational costs across the four shrinkage methods and the OLS baseline when estimable as sample size increases.

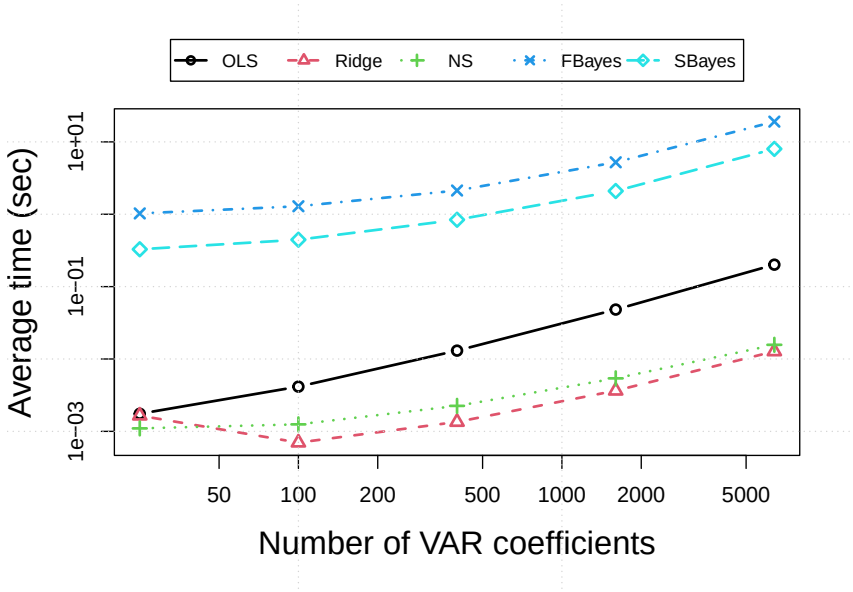



**Figure 7:** Comparison of computational costs across the four shrinkage methods and the OLS baseline as the number of VAR coefficients increases.

with natural conjugate and non-conjugate priors (Ni and Sun, 2005), and semiparametric Bayesian methods (Lee et al., 2016). This design allows users to compare OLS and alternative shrinkage estimators while retaining familiar tools for coefficient extraction, fitted values, residuals, forecasting, impulse responses, forecast error variance decompositions, diagnostic tests, and visualization.

The package is intended for applied researchers, students, and analysts who use the **vars** workflow but need more stable estimation than ordinary least squares can provide. Such settings arise when the number of variables is large relative to the number of observations, as in macroeconomic forecasting, financial systems, gene regulatory network analysis, and dynamic brain connectivity analysis (Harrison et al., 2003; Opgen-Rhein and Strimmer, 2007; Bańbura et al., 2010). In these applications, shrinkage is useful not only for forecasting but also for stabilizing coefficient estimates used to study dynamic relationships, partial correlation structures, impulse responses, and forecast error variance decompositions.

The structural analysis tools in **VARshrink** come from this interoperability with **vars**. Because `varshrinkest` objects inherit from `varest`, users can apply familiar post-estimation tools from **vars**. For impulse responses, uncertainty calculation is estimator-specific: bootstrap re-estimation can be costly for shrinkage estimators, whereas stored MCMC draws can be used by `irf()` to form Bayesian credible intervals when the full Bayesian method is fitted with `store_mcmc = TRUE`. Users whose main goal is advanced structural identification, stochastic volatility, or sign, zero, and narrative restrictions may therefore prefer packages designed specifically for those purposes.

Several limitations remain. **VARshrink** does not yet implement the full range of modern Bayesian shrinkage priors, such as Minnesota-type, global-local, or hierarchical priors used in macroeconomic forecasting (Doan et al., 1984; Litterman, 1986; Kastner and Huber, 2017; Giannone et al., 2015). It also inherits structural identification from **vars** rather than developing a separate structural VAR framework. Finally, fully Bayesian estimation provides richer posterior information but is more computationally expensive than ridge or nonparametric shrinkage, especially as the number of variables grows. These limitations define the current scope of the package and suggest future extensions, including additional prior structures, structured penalties, and closer integration with modern structural VAR workflows.

## 6 Conclusions

Shrinkage methods are useful for VAR models when the number of parameters is large relative to the sample size. The **R** package **VARshrink** provides an OLS baseline and ridge, nonparametric, fully Bayesian, and semiparametric shrinkage estimators through a unified interface connected to the familiar **vars** class system. This allows users to compare OLS with shrinkage methods and then apply standard VAR tools for forecasting, visualization, diagnostics, impulse responses, and forecast error variance decompositions.

**VARshrink** complements specialized software for penalized forecasting, Bayesian prior selection, stochastic volatility, and structural identification. Its main role is to provide a common workflow for stable VAR coefficient estimation in high-dimensional settings. Future development can extend this framework with additional prior structures, structured penalties, and richer structural analysis tools while preserving compatibility with established VAR workflows in **R**.

## Computational details

The results in this manuscript were obtained using **R** 4.5.3 and **VARshrink** 0.5.0 on a MacBook running macOS 26.3.1, equipped with an Apple M4 Pro CPU and 24GB of memory.

## Supplementary materials

varshrink-rjournal.R: R code to reproduce all the results in the manuscript.

## Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grants funded by the Ministry of Science and ICT (MSIT) (RS-2024-00336424, RS-2024-00358572) and the Ministry of Education (MOE) (RS-2025-25415913) of the Republic of Korea. N. Lee was supported by the Regional Innovation System & Education (RISE) program through the Gangwon RISE Center funded by the MOE and the Gangwon State (G.S.), Republic of Korea (2026-RISE-10-002).

## Conflict of interest

The authors have no competing interests to declare that are relevant to the content of this article.

## Data availability

The Canada dataset was deposited by other researchers and is available in the **R** package **vars** at the CRAN repository, https://CRAN.R-project.org/package=vars.

## References

M. Bańbura, D. Giannone, and L. Reichlin. Large Bayesian vector auto regressions. *Journal of Applied Econometrics*, 25(1):71–92, 2010. doi: 10.1002/jae.1137. [p269, 282]

L. Beltrachini, N. von Ellenrieder, and C. H. Muravchik. Shrinkage approach for spatiotemporal EEG covariance matrix estimation. *IEEE Transactions on Signal Processing*, 61(7):1797–1808, 2013. doi: 10.1109/TSP.2013.2238532. [p269]

H. Böhm and R. von Sachs. Shrinkage estimation in the frequency domain of multivariate time series. *Journal of Multivariate Analysis*, 100(5):913 – 935, 2009. doi: 10.1016/j.jmva.2008.09.009. [p269]

T. Doan, R. Litterman, and C. Sims. Forecasting and conditional projection using realistic prior distributions. *Econometric Reviews*, 3(1):1–100, 1984. doi: 10.1080/07474938408800053. [p269, 282]

T. Eltoft, T. Kim, and T.-W. Lee. On the multivariate laplace distribution. *IEEE Signal Processing Letters*, 13(5):300–303, 2006. doi: 10.1109/LSP.2006.870353. [p271]

M. Fiecas and H. Ombao. The generalized shrinkage estimator for the analysis of functional connectivity of brain signals. *The Annals of Applied Statistics*, 5(2A):1102–1125, 2011. doi: 10.1214/10-AOAS396. [p269]

J. Friedman, T. Hastie, R. Tibshirani, N. Simon, B. Narasimhan, and J. Qian. glmnet: Lasso and elastic-net regularized generalized linear models. R Package Version 2.0-16, 2018. Available at https://cran.r-project.org/web/packages/glmnet. [p270]

D. Giannone, M. Lenza, and G. E. Primiceri. Prior selection for vector autoregressions. *The Review of Economics and Statistics*, 97(2):436–451, 2015. doi: 10.1162/REST_a_00483. [p269, 280, 282]

G. H. Golub, M. Heath, and G. Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223, 1979. doi: 10.1080/00401706.1979.10489751. [p270, 275, 280]

L. Gruber, S. Haan, and G. Kastner. bayesianVARs: MCMC estimation of Bayesian vector autoregressions. R Package Version 0.1.8, 2026. Available at https://cran.r-project.org/package=bayesianVARs. [p269, 280]

J. D. Hamilton. *Time Series Analysis*. Princeton University Press, Princeton, 1994. [p269, 270, 280]

L. Harrison, W. D. Penny, and K. Friston. Multivariate autoregressive modeling of fmri time series. *NeuroImage*, 19:1477–1491, 2003. [p269, 282]

A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970. doi: 10.1080/00401706.1970.10488634. [p270, 271, 275, 280]

R. J. Hyndman and R. Killick. CRAN Task View: Time Series Analysis. Version 2026-06-09, 2026. URL https://CRAN.R-project.org/view=TimeSeries. [p280]

G. Kastner and F. Huber. Sparse Bayesian vector autoregressions in huge dimensions. arXiv:1704.03239, 2017. URL https://arxiv.org/abs/1704.03239. [p282]

G. Koop and D. Korobilis. Bayesian multivariate time series methods for empirical macroeconomics. *Foundations and Trends in Econometrics*, 3(4):267–358, 2010. doi: 10.1561/0800000013. [p269]

F. Krueger. bvarsv: Bayesian analysis of a vector autoregressive model with stochastic volatility and time-varying parameters. R Package Version 1.1, 2015. Available at https://cran.r-project.org/web/packages/bvarsv. [p269, 280]

N. Kuschnig and L. Vashold. BVAR: Bayesian vector autoregressions with hierarchical prior selection in R. *Journal of Statistical Software*, 100(14):1–27, 2021. doi: 10.18637/jss.v100.i14. [p269, 280]

O. Ledoit and M. Wolf. A well-conditioned estimator for large-dimensional covariance matrices. *Journal of Multivariate Analysis*, 88(2):365 – 411, 2004. doi: 10.1016/S0047-259X(03)00096-4. [p269]

N. Lee, H. Choi, and S.-H. Kim. Bayes shrinkage estimation for high-dimensional var models with scale mixture of normal distributions for noise. *Computationl Statistics & Data Analysis*, 101:250–276, 2016. doi: 10.1016/j.csda.2016.03.007. [p269, 270, 271, 272, 275, 282]

R. B. Litterman. Forecasting with Bayesian vector autoregressions: Five years of experience. *Journal of Business & Economic Statistics*, 4(1):25–38, 1986. doi: 10.2307/1391384. [p269, 282]

F. X. Mohr. bvartools: Bayesian inference of vector autoregressive and error correction models. R Package Version 0.2.4, 2024. Available at https://cran.r-project.org/package=bvartools. [p269, 280]

S. Moritz and E. Cule. ridge: Ridge regression with automatic selection of the penalty parameter. R Package Version 2.3, 2018. Available at https://cran.r-project.org/web/packages/ridge. [p270]

S. Ni and D. Sun. Bayesian estimates for vector autoregressive models. *Journal of Business & Economic Statistics*, 23(1):105–117, 2005. doi: 10.1198/073500104000000622. [p269, 271, 272, 275, 282]

W. Nicholson, D. Matteson, and J. Bien. BigVAR: Dimension reduction methods for multivariate time series. R Package Version 1.1.5, 2026. Available at https://cran.r-project.org/package=BigVAR. [p269, 280]

R. Opgen-Rhein and K. Strimmer. Learning causal networks from systems biology time course data: an effective model selection procedure for the vector autoregressive process. *BMC Bioinformatics*, 8 (2):S3, 2007. doi: 10.1186/1471-2105-8-S2-S3. [p269, 270, 271, 275, 280, 282]

B. Pfaff and M. Stigler. vars: Var modelling. R Package Version 1.5-3, 2018. Available at https://cran.r-project.org/web/packages/vars. [p269]

G. E. Primiceri. Time varying structural vector autoregressions and monetary policy. *The Review of Economic Studies*, 72(3):821–852, 2005. doi: 10.1111/j.1467-937X.2005.00353.x. [p269, 280]

B. Ripley, B. Venables, D. M. Bates, K. Hornik, A. Gebhardt, and D. Firth. MASS: Support functions and datasets for venables and ripley's mass. R Package Version 7.3-51.1, 2018. Available at https://cran.r-project.org/web/packages/MASS. [p270, 271, 275]

J. Schäfer and K. Strimmer. A shrinkage approach to large-scale covariance matrix estimation and implications for functional genomics. *Statistical Applications in Genetics and Molecular Biology*, 4(1):32, 2005. doi: 10.2202/1544-6115.1175. [p271, 275]

J. Schäfer, R. Opgen-Rhein, V. Zuber, M. Ahdesmäki, A. P. D. Silva, and K. Strimmer. corpcor: Efficient estimation of covariance and (partial) correlation. R Package Version 1.6.9, 2017. Available at http://CRAN.R-project.org/package=corpcor. [p269, 271, 275]

D. Sun and S. Ni. Bayesian analysis of vector-autoregressive models with noninformative priors. *Journal of Statistical Planning and Inference*, 121(2):291 – 309, 2004. doi: 10.1016/S0378-3758(03)00116-2. [p269, 271, 275]

R. S. Tsay. *Analysis of Financial Time Series*. Wiley series in probability and statistics. John Wiley & Sons, Hoboken, NJ, 2nd edition, 2005. ISBN 978-0-471-69074-0. [p270, 280]

R. S. Tsay and D. Wood. MTS: All-purpose toolkit for analyzing multivariate time series (mts) and estimating multivariate volatility models. R Package Version 1.0, 2018. Available at https://cran.r-project.org/web/packages/MTS. [p269]

X. Wang and T. Woźniak. bsvarSIGNs: Bayesian SVARs with sign, zero, and narrative restrictions. R Package Version 2.0, 2025. Available at https://cran.r-project.org/package=bsvarSIGNs. [p269, 280]

M. West. Outlier models and prior distributions in Bayesian linear regression. *Journal of the Royal Statistical Society. Series B (Methodological)*, 46(3):431–439, 1984. [p271]

I. Wilms, D. S. Matteson, J. Bien, S. Basu, W. Nicholson, and E. Wegner. bigtime: Sparse estimation of large time series models. R Package Version 0.2.3, 2023. Available at https://cran.r-project.org/package=bigtime. [p269, 280]

*Namgil Lee*
*Kangwon National University*
*Department of Information Statistics, Kangwon National University, Chuncheon, Gangwon 24341, Republic of Korea*
*Bionsight, Chuncheon, Gangwon 24341, Republic of Korea*
*ORCID: 0000-0003-0593-9028*
namgil.lee@kangwon.ac.kr

*Sung-Ho Kim*
*KAIST*
*Department of Mathematical Sciences, KAIST, Daejeon 34141, Republic of Korea*
sung-ho.kim@kaist.edu

# sugarglider: Create Glyph-Maps of Spatiotemporal Data

*by Maliny Po, S. Nathan Yang, H. Sherry Zhang, and Dianne Cook*

**Abstract** Spatiotemporal data visualisation allows users to explore events and interactions over space and time, making it easier to discover patterns, anomalies, and relationships within the data. Glyph maps use specialized multivariate glyph plots that depict and summarize data at each spatial location over time. A variant of glyph maps, the line glyph, illustrates changes over time but falls short in capturing the diverse components of spatiotemporal data. To address this limitation, Sugarglider extends glyph maps with ribbon and segment geometries to depict seasonal trends, variations, and data ranges.

## 1 Introduction

Spatiotemporal data visualisation allows users to explore events and interactions across space and time. This approach enables the discovery of temporal patterns, anomalies, and relationships within the data. As Kim et al. (2017) explains, a natural way to analyze spatiotemporal data is by plotting it on a map and incorporating animation controls or small multiple views to visualize each time step. This method helps analysts gain insights into the spatial distribution of the data and the patterns or correlations that emerge over time within these distributions.

One method for visualising spatiotemporal data is `glyph maps`. Glyph maps are specialized visual tools that condense multivariate data into a single graphical element positioned at a specific location on the map. Each glyph comprises two main structural components: spatial coordinates and data values. This visualisation allows for the simultaneous viewing of local patterns (specific to individual locations) and global patterns (across all locations). A variation of the glyph map is the Line Glyph. However, one limitation of this type of plot is that it can only display changes over time and does not effectively capture different data components.

To address this limitation, the `sugarglider` package was developed, enhancing glyph maps with the addition of ribbon and segment glyphs specifically designed for exploring variations and seasonal components in spatiotemporal data. This package includes two core functions: `geom_glyph_ribbon()` and `geom_glyph_segment()`, which enable the creation of ribbon and segment glyph maps, respectively. The `geom_glyph_ribbon()` function visualizes intervals between two values, making it perfect for depicting ranges or uncertainties. Meanwhile, the `geom_glyph_segment()` function connects data points to showcase trends and transitions.

`sugarglider` offers extensive customization features such as adjustable glyph aspect ratios, global or individual scaling, and custom rescaling, allowing users to tailor their visualisations to specific analytical needs. This versatility makes `sugarglider` a valuable tool for researchers and analysts working with spatiotemporal datasets across various fields, including transportation and climate, as discussed in the application section of this paper.

The following sections will explore how `sugarglider` enables users to visualize spatiotemporal data effectively, providing practical examples and highlighting the package's flexibility in creating insightful and customizable glyph maps.

## 2 Literature Review

### 2.1 Spatiotemporal visualisation

Data visualisation is often seen as converting unstructured raw data into structured representations that enhance interpretation. As Spence (2001) argues, visualisation involves creating a mental model or image. In this sense, the general workflow for data visualisation consists of four fundamental stages: collecting and storing data, preprocessing data into a desirable format, rendering displays through hardware and algorithms, and interpreting these visualisations through the human perceptual system (Waddell (2004)).

Spatiotemporal visualisations offer a global view of activities, enabling the detection of trends and evolution, making them indispensable tools for spatial and temporal analysis and decision-making (Zhong et al. (2012)). Various techniques are used in spatiotemporal data visualisation, such as multiple maps representing different time moments, dynamic maps with user control, change

maps highlighting differences between two time points, and the space-time cube, a 3D visualisation integrating geographical space and time (Andrienko et al. (2003)).

### 2.2 Glyph-maps

Glyph maps are specialized forms of multivariate glyph plots that represent each spatial location with a glyph, summarizing data collected over time at that location. As detailed in the paper by Wickham et al. (2012), each spatial location is represented by a single glyph that encapsulates temporal data collected at that point, helping uncover both local and global patterns, with particular emphasis on temporal relationships.

The construction of a glyph map involves a transformation technique that uses linear algebra to convert temporal coordinates (minor coordinates) into spatial coordinates (major coordinates). This transformation is implemented in packages such as `GGally` by Schloerke et al. (2024) and `cubble` by Zhang et al. (2024), which support a more integrated approach to spatiotemporal data visualisation.

Although `GGally` and `cubble` perform coordinate transformations as a preprocessing step and provide tools for creating glyph maps, both packages depict each glyph solely as a time series. Neither, however, includes a geom for encoding intervals, which is crucial for visualising uncertainty or data ranges. To address this limitation, `sugarglider` introduces Glyph Ribbons and Glyph Segments, allowing users to visualize uncertainties and seasonal patterns in spatiotemporal data. These features are described in more detail in a later section.

For a plot to be effective in a specific data analysis task, it should make key comparisons easy to see. Position mapping is among the most intuitively perceived visual properties (Cleveland and McGill (1984)). Glyph maps reveal changes in slope, trend, average value, and variance over time, arranging graphical elements to support temporal comparisons. However, when glyph maps use linear trends as icons, they can be influenced by the Zöllner Illusion, which may distort the appearance of straight lines. Thus, to ensure accurate perception of change, implementing reference frames and proper scaling is essential (Wickham et al. (2012)). Recent developments, such as those in `sugarglider`, address this challenge by using reference lines as visual anchors to reduce perceptual distortion and enhance the interpretability of glyph maps, an approach discussed in more detail later.

### 2.3 `ggplot2` extensions

The architecture of `ggplot2` is based on the `ggproto` object-oriented programming system. Initially, `ggplot2` relied on the proto system for object-oriented tasks, an S3 subclass of the R environment class. Proto objects are manipulated using the `proto()` function, which sets the parent environment, evaluates expressions, and manages lazy evaluation of arguments (Grothendieck et al. (2016)).

The `sugarglider` package is implemented as a `ggproto` extension of `ggplot2`, the same mechanism used by `cubble` to build `geom_glyph()` (Zhang et al. (2024)). This approach was chosen over defining a new geom from scratch because `ggproto` allows the necessary coordinate transformation and other data processing to be handled internally within the geom, keeping `sugarglider`'s functions compatible with standard ggplot2 layer syntax and consistent with the existing glyph-map ecosystem.

## 3 Software

The `sugarglider` package extends the capabilities of `ggplot2` by introducing functions designed explicitly for visualising seasonal patterns in spatiotemporal data. It includes `geom_glyph_ribbon()` and `geom_glyph_segment()`, which visualise measurements recorded over time at specific locations using glyph maps. These functions enable explicit depictions of seasonal trends by leveraging the combination of *x_major* and *y_major* coordinates.

Both `geom_glyph_ribbon()` and `geom_glyph_segment()` require the same data structure, combining spatial (longitude and latitude) and temporal elements with minimum and maximum values to establish the lower and upper bounds of the ribbon or segment. These functions share the same input structure, each offering distinct advantages while accommodating both continuous ranges and discrete start–end values.

These geometries enable users to visualize uncertainty and seasonal trends in time series plots. `geom_glyph_ribbon()` is particularly suitable for visualising uncertainty because it draws continuous lines, whereas `geom_glyph_segment()` is useful for detecting NA values along the x_major axis. Since `geom_glyph_ribbon()` is built upon ggplot2's `geom_ribbon()`, it connects the end of a previous segment to the beginning of the next when NA values are present. In contrast, `geom_glyph_segment()` leaves a gap that makes missing values easy to identify. During data preprocessing, the package automatically

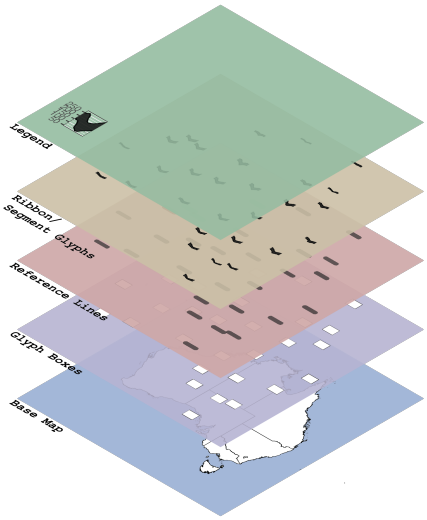



**Figure 1:** The diagram depicts the structure of a glyph map. The first layer represents the base map. Subsequent layers comprise glyph boxes and reference lines. The fourth layer encompasses the glyph itself, allowing users to depict ribbon glyphs. The legend layer is optional. It enables users to display a legend—a magnified version of one of the glyphs.

removes rows with missing values. Users are notified through warning messages printed to the console.

The structure of glyph maps in `sugarglider` consists of four main layers: the base map, glyph boxes, reference lines, and ribbon or segment glyphs. Additionally, users can create a legend, adding an extra layer to the glyph maps. In addition to the base map, `sugarglider` provides functionality to generate all the elements of a comprehensive glyph map, as illustrated in the figure below.

Each layer can be plotted independently, and the package supports creating glyph maps using ribbon and segment geometries. The core functionality includes:

- `geom_glyph_ribbon()`: Displays an interval on the y-axis for each *x_minor* value, with the bounds defined by *ymin_minor* and *ymax_minor*. This function draws ribbon geometry using the `geom_ribbon()` function from `ggplot2`. Each glyph is plotted by combining *x_major* and *y_major* coordinates. This functionality is handy for visualising data ranges or uncertainties.
- `geom_glyph_segment()`: Connects *y_minor* to *yend_minor* with a straight line using `geom_segment()` from `ggplot2`, resulting in segment glyphs. Each glyph is plotted by combining *x_major* and *y_major* coordinates.

In addition to these two functions, `sugarglider` offers several other features that enhance the aesthetic and interpretability of glyph maps. The `add_glyph_boxes()` function introduces reference boxes that visually frame individual glyphs, helping to define boundaries and distinguish them from one another. The `add_ref_lines()` function draws a horizontal midpoint for each glyph, facilitating comparisons across data points and serving as a visual anchor to reduce perceptual distortion as a result of the Zöllner illusion and improve the accuracy of interpreting glyph maps. The `add_glyph_legend()` displays an enlarged version of a randomly chosen glyph in the bottom-left corner of the panel, enabling users to visualize the data range. Lastly, the `theme_glyph()` function provides a customized theme for glyph maps, built on top of `theme_map()` from `ggthemes`. It adjusts the plot's appearance, including the legend position, text styles, and background settings, to create a clean, visually consistent layout for glyph maps.

```
library(ggplot2)
library(tidyverse)
library(sugarglider)
library(ozmaps)
library(gridExtra)
```

```
# Define a color palette
color_palette <- c("deepskyblue4", "coral3")

# Filter the data to three weather stations
vic_temp <- aus_temp |>
  filter(id %in% c("ASN00026021", "ASN00085291", "ASN00084143"))

# Ribbon glyph
p1 <- vic_temp |>
  ggplot(aes(x_major = long,
             y_major = lat,
             x_minor = month,
             ymin_minor = tmin,
             ymax_minor = tmax)) +
  geom_sf(data = abs_ste |> filter(NAME == "Victoria"),
          fill = "antiquewhite", color = "white", inherit.aes = FALSE)  +
  # Customize the size of each glyph box using the width and height parameters.
  add_glyph_boxes(width = rel(2.5), height = rel(1.5),
                  color = color_palette[1]) +
  add_ref_lines(width = rel(2.5), height = rel(1.5),
                color = color_palette[1]) +
  geom_glyph_ribbon(width = rel(2.5), height = rel(1.5),
                    color = color_palette[1], fill = color_palette[1]) +
  # Theme and aesthetic
  theme_glyph() +
  labs(title = "geom_glyph_ribbon()") +
  theme(plot.title = element_text(hjust = 0.5),
        title = element_text(color = color_palette[1],
                             family  = "mono"))

# Segment glyph
p2 <- vic_temp |>
  ggplot(aes(x_major = long,
             y_major = lat,
             x_minor = month,
             y_minor = tmin,
             yend_minor = tmax)) +
  geom_sf(data = abs_ste |> filter(NAME == "Victoria"),
          fill = "antiquewhite", color = "white", inherit.aes = FALSE)  +
  # Customize the size of each glyph box using the width and height parameters.
  add_glyph_boxes(width = rel(2.5), height = rel(1.5),
                  color = color_palette[2]) +
  add_ref_lines(width = rel(2.5), height = rel(1.5),
                color = color_palette[2]) +
  geom_glyph_segment(width = rel(2.5), height = rel(1.5),
                     color = color_palette[2]) +
  # Theme and aesthetic
  theme_glyph() +
  labs(title = "geom_glyph_segment()") +
  theme(plot.title = element_text(hjust = 0.5),
        title = element_text(color = color_palette[2]))

grid.arrange(p1, p2, ncol = 2)
```

The computational performance of glyph maps depends on both the number of locations and dataset size. Rendering glyph maps with hundreds of locations or high-frequency time series may result in slower plotting times. For large datasets, users are encouraged to aggregate the data to a monthly or yearly level before plotting. Currently, `sugarglider` produces static glyph maps, which can become overcrowded when visualising high-density spatial data. To mitigate this issue, users are encouraged to display only a limited number of locations or integrate `leaflet` to provide an interactive visualisation. A more detailed discussion of these limitations and potential solutions is presented in the Conclusion.

The `sugarglider` package provides various customization options for more flexible visualisation. It includes features such as the `global_rescale` argument, which allows users to choose between

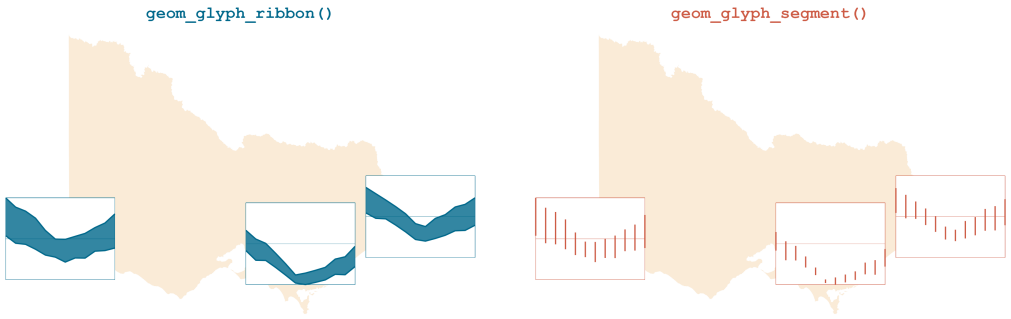



**Figure 2:** Comparison of ribbon and segment glyph maps using the dataset from the 'sugarglider' package. Glyph boxes and reference lines are added to improve interpretation of the glyph structure. Glyph dimensions are customised using the width and height arguments.

global or individual glyph scaling. Users can also adjust the scaling of minor values within grid cells and the overall width and height of glyphs. This ensures that the glyph map can be finely tuned to meet specific data visualisation needs. The following section will explore these features in greater detail and provide practical examples illustrating their application within different visualisation contexts.

### 3.1 Aesthetics

The functions in `sugarglider` expect spatial coordinates as the major axis and temporal data, along with some measurements, as minor axes. For minor axes, `sugarglider` adopts the same aesthetics as `ggplot2::geom_ribbon()` and `ggplot2::geom_segment()`, appending `_minor` to each aesthetic name for use in `geom_glyph_ribbon()` and `geom_glyph_segment()`. To incorporate a variable into the glyph plot, define it explicitly as an aesthetic. To produce glyph maps, the following aesthetics are required:

| Aesthetics | Description |
|---|---|
| `x_major`, `y_major` | Spatial coordinates that define the position of glyphs. |
| `x_minor` | Represents temporal data associated with each glyph. |
| `ymin_minor`, `ymax_minor` | Used by `geom_glyph_ribbon()` to establish the lower and upper bounds of the ribbon geometry within each glyph. |
| `y_minor`, `yend_minor` | Used by `geom_glyph_segment()` to set the start and end points of the segment geometry within each glyph. |

The functions `add_glyph_boxes()`, `add_ref_lines()`, and `add_glyph_legend()` are compatible with either *ymin_minor*, *ymax_minor*, or *y_minor*, *yend_minor*. Additionally, `sugarglider` introduces several customizable options to tailor the visual aspects further:

| Option | Default | Description |
|---|---|---|
| `colour` | `"black"` | Sets the color for line segments and borders. |
| `linewidth` | `0.5` | Specifies the width of the line for borders. |
| `linetype` | `1` | Defines the style of the line for borders. |
| `fill` | `"black"` | Determines the color of the interior area of the geometries. |
| `alpha` | `0.8` | Controls the transparency level of the glyphs. |

### 3.2 Options

Options within the `sugarglider` package let you tailor your visualisations to the specific needs of your analysis. The *global_rescale* argument controls whether rescaling is applied globally to all data points

or individually for each glyph. The effects of these two approaches are as follows:

- Local Rescale (global_rescale = FALSE): Each glyph is rescaled independently using its own data range, emphasising the variation and temporal patterns within each location.
- Global Rescale (global_rescale = TRUE): All glyphs are rescaled using a common data range, preserving magnitude differences across locations and enabling direct comparisons between glyphs.

`sugarglider` also offers customizable features to enhance the flexibility and precision of visualisations. For example, it facilitates the scaling of minor values within the glyph along the x- and y-axes. Users can specify their rescale function by replacing *"identity"* with a custom function in *x_scale* and *y_scale*. If a user wishes to modify the rescaling function on only one axis, they can replace the corresponding parameter with their chosen function and retain "identity" for the other. In this package, "identity" rescales the minor axes to an interval of [-1,1]. The impact of rescaling on glyphs and its implications for visual interpretation will be discussed in the upcoming section.

Additionally, the *width* and *height* of glyphs are fully adjustable, allowing users to customise their appearance to suit the dimensions and scale of the data being visualised. By default, both dimensions are determined by the smallest distance between two consecutive coordinates. Depending on the proximity of locations, this behaviour may produce glyphs that are either too small or too large. Users can therefore override the default values by specifying the *width* and *height* arguments to achieve an appropriate glyph size. An example is shown in Figure 2, where the glyph dimensions were customised using these arguments in the `add_glyph_boxes()` function. For high-density datasets, users may also choose to display only a subset of locations or combine `sugarglider` with Leaflet to create an interactive map that displays all locations. These customisation options enable sugarglider to adapt to a wide range of datasets and visualisation requirements, making it a versatile tool for seasonal spatiotemporal data visualisation.

| Option | Default | Description |
|---|---|---|
| `x_scale` | `"identity"` | This function scales each set of minor values within a grid cell along the x-dimension. |
| `y_scale` | `"identity"` | This function scales each set of minor values within a grid cell along the y-dimension. |
| `width` | `"default"` | The width of each glyph. The `default` is set to the smallest distance between two consecutive coordinates, converted from meters to degrees of latitude using the Haversine method. This default behaviour can be freely overridden by specifying a custom value. |
| `height` | `"default"` | The height of each glyph. The `default` is calculated using the ratio (1:1.618) relative to the `width`, to maintain a consistent aspect ratio. This default behaviour can be freely overridden by specifying a custom value. |
| `global_rescale` | `TRUE` | Controls whether the minor-axis values are rescaled using a common range across all glyphs (*TRUE*) or independently within each glyph (*FALSE*). |

### 3.3 Data structure

When using the `sugarglider` package to create glyph plots, the first step is to ensure that your data is in the correct format. Per Zhang et al. (2024), there are two data structures to consider, and `sugarglider` is compatible with one of them. The package supports data structured in an extended format with temporal and spatial elements.

The package includes a dataset named `aus_temp`, sourced from the National Oceanic and Atmospheric Administration (NOAA). This dataset provides comprehensive climate data for 2020 from 29 stations across Australia. It includes crucial climate variables such as precipitation and temperature, along with key spatial elements (longitude and latitude), temporal elements (month), and temperature ranges. These temperature ranges are crucial for determining the widths of the ribbon and segment plots in glyph maps.

```
glimpse(aus_temp)
```

```
#> Rows: 348
#> Columns: 7
```

```
#> $ id    <chr> "ASN00001020", "ASN00001020", "ASN00001020", "ASN00001020", "ASN~
#> $ long  <dbl> 126.3867, 126.3867, 126.3867, 126.3867, 126.3867, 126.3867, 126.~
#> $ lat   <dbl> -14.0900, -14.0900, -14.0900, -14.0900, -14.0900, -14.0900, -14.~
#> $ month <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6, 7, 8, 9~
#> $ tmin  <dbl> 253.4516, 248.6786, 253.6129, 244.0357, 220.4138, 202.3667, 153.~
#> $ tmax  <dbl> 319.0000, 322.6071, 333.1935, 340.9310, 331.9333, 310.9000, 291.~
#> $ prcp  <dbl> 163.87096774, 162.74074074, 42.00000000, 21.57142857, 0.00000000~
```

Datasets may not always include both spatial and temporal elements. Analysts often start with station data that provides geographic locations, recorded variables, and time periods. To extract relevant data, they can query temporal variables for specific stations of interest. When starting with purely spatial or purely temporal data, they first need additional elements to transform it into a spatiotemporal format.

For these situations, the `cubble` package offers functions such as `make_cubble()` to help users structure their data into `cubble` objects, optimised for glyph maps. This structuring facilitates detailed, insightful spatiotemporal visualisations and seamless integration into the `sugarglider` package.

### 3.4 Rescale

In `sugarglider`, rescaling is a crucial preprocessing step applied to the minor axes, the data used to plot individual glyphs. Before a glyph can be positioned on a map, its time-series data must first be normalised into a common scale. Normalising the data ensures that glyphs are comparable across locations and provides a consistent input for the spatial-temporal transformation described in the following section. The rescaling mechanism is governed by two parameters: *x_scale* and *y_scale*. *x_scale* adjusts the minor values along the x-dimension within each glyph, while *y_scale* modifies them along the y-dimension.

By default, the rescaling function is set to "identity", which adjusts the minor axes to fit within the interval [-1, 1]. However, users can customize the rescaling function by replacing the default *x_scale* and *y_scale* settings with functions. For example, the following code demonstrates a custom rescale function that maps values to the interval [0, 1]. When this custom rescale is applied, the resulting ribbon in the plot appears significantly thinner than in the previous example, which used the default rescaling settings.

```
# Default rescale
nsw_temp |>
   ggplot(aes(x_major = long,
              y_major = lat,
              x_minor = month,
              ymin_minor = tmin,
              ymax_minor = tmax)) +
  geom_glyph_ribbon() +
  theme_glyph()

# Custom rescale
custom_rescale <- function(dx) {
  rng <- range(dx, na.rm = TRUE)
  # Rescale dx to [0,1]
  rescaled <- (dx - rng[1]) / (rng[2] - rng[1])
}

nsw_temp |>
   ggplot(aes(x_major = long,
              y_major = lat,
              x_minor = month,
              ymin_minor = tmin,
              ymax_minor = tmax)) +
  geom_glyph_ribbon(x_scale = custom_rescale,
                    y_scale = custom_rescale) +
  theme_glyph()
```

To fully grasp the impact of rescaling on mapping temporal data to glyphs, it's important to consider how this process applies to both `geom_glyph_ribbon()` and `geom_glyph_segment()`. The

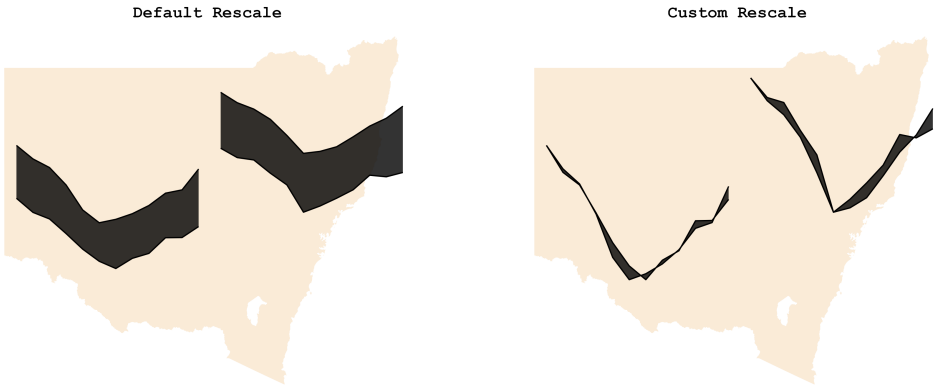



**Figure 3:** The figure illustrates the effect of rescaling on ribbon glyphs. With the default rescaling, all minor axes are adjusted to fit within the interval [-1, 1], whereas the custom rescale function adjusts the minor axes to the interval [0, 1]. Additional code is required to plot the base map alongside the rescaled glyphs.

transformation of spatiotemporal data into visual representations will be explored in greater detail in the next section.

Additionally, `sugarglider` allows users to choose whether rescaling is applied globally across all glyphs or individually for each glyph. This behaviour is controlled by the `global_rescale` parameter, which defaults to *TRUE*. When `global_rescale` is set to *FALSE*, users can implement local rescaling, allowing each glyph to be scaled independently. The difference between global and local rescaling is evident in the following example:

```
# Global rescale
aus_temp |>
  ggplot(aes(
    x_major = long,
    y_major = lat,
    x_minor = month,
    y_minor = tmin,
    yend_minor = tmax)) +
  add_glyph_boxes() +
  add_ref_lines() +
  geom_glyph_segment(global_rescale = TRUE) +
  theme_glyph()

# Local Rescale
aus_temp |>
  ggplot(aes(
    x_major = long,
    y_major = lat,
    x_minor = month,
    y_minor = tmin,
    yend_minor = tmax)) +
  add_glyph_boxes() +
  add_ref_lines() +
  geom_glyph_segment(global_rescale = FALSE) +
  theme_glyph()
```

The rescaling process in `sugarglider` involves several steps. First, the function checks for custom scaling based on the `x_scale` and `y_scale` parameters. It then groups the data based on the designated grouping variable to ensure each glyph is drawn as a distinct path. If `global_rescale` is set to *TRUE*, the data is ungrouped before rescaling the minor axes, ensuring consistent scaling across all glyphs. Conversely, if `global_rescale` is set to *FALSE*, the data remains grouped, resulting in local scaling for

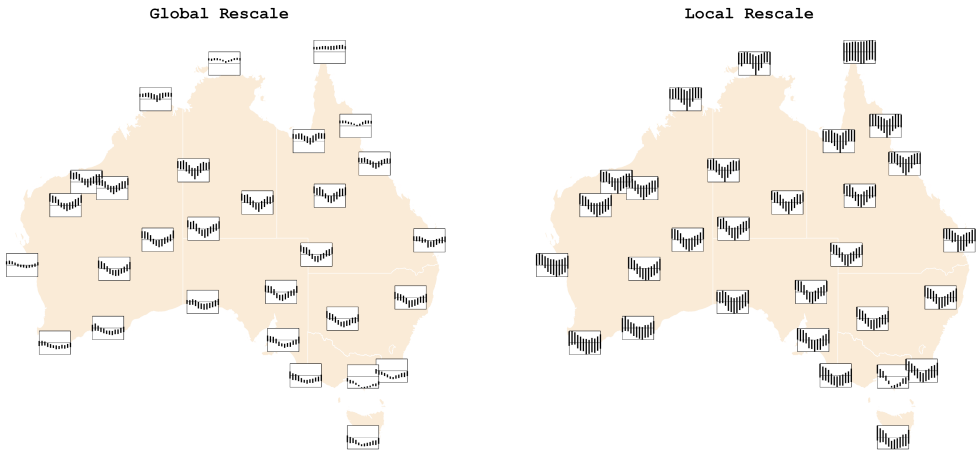



**Figure 4:** The figure highlights the impact of global and local rescaling on segment glyphs. With global rescaling, the temperature range is uniform across all glyphs, allowing users to compare variations in temperature between stations across Australia. In contrast, with local rescaling, the temperature range varies between glyphs, enabling more detailed insights into the temperature distribution at individual stations. Additional code is required to plot the base map and specify the xlim for the sf coordinates.

each glyph.

For both `geom_glyph_ribbon()` and `geom_glyph_segment()`, rescaling is applied separately to the `ymin_minor` and `ymax_minor` values or the `y_minor` and `yend_minor` values, respectively. This ensures that the lower and upper bounds are adjusted independently to fit within the specified scale, maintaining accuracy in the rescaled value.

### 3.5 Spatial-temporal transformation

After rescaling, the normalised minor-axis values are converted to spatial coordinates to position each glyph on the base map. The construction of a glyph map, as described in Wickham et al. (2012), involves a linear combination of two key structural components: spatial location and data values. In this context, the major axes represent the spatial coordinates, latitude ($y_{major}$) and longitude ($x_{major}$). In contrast, the minor axes correspond to time ($x_{minor}$) and a measurement of interest ($ymax_{minor}$ and $ymin_{minor}$). For segment glyphs, the measurement is represented by $y_{minor}$ and $yend_{minor}$.

Once the minor axes are rescaled to the interval [-1, 1], the final coordinates for the ribbon glyph are determined through a linear transformation as follows:

$$\mathrm{x} = \mathrm{x}_{\mathrm{major}} + \frac{\mathrm{width}}{2} \cdot \mathrm{x}_{\mathrm{minor}} \tag{1}$$

$$\mathrm{ymin} = \mathrm{y}_{\mathrm{major}} + \frac{\mathrm{height}}{2} \cdot \mathrm{ymin}_{\mathrm{minor}} \tag{2}$$

$$\mathrm{ymax} = \mathrm{y}_{\mathrm{major}} + \frac{\mathrm{height}}{2} \cdot \mathrm{ymax}_{\mathrm{minor}} \tag{3}$$

Similarly, the coordinates for the segment glyph are computed as:

$$\mathrm{x} = \mathrm{x}_{\mathrm{major}} + \frac{\mathrm{width}}{2} \cdot \mathrm{x}_{\mathrm{minor}} \tag{4}$$

$$\mathrm{y} = \mathrm{y}_{\mathrm{major}} + \frac{\mathrm{height}}{2} \cdot \mathrm{y}_{\mathrm{minor}} \tag{5}$$

$$\mathrm{yend} = \mathrm{y}_{\mathrm{major}} + \frac{\mathrm{height}}{2} \cdot \mathrm{yend}_{\mathrm{minor}} \tag{6}$$

This linear transformation ensures that the temporal and data components are appropriately aligned with the spatial coordinates, enabling a clear and accurate visualisation of spatiotemporal data.

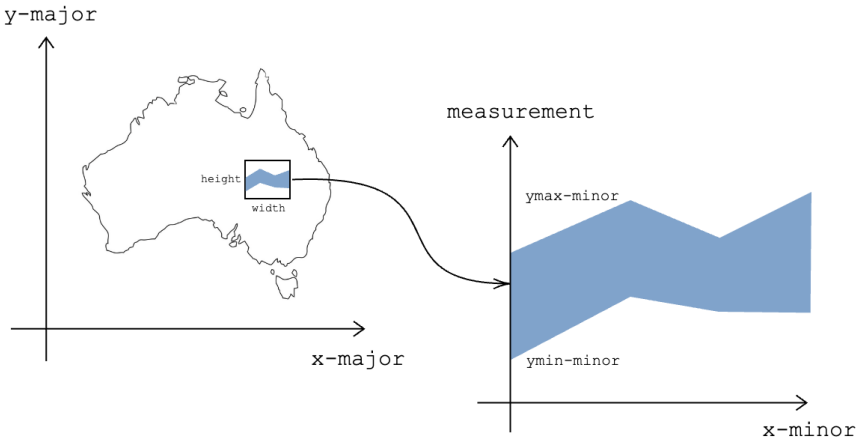



**Figure 5:** The diagram highlights how spatial data (geographical location) combines temporal data (measurements over time) to create a spatiotemporal visualisation. In sugarglider, the transformation maps each station's temporal measurements into a visual glyph, allowing users to see patterns across different spatial locations over time.

### 3.6 Examples

The `aus_temp` dataset is used to demonstrate the functionality of the `sugarglider` package. Using the default rescaling parameters, we can visualize temperature data with `geom_glyph_segment()` alongside `geom_point()` that marks the location of each weather station. Each segment glyph represents local climate data, providing an intuitive way to explore temperature variations across Australia.

```
aus_temp |>
  ggplot(aes(
    x_major = long,
    y_major = lat,
    x_minor = month,
    y_minor = tmin,
    yend_minor = tmax)) +
  add_glyph_boxes() +
  geom_point(aes(x = long, y = lat)) +
  geom_glyph_segment() +
  theme_glyph()
```

Building on the analysis, precipitation data across Australia can be visualized using `geom_glyph_ribbon()`. The glyphs are colour-coded to represent different rainfall levels. In contrast, reference lines and glyph boxes are added to improve clarity and facilitate easy comparisons of precipitation levels across the country.

```
aus_temp |>
   group_by(id) |>
   mutate(prcp = mean(prcp, na.rm = TRUE)) |>
   ggplot(aes(x_major = long, y_major = lat,
              x_minor = month, ymin_minor = tmin,
              ymax_minor = tmax,
              fill = prcp, color = prcp)) +
   add_glyph_boxes() +
   add_ref_lines() +
   geom_glyph_ribbon() +
   theme_glyph()
```

The following example uses the `historical_temp` dataset included in the `sugarglider` package. This dataset contains aggregated monthly average minimum and maximum temperatures, along with precipitation, for selected Australian weather stations during 2021 and 2022.

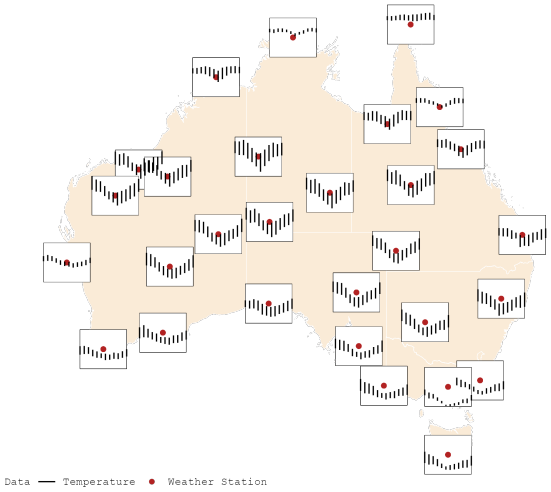



**Figure 6:** Daily temperature variations across Australian weather stations are visualized using segment glyphs and geom points to mark the locations of the stations on the map. Additional coding is needed to integrate a base map into the visualisation. This detailed view reveals that the temperature variation in central Australia is more pronounced than in the coastal regions. Additionally, temperatures in the southeastern areas, such as Victoria and Tasmania, are generally lower than those in the northern regions, highlighting regional climate differences across Australia.

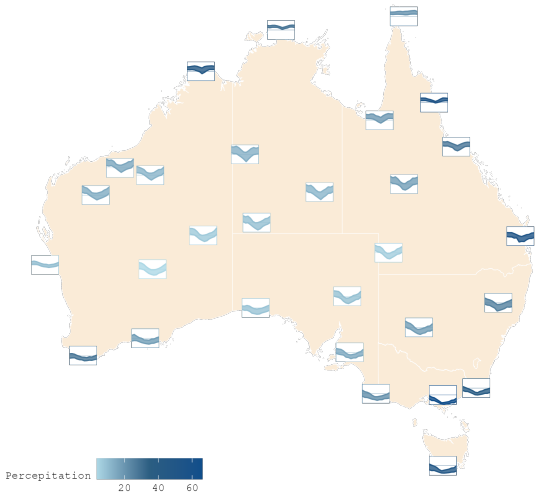



**Figure 7:** Precipitation and Temperature Ranges Across Australia. This visualisation uses color-coded glyphs to depict variations in precipitation levels, with sky blue representing the lowest and navy blue the highest levels. Additional code is required to incorporate a base map and customize the theme further. From the plot, it is evident that coastal areas receive more rainfall than the central regions.

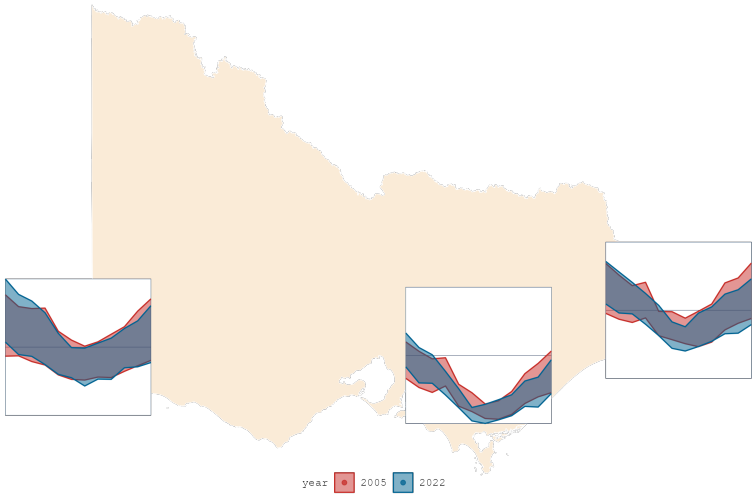



**Figure 8:** Temperature Trends in Selected Victorian Weather Stations. Additional coding is required for the base map, title, and theme customization. In both years, a noticeable seasonal pattern in temperature is evident, with peaks occurring in the summer months and troughs in the winter.

```
glimpse(historical_temp)
```

```
#> Rows: 696
#> Columns: 8
#> $ id    <chr> "ASN00001020", "ASN00001020", "ASN00001020", "ASN00001020", "ASN~
#> $ long  <dbl> 126.3867, 126.3867, 126.3867, 126.3867, 126.3867, 126.3867, 126.~
#> $ lat   <dbl> -14.0900, -14.0900, -14.0900, -14.0900, -14.0900, -14.0900, -14.~
#> $ month <dbl> 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10, 11~
#> $ year  <dbl> 2005, 2022, 2005, 2022, 2005, 2022, 2005, 2022, 2005, 2022, 2005~
#> $ tmin  <dbl> 254.6667, 253.4516, 256.9615, 248.6786, 252.8750, 253.6129, 254.~
#> $ tmax  <dbl> 330.0769, 319.0000, 334.8889, 322.6071, 342.5000, 333.1935, 345.~
#> $ prcp  <dbl> 55.72413793, 163.87096774, 15.07692308, 162.74074074, 181.161290~
```

To compare temperature trends across different years for specific regions in Victoria, `geom_glyph_ribbon()` offers an effective way to visualize how temperatures have evolved. Each year is distinguished by a different colour, making the trends clear and easy to interpret.

```
historical_temp |>
  filter(id %in% c("ASN00026021", "ASN00085291", "ASN00084143")) |>
   ggplot(aes(color = factor(year), fill = factor(year),
              group = interaction(year,id),
              x_major = long, y_major = lat,
              x_minor = month, ymin_minor = tmin,
              ymax_minor = tmax)) +
   add_glyph_boxes() +
   add_ref_lines() +
   geom_glyph_ribbon() +
   theme_glyph()
```

To further improve map readability, the `add_glyph_legend()` function integrates an enlarged version of one of the glyphs in the bottom left corner of the plot. This legend helps users interpret the data scale more effectively.

In the example below, a ribbon glyph map is created using `geom_glyph_ribbon()` and overlaid on a base map to depict daily temperature variations across Australian weather stations. A legend is added via `add_glyph_legend()`, allowing users to easily interpret the range of daily temperature values based on a randomly selected weather station. Since the legend data is drawn from a single, randomly chosen station, it is essential to set a seed for reproducibility to ensure consistent results.

```
set.seed(28493)
aus_temp |>
```

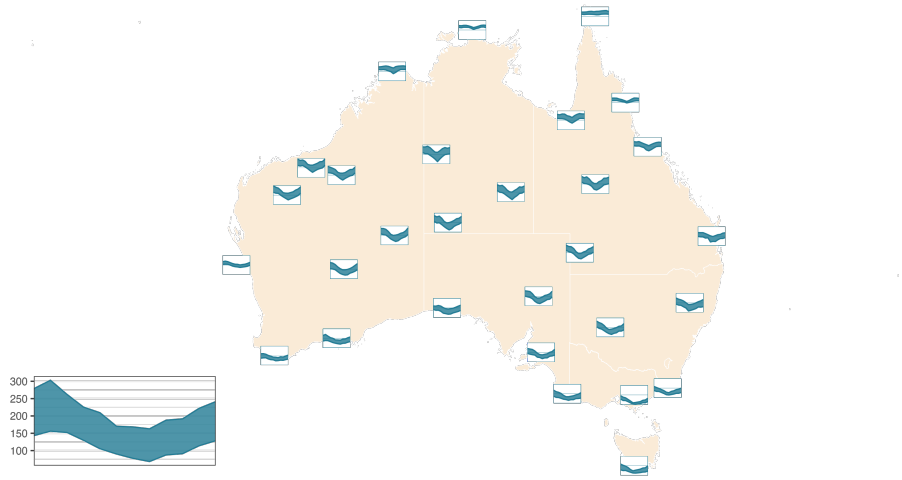



**Figure 9:** Temperature Ranges Across Australia with Glyph Legend. Additional coding is required for the base map and further theme customization. In this example, the glyphs are uniformly scaled, representing a temperature range from approximately 5.0 to 35.0 degrees Celsius. Temperature values are stored in tenths of a degree Celsius, following the NOAA data convention.

```
  ggplot(aes(x_major = long, y_major = lat,
             x_minor = month, ymin_minor = tmin,
             ymax_minor = tmax)) +
add_glyph_boxes() +
add_ref_lines() +
add_glyph_legend() +
geom_glyph_ribbon() +
theme_glyph()
```

## 4 Application

This section presents four examples showcasing the applications of the `sugarglider` package. The first two examples use the `Train Service Passenger Counts data` from `DataVic`. The first example demonstrates how to create ribbon glyph maps with Leaflet to explore variations in train service passenger counts. The second uses segment glyphs to analyse daily variations in passenger counts on train services in Melbourne City LGA. The remaining two examples are based on the `Airline Flight Delay and Cancellation` data from `Kaggle`. The third features interactive segment glyphs, developed with `ggiraph`, to analyse flight variations at the top 10 US airports with high cancellation rates. Finally, the fourth uses ribbon glyphs to compare flight variations between the western and southern regions.

### 4.1 Train service passenger counts

The `Train Service Passenger Counts` dataset, published by `DataVic`, provides estimated numbers of passengers boarding and alighting from Metro Train and Regional Train services for the 2023-2024 period. These estimates include all individuals aged five and over, excluding train drivers and station staff. For this analysis, the data have been aggregated to the hourly level and categorized into maximum and minimum passenger counts for weekdays, weekends, and holidays (including public and school holidays).

#### Victorian train station patronage

Interactive graphics are particularly useful for spatiotemporal data, enabling users to explore it from multiple perspectives. The `cubble` package exemplifies this by creating linked interactive plots using `crosswalk::bscols()` (Zhang et al. (2024)). This paper demonstrates how to create interactive glyph maps using Leaflet.

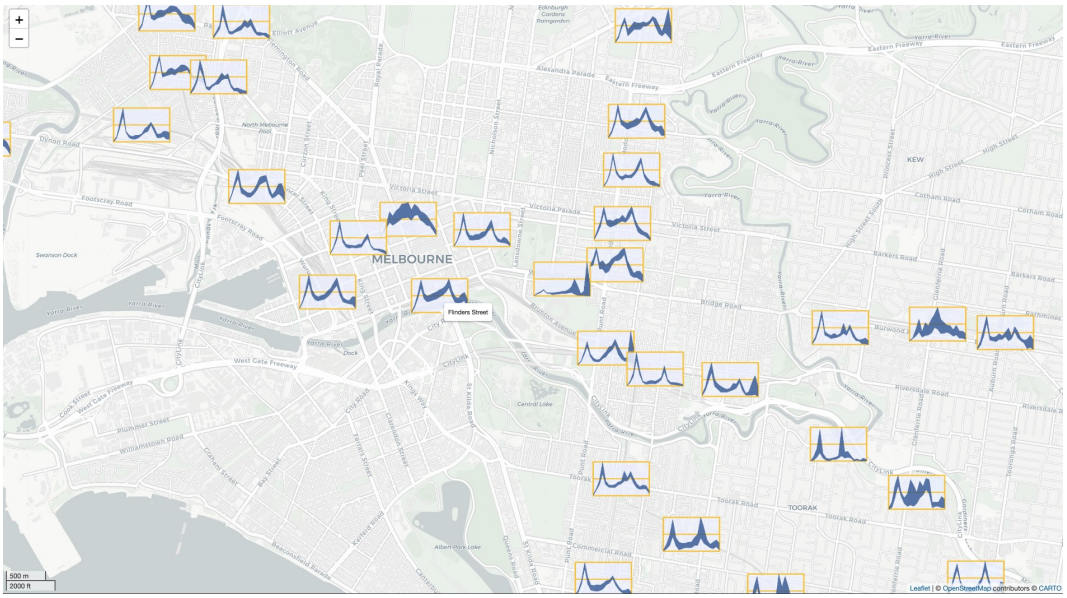



**Figure 10:** Screenshot of the hourly train station traffic in Melbourne. Each glyph represents hourly traffic, with peaks occurring during typical commuting hours (morning and evening rush hours)

Making a glyph map with Leaflet is helpful for high-density data where many observations are clustered or overlap, as is the case with the Victorian Train data. The main payoff is that Leaflet's ability to pan in and out lets us display all glyphs, regardless of how close they are to each other.

To create interactive glyph maps with Leaflet, we need to save each glyph as an image and add these to the Leaflet base map as icons. The process begins by creating a list of all unique train stations that service the Metro and Vline. We then iterate over each station and generate ribbon glyphs using `geom_glyph_ribbon()`. Each glyph is saved as an image, and the file paths for all the images are stored in an object for the next step.

The workflow begins by identifying unique train stations and generating ribbon glyphs to represent the daily passenger traffic patterns at each location. These glyphs are saved as image files and linked to their respective stations. An interactive base map is then constructed using Leaflet, onto which the glyph images are added as custom markers at each station's geospatial coordinates.

The resulting plot visualizes hourly train traffic across various stations in Victoria. Each glyph represents a train station and depicts fluctuations in train traffic at that location throughout the day.

In central Melbourne, especially around major hubs like Melbourne Central Station, traffic varies significantly, with peaks in the middle of the day. This indicates a substantial influx of people, highlighting the central business district (CBD) as the city's primary transit hub.

Some Eastern and South Eastern train stations exhibit pronounced peaks, indicating that they are key transit corridors for residents commuting into the city centre. These stations are situated along major rail lines, such as the `Cranbourne/Pakenham` and `Lilydale/Belgrave` lines, which cater to large suburban populations.

Moving away from the city centre and major rail lines, the glyphs illustrate smaller peaks and greater fluctuations in traffic levels. This suggests that train services are either less frequent or used less in the outer suburbs than in central areas.

### Daily train passenger variations in Melbourne City LGA

```
weekday <- station |>
  group_by(station_name) |>
  ggplot(aes(x_major = long, y_major = lat,
             x_minor = hour, y_minor = min_weekday,
             yend_minor = max_weekday,
             color = services, fill = services)) +
  add_glyph_boxes() +
  add_ref_lines() +
  geom_glyph_segment(global_rescale = FALSE) +
  theme_glyph() +
```

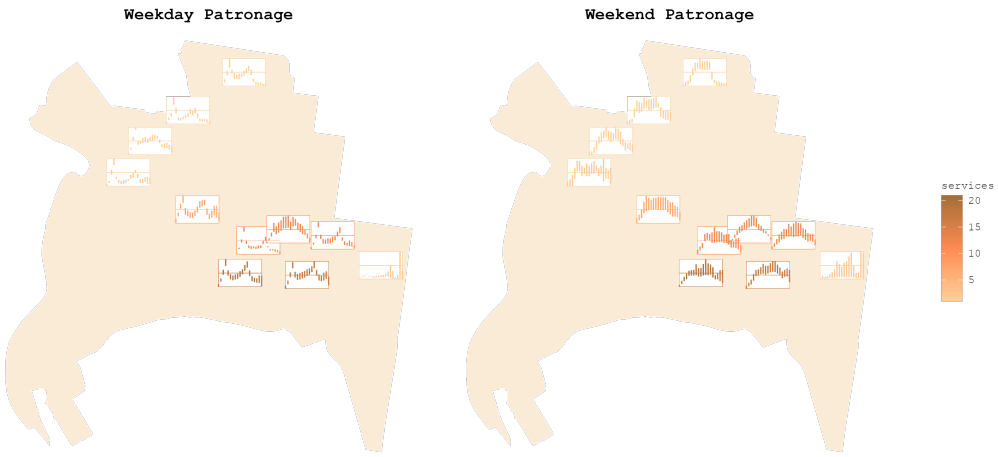



**Figure 11:** Patronage at various train stations in Melbourne City LGA on weekdays versus weekends. Each glyph is colour-coded by the number of train services at each station. Additional code is required to construct the base map and customize the plot further. The plot illustrates the behavioral patterns of train patrons, showing high demand during rush hours on weekdays and consistent, elevated demand throughout the weekend.

```
  labs(title = "Weekday Patronage")

weekend <- station |>
  group_by(station_name) |>
  ggplot(aes(x_major = long, y_major = lat,
             x_minor = hour, y_minor = min_weekend,
             yend_minor = max_weekend,
             fill = services, color = services)) +
  add_glyph_boxes() +
  add_ref_lines() +
  geom_glyph_segment(global_rescale = FALSE) +
  theme_glyph() +
  labs(title = "Weekend Patronage")
```

The plot provides insightful observations into train service traffic within the Melbourne City LGA, highlighting differences between weekday and weekend patronage. A significant volume of train services is apparent in the city loop, where major connecting hubs such as Flinders Street Station are situated. These stations are depicted with darker shades of brown, indicating up to 20 services, starkly contrasting with the lighter color gradients that signify lower train services.

The analysis further reveals a discrepancy in traffic between weekdays and weekends. At most train stations, there is a greater variation in patronage on weekends than on weekdays. This pattern is consistent across all stations, regardless of the number of train lines each station accommodates. Additionally, there are fewer pronounced peaks on weekends, as traffic is high but consistent throughout the day. In contrast, on weekdays, rush hours see significant spikes due to large numbers of people boarding trains in the morning and evening for school and work, with fewer patrons in between the peaks. This trend underscores the reliance on public transport at certain times of day and could be pivotal for future decisions on scheduling and resource allocation within the transit system.

### 4.2 Flight Delay and Cancellation

The U.S. Department of Transportation (DOT) Bureau of Transportation Statistics monitors the on-time performance of domestic flights operated by major U.S. carriers. The DOT publishes the Air Travel Consumer Report monthly, which summarizes data on on-time, delayed, canceled, and diverted flights.

This dataset, sourced from Kaggle's `Airline Flight Delay and Cancellation` data, has been processed and aggregated to focus on the minimum and maximum number of flights originating from the top 10 U.S. airports with the highest cancellation rates.

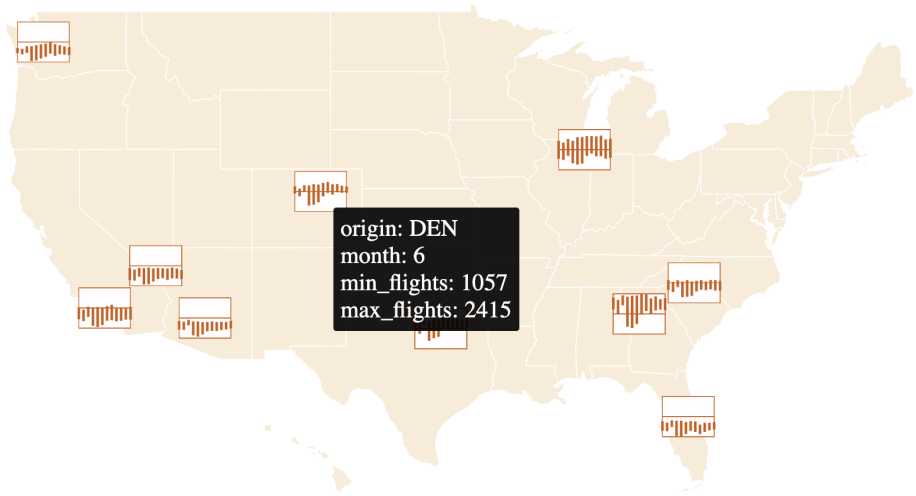



**Figure 12:** Monthly Flight Variability Based on the top 10 US airports with high cancellation rates. Additional codes are needed for the base map and additional theme customization. The graph highlights that airports in certain regions experience more variability than others. Each line segment represents the gap between the minimum and maximum flights. A longer segment indicates more significant month-to-month fluctuations in flight operations.

### Monthly Flight Variation in US Airport with High Cancellation

This example demonstrates the functionality of `geom_glyph_segment()`, which shows the monthly flight range for each airport and provides insights into how flight numbers change over time. Additionally, `geom_glyph_ribbon()` visualizes the variation in the minimum and maximum number of flights per airport, providing a clear depiction of the spread in flight activity.

Users can generate interactive glyphs with `ggiraph::girafe()` using `sugarglider`. First, users need to specify tooltips, which appear when hovering over glyphs. In this example, the tooltips include the station ID, month, minimum, and maximum temperatures across all months.

Tooltips must be provided in the `tooltip` argument's aesthetic. The user can then plot their desired glyph map and save it as a ggplot object. This ggplot object is converted into an interactive glyph map using the `girafe()` function.

```
# Specify tooltip for ggiraph
flights <- flights |>
  mutate(tooltip = paste("origin: ",origin,
                         "\nmonth: ", month,
                         "\nmin_flights: ", min_flights,
                         "\nmax_flights: ", max_flights))

fl <- flights |>
  ggplot(aes(x_major = long, y_major = lat,
             x_minor = month, y_minor = min_flights,
             yend_minor = max_flights, group = origin,
             tooltip = tooltip)) +
  add_glyph_boxes() +
  add_ref_lines() +
  geom_glyph_segment() +
  theme_glyph()

# Interactive plot using ggiraph
girafe(ggobj = fl)
```

This visualisation shows the range of departing flights from each airport on a map. Each line represents a flight range for one airport, with line length showing variation in flight numbers across

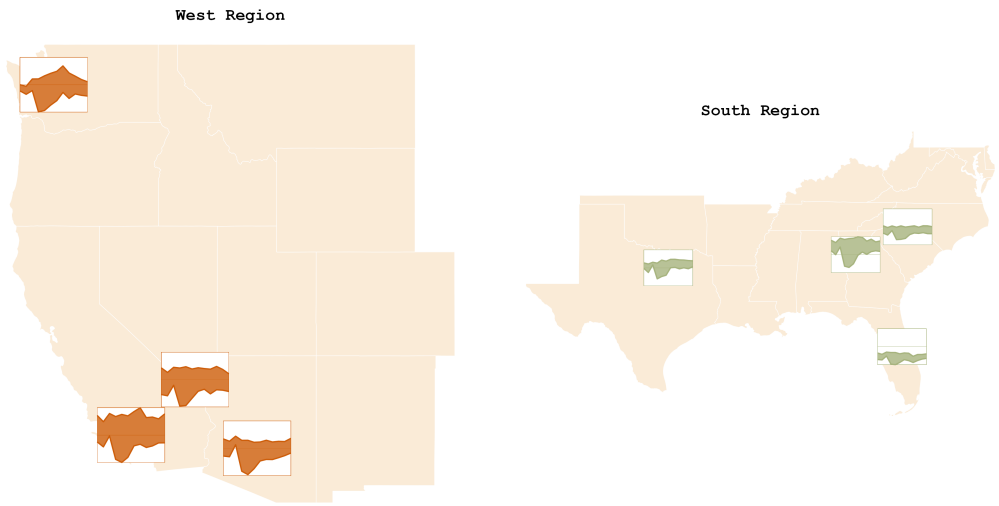



**Figure 13:** Comparison of flight patterns between the western and southern regions. Additional codes are needed for the base map and additional theme customization.

locations. Airports like ATL and ORD generally handle more departing flights than airports such as MCO and PHX. Airport traffic fluctuates throughout the year, with broader intervals in the middle of the year and narrower ones during the holiday season at year-end.

### Flight Pattern Between Western and Southern Region

```
# South Region
flights |>
  filter(origin %in% c("ATL", "CLT", "MCO", "DFW")) |>
  ggplot(aes(x_major = long, y_major = lat,
             x_minor = month, ymin_minor = min_flights,
             ymax_minor = max_flights)) +
  add_glyph_boxes() +
  add_ref_lines() +
  geom_glyph_ribbon() +
  theme_glyph()

# West region
flights |>
  filter(origin %in% c("PHX", "LAS", "LAX", "SEA")) |>
  ggplot(aes(x_major = long, y_major = lat,
             x_minor = month, ymin_minor = min_flights,
             ymax_minor = max_flights)) +
  add_glyph_boxes() +
  add_ref_lines() +
  geom_glyph_ribbon() +
  theme_glyph()
```

`geom_glyph_ribbon()` highlights the disparities in flight volume between regions. The Western region exhibits a more significant variation in the number of flights than the Southern region, as evidenced by the thicker ribbons. Notably, both regions display a significant increase in flight volume discrepancies during the mid-year, reinforcing the findings observed with `geom_glyph_segment()`.

## 5 Conclusion

This paper introduced `sugarglider`, an R package that extends glyph maps with ribbon and segment geometries for exploring spatiotemporal data. Across two applications—Victorian train patronage and flight delays and cancellations—the package demonstrated how ribbon and segment glyph maps can effectively visualise multivariate temporal patterns across numerous spatial locations simultaneously.

Two potential directions for future development include handling different data types and improving the glyph overlap handling. Currently, `sugarglider` expects data in a tidy format, but expanding its functionality to work with lists of vectors or matrices would enhance its usability. Additionally, static maps can become overcrowded when too many glyphs are plotted, making it difficult to interpret the visualisations. A possible solution is to collapse overlapping glyphs into a single glyph, with an accompanying data summary for all the collapsed glyphs. Another approach is to repel glyphs away from each other to prevent overlap. These improvements would increase the flexibility and scalability of `sugarglider` for a broader range of data visualisation tasks.

### 5.1 Acknowledgements

This work is funded by Google Summer of Code in affiliation with the R Project for Statistical Computing. This article is created using the R packages `knitr` (Xie (2014)) and `rjtools` (O'Hara-Wild et al. (2024)). The source code for reproducing this paper is available in the supplementary files and also on Github at https://github.com/maliny12/paper-sugarglider. Additionally, the `sugarglider` package is available on CRAN, with its source code available at https://github.com/maliny12/sugarglider .

### 5.2 CRAN Packages Dependencies

| Type | Dependency |
|---|---|
| Imports | ggplot2, dplyr, ggplotify, ggthemes |
| Suggests | cubble, testthat, tidyr, grid, lubridate, knitr, rmarkdown, gridExtra, ozmaps, sf, tidyverse, viridis, vdiffr, kableExtra, ggiraph, jsonlite, httr, usmap, geosphere, purrr |

## References

N. Andrienko, G. Andrienko, and P. Gatalsky. Exploratory spatio-temporal visualization: an analytical review. *Journal of Visual Languages & Computing*, 14(6):503–541, 2003. doi: 10.1016/S1045-926X(03)00046-6. [p287]

W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984. ISSN 01621459, 1537274X. doi: http://dx.doi.org/10.2307/2288400. URL http://www.jstor.org/stable/2288400. [p287]

G. Grothendieck, L. Kates, and T. Petzoldt. *proto: Prototype Object-Based Programming*, 2016. URL https://CRAN.R-project.org/package=proto. R package version 1.0.0. [p287]

S. Kim, S. Jeong, I. Woo, Y. Jang, R. Maciejewski, and D. S. Ebert. Data flow analysis and visualization for spatiotemporal statistical data without trajectory information. *IEEE transactions on visualization and computer graphics*, 24(3):1287–1300, 2017. doi: 10.1109/TVCG.2017.2666146. [p286]

M. O'Hara-Wild, S. Kobakian, H. S. Zhang, D. Cook, S. Urbanek, and C. Dervieux. *rjtools: Preparing, Checking, and Submitting Articles to the 'R Journal'*, 2024. URL https://github.com/rjournal/rjtools. R package version 1.0.14. [p303]

B. Schloerke, D. Cook, J. Larmarange, F. Briatte, M. Marbach, E. Thoen, A. Elberg, and J. Crowley. *GGally: Extension to 'ggplot2'*, 2024. URL https://CRAN.R-project.org/package=GGally. R package version 2.2.1. [p287]

R. Spence. Information visualization, 2001. [p286]

P. Waddell. Introduction to urban simulation: Design and development of operational models. *Transport Geography and Spatial Systems/Pergammon Press*, 2004. doi: 10.1108/9781615832538-013. [p286]

H. Wickham, H. Hofmann, C. Wickham, and D. Cook. Glyph-maps for visually exploring temporal patterns in climate data and models. *Environmetrics*, 23(5):382–393, 2012. doi: 10.1002/env.2152. [p287, 294]

Y. Xie. knitr: A comprehensive tool for reproducible research in R. In V. Stodden, F. Leisch, and R. D. Peng, editors, *Implementing Reproducible Computational Research*. Chapman and Hall/CRC, 2014. doi: 10.32614/CRAN.package.knitr. ISBN 978-1466561595. [p303]

H. S. Zhang, D. Cook, U. Laa, N. Langrené, and P. Menéndez. cubble: An r package for organizing and wrangling multivariate spatio-temporal data. *Journal of Statistical Software*, 110(7):1–27, 2024. doi: 10.18637/jss.v110.i07. URL https://www.jstatsoft.org/index.php/jss/article/view/v110i07. [p287, 291, 298]

C. Zhong, T. Wang, W. Zeng, and S. Müller Arisona. Spatiotemporal visualisation: A survey and outlook. *Digital Urban Modeling and Simulation*, pages 299–317, 2012. doi: 10.1007/978-3-642-29758-8_16. [p286]

*Maliny Po*
*Monash University*
*Department of Econometrics and Business Statistics*
*Melbourne, Australia*
*ORCID: 0009-0008-4686-6631*
malinypo12@gmail.com

*S. Nathan Yang*
*Duke University*
*Durham, North Carolina*
*ORCID: 0009-0002-9985-1042*
nathan.s.yang@gmail.com

*H. Sherry Zhang*
*University of Texas at Austin*
*Department of Statistics and Data Sciences*
*Austin, Texas*
*ORCID: 0000-0002-7122-1463*
huizezhangsh@gmail.com

*Dianne Cook*
*Monash University*
*Department of Econometrics and Business Statistics*
*Melbourne, Australia*
*ORCID: 0000-0002-3813-7155*
dicook@monash.edu

# iAR: An R Package for Autoregressive Modeling Irregularly Observed Time Series

*by Felipe Elorrieta, Susana Eyheramendy, Wilfredo Palma, and Cesar Ojeda*

**Abstract** iAR is an R package that provides tools for handling autoregressive models irregularly observed, for one or two dimensions, stationary time series. The standard autoregressive (AR) model is not suitable for fitting irregularly observed time series, especially when the time gaps are large or randomly distributed. The iAR package implements a class of irregular autoregressive models designed to overcome this limitation. It includes the irregular autoregressive (iAR) model, the complex-valued extension (CiAR) that allows the estimation of negative values for the parameter, and a bivariate autoregressive model (BiAR). It further implements extensions of the iAR process for t-distributed and Gamma-distributed conditional models. The package includes tools for parameter estimation via maximum likelihood, as well as functions for simulation, forecasting, and interpolation. This paper provides a comprehensive overview of the iAR package, detailing its functionality and illustrating its application through examples on both simulated and real-world data.

## 1 Introduction

Time series analysis is typically performed on data observed at regular intervals, under the assumption of evenly spaced discrete time points (see, for instance, Box and Jenkins (1970); Brockwell and Davis (1991); Hyndman and Athanasopoulos (2021)). When the data are recorded at irregular time intervals, two main strategies are commonly adopted: modeling the series in continuous time, or transforming it to a regularly spaced sequence through interpolation (see Rehfeld et al. (2011)).

Continuous-time models rely on the assumption that consecutive observations are closely spaced. However, this condition is not always met, especially in applications with sparse or unevenly sampled data. The package introduced in this paper addresses this challenge by providing discrete-time models specifically designed for unequally spaced observations.

Although several general-purpose time series packages are available in R, most notably those described in Hyndman et al. (2026) and Hyndman and Khandakar (2008), which provide a comprehensive framework for modeling, forecasting, and evaluating regularly observed time series, there are currently few tools available for modeling irregularly observed time series. For instance, two notable packages that implement Continuous ARMA (CARMA) models are **growth** and **yuima** (Brouste et al., 2014). The main distinction is that **growth** allows incorporating fixed effects into the CARMA specification, whereas this is not supported in **yuima**. An extension of the CARMA model allowing a Hurst parameter $H$ to represent long-term memory is implemented in the **carfima** package.

CARMA models can also be estimated through more general frameworks for stochastic differential equations with irregular time points, such as the **sde** (McMillen, 2008) and ctsmr (Juhl et al., 2016) packages. These packages use maximum likelihood estimation combined with Kalman filtering techniques.

Multivariate irregular time series can be modeled using **Rdrw** and **ctsem**, which implement damped random walk processes. Additionally, several other packages in R are designed for specific tasks on irregular data. For instance, the **lomb** package (Ruf, 1999) implements the Lomb–Scargle periodogram, while the **RobPer** package (Thieler et al., 2016) allows the estimation of its extension, known as the Generalized Lomb–Scargle (GLS) periodogram (Zechmeister and Kürster, 2009), both of which are used for period estimation in irregularly observed time series.

Another example is the sour package (Edelson et al., 2017) which allows one to obtain the Discrete Correlation Function (DCF) with which one can obtain an estimator of the autocorrelation function under irregular sampling. Moreover, the package **BINCOR** (Polanco-Martinez et al., 2019) addresses the problem of estimating the correlation (and cross-correlation) between two series that are unevenly spaced — and possibly sampled at different timepoints — by using a "binned correlation" approach which resamples the irregular series into regular bins (taking into account persistence/autocorrelation) and then computes correlations or cross-correlations.

This paper presents **iAR**, a new R package that introduces novel discrete-time models to fit irregularly observed time series. Unlike the packages mentioned above, which primarily rely on continuous-time formulations, **iAR** adopts a discrete-time perspective, offering greater flexibility in the modeling and estimation process.

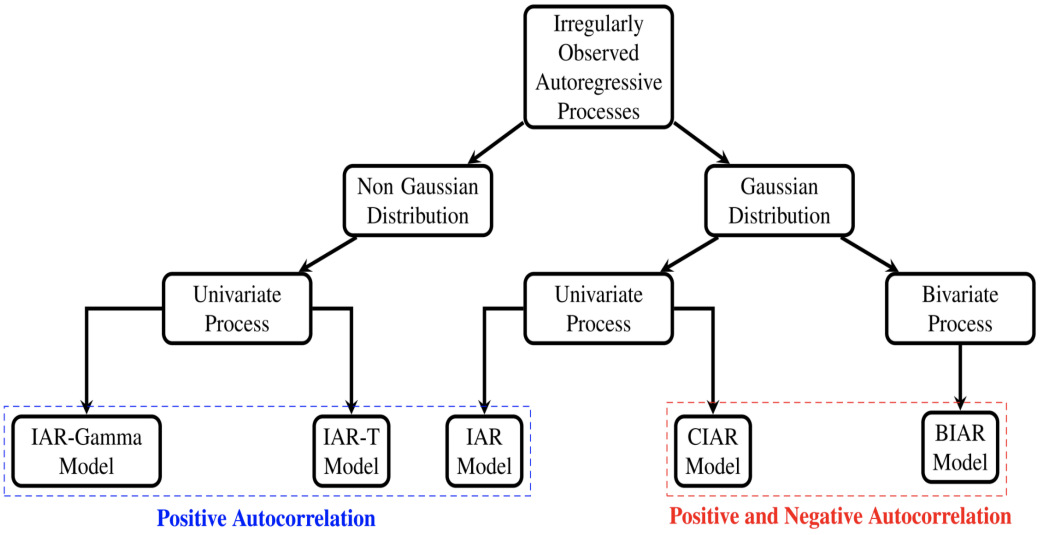



**Figure 1:** Flowchart of Irregularly Observed Autoregressive Processes.

The models implemented in **iAR** include the irregular autoregressive (iAR) process, its complex-valued extension (CiAR), and a bivariate version (BiAR). Additionally, the package provides tools to handle non-Gaussian data through extensions of the iAR model for $t$ and Gamma-distributed observations. These models allow for the estimation of negative autocorrelation, which cannot be captured by CARMA-based approaches.

**iAR** offers a range of functionalities for irregularly spaced data, including: (i) random generation of observation times; (ii) simulation of the irregular autoregressive processes; (iii) estimation of autocorrelation; (iv) model fitting via maximum likelihood; and (v) interpolation of missing values and forecasting. Furthermore, all models implemented in **iAR** are also compatible with regularly spaced stationary series, such as standard AR processes.

The package is built using the S7 object-oriented system (Vaughan et al., 2024), which provides a robust and modular framework for class and method definitions. Compared to earlier systems like S3 and S4, S7 enforces a clearer separation between class specification and method implementation, supports formal validation, and ensures consistent method dispatch. This structure improves code clarity, extensibility, and integration with other object-oriented tools in R. Internally, the package relies on the **zoo** package (Zeileis and Grothendieck, 2005) to manage indexed irregular time series efficiently, supporting multiple time representations, including numeric time scales, calendar-based indices (e.g., `Date` and `POSIXct`), and user-defined irregular sampling schemes.

The remainder of this paper is organized as follows: Section 2 introduces the modeling framework for irregularly observed autoregressive processes and discusses their applicability. Section 3 presents the procedures to obtain the maximum likelihood estimators for each of the models available in the **iAR** package. Section 4 describes the structure and methods implemented in the package through examples using simulated data. Section 5 reports the computational times of the estimation procedures and compares them with similar functions available in other packages. Section 6 presents applications of the **iAR** package to astronomical data. Finally, Section 7 concludes the paper with a discussion.

## 2 Irregularly observed autoregressive models

The five models available in the **iAR** package are presented in this section and are depicted in Figure 1. The choice of each model depends on the type of time series to be fitted. For example, if the data available are non-Gaussian distributed or not positively autocorrelated the appropriate model to fit the data may vary. In the following, each of the five models is briefly described.

### 2.1 iAR Model

The irregular Autoregressive (iAR) model was introduced by Eyheramendy et al. (2018). We present here a brief description of the model. Let $\{t_j\}$ be the irregular observational times for $j = 1, \ldots, n$. It is assumed that in general $t_j - t_{j-1}$ are not equal for $j = 2, \ldots, n$. The iAR model is defined by,

$$y_{t_j} = \phi^{t_j - t_{j-1}} y_{t_{j-1}} + \sigma \sqrt{1 - \phi^{2(t_j - t_{j-1})}}\, \varepsilon_{t_j}, \tag{1}$$

where $\varepsilon_{t_j}$ is a white noise sequence with zero mean and unit variance, $\sigma$ is the standard deviation of the sequence of observations $y_{t_j}$. Note that $\phi$ is the parameter that describes the autocorrelation of the process. The iAR model can be represented in a state-space system as follows,

$$X_{t_j} = \phi^{t_j - t_{j-1}} X_{t_{j-1}} + \epsilon_{t_j} \tag{2}$$

$$y_{t_j} = X_{t_j} + \omega_{y_{t_j}} \tag{3}$$

Note that, in this representation the transition matrix of dimension $1 \times 1$ is defined as $F_{t_j} = \phi^{t_j - t_{j-1}}$. On the other hand, the observation matrix is defined by $G_{t_j} = 1$. The measurement error of $y_{t_j}$ is defined by $\omega_{y_{t_j}}$ with known variance $\delta^2_{t_j}$ and the error of the transition equation is $\epsilon_{t_j}$ with variance $Q_{t_j} = \sigma^2(1 - \phi^{2(t_j - t_{j-1})})$.

Assuming Gaussian errors $\varepsilon_{t_j}$, i.e. $\varepsilon_{t_j} \sim N(0, 1)$, the iAR model is equivalent to the CARMA(1,0) or CAR(1) model. However, the iAR model can be extended to allow for non-Gaussian distributed data. The extensions implemented in the iAR package assume that the conditional moments of the iAR process follow either a Gamma or Student-t distribution. These two models are described next.

### 2.2 iAR-Gamma Model

The conditional mean and variance of the iAR model are:

$$\begin{aligned} \mathbb{E}(y_{t_j} | y_{t_{j-1}}) &= \mu + \phi^{t_j - t_{j-1}} y_{t_{j-1}} \\ \mathbb{V}(y_{t_j} | y_{t_{j-1}}) &= \sigma^2 \left(1 - \phi^{2(t_j - t_{j-1})}\right) \end{aligned}$$

respectively. If $y_{t_j} | y_{t_{j-1}} \sim \text{Gamma}(\alpha_{t_j}, \beta_{t_j})$, where $\alpha_{t_j}$ and $\beta_{t_j}$ are the shape and scale parameters respectively, then $y_{t_j}$ will be an iAR-Gamma process if $\alpha_{t_j}$ and $\beta_{t_j}$ are such that the following two equations are satisfied,

$$\begin{aligned} \alpha_{t_j} \beta_{t_j} &= \mu + \phi^{t_j - t_{j-1}} y_{t_{j-1}} \\ \alpha_{t_j} \beta^2_{t_j} &= \sigma^2 \left(1 - \phi^{2(t_j - t_{j-1})}\right) \end{aligned} \tag{4}$$

### 2.3 iAR-T Model

A similar approach can be used to define an iAR-T model. Suppose that $y_{t_j} | y_{t_{j-1}} \sim t_\nu(\lambda_{t_j}, \tau^2_{t_j})$ is a non-standardized Student-T distribution with mean $\lambda_{t_j}$, variance $\tau^2_{t_j}$ and degrees of freedom $\nu$. Then $y_{t_j}$ will be an iAR-T process if $\lambda_{t_j}$ and $\tau^2_{t_j}$ are defined by,

$$\begin{aligned} \lambda_{t_j} &= \mu + \phi^{t_j - t_{j-1}} y_{t_{j-1}} \\ \tau^2_{t_j} &= \frac{\nu - 2}{\nu} \sigma^2 \left(1 - \phi^{2(t_j - t_{j-1})}\right). \end{aligned} \tag{5}$$

### 2.4 CiAR Model

For irregularly observed time series, the models described above are well defined for positive values of the autocorrelation parameter $\phi$. The reason for this is that the time gap $\Delta_j = t_j - t_{j-1}$ can take any positive real value. Consequently, the $\phi$ parameter must be greater than zero since the result of a negative $\phi$ to the power of a real number, in general, is a complex number.

In order to address this issue, equation (1) can be extended to a complex process which allows for negative values of the autocorrelation of the process. This extension of the iAR model is called the Complex irregular Autoregressive (CiAR) model (Elorrieta et al., 2019). The CiAR process is characterized by a complex autocorrelation parameter defined by $\phi = \phi^R + i\phi^I$. The CiAR model can be represented by the following state space system,

$$\begin{pmatrix} y^*_{t_j} \\ z^*_{t_j} \end{pmatrix} = \begin{pmatrix} \alpha_{t_j} & -\beta_{t_j} \\ \beta_{t_j} & \alpha_{t_j} \end{pmatrix} \begin{pmatrix} y^*_{t_{j-1}} \\ z^*_{t_{j-1}} \end{pmatrix} + \sigma\sqrt{1 - \left|\phi^{t_j - t_{j-1}}\right|^2} \begin{pmatrix} \varepsilon_{y_{t_j}} \\ \varepsilon_{z_{t_j}} \end{pmatrix} \tag{6}$$

$$y_{t_j} = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} y^*_{t_j} \\ z^*_{t_j} \end{pmatrix} + \omega_{y_{t_j}} \tag{7}$$

In this case, the transition equation is two-dimensional with a transition matrix $F_{t_j} = \begin{pmatrix} \alpha_{t_j} & -\beta_{t_j} \\ \beta_{t_j} & \alpha_{t_j} \end{pmatrix}$. In addition, the observation matrix is defined by $G_{t_j} = \begin{pmatrix} 1 & 0 \end{pmatrix}$. $\alpha_{t_j}$ and $\beta_{t_j}$ are the reparameterized coefficients of the model, given by $\alpha_{t_j} = |\phi|^{\delta_j}\cos(\delta_j\psi)$ and $\beta_{t_j} = |\phi|^{\delta_j}\sin(\delta_j\psi)$ respectively, where $\delta_j = t_j - t_{j-1}$ and $\psi = \arccos\left(\frac{\phi^R}{|\phi|}\right)$. The variance of the errors of the transition and observations equation are defined by $Q_{t_j} = \sigma^2(1 - \left|\phi^{t_j - t_{j-1}}\right|^2)$ and $\delta^2_{t_j}$ respectively. As in the iAR model, $\sigma^2$ denotes the marginal variance of the process.

### 2.5 BiAR Model

We propose a bivariate extension of the CiAR model which we called the bivariate irregular autoregressive (BiAR) (Elorrieta et al., 2021) model. Unlike the CiAR model, the BiAR model is defined by a two-dimensional observation equation. The state space representation of the BiAR model is defined as,

$$\begin{pmatrix} y^*_{t_j} \\ z^*_{t_j} \end{pmatrix} = \begin{pmatrix} \alpha_{t_j} & -\beta_{t_j} \\ \beta_{t_j} & \alpha_{t_j} \end{pmatrix} \begin{pmatrix} y^*_{t_{j-1}} \\ z^*_{t_{j-1}} \end{pmatrix} + \begin{pmatrix} \varepsilon_{y_{t_j}} \\ \varepsilon_{z_{t_j}} \end{pmatrix} \tag{8}$$

$$\begin{pmatrix} y_{t_j} \\ z_{t_j} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} y^*_{t_j} \\ z^*_{t_j} \end{pmatrix} + \begin{pmatrix} \omega_{y_{t_j}} \\ \omega_{z_{t_j}} \end{pmatrix} \tag{9}$$

where $\varepsilon_{t_j} = \begin{pmatrix} \varepsilon_{y_{t_j}} \\ \varepsilon_{z_{t_j}} \end{pmatrix}$ can be decomposed as $\varepsilon_{t_j} = L_{t_j}\xi_{t_j}$, where $L_{t_j}$ is a lower triangular matrix obtained by the Cholesky decomposition of the covariance matrix of $\varepsilon_{t_j}$ and $\xi_{t_j}$ is a bivariate normal random variable with zero mean and variance $\Sigma_\xi = \begin{pmatrix} 1 & \rho_\xi \\ \rho_\xi & 1 \end{pmatrix}$. As in the CiAR model, $\alpha_{t_j}$ and $\beta_{t_j}$ are a reparameterization of the parameters $\phi^R$ and $\phi^I$. Note that, the BiAR model is defined by three parameters $\phi^R$,$\phi^I$ and $\rho_\xi$ which can be interpreted as the autocorrelation parameter $\phi^R$, the cross correlation parameter $\phi^I$ and the contemporary correlation parameter $\rho_\xi$.

### 2.6 Considerations

All irregularly observed time series models implemented in this package depend on the observational gaps through both the autoregressive coefficient and the innovation variance. This may lead to some edge cases. For instance, when two consecutive observation times are identical ($\Delta_j = 0$), the autoregressive coefficient satisfies $\phi^{\Delta_j} = 1$ and the innovation variance becomes zero. In practice, this may lead to numerical instability during estimation procedures. On the other hand, for very large time gaps, the autoregressive coefficient tends to zero, implying that consecutive observations become approximately independent.

## 3 Estimation algorithms

### 3.1 Likelihood of the processes

Assume that the $y_{t_j}$ sequence follows an iAR model, if we further assume a Gaussian distribution, the negative log-likelihood of the $y_{t_j}$ sequence is given by,

$$\ell(\theta) = \frac{n}{2}\log(2\pi) + \frac{1}{2}\sum_{j=1}^{n}\log\nu_{t_j} + \frac{1}{2}\sum_{j=1}^{n}\frac{e^2_{t_j}}{\nu_{t_j}}, \tag{10}$$

where $\theta = (\phi, \sigma^2)$ is the vector of the parameters of the model. The initial values are defined

by $e_{t_1} = y_{t_1}$, $\nu_{t_1} = \sigma^2 + \delta_{t_1}^2$ and $\hat{y}_{t_1} = 0$, while $\hat{y}_{t_j} = \hat{\phi}^{t_j - t_{j-1}} y_{t_{j-1}}$, $e_{t_j} = y_{t_j} - \hat{y}_{t_j}$ and $\nu_{t_j} = \sigma^2(1 - \phi^{2(t_j - t_{j-1})}) + \delta_{t_j}^2$ for $j = 2, \ldots, n$.

For the iAR-Gamma model, the log-likelihood of the conditional distribution of $y_{t_j}|y_{t_{j-1}}$ can be written as,

$$\ell(\theta) = -\left(\alpha_{t_j}\right)\log\beta_{t_j} - \log\Gamma\left(\alpha_{t_j}\right) - \frac{1}{\beta_{t_j}}y_{t_j} + \left(\alpha_{t_j} - 1\right)\log y_{t_j} \tag{11}$$

where $\theta = (\alpha_{t_j}, \beta_{t_j})$ is the vector of the parameters of the model. Note from equation (4) that $\alpha_{t_j}$ and $\beta_{t_j}$ depend on the parameters $\phi$, $\sigma^2$ and $\mu$. The initial value is defined as having a Gamma distribution with shape and scale parameters equal to one, i.e. $y_{t_1} \sim \text{Gamma}(1, 1)$.

For the iAR-T model, the log-likelihood of the conditional distribution of $y_{t_j}|y_{t_{j-1}}$ can be written as,

$$\ell(\theta) = -\frac{1}{2}\log\tau_{t_j}^2 - \frac{\nu+1}{2}\log\left(1 + \frac{1}{\nu}\left(\frac{y_{t_j} - \lambda_{t_j}}{\tau_{t_j}}\right)^2\right) \tag{12}$$

where $\theta = (\nu, \tau_{t_j}, \lambda_{t_j})$ is the vector of the parameters of the model. Note from equation (5) that $\tau_{t_j}$ and $\lambda_{t_j}$ depend on the parameters $\phi$, $\sigma^2$. The initial value is defined by $y_{t_1} \sim N(0, 1)$.

### 3.2 Kalman recursions

The state-space representations of the models described in Section 2 allow one to obtain the maximum likelihood estimators of the parameters of the models using the Kalman recursions. Given the transition matrix $F_{t_j}$, the observation matrix $G_{t_j}$, the transition error variance $Q_{t_j}$ and the observation error variance $R_{t_j}$ the Kalman recursions can be obtained as,

$$\begin{aligned}
\Lambda_{t_j} &= G\Omega_{t_j}G' + R_{t_j} \\
\Theta_{t_j} &= F_{t_j}\Omega_{t_j}G' \\
\Omega_{t_{j+1}} &= F_{t_j}\Omega_{t_j}F_{t_j}' + Q_{t_j} - \Theta_{t_j}\Lambda_{t_j}^{-1}\Theta_{t_j}' \\
\nu_{t_j} &= X_{t_j} - G\hat{X}_{t_j} \\
\hat{X}_{t_{j+1}} &= F_{t_j}\hat{X}_{t_j} + \Theta_{t_j}\Lambda_{t_j}^{-1}\nu_{t_j}
\end{aligned} \tag{13}$$

Using the Kalman recursions parameters, we can obtain the maximum likelihood estimator from the log-likelihood function defined as,

$$\ell(\phi) \propto -\frac{1}{2}\sum_{j=1}^{n}\left(\log(|\Lambda_{t_j}|) + \nu_{t_j}'\Lambda_{t_j}^{-1}\nu_{t_j}\right) \tag{14}$$

## 4 Structure and usage of the iAR package

### 4.1 Object-oriented system

The iAR package is structured using the S7 object-oriented system, which provides a formal and modular framework for defining classes and methods. This design enables a coherent and consistent internal organization across the different autoregressive models implemented in the package: iAR, CiAR, and BiAR.

Each model class provides a standardized set of methods for tasks such as maximum likelihood estimation, computation of fitted values, forecasting, and interpolation. The implementation of S7 ensures that these methods are defined uniformly across models, while still allowing for model-specific behavior when needed. Also, extensive validations have been implemented within the package functions to ensure robust behavior under a variety of input scenarios and model configurations.

For handling irregular time indices, the package leverages the zoo package, which offers flexible and efficient tools for managing time-based data in R. This integration allows for seamless manipulation and alignment of time series with irregular observational times, a fundamental feature for the models supported by the package.

### 4.2 The R interface

Our package can be installed and loaded in R by the usual commands:

```
install.packages("iAR")
library("iAR")
```

The classes and methods implemented in the **iAR** package are summarized in Table1. Each class—iAR, CiAR, and BiAR—includes a core set of methods for simulating time series (sim), estimating parameters via maximum likelihood (kalman), computing fitted values (fit), summarizing fitted models (summary), interpolating missing observations (interpolation), forecasting future values (forecast), and visualizing results through various plotting functions (plot, plot_fit, plot_forecast). The iAR class additionally includes loglik for computing the log-likelihood directly.

**Table 1:** Classes and methods of the iAR package.

| Class | Method | Description |
|---|---|---|
| iAR | sim | Sampling from an iAR model |
| | loglik | MLE estimation of iAR model |
| | kalman | MLE estimation of iAR model |
| | fit | Fitted values of iAR model |
| | summary | Summary of the iAR model |
| | interpolation | Interpolation from the iAR model |
| | forecast | Forecast from iAR model |
| | plot | Plot of the series |
| | plot_fit | Plot of the fitted values |
| | plot_forecast | Plot of the forecasted values |
| CiAR | sim | Sampling from a CiAR model |
| | kalman | MLE estimation of CiAR model |
| | fit | Fitted values of CiAR model |
| | summary | Summary of the CiAR model |
| | interpolation | Interpolation from the CiAR model |
| | forecast | Forecast from CiAR model |
| | plot | Plot of the series |
| | plot_fit | Plot of the fitted values |
| | plot_forecast | Plot of the forecasted values |
| BiAR | sim | Sampling from a BiAR model |
| | kalman | MLE estimation of BiAR model |
| | fit | Fitted values of BiAR model |
| | summary | Summary of the BiAR model |
| | interpolation | Interpolation from the BiAR model |
| | forecast | Forecast from BiAR model |
| | plot | Plot of the series |
| | plot_fit | Plot of the fitted values |
| | plot_forecast | Plot of the forecasted values |
| utilities | gentime | Generating time points based on a statistical distribution |
| | pairingits | Pairing the observational times of two time series |
| | phase | Computing phased time series |
| | harmonicfit | Harmonic fit to time series |

Most methods are consistently available across the three main classes, with the only exception being the loglik method. This method is implemented exclusively for the iAR class, while parameter estimation for the CiAR and BiAR classes relies entirely on the Kalman filtering framework. Furthermore, since the same methods are used for the Gaussian and non-Gaussian versions of the iAR model (e.g., iAR-Gamma and iAR-T), the underlying distributional assumptions are controlled via arguments passed to these functions. This unified interface simplifies the application of various models to irregularly observed time series.

Additionally, the utilities class provides support functions that enhance the modeling workflow. The gentime method generates irregular time points based on a statistical distribution, which is useful when the time slot is not provided by the user. The pairingits method allows pairing observational

times from two irregularly observed time series, facilitating multivariate modeling with the BiAR class. The phase method computes phased time series, which is particularly useful for visualizing periodic behaviors in irregularly observed data. Finally, the harmonicfit method performs harmonic fitting on time series, providing a flexible tool for removing dominant periodic components before autoregressive modeling.

To improve computational efficiency, several functions in the methods were implemented in C++ using the package **Rcpp** (Eddelbuettel and Francois, 2011). In particular, we use the C++ linear algebra library Armadillo, accessed via the **RcppArmadillo** package (Eddelbuettel and Sanderson, 2014).

### 4.3 Examples

The sim method allows the user to generate realizations from any of the five models available in this package. To simulate a standard iAR process, we first create an object of class iAR as follows:

```
model_norm <- iAR(family = "norm", coef = 0.9)
```

Here, the model is defined by specifying the distribution family and the autoregressive coefficient. To simulate an iAR process, the argument coef, which corresponds to the model autoregressive parameter, must take a value between 0 and 1.

The time points can be specified by the user when the iAR object is defined. However, if the times argument is left empty, the sim method will automatically generate a time array of length n, using a random distribution defined by the gentime method from the utilities class. By default, an array of size n=100 is generated from a mixture of two exponential distributions with rate parameters lambda1 = 130 and lambda2 = 6.5, and weights p1 = 0.15 and p2 = 0.85, respectively.

Once the model is defined, the process can be simulated using the sim method as follows:

```
model_norm <- sim(model_norm)

#> Generating irregular time points using the gentime method of the class utilities...
```

To visualize the simulated process, the plot method from the iAR package can be used:

```
plot(model_norm, main = "", ylab = "Values", xlab = "Time", ylim = c(-3, 3),type = "o",pch=20)
```

The simulated iAR process is shown in Figure 2. In this figure, the irregular time gaps and the stationary behavior of the process can be observed. To estimate the parameter phi, the kalman method is available in our package. This method uses Kalman recursions to obtain the maximum likelihood estimator based on the state-space representation of the iAR model. Applying this method to the previously simulated process yields the following result:

```
model <- kalman(model_norm)
print(model@coef)

#> [1] 0.8588356

print(model@kalmanlik)

#> [1] 0.5328931
```

Another option for estimating the phi parameter of the iAR model is to use the loglik method. This method computes the maximum likelihood estimator based on the full likelihood function. The loglik method can be applied as follows:

```
model <- loglik(model_norm)
print(model@coef)

#> [1] 0.8548406

print(model@loglik)

#> [1] 117.7532
```

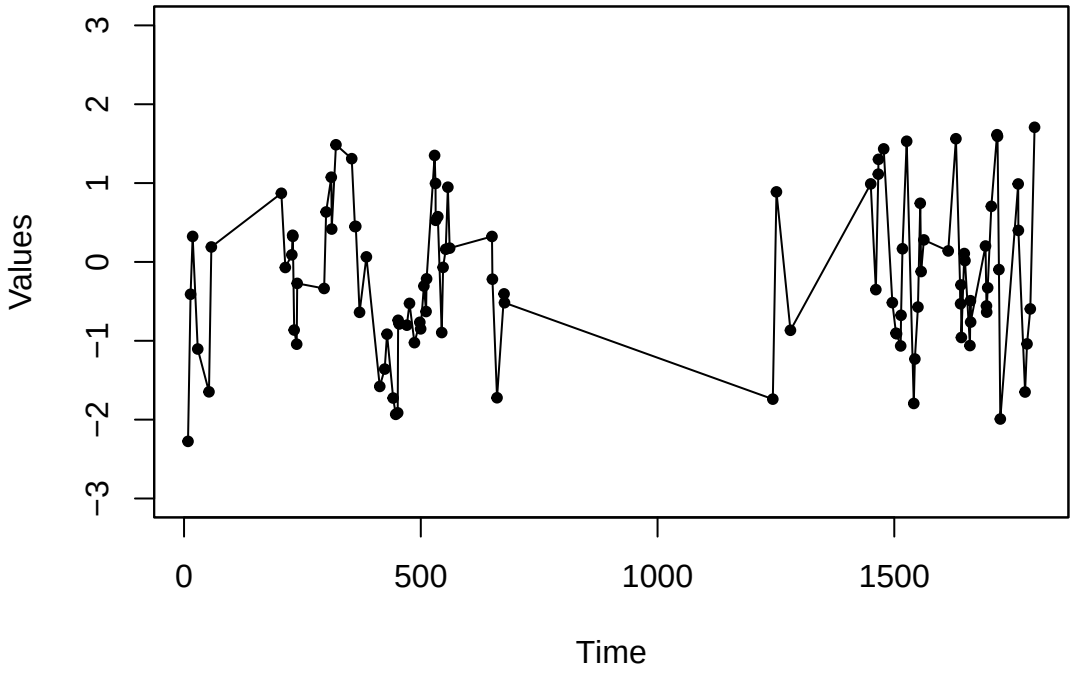



**Figure 2:** Simulated Irregularly Observed Autoregressive process.

To assess whether the parameter $\phi$ is significantly different from zero, the summary method can be used. This method returns the p-value of the t-test associated with this parameter. To use it properly, the hessian slot must be set to TRUE in the iAR object beforehand. If the p-value returned by this function is less than a given significance level, we can conclude that the parameter is significantly greater than zero. Furthermore, the summary method also provides additional information about the fitted model, including information criteria and residual diagnostics. The usage of this method is as follows:

```
model_norm@hessian=TRUE
model <- loglik(model_norm)
summary(model)

#> iAR model
#>
#> Family:
#>   norm
#>
#> Coefficients:
#>      Estimate St. Error t value Pr(>|t|)
#> phi     0.85      0.03   29.51     0.00
#>
#> Information criteria:
#> AIC: 237.51
#> BIC: 240.11
#>
#> Residual diagnostics:
#> ACF lag 1: -0.04
#> Ljung-Box test p-value (lag=1): 0.68
#> Ljung-Box test p-value (lag=10): 0.71
```

As shown in this example, the obtained p-value is very small. Consequently, the parameter is

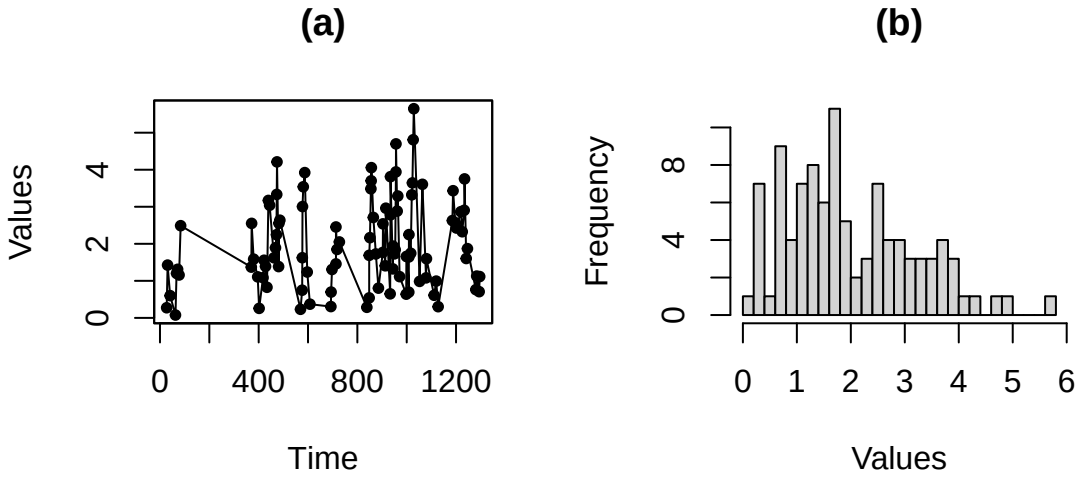



**Figure 3:** Simulated iAR-Gamma process. Figure a) shows the simulated irregularly observed time series. Figure b) shows the histogram of the values of the simulated process.

significantly greater than zero. Furthermore, the diagnostics of the fitted model suggest no remaining autocorrelation in the residuals, supporting the whiteness of the errors.

For the remaining models available in this package, the methods are structured similarly to the previous example (as shown in Table 1). Thus, the procedure for generating a non-Gaussian iAR model begins by creating an object of the iAR object with either a gamma or t distribution. Below, we illustrate the simulation of an iAR-Gamma process.

```
model_gamma <- iAR(family = "gamma", coef = 0.9, variance=1, mean=1, hessian=TRUE)
model_gamma <- iAR::sim(model_gamma)

#> Generating irregular time points using the gentime method of the class utilities...
```

where the additional parameters mean and variance, representing the mean and variance of the simulated process, respectively, must be defined. Figure 3 shows a simulated iAR-Gamma process with the distribution of the simulated data.

```
par(mfrow=c(1,2))
plot(model_gamma,main="(a)",ylab="Values",xlab="Time",type = "o",pch=20)
hist(model_gamma@series,breaks=25,main="(b)",xlab="Values")
```

Note in Figure 3-b that the simulated values have a left-skewed distribution typical of the gamma family. To estimate the parameters of the iAR-Gamma process, the loglik method can be used as follows:

```
model_gamma <- loglik(model_gamma)
summary(model_gamma,diagnostic=FALSE)

#> iAR model
#>
#> Family:
#>   gamma
#>
#> Coefficients:
#>           Estimate St. Error t value Pr(>|t|)
#> phi           0.90      0.01   83.80     0.00
#> mean          1.01      0.04   22.75     0.00
#> variance      0.83      0.14    5.88     0.00
#>
#> Information criteria:
#> AIC: 179.25
#> BIC: 187.07
```

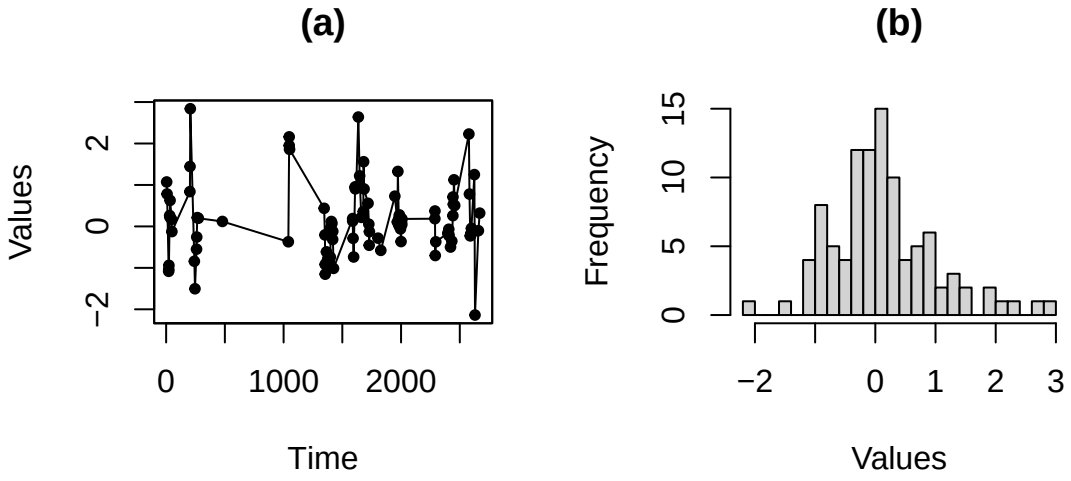



**Figure 4:** Simulated iAR-T process. Figure a) shows the simulated irregularly observed time series. Figure b) shows the histogram of the values of the simulated process.

In this example, the estimated values for the three parameters are close to their true values, indicating that the estimation procedure performs well. Additionally, the near-zero p-values confirm the statistical significance of the parameter estimates. Note that we use diagnostic = FALSE in the summary method, which suppresses the residual diagnostics to simplify the presentation of the output.

To generate a heavy-tailed iAR process, the method sim can be applied to an iAR object of the family t. The family t requires the additional argument df which specifies the degrees of freedom of the conditional Student's t distribution. The procedure to generate an iAR-t process is as follows:

```
model_t <- iAR(family = "t", coef = 0.9, sigma=1, df=3, hessian=TRUE)
model_t <- iAR::sim(model_t)

#> Generating irregular time points using the gentime method of the class utilities...

par(mfrow=c(1,2))
plot(model_t,main="(a)",ylab="Values",xlab="Time",type = "o",pch=20)
hist(model_t@series,breaks=25,main="(b)",xlab="Values")
```

Note in Figure 4 that the simulated process exhibits a heavy tail distribution. In order to estimate the parameters of this process via maximum likelihood, the method loglik must be used as follows:

```
model_t <- loglik(model_t)
summary(model_t,diagnostic=FALSE)

#> iAR model
#>
#> Family:
#>   t
#>
#> d.f.:
#>   3
#>
#> Coefficients:
#>        Estimate St. Error t value Pr(>|t|)
#> phi       0.90      0.02   42.62     0.00
#> sigma     0.96      0.23    4.25     0.00
#>
#> Information criteria:
#> AIC: 175.25
#> BIC: 180.46
```

This method allows estimation of the parameters phi and sigma from the iAR-T model. The standard errors and p-values indicate that the estimates are statistically significant.

To generate a CiAR process, a CiAR class object must first be created. The argument coef, which corresponds to a vector containing the coefficients of the CiAR model must be defined. Finally, the sim method is used to simulate the CiAR process. Note that in the first examples, the observational times are not provided, and are therefore generated by default. To illustrate how observational times can be specified by the user, the gentime function is manually used to generate them. In this case, a mixture of two exponential distributions is considered with means lambda1 = 15 and lambda2 = 2.5, and weights p1 = 0.15 and p2 = 0.85, respectively, as follows:

```
times<-gentime(n=200, distribution = "expmixture", lambda1 = 15, lambda2 = 2.5,
               p1 = 0.15, p2 = 0.85)
model_CiAR <- CiAR(coef = c(0.9, 0.0),times=times@times)
model_CiAR <- sim(model_CiAR)
```

To estimate the parameters of the CiAR model, the kalman method must be used. If the argument hessian = TRUE is specified, the significance of the estimated parameters can also be obtained, as shown below:

```
model_CiAR@hessian=TRUE
model_CiAR <- kalman(model_CiAR)
summary(model_CiAR,diagnostic=FALSE)

#> CiAR model
#>
#> Coefficients:
#>      Estimate St. Error t value Pr(>|t|)
#> phiR     0.91       0.10   8.86     0.00
#> phiI     0.04       0.48    0.08     0.94
#>
#> Information criteria:
#> AIC: 298.27
#> BIC: 304.86
```

As shown, this method provides both the parameter estimates and the reduced negative log-likelihood evaluated at the coefficients phiR and phiI. The estimates obtained are accurate for both parameters.

Additionally, to obtain the fitted values from the CiAR model, the fit method must be applied. This method returns the fitted values of a time series observed at irregular time points, given the parameters phiR and phiI, as follows:

```
model_CiAR <- fit(model_CiAR)
plot_fit(model_CiAR,type = "o",pch=20,ylab="Values",xlab="Time")
```

Figure 5 shows the fitted values estimated with the method fit over the simulated CiAR process. The method forecast applied on a CiAR object allows forecasting from the CiAR model given the coefficients and the tAhead which corresponds to the time gap in which the forecast is made. In the following example, a forecast 1.5 days in the future is performed:

```
forCIAR <- forecast(model_CiAR, tAhead=1.5)
forCIAR@forecast

#> [1] -0.6433988
```

It is important to notice that the forecast predictor of the CiAR process (as well as for any autoregressive process) converges to the mean for large prediction horizons.

The sim method applied on a CiAR object also allows one to generate an irregularly observed autoregressive process with negative autocorrelation as follows:

```
par(mfrow=c(1,2))
model_CiAR <- CiAR(times = times@times,coef = c(0.9, 0))
model_CiAR <- sim(model_CiAR)
```

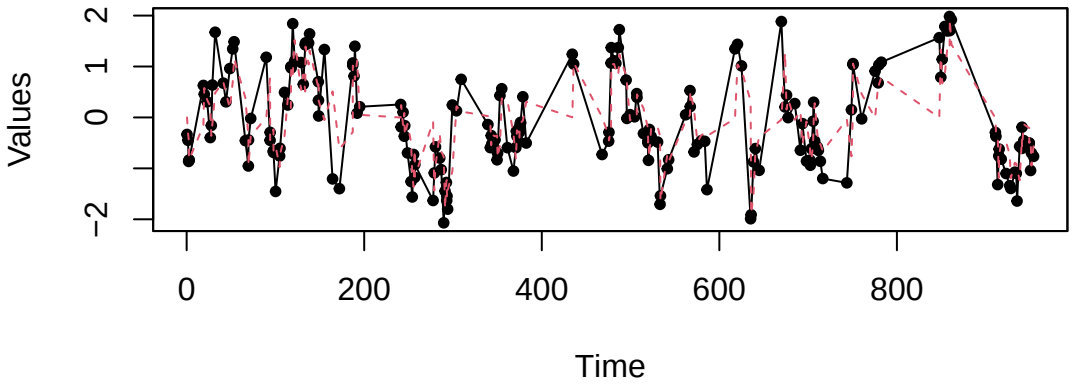



**Figure 5:** Simulated Complex Irregularly Observed Autoregressive process. The blue line corresponds to the fitted values estimated with the Kalman recursions.

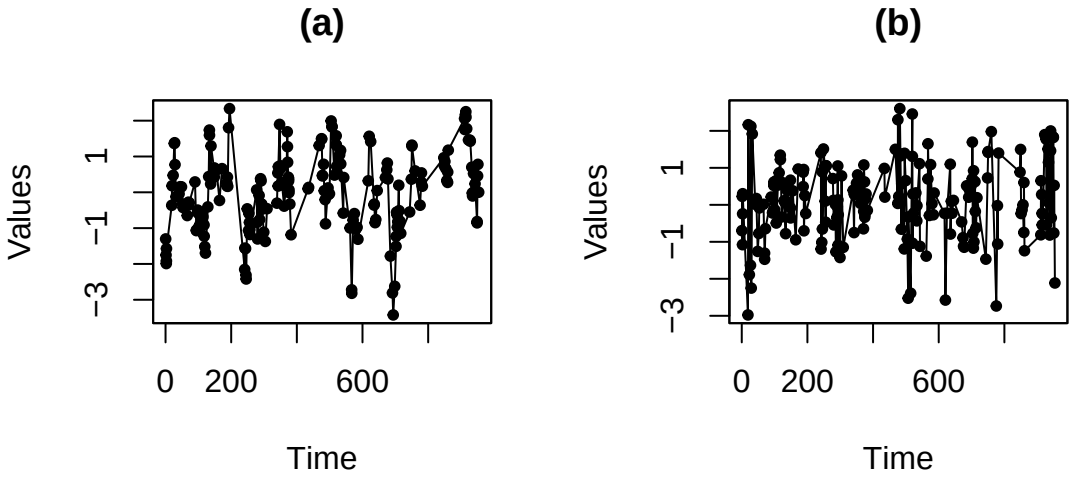



**Figure 6:** Simulated Complex Irregularly Observed Autoregressive processes with positive autocorrelation of order one (Figure (a)) and negative autocorrelation of order one (Figure (b)).

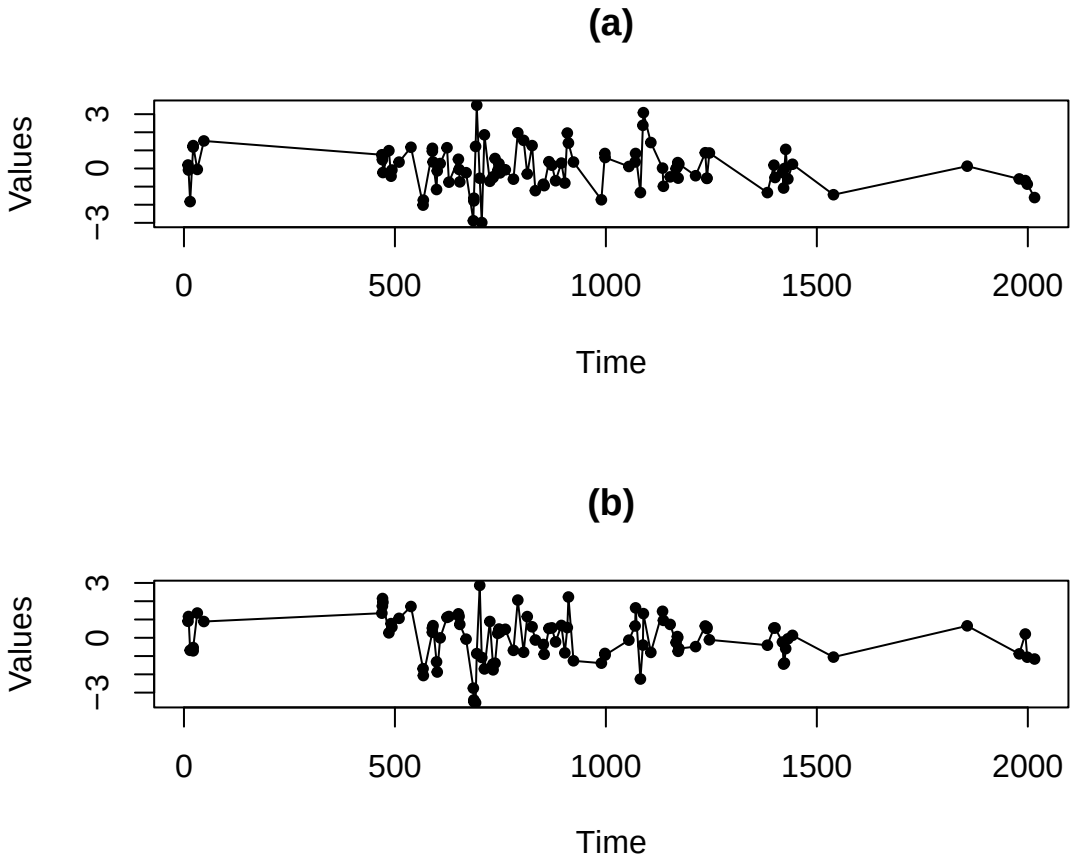



**Figure 7:** Simulated Bivariate Irregularly Observed Autoregressive process.

```
plot(model_CiAR,ylab="Values",xlab="Time",main="(a)",type = "o",pch=20)
model_CiAR <- CiAR(times = times@times,coef = c(-0.9, 0))
model_CiAR <- sim(model_CiAR)
plot(model_CiAR,ylab="Values",xlab="Time",main="(b)",type = "o",pch=20)
```

Figure 6 shows the CiAR process simulated with both positive and negative autocorrelation. Note that the time series with negative autocorrelation shows more frequent oscillations around zero, while the positively correlated process exhibits a more persistent behavior.

The last model implemented in this package is the BiAR process. The BiAR class provides functionality for working with this model. To simulate a BiAR process, the argument coef must be defined as a vector of length two, corresponding to the model coefficients. As we mentioned for the previous examples, the time points can be provided by the user or automatically generated by a random distribution. Additionally, a parameter rho can be specified to indicate the correlation between the two components of the bivariate process. The use of the sim method in this class is as follows:

```
model_BiAR <- BiAR(coef = c(0.9, 0.3), rho = 0.9)
model_BiAR <- sim(model_BiAR)

#> Generating irregular time points using the gentime method of the class utilities...

par(mfrow=c(2,1))
plot(model_BiAR@times,model_BiAR@series[,1],main="(a)",xlab="Time",ylab="Values",
     type = "o",pch=20)
plot(model_BiAR@times,model_BiAR@series[,2],main="(b)",xlab="Time",ylab="Values",
     type = "o",pch=20)
```

Figure 7 shows the simulated BiAR process. As expected, the two time series display similar behavior due to the value of the contemporary correlation parameter $\rho = 0.9$. To estimate the

parameters of the BiAR model and assess their significance, the method kalman should be used with the argument hessian = TRUE, as shown below:

```
model_BiAR@hessian=TRUE
model_BiAR <- kalman(model_BiAR, niter = 20)
summary(model_BiAR,diagnostic=FALSE)

#> BiAR model
#>
#> Coefficients:
#>      Estimate St. Error t value Pr(>|t|)
#> phiR     0.90       0.01   142.40     0.00
#> phiI     0.30       0.01    33.38     0.00
#>
#> Information criteria:
#> AIC: -7890.85
#> BIC: -7885.64
```

In addition, the method fit allows estimation of the contemporary correlation parameter rho. The usage of this method is the following:

```
model_BiAR <- fit(model_BiAR)
model_BiAR@rho

#> [1] 0.8887763
```

These results show accurate estimation of the three parameters of the BiAR model.

## 5 CPU Performance

In this section, we evaluate the computational performance of the functions implemented in the **iAR** package for R. First, the computational times are compared with respect to other functions available in R for fitting irregularly spaced time series models. The functions included in the comparison are the following: carma function from the **growth** package, drw function from the **Rdrw** package and the loglik method from the iAR class of our **iAR** package.

The microbenchmark function from the package of the same name is used for the comparison of the computational times of the functions mentioned above. For this experiment, an irregularly observed time series of length 200 was generated. Each function was executed 100 times on the generated time series. The distribution of execution times (in microseconds) is presented in Figure 8. Note that the loglik method from our package estimates the model parameters in significantly less time than the other functions analyzed.

In a second experiment, we assessed how the computational time of the estimation methods implemented in our package scales with the sample size of simulated time series. Figure 9 shows the mean computation times of the `loglik` and `kalman` methods from the `iAR` class, and the `kalman` method from the `CiAR` and `BiAR` classes, for sample sizes ranging from 50 to 1000 observations, after 100 repetitions. The results show an approximately linear growth of computation time with sample size. As expected, the estimation methods for the `iAR` model are considerably faster than those for the `CiAR` and `BiAR` models, reflecting their lower computational complexity.

In addition, Table 2 reports the mean CPU times (in seconds). For example, the iAR model increases from $5 \times 10^{-4}$ seconds (n = 50) to 0.0035 seconds (n = 1000), while the CiAR and BiAR models increase from 0.0073 to 0.1552 and from 0.0108 to 0.2185 seconds, respectively. These results are consistent with the approximately linear growth observed in the previous figure. In particular, when n increases by a factor of 20, the computation time increases by a factor of about 20 for CiAR and BiAR, and by a factor of about 5 for iAR.

## 6 Applications of the iAR package on real datasets

### 6.1 Identification of Multiperiodic time series

In astronomy, there are surveys where finding irregular time series with periodical behavior is not rare. Furthermore, some astronomical objects can have more than one significant period. Generally, this

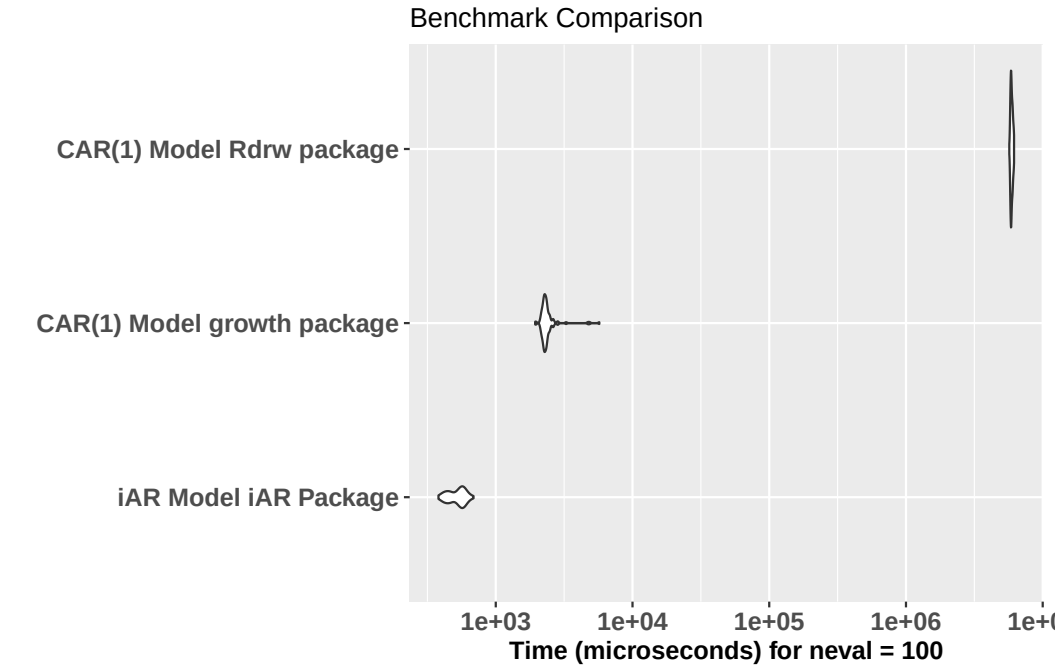



**Figure 8:** Distribution of the computation times (in microseconds) of the functions carma (growth package), drw (Rdrw package) and the loglik method (iAR package).

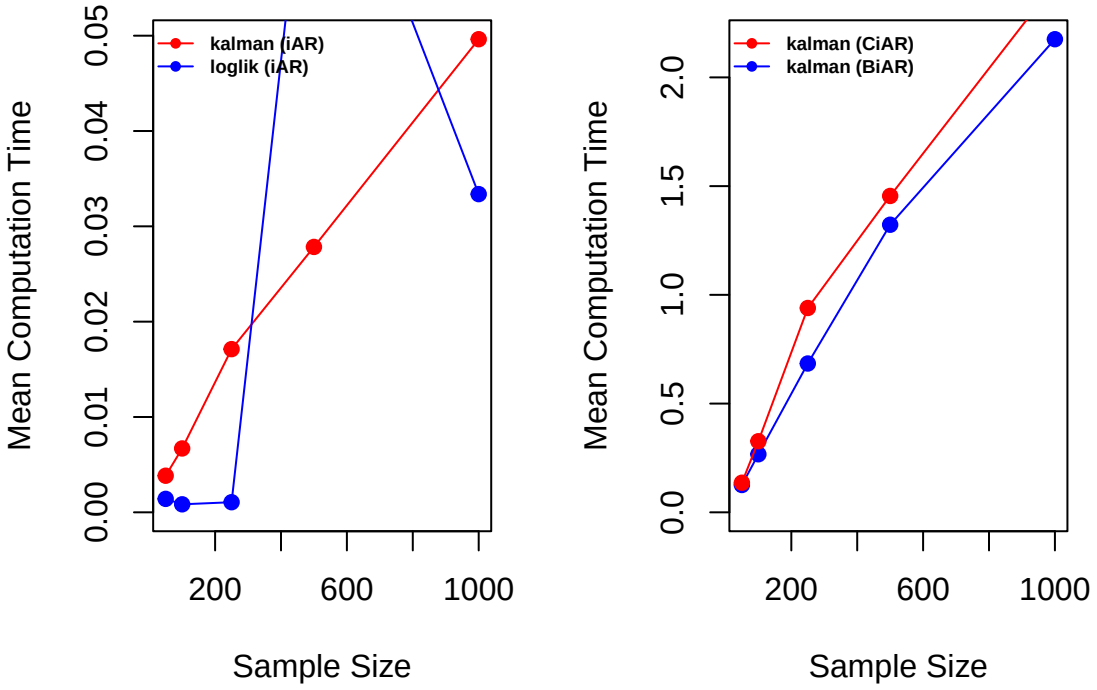



**Figure 9:** Mean computation time (in seconds) of the estimation methods in the iAR package.

**Table 2:** Mean computation time (in seconds) of the estimation methods in the iAR package

| n | iAR | CiAR | BiAR |
|---|---|---|---|
| 50 | 0.0005 | 0.0073 | 0.0108 |
| 100 | 0.0006 | 0.0163 | 0.0238 |
| 250 | 0.0009 | 0.0399 | 0.0563 |
| 500 | 0.0017 | 0.0760 | 0.1125 |
| 1000 | 0.0035 | 0.1552 | 0.2185 |

happens in specific types of variable stars, which are called multi-periodic variable stars. In the **iAR** package an example of this type of irregular time series can be found. This dataset is called dmcep and corresponds to the time series of the magnitude of the brightness of a double mode cepheid (DMCEP) variable star observed by the OGLE survey (Udalski et al., 2015). This time series has been irregularly observed and each observation has associated an observational error. The dataset contains 191 observations and three attributes: the observational times (t), the brightness magnitudes (m), and the known variance of the measurement errors (merr). The dominant frequency of this time series is 0.741, while the second frequency is 0.543. Figure 10 (a) shows the raw time series of this DMCEP variable star. In addition, we folded this time series by its first period, using the method phase of this package. The folded time series is presented in the Figure 10 (b) and reflects the periodical behavior of the brightness of this astronomical object.

Due to the periodic behavior of this irregular time series, a harmonic model must be fitted to it. For this purpose the harmonicfit method of the iAR package can be used. This function requires specifying the number of harmonic components (nham) to include in the model. In this application, the default value nham=4 was used. Later, we fitted an iAR model to the residuals of the harmonic model to verify if the whole time dependence structure of this time series was explained by this model. The parameter estimated by the iAR model was $\hat{\phi} = 0.02$ , with an associated p-value of 0.42, suggesting no significant autocorrelation after the harmonic fit.

```
data(dmcep)
f1=0.741
o1<-harmonicfit(data=dmcep,f1=f1)
y=o1@residuals
y=y/sd(y)
merr=dmcep$merr/sd(y)
x <- iAR(family = "norm", times = dmcep[,1], series = y, series_esd = merr, hessian=TRUE)
x = loglik(x)
summary(x)

#> iAR model
#>
#> Family:
#>   norm
#>
#> Coefficients:
#>      Estimate St. Error t value Pr(>|t|)
#> phi     0.02      0.02   0.81     0.42
#>
#> Information criteria:
#> AIC: 528.77
#> BIC: 532.02
#>
#> Residual diagnostics:
#> ACF lag 1: -0.13
#> Ljung-Box test p-value (lag=1): 0.06
#> Ljung-Box test p-value (lag=10): 0.03
```

However, the residual diagnostics indicate some remaining dependence structure. This result is further confirmed by fitting the CiAR model to the residuals of the harmonic model, which yields an estimated autocorrelation parameter of $-0.63$. This result indicates the presence of negative autocorrelation in the residuals, which explains why it is captured by the CiAR model.

Figure 10 (d) shows that the residuals of the harmonic fit folded using the second period of this variable star exhibit a clear serial correlation. Later, this serial correlation can be modeled using a new harmonic model that considers the second period.

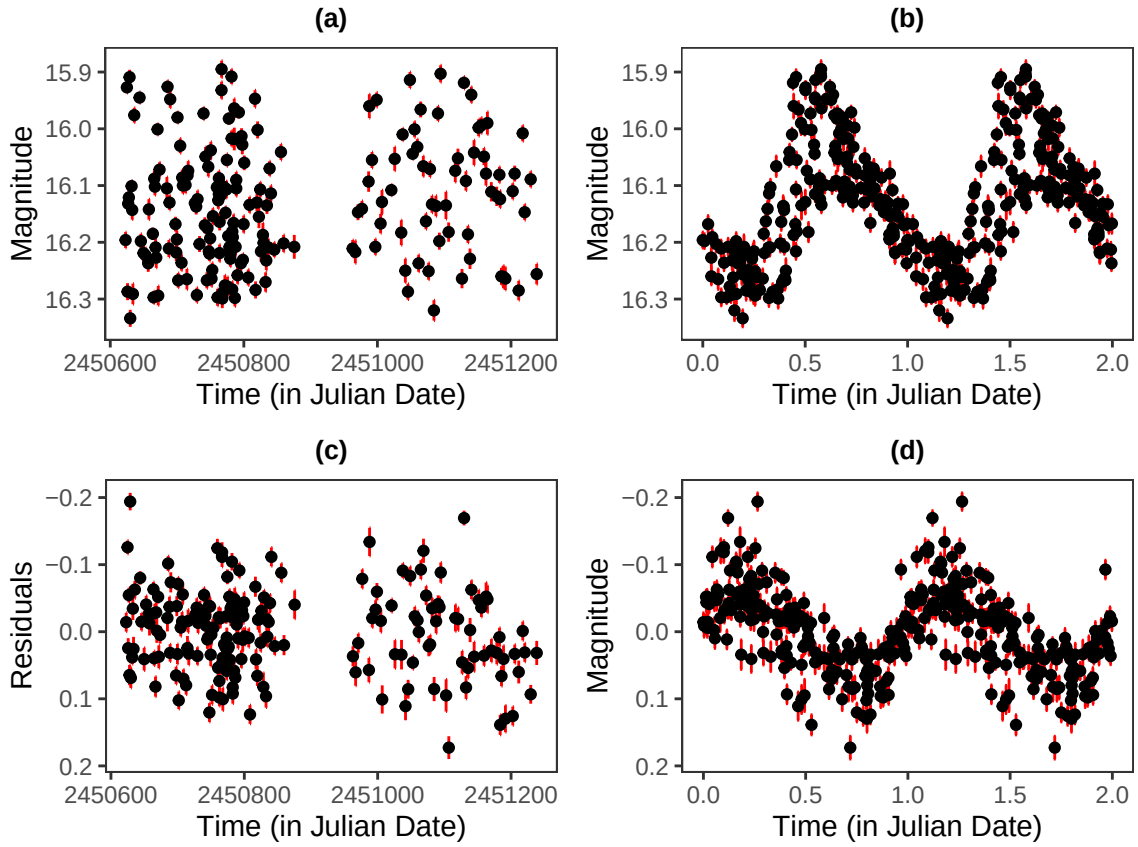



**Figure 10:** Figure a) shows the raw DMCEP time series and Figure b) shows this time series folded by the first period of the variable star. Figure c) shows the residuals of the harmonic fit and Figure d) shows this residuals folded by the second period of the variable star.

### 6.2 Pairing two astronomical time series

Among the models available in the **iAR** package, the BiAR model allows two time series to be modeled jointly. However, this model assumes that two time series are measured at the same times. This assumption is not necessarily satisfied. For example, in astronomy an object can be observed with multiple filters, producing distinct time series of brightness that are not perfectly synchronized. To address this problem, a two-step procedure is proposed. The first step consists of pairing two time series by their observational times, in order to pair the measurements taken at the same time. In the case that for a given time there is a measurement in only one time series, a missing value is assigned in the remaining one. In the second step these missing values are imputed with an interpolation procedure.

To illustrate this procedure, we use two time series available in the **iAR** package. These time series, called cvnovag and cvnovar, correspond to the brightness magnitude of a cataclysmic variable object observed both in the g and r filters of the ZTF survey respectively and processed by the ALeRCE broker (Förster et al., 2021). Prior to pairing, a smooth trend is removed from each series using the `loess` function in order to obtain a stationary time series. This choice is not restrictive, and other smoothing or detrending methods available in R could be used instead (see, for instance, `smooth.spline` in base R, or functions available in packages such as **mgcv** (Wood (2001),Wood (2017)) and **forecast** (Hyndman and Khandakar, 2008)). To pair these two time series, we use the method `pairingits` from the **iAR** package as follows:

```
data(cvnovag)
data(cvnovar)
lo.g <- loess(m ~ t, cvnovag)
lo.r <- loess(m ~ t, cvnovar)
datag=cvnovag
datar=cvnovar
datag$m=datag$m-lo.g$fitted
datar$m=datar$m-lo.r$fitted
o1<-pairingits(datag,datar,tol=0.1)
pargr1=na.omit(o1@paired)
```

This function causes that the datasets cvnovag and cvnovar of 67 and 65 observations respectively, to be merged into a single dataset with 97 observations which contains the measurements of both time series. Out of the total 97 observations in the merged dataset, 35 were matched based on their observation times, while in the remaining 62 observations there is a missing value in any of the two time series.

In order to fill these missing values we use the method interpolation. To implement this function we fit the BiAR model using the 35 observational times in which we have measures in the two time series as follows:

```
st=apply(pargr1[,c(1,4)],1,mean)
model_BiAR <- BiAR(times = st,series=pargr1[,c(2,5)],series_esd=pargr1[,c(3,6)])
model_BiAR@hessian=T
model_BiAR <- kalman(model_BiAR, niter = 20)
summary(model_BiAR,diagnostic=FALSE)

#> BiAR model
#>
#> Coefficients:
#>      Estimate St. Error t value Pr(>|t|)
#> phiR     0.93      0.02   52.44     0.00
#> phiI     0.00      0.01    0.81     0.42
#>
#> Information criteria:
#> AIC: -427.97
#> BIC: -424.86

model_BiAR <- fit(model_BiAR)
print(model_BiAR@rho)

#> [1] 0.9810806
```

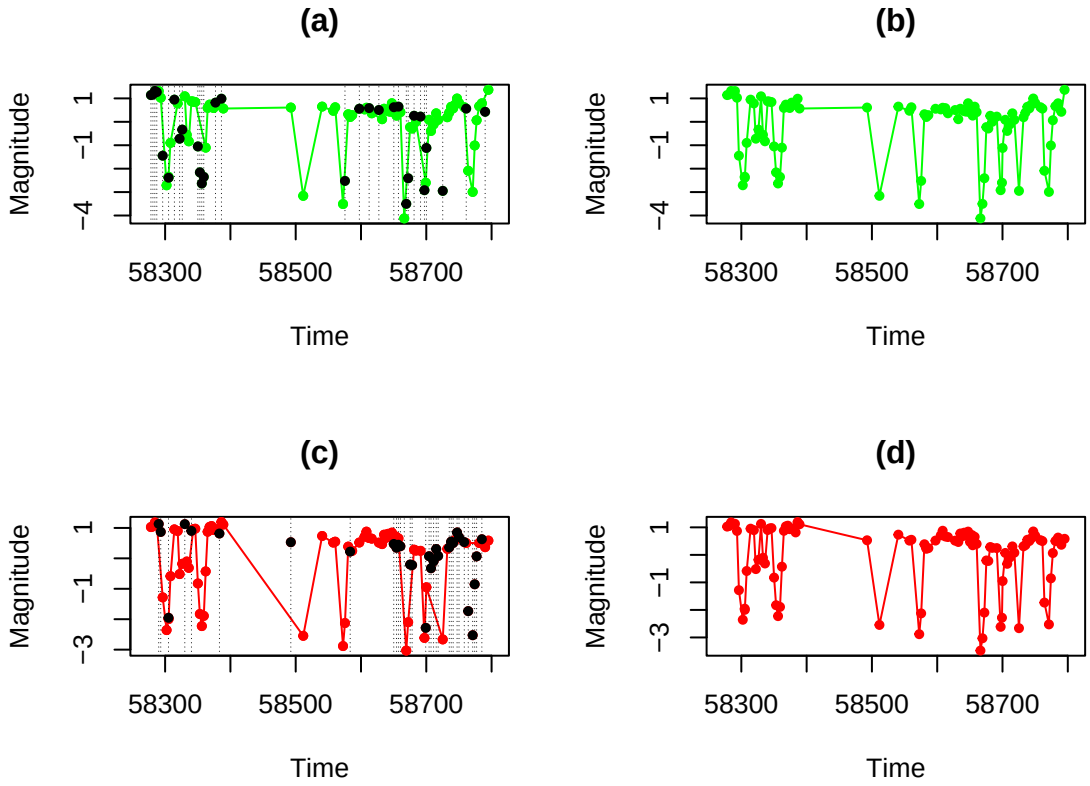



**Figure 11:** Figures a) and b) shows the time series cvnovag (green) and cvnovar (red) respectively. The vertical dotted lines show the observation times where a value must be imputed. The black dots are the estimated values at each of these observational times. Figures c) y d) shows both time series after the smoothing procedure.

Note that the contemporary correlation between both time series estimated by the BiAR model is very high (rho=0.98). Using the estimated parameters by the BiAR model (phiR= 0.929 and phiI= 0.004), we estimate each missing value recursively, so that only one missing value is estimated per iteration. Note that by using the bivariate model to perform the smoothing, we achieve that the time series that has an observation at a given time helps the imputation of the time series that has the missing value at the same time. This is particularly useful for highly correlated series such as the one presented in this example.

Figures 11 a and 11 b show both time series after the pairing procedure, and Figures 11 c and 11 d show both time series after the smoothing procedure. Note that due to the high correlation between the raw time series, the imputed time series become very similar.

## 7 Summary and discussion

A new R package called **iAR** is presented in this article. This package provides a flexible and consistent framework for modeling autoregressive processes with irregularly observed data. Its implementation using the S7 object-oriented system ensures a modular and extensible architecture, where each model class supports common methods for estimation, prediction, and interpolation. The use of S7 facilitates a coherent internal organization and simplifies future extensions to higher-dimensional models to be incorporated naturally.

This package implements novel models to fit irregularly observed time series. The methods implemented are the iAR, iAR-Gamma, iAR-T, CiAR and BiAR models. These models allow fitting a wide variety of autoregressive irregularly observed time series, some of which cannot be properly explained by models that assume regular or continuous time. For example, the CiAR model can fit irregularly observed time series with negative autocorrelation. Another feature is the ability to fit non-Gaussian data. The contexts in which each of these models is suitable are described throughout the article.

In the real-data applications presented in this paper, we showed that the models implemented in **iAR** are able to detect autoregressive dependence structures in irregularly observed time series and to generate interpolations in a model-based way. In this sense, the package provides an alternative to approaches that focus on interpolation of missing values, such as those implemented in **imputeTS** (Moritz and Bartz-Beielstein, 2017), by explicitly modeling the underlying dependence structure rather than treating interpolation as a preprocessing step.

From a computational perspective, the current implementation is not parallelized, but the estimation procedures scale approximately linearly with the length of the time series. Empirical results show that the implemented functions allow estimating model parameters in less computational time than other R packages commonly used for this type of data.

The interface and functionality of **iAR** were developed with data analysts working with irregularly observed time series in mind, rather than exclusively with expert R programmers. Such data frequently arise in fields such as astronomy, remote sensing, climatology, finance, and medical studies, where observation times may be limited by external constraints. The proposed models are designed for stationary time series whose dependency structure is adequately described by the assumed autoregressive formulation, which is a natural limitation of the current framework.

Furthermore, an interactive application developed using the **shiny** package (Chang et al., 2020) allows users to fit the models implemented in **iAR** to their own data through a graphical interface. The application is publicly available at: https://iargem.shinyapps.io/iarFit/.

Overall, this package contributes novel methodological tools to the analysis of irregularly observed time series, a setting for which dedicated statistical software remains relatively scarce in the R ecosystem, despite its relevance across a wide range of applied disciplines (see, for instance, Ojeda et al. (2023a); Ojeda et al. (2023b); Godoy Pulecio and Ojeda Echeverri (2025)).

## 8 Computational details

The results in this paper were obtained using R~4.4.2 and the following packages: **ggplot2**~4.0.0, **zoo**~1.8-12, **S7**~0.2.0, **Rcpp**~1.1.0, **Rdpack**~2.6.2. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/.

## Acknowledgments

The authors were supported by the ANID Millennium Science Initiative ICN12_009, awarded to the Millennium Institute of Astrophysics. F.E acknowledges the support of the ANID grant Fondecyt Iniciacion 11200590 and the CIRAS-AI Project code FIUF137139-USACH.

## References

G. E. P. Box and G. M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco, 1970. URL https://books.google.com/books/about/Time_Series_Analysis.html?id=5BVfnXaq03oC. [p305]

P. J. Brockwell and R. A. Davis. *Time Series: Theory and Methods*. Springer Series in Statistics. Springer, 2 edition, 1991. doi: 10.1007/978-1-4419-0320-4. URL https://link.springer.com/book/10.1007/978-1-4419-0320-4. [p305]

A. Brouste, M. Fukasawa, H. Hino, S. Iacus, K. Kamatani, Y. Koike, H. Masuda, R. Nomura, T. Ogihara, Y. Shimuzu, M. Uchida, and N. Yoshida. The yuima project: A computational framework for simulation and inference of stochastic differential equations. *Journal of Statistical Software, Articles*, 57(4):1–51, 2014. ISSN 1548-7660. doi: 10.18637/jss.v057.i04. URL https://www.jstatsoft.org/v057/i04. [p305]

W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *shiny: Web Application Framework for R*, 2020. URL https://CRAN.R-project.org/package=shiny. [p324]

D. Eddelbuettel and R. Francois. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. doi: 10.18637/jss.v040.i08. URL https://www.jstatsoft.org/index.php/jss/article/view/v040i08. [p311]

D. Eddelbuettel and C. Sanderson. Rcpparmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics & Data Analysis*, 71:1054–1063, 2014. ISSN 0167-9473. doi: https:

//doi.org/10.1016/j.csda.2013.02.005. URL https://www.sciencedirect.com/science/article/pii/S0167947313000492. [p311]

R. Edelson, J. Gelbord, E. Cackett, S. Connolly, C. Done, M. Fausnaugh, E. Gardner, N. Gehrels, M. Goad, K. Horne, I. McHardy, B. M. Peterson, S. Vaughan, M. Vestergaard, A. Breeveld, A. J. Barth, M. Bentz, M. Bottorff, W. N. Brandt, S. M. Crawford, E. D. Bontà, D. Emmanoulopoulos, P. Evans, R. F. Jaimes, A. V. Filippenko, G. Ferland, D. Grupe, M. Joner, J. Kennea, K. T. Korista, H. A. Krimm, G. Kriss, D. C. Leonard, S. Mathur, H. Netzer, J. Nousek, K. Page, E. Romero-Colmenero, M. Siegel, D. A. Starkey, T. Treu, H. A. Vogler, H. Winkler, and W. Zheng. Swift monitoring of NGC 4151: Evidence for a second x-ray/UV reprocessing. *The Astrophysical Journal*, 840(1):41, may 2017. doi: 10.3847/1538-4357/aa6890. URL https://doi.org/10.3847/1538-4357/aa6890. [p305]

F. Elorrieta, S. Eyheramendy, and W. Palma. Discrete-time autoregressive model for unequally spaced time-series observations. *A&A*, 627:A120, 2019. doi: 10.1051/0004-6361/201935560. URL https://doi.org/10.1051/0004-6361/201935560. [p307]

F. Elorrieta, S. Eyheramendy, W. Palma, and C. Ojeda. A novel bivariate autoregressive model for predicting and forecasting irregularly observed time series. *Monthly Notices of the Royal Astronomical Society*, 505(1):1105–1116, 05 2021. ISSN 0035-8711. doi: 10.1093/mnras/stab1216. URL https://doi.org/10.1093/mnras/stab1216. [p308]

S. Eyheramendy, F. Elorrieta, and W. Palma. An irregular discrete time series model to identify residuals with autocorrelation in astronomical light curves. *Monthly Notices of the Royal Astronomical Society*, 481(4):4311–4322, 09 2018. ISSN 0035-8711. doi: 10.1093/mnras/sty2487. URL https://doi.org/10.1093/mnras/sty2487. [p306]

F. Förster, G. Cabrera-Vives, E. Castillo-Navarrete, P. A. Estévez, P. Sánchez-Sáez, J. Arredondo, F. E. Bauer, R. Carrasco-Davis, M. Catelan, F. Elorrieta, S. Eyheramendy, P. Huijse, G. Pignata, E. Reyes, I. Reyes, D. Rodríguez-Mancini, D. Ruz-Mieres, C. Valenzuela, I. Álvarez-Maldonado, N. Astorga, J. Borissova, A. Clocchiatti, D. D. Cicco, C. Donoso-Oliva, L. Hernández-García, M. J. Graham, A. Jordán, R. Kurtev, A. Mahabal, J. C. Maureira, A. Muñoz-Arancibia, R. Molina-Ferreiro, A. Moya, W. Palma, M. Pérez-Carrasco, P. Protopapas, M. Romero, L. Sabatini-Gacitua, A. Sánchez, J. S. Martín, C. Sepúlveda-Cobo, E. Vera, and J. R. Vergara. The automatic learning for the rapid classification of events (ALeRCE) alert broker. *The Astronomical Journal*, 161(5):242, apr 2021. doi: 10.3847/1538-3881/abe9bc. URL https://doi.org/10.3847/1538-3881/abe9bc. [p322]

D. A. Godoy Pulecio and C. A. Ojeda Echeverri. An autoregressive moving average model for time series with irregular time intervals. *Computer Sciences & Mathematics Forum*, 11(1):8–0, 2025. ISSN 2813-0324. doi: 10.3390/cmsf2025011008. URL https://www.mdpi.com/2813-0324/11/1/8. [p324]

R. Hyndman, G. Athanasopoulos, C. Bergmeir, G. Caceres, L. Chhay, M. O'Hara-Wild, F. Petropoulos, S. Razbash, E. Wang, and F. Yasmeen. *forecast: Forecasting functions for time series and linear models*, 2026. URL https://pkg.robjhyndman.com/forecast/. R package version 9.0.0. [p305]

R. J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 3 edition, 2021. URL https://otexts.com/fpp3/. [p305]

R. J. Hyndman and Y. Khandakar. Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software*, 27(3):1–22, 2008. doi: 10.18637/jss.v027.i03. [p305, 322]

R. Juhl, J. K. Møller, and H. Madsen. ctsmr - continuous time stochastic modeling in R, 2016. [p305]

T. McMillen. Simulation and inference for stochastic differential equations: With R examples. *Journal of Statistical Software, Book Reviews*, 26(2):1–3, 2008. ISSN 1548-7660. doi: 10.18637/jss.v026.b02. URL https://www.jstatsoft.org/v026/b02. [p305]

S. Moritz and T. Bartz-Beielstein. imputets: Time series missing value imputation in R. *The R Journal*, 9: 207–218, 2017. ISSN 2073-4859. doi: 10.32614/RJ-2017-009. https://doi.org/10.32614/RJ-2017-009. [p324]

C. Ojeda, W. Palma, S. Eyheramendy, and F. Elorrieta. Extending time-series models for irregular observational gaps with a moving average structure for astronomical sequences. *RAS Techniques and Instruments*, 2(1):33–44, Jan. 2023a. ISSN 2752-8200. doi: 10.1093/rasti/rzac011. URL https://academic.oup.com/rasti/article/2/1/33/6987645. [p324]

C. Ojeda, W. Palma, S. Eyheramendy, and F. Elorrieta. A novel first-order autoregressive moving average model to analyze discrete-time series irregularly observed. In O. Valenzuela, F. Rojas, L. J. Herrera, H. Pomares, and I. Rojas, editors, *Theory and Applications of Time Series Analysis and*

*Forecasting*, pages 91–103. Springer International Publishing, Cham, 2023b. ISBN 9783031141966 9783031141973. doi: 10.1007/978-3-031-14197-3_7. URL https://link.springer.com/10.1007/978-3-031-14197-3_7. [p324]

J. M. Polanco-Martinez, M. A. Medina-Elizalde, M. F. S. Goni, and M. Mudelsee. Bincor: An R package for estimating the correlation between two unevenly spaced time series. *The R Journal*, 11:170–184, 2019. ISSN 2073-4859. doi: 10.32614/RJ-2019-035. https://doi.org/10.32614/RJ-2019-035. [p305]

K. Rehfeld, N. Marwan, J. Heitzig, and J. Kurths. Comparison of correlation analysis techniques for irregularly sampled time series. *Nonlinear Processes in Geophysics*, 18(3):389–404, June 2011. doi: 10.5194/npg-18-389-2011. [p305]

T. Ruf. The lomb-scargle periodogram in biological rhythm research: Analysis of incomplete and unequally spaced time-series. *Biological Rhythm Research*, 30(2):178–201, 1999. doi: 10.1076/BRHM.30.2.178.1422. URL https://doi.org/10.1076/BRHM.30.2.178.1422. [p305]

A. M. Thieler, R. Fried, and J. Rathjens. RobPer: An R package to calculate periodograms for light curves based on robust regression. *Journal of Statistical Software*, 69(9):1–36, 2016. doi: 10.18637/jss.v069.i09. [p305]

A. Udalski, M. K. Szymański, and G. Szymański. OGLE-IV: Fourth Phase of the Optical Gravitational Lensing Experiment. *Acta Astronomica*, 65:1–38, Mar. 2015. [p320]

D. Vaughan, J. Hester, T. Kalinowski, W. Landau, M. Lawrence, M. Maechler, L. Tierney, and H. Wickham. *S7: An Object Oriented System Meant to Become a Successor to S3 and S4*, 2024. URL https://CRAN.R-project.org/package=S7. R package version 0.2.0. [p306]

S. N. Wood. Mgcv: Gams and generalized ridge regression for R. *R News*, 1:20–25, 2001. ISSN 1609-3631. URL https://journal.r-project.org/articles/RN-2001-015/. [p322]

S. N. Wood. *Generalized Additive Models: An Introduction with R*. Chapman and Hall/CRC, 2 edition, 2017. doi: 10.1201/9781315370279. URL https://doi.org/10.1201/9781315370279. [p322]

M. Zechmeister and M. Kürster. The generalised Lomb-Scargle periodogram. A new formalism for the floating-mean and Keplerian periodograms. *Astronomy and Astrophysics*, 496:577–584, Mar. 2009. doi: 10.1051/0004-6361:200811296. [p305]

A. Zeileis and G. Grothendieck. zoo: S3 infrastructure for regular and irregular time series. *Journal of Statistical Software*, 14(6):1–27, 2005. doi: 10.18637/jss.v014.i06. [p306]

*Felipe Elorrieta*
*Universidad de Santiago de Chile*
*Department of Mathematics and Computer Science*
*Av. Libertador Bernardo O'Higgins 3663, Estación Central, Santiago, Chile*
*ORCID: 0000-0002-1835-7433*
felipe.elorrieta@usach.cl

*Susana Eyheramendy*
*Universidad Adolfo Ibañez*
*Faculty of Engineering and Sciences*
*Diagonal Las Torres 2700, Peñalolén, Santiago, Chile*
*ORCID: 0000-0003-4723-9660*
susana.eyheramendy@gmail.com

*Wilfredo Palma*
*Millennium Institute of Astrophysics*
*Nuncio Monsenor Sotero Sanz 100, Of. 104, Providencia, Santiago, Chile*
wilfredo.palma@gmail.com

*Cesar Ojeda*
*Universidad del Valle*
*Escuela de Estadística*
*Cali, Colombia*
*ORCID: 0000-0002-4042-8920*
cesar.ojeda@correounivalle.edu.co

# R Package dbrobust: Robust Distance-Based Visualization and Analysis of Mixed-Type Data

*by Eva Boj, Aurea Grané and Marcos Álvarez*

**Abstract** I n many applied fields, such as socioeconomic research, quality of life analysis or gender inequality studies, datasets often contain a mixture of quantitative, binary and categorical variables. Traditional multivariate techniques face this complexity, especially when robustness to outliers and interpretability in high-dimensional spaces is required. To overcome these challenges and address the lack of integrated tools for robust distance-based (DB) analysis of mixed-type data, we introduce **dbrobust**, a new R package that provides a unified and extensible framework for computing, visualizing and analyzing robust distances between observations. The package builds on classical measures such as Euclidean, Manhattan, Mahalanobis and Gower distances, and extends them through robust and weighted generalizations, including a novel implementation of a family of hybrid distances via Related Metric Scaling (RelMS). It integrates visualization techniques such as distance heatmaps, graph-based layouts and multidimensional scaling (MDS) maps, including conditional MDS to highlight trimmed individuals or reveal structural patterns across groups, and is fully compatible with **dbstats**, which implements the distance-based predictive models.

## 1 Introduction

Contemporary applied statistics faces growing complexity in domains such as social sciences, quality of life assessment, environmental monitoring, and gender inequality research. Analysts increasingly encounter datasets of mixed-type that combine quantitative, binary, and multiclass categorical variables, reflecting multifaceted phenomena in the real world. Traditional statistical models frequently rely on assumptions of distributional homogeneity and well-behaved data structures that rarely hold in practice. These limitations are compounded by challenges including influential outliers, measurement inconsistencies, and the curse of dimensionality, which collectively undermine the efficacy of classical approaches in extracting meaningful insights from complex datasets.

The analysis of mixed-type variables presents substantial methodological hurdles. Conventional multivariate techniques, such as principal components analysis and multivariate analysis of variance, are fundamentally designed for continuous data and often yield biased or uninterpretable results when applied to categorical or binary variables. Their sensitivity to anomalous observations further distorts both parametric estimation and graphical representations. Perhaps most critically, the absence of a standardized framework for quantifying dissimilarity across diverse variable types creates significant obstacles for both exploratory data examination and confirmatory statistical inference, necessitating specialized methodological approaches.

DB methodologies offer a promising alternative by reducing multivariate complexity to pairwise dissimilarity computations. While established measures exist for homogeneous data (e.g., Euclidean and Mahalanobis distances for continuous variables; Jaccard and Sokal similarity coefficients for binary data), the Gower similarity coefficient provides a unified approach for mixed-type variables. However, conventional distance measures often lack robustness against data contamination. Recent advances have introduced more robust and adaptive formulations, but their implementation remains fragmented across statistical ecosystems, limiting accessibility for applied researchers.

This work addresses these challenges through the development and comprehensive documentation of **dbrobust** (Álvarez et al., 2026), a novel R package (R Core Team, 2026) designed for robust DB analysis. It pursues five core objectives: (1) Unified distance computation; (2) Novel robust methodologies; (3) Enhanced visualization capabilities; (4) Contextual analysis framework; (5) Seamless interoperability with **dbstats** (Boj et al., 2025) and other R packages, including **vegan** (Oksanen et al., 2026), **cluster** (Maechler et al., 2026), **proxy** (Meyer and Buchta, 2025), **qgraph** (Epskamp et al., 2012), and **ggplot2** (Wickham, 2016).

The **dbrobust** package aims to bridge the gap between methodological innovation and practical implementation, providing social scientists and applied researchers with accessible tools for robust analysis of complex modern datasets.

The paper proceeds as follows. In Section 2 we provide a brief summary of common distance measures and similarity coefficients, and a review of the general formulation that enables the computation of the novel family of hybrid distance measures. In Section 3 we explain the core functions of the **dbrobust** package. Section 4 contains examples to illustrate the methodology, and we conclude in Section 5.

## 2 Methodology

The notion of distance between elements (objects, units or individuals) within a set $\Omega$ underlies numerous classical multivariate statistical methods. By defining such distances, we can represent the elements as points in a suitable metric space, enabling a geometric interpretation of these procedures. This geometric perspective is not restricted to cases involving only quantitative variables; it becomes even more valuable when any kind of proximity or similarity can be quantified between the units in $\Omega$.

### 2.1 Preliminary notation

Let $\Omega = \{1, \ldots, n\}$ denote a set of units. In what follows we review the definitions of distance, dissimilarity and similarity and related concepts.

#### Definitions of dissimilarity and distance

A mapping $\delta : \Omega \times \Omega \to \mathbb{R}$ is called a dissimilarity if it satisfies the following properties:

(i) Non-negativity: $\delta_{ij} \geq 0 \quad \forall\, i, j \in \Omega$,

(ii) Identity of indiscernibles: $\delta_{ii} = 0 \quad \forall\, i \in \Omega$,

(iii) Symmetry: $\delta_{ij} = \delta_{ji} \quad \forall\, i, j \in \Omega$.

A dissimilarity is called a semi-metric if it satisfies the following additional property:

(iv) Triangular inequality: $\delta_{ij} \leq \delta_{ik} + \delta_{kj} \quad \forall\, i, j, k \in \Omega$.

A semi-metric is called a metric if the following additional condition is fulfilled:

(v) Definiteness: $\delta_{ij} = 0 \iff i = j \quad \forall\, i, j \in \Omega$.

In general, the term distance may refer to either a metric or a semi-metric, depending on the context. In this work, those matrices containing either pairwise distances or pairwise dissimilarities will be referred to as distance matrices.

#### Definition of similarity

In many practical contexts, it is more informative to work with similarities rather than distances. A similarity quantifies the degree of proximity or likeness between two units, serving as the dual concept of a distance. More formally, a mapping $s : \Omega \times \Omega \longrightarrow \mathbb{R}$ is called a similarity if it satisfies:

(i) Boundedness and normalization[1]: $0 \leq s_{ij} \leq s_{ii} = 1 \quad \forall\, i, j \in \Omega$,

(ii) Symmetry: $s_{ij} = s_{ji} \quad \forall\, i, j \in \Omega$.

It is straightforward to derive a dissimilarity from a similarity, and vice versa. There are several transformations available, like $\delta_{ij} = 1 - s_{ij}$ or $\delta_{ij} = \sqrt{1 - s_{ij}}$, among others (Gower and Legendre, 1986). In this work, we use the general transformation given by Gower (1966):

$$\delta_{ij}^2 = s_{ii} + s_{jj} - 2s_{ij},$$

where $s_{ij}$ is the similarity between units $i$ and $j$, since this general transformation yields a positive distance measure even when $s_{ij}$ does not lie within the $[0, 1]$-interval, or when $s_{ii} \neq 1$.

#### The weighted Gram matrix and the geometric variability of a distance

Let $\delta_{ij}$ be a dissimilarity measure defined for each pair of units $i, j \in \Omega$, and consider the matrix of pairwise squared distances $\mathbf{\Delta} = \left(\delta_{ij}^2\right)_{1 \leq i,j \leq n}$.

For each individual in $\Omega$, consider a constant positive weight $w_i \in (0, 1)$, and let $\mathbf{w} = (w_1, \ldots, w_n)^\top$ be the $n \times 1$ weight vector, such that $\mathbf{1}^\top \cdot \mathbf{w} = 1$, where $\mathbf{1}$ represents the $n \times 1$ vector of ones. We define, on the one hand, the weighted Gram matrix as:

$$\mathbf{G}_\mathbf{w} = -\frac{1}{2}\mathbf{J}_\mathbf{w} \cdot \mathbf{\Delta} \cdot \mathbf{J}_\mathbf{w}^\top, \tag{1}$$

[1] Although this is the most accepted general definition, there are similarity coefficients that do not lie within the $[0, 1]$-interval, or that $s_{ii} \neq 1$.

where $\mathbf{J_w} = \mathbf{I_n} - \mathbf{1} \cdot \mathbf{w}^\top$ is the weighted centering matrix, with $\mathbf{I_n}$ the identity matrix of size $n \times n$, and on the other hand, its standardized version as:

$$\mathbf{F_w} = \mathbf{D_w}^{1/2} \cdot \mathbf{G_w} \cdot \mathbf{D_w}^{1/2}, \tag{2}$$

where $\mathbf{D_w} = \text{diag}(\mathbf{w})$ is a diagonal matrix whose diagonal entries are the individual weights in $\mathbf{w}$. If all weights are equal (i.e., $w_i = \frac{1}{n}$, $\forall i$), the weighted concepts coincide with the classical definitions in Borg and Groenen (2005).

A key element in both the comparison and combination of different dissimilarities is the geometric variability of a distance matrix $\mathbf{\Delta}$. It was introduced by Cuadras and Fortiana (1995) as a generalization of the concept of total variation, and within the weighted framework, it is computed as:

$$V_{\mathbf{\Delta}} = \text{tr}(\mathbf{F_w}) = \frac{1}{2}\mathbf{w}^\top \cdot \mathbf{\Delta} \cdot \mathbf{w}. \tag{3}$$

In this work, the concept of geometric variability is used to ensure the commensurability of all distance matrices to be combined.

### The Euclidean property

Once the concepts of distance, dissimilarity, and similarity are established, it becomes essential to determine whether a given dissimilarity allows a geometric representation in a Euclidean space. This condition, known as the *Euclidean property* or *Euclideanarity*, a neologism coined by John Gower when he described that property in Gower and Legendre (1986), is fundamental in multivariate analysis, as it enables the visualization and interpretation of data through classical techniques such as metric MDS. MDS provides a geometric representation of a distance matrix in a low-dimensional Euclidean space, typically two or three dimensions. Its objective is to place observations as points in a Euclidean space so that the distances between points reproduce, as closely as possible, the original dissimilarities. As a result, observations that are similar are displayed close together, whereas observations that are more dissimilar appear farther apart, facilitating the visual exploration of the data structure. When the distance matrix fulfills the Euclidean property, MDS provides an exact geometric representation of the observations. Euclideanarity is a useful property because it ensures that the MDS representation will produce point coordinates that can be entirely represented in Euclidean space, without production of negative eigenvalues and complex point coordinates. In any case, this property prevents imaginary dimensions in the final metric.

A dissimilarity is said to satisfy the Euclidean property when it is possible to embed the set of units into a finite-dimensional Euclidean space such that the pairwise dissimilarity between units in the original space corresponds to the $\ell^2$-distance between units in the Euclidean space. More specifically, a dissimilarity fulfills the Euclidean property if and only if the associated weighted Gram matrix $\mathbf{G_w}$ is positive semi-definite (Boj et al., 2010).

When this condition is not met, it is necessary to apply certain transformations to $\mathbf{\Delta}$ in order to satisfy this requirement, as described in Borg and Groenen (2005). In particular, in the **dbrobust** package, the following transformation is implemented in function `make_euclidean`. For a given matrix of pairwise squared distances $\mathbf{\Delta}$, the transformation consists of computing the associated weighted Gram matrix $\mathbf{G_w}$ defined in (1), and check for its positive semi-definiteness. If $\mathbf{G_w}$ contains negative eigenvalues, then $\mathbf{\Delta}$ is corrected by an additive constant shift (Lingoes, 1971; Mardia, 1978):

$$\mathbf{\Delta}_{\text{new}} = \mathbf{\Delta} + 2c\,\mathbf{1} \cdot \mathbf{1}^\top - 2c\,\mathbf{I},$$

where $c \geq |\lambda_{\min}|$ and $\lambda_{\min}$ is the smallest eigenvalue of $\mathbf{G_w}$. In the implementation provided by the `make_euclidean()` function in `dbrobust` (see Section 3.2 for details), the constant is taken as $c = |\lambda_{\min}|$, corresponding to the minimum correction required to ensure that

the weighted Gram matrix is positive semi-definite. This is the value used throughout the package. This correction ensures that the updated Gram matrix becomes positive semi-definite, and the modified dissimilarities fulfill the Euclidean requirement.

### 2.2 Overview of common dissimilarity measures and similarity coefficients

There is a wide variety of dissimilarity measures and similarity coefficients used in multivariate analysis, each with its own mathematical properties and assumptions. They differ based on several key characteristics:

- Metric properties: Whether the measure satisfies the properties of a true metric (non-negativity, identity of indiscernibles, symmetry, and the triangle inequality),
- Sensitivity to scaling and translation: Whether it is affected by changes in the scale or location, and
- Type of data handled: Whether the measure is suitable for quantitative, multiclass categorical, binary, or mixed-type data.

The choice of the dissimilarity depends heavily on the nature of the data and the specific objectives of the analysis. For example, Euclidean and Mahalanobis distances are typically used for continuous variables, while similarity measures such as Jaccard or Dice are more appropriate for binary data. In the case of mixed-type data, hybrid measures like Gower distance are recommended.

Despite this theoretical diversity, the practical implementation of these measures in R remains fragmented and inconsistent. The base function `dist` from the **stats** package supports only quantitative data, while other packages such as **ade4** (Dray et al., 2007), **StatMatch** (D'Orazio, 2025), **cluster** (Maechler et al., 2026), or **gower** (van der Loo, 2024) address binary, categorical, or mixed data types, each with distinct syntax and assumptions.

Although several functions are available in R, the incorporation of robustness is not consistently supported across implementations. In many cases, users must rely on external procedures for robust covariance estimation when resistance to outliers is required. Moreover, existing approaches do not incorporate observation weights in the covariance estimation process. This situation complicates the creation of unified analytical pipelines across data types and limits accessibility for non-expert users.

Observation weights are typically determined by the application rather than selected or tuned by the analyst. For example, in aggregated datasets or survey data, weights often reflect the number of individuals represented by each observation. In such settings, incorporating weights into the distance computation is important because they directly affect the estimation of the covariance and association matrices and, consequently, the Mahalanobis-type distances themselves. This consideration was one of the main motivations for the development of the weighted robust covariance estimators implemented in **dbrobust**. While several R packages provide robust covariance estimators, we were not able to identify implementations that simultaneously account for robustness and observation weights in covariance estimation. The proposed DB-trimming estimator (Boj and Grané, 2024) and the weighted Minimum Covariance Determinant (wMCD) estimator (Boj et al., 2026) were developed to address this gap.

Given the extensive range of existing dissimilarity functions, in what follows we summarize and present the most commonly used and widely accepted ones.

#### Distances for quantitative data

Let $\mathbf{z}_i = (z_{i1}, \ldots, z_{ip})^\top$, and $\mathbf{z}_j = (z_{j1}, \ldots, z_{jp})^\top$ represent the observed values for $p$ quantitative variables $Z_1, \ldots, Z_p$ corresponding to two different individuals $i, j \in \Omega$. Table 1 contains a summary of the most widely used distance measures for quantitative data.

**Table 1:** Distance measures for quantitative data

| Name | Formula | Considerations | Scale invariance |
|---|---|---|---|
| Manhattan ($\ell^1$) | $\delta_1(i,j) = \sum_{k=1}^{p} \lvert z_{ik} - z_{jk} \rvert$ | Metric. Less sensitive to outliers than $\ell^2$. | No |
| Euclidean ($\ell^2$) | $\delta_E(i,j) = \left[(\mathbf{z}_i - \mathbf{z}_j)^\top (\mathbf{z}_i - \mathbf{z}_j)\right]^{1/2}$ | Appropriate when variables are uncorrelated and have unit variance. Common default in software. | No |
| Minkowski ($\ell^q$) | $\delta_q(i,j) = \left[\sum_{k=1}^{p} \lvert z_{ik} - z_{jk} \rvert^q\right]^{1/q}, q \geq 1$ | Generalization of Euclidean. Euclidean property preserved only for $q = 2$. | No |
| Chebyshev ($\ell^\infty$) | $\delta_\infty(i,j) = \max_k \lvert z_{ik} - z_{jk} \rvert$ | Metric. Reflects maximum feature deviation. | No |
| Canberra | $\delta_C(i,j) = \sum_{k=1}^{p} \frac{\lvert z_{ik} - z_{jk} \rvert}{\lvert z_{ik} \rvert + \lvert z_{jk} \rvert}$ | Sensitive to small values. Convention: $\frac{0}{0} := 0$. | Yes |
| Standardized Euclidean | $\delta_K(i,j) = \left[\sum_{k=1}^{p} \frac{(z_{ik} - z_{jk})^2}{s_k^2}\right]^{1/2}$ | Weights variables by inverse variance, $s_k^2 = var(Z_k)$. Assumes uncorrelated features. | Yes |
| Mahalanobis | $\delta_M(i,j) = \left[(\mathbf{z}_i - \mathbf{z}_j)^\top \mathbf{S}^{-1} (\mathbf{z}_i - \mathbf{z}_j)\right]^{1/2}$ | Accounts for variable correlations ($\mathbf{S}$ covariance matrix of the dataset). Particularly suitable for multivariate Gaussian data. | Yes |
| Range Normalized Manhattan | $\delta_G(i,j) = \sum_{k=1}^{p} \frac{\lvert z_{ik} - z_{jk} \rvert}{\max(Z_k) - \min(Z_k)}$ | Normalizes by range of each variable across all observations. Used in Gower's distance for numeric variables. | Yes |

Source: Adapted from Gower and Legendre (1986) and Cuadras (1989)

A variety of R functions are available for distance computation, each with its own scope and specific characteristics. Although the following list is not intended to be exhaustive, it includes some of the most commonly used functions for computing distances on quantitative data. The base function `dist()` in package **stats** supports Euclidean, Minkowski, Manhattan, and Canberra distances, whereas `cluster::daisy()` provides similar functionality with slightly different syntax for Euclidean and Manhattan distances.

For Mahalanobis distance, `stats::mahalanobis()` computes the squared distance of each observation from a reference center, whereas `StatMatch::mahalanobis.dist()` computes pairwise distances between observations and directly returns a distance matrix. In addition, by setting the argument `rob.vc = TRUE`, `StatMatch::mahalanobis.dist()` estimates a robust covariance matrix using the function `cov.rob()` from package **MASS** (Ripley and Venables, 2002), providing a robust alternative for distance computation.

Robust covariance estimation is available in several R packages. The function `cov.rob()` from package **MASS** provides robust estimates of location and covariance, whereas `covMcd()` from package **robustbase** (Maechler et al., 2025) implements the Minimum Covariance Determinant (MCD) estimator. Package **rrcov** (Todorov and Filzmoser, 2009) offers similar functionality through `CovMcd()`, together with a broader collection of robust multivariate estimators. The resulting covariance matrices can be employed to compute robust Mahalanobis distances.

A limitation of these approaches is that none of them incorporates observation weights in the estimation of the covariance matrix. To address this issue, package **dbrobust** implements two alternative approaches for the computation of weighted robust covariance matrices: a DB-trimming estimator and a wMCD estimator. These approaches extend the available robust covariance estimation framework by explicitly incorporating observation weights in the estimation process, which in turn enables the computation of weighted robust Mahalanobis distances.

The interaction between observation weights and robust trimming procedures deserves further investigation. In particular, the extent to which highly weighted observations may be retained or excluded by a robust procedure depends on both the weighting scheme and the structure of the data. A comprehensive study of this interaction is beyond the scope of the present paper and represents an interesting avenue for future research.

### Similarities for binary data

In this case, for each unit in $\Omega$ a vector of $p$ binary variables taking values in $\{0,1\}$ is observed. The similarity $s_{ij}$ between each pair of units $i$ and $j$ in $\Omega$ is often quantified through pairwise similarity coefficients. In particular, the $s_{ij}$'s are computed as functions of four well-known coefficients, namely $a, b, c, d$, each of which depends on each pair of units $i$ and $j$, although we do not explicit this dependence for ease of notation. They are defined as follows: $a$: It is the frequency of variables with response 1 in both units; $b$: It is the frequency of variables with response 0 in unit $i$, and response 1 in unit $j$; $c$: It is the frequency of variables with response 1 in unit $i$, and response 0 in unit $j$; $d$: It is the frequency of variables with response 0 in both units. As an important consideration, $a+b+c+d=p$, since each variable falls into exactly one of these four categories. More formally,

$$\begin{aligned} a &= \#\{k \in \{1,\dots,p\} : z_{ik}=1,\ z_{jk}=1\},\\ b &= \#\{k \in \{1,\dots,p\} : z_{ik}=0,\ z_{jk}=1\},\\ c &= \#\{k \in \{1,\dots,p\} : z_{ik}=1,\ z_{jk}=0\},\\ d &= \#\{k \in \{1,\dots,p\} : z_{ik}=0,\ z_{jk}=0\}, \end{aligned}$$

where # denotes set cardinality. By construction, $a+b+c+d=p$.

Many different similarity coefficients exist and in Table 2 we reproduce the most widely used.

**Table 2:** Similarity coefficients for binary data, their Euclidean property and value ranges

| Name | Formula | Euclidean | Range |
|---|---|---|---|
| Sokal-Michener | $s_{ij}=\frac{a+d}{a+b+c+d}$ | Yes | $[0,1]$ |
| Ochiai (Cosine) | $s_{ij}=\frac{a}{\sqrt{(a+b)(a+c)}}$ | Yes | $[0,1]$ |
| Jaccard | $s_{ij}=\frac{a}{a+b+c}$ | Yes | $[0,1]$ |
| Dice | $s_{ij}=\frac{2a}{2a+b+c}$ | Yes | $[0,\frac{2}{p}]$ |
| Russell-Rao | $s_{ij}=\frac{a}{a+b+c+d}$ | Yes | $[0,1]$ |
| Kulczynski | $s_{ij}=\frac{1}{2}\left(\frac{a}{a+b}+\frac{a}{a+c}\right)$ | No | $[0,1]$ |
| Sokal-Sneath | $s_{ij}=\frac{a}{a+\frac{1}{2}(b+c)}$ | Yes | $[0,1]$ |

Source: Adapted from Gower and Legendre (1986) and Cuadras (1989)

Among existing libraries, the **ade4** package offers the broadest coverage through the function `ade4::dist.binary()`, which supports most classical coefficients such as Jaccard, Dice, Ochiai, and Sokal–Sneath. However, it does not implement the Kulczynski distance, and all dissimilarities are computed as $\delta=\sqrt{1-s}$. Robust or weighted extensions are not natively available.

### Similarities for multiclass categorical data

In this case, for each unit in $\Omega$ a vector of $p$ multiclass categorical variables are observed, each of which may have a different number of possible categories. For a given pair of units $i$, and $j$, denote by $\alpha_{ij}$ the number of agreements (or matches) along the $p$ variables, and by $p-\alpha_{ij}$ the number of disagreements (or mismatches).

The most famous similarity coefficient for multiclass categorical data is the matching coefficient, defined as:

$$m_{ij}=\frac{\alpha_{ij}}{p}. \tag{4}$$

The coefficient has a bounded range between 0 and 1, and meets the requirements to be considered a Euclidean similarity. Moreover, when dealing with binary variables, the count

of agreements $a_{ij} = a + d$, meaning that the similarity coefficient introduced in equation (4), is equivalent to Sokal-Michener's coefficient.

The distance associated with this similarity coefficient is known as the Hamming distance, which simply counts the number of mismatches between two categorical vectors and is defined as:

$$\delta_H(i,j) = 1 - m_{ij}, \tag{5}$$

and lies in the interval $[0,1]$. Thus, the matching coefficient and the normalized Hamming distance are complementary measures, with one expressing similarity and the other dissimilarity between the same pair of units. In the context of multiclass categorical data, the matching coefficient remains the most prominent and commonly applied similarity measure. While alternative coefficients do exist, they are infrequently cited, even within academic literature, and their practical utility is typically confined to highly specific cases.

R implementation for multiclass similarity coefficients is less extensive than for quantitative data. The matching coefficient, equivalent to Sokal-Michener for binary variables, is implemented in the function `sm()` from the **nomclust** package (Sulc et al., 2025), which computes and stores results in a `dist` object.

Additionally, the latest version of **dbrobust** incorporates the distance measure for categorical variables recently proposed by Grané et al. (2026), available through `calculate_distances()` using `method = "cramer"`. This novel Mahalanobis-type distance explicitly accounts for the association structure among categorical variables, extending the package's capabilities for the analysis of categorical and mixed data. It is based on Cramér's $V$ association coefficient, and it allows the use of weighted observations. Given two units $i$ and $j$ for which $p$ categorical variables have been observed, the Cramér-based distance (in squared value) is defined as

$$\delta_C^2(i,j) = \delta_H(i,j)\, \mathbf{A}^{-1}\, \delta_H(i,j), \tag{6}$$

where $\mathbf{A}$ is a $p \times p$ matrix that contains the pair-wise Cramér's $V$ association coefficients for the $p$ categorical variables and $\delta_H$ is the Hamming distance defined in (5). A shrinkage scheme is applied (when needed) inside `method = "cramer"`, as well as the Euclideanarity check, to asses both for the positive definiteness of $\mathbf{A}$ (see Grané et al., 2026 for the details) and the semi-metric requirement (Gower and Legendre, 1986).

### Distances for mixed-type data

To start this section, we consider that the set $\Omega$ is composed by $n$ units for which a total of $p$ variables of mixed-type are measured, $Z_1, \ldots, Z_p$, and let $\mathbf{z}_i = (z_{i1}, \ldots, z_{ip})^\top$, and $\mathbf{z}_j = (z_{j1}, \ldots, z_{jp})^\top$ represent the observed values for two different individuals $i, j \in \Omega$. In particular, we observe $p_1$ quantitative variables, $p_2$ binary variables, and $p_3$ categorical variables, such that $p = p_1 + p_2 + p_3$. This case has been extensively studied over time. Notably, Estabrook and Rogers (1966) contributed to the field, and Gower (1971) introduced several methods, among which the most renowned is Gower's similarity coefficient:

$$s_{ij} = \frac{\sum_{h=1}^{p_1} \left(1 - \frac{|z_{ih} - z_{jh}|}{G_h}\right) + a + \alpha_{ij}}{p_1 + (p_2 - d) + p_3}, \quad \text{where} \quad 0 \leq s_{ij} \leq 1, \tag{7}$$

in which $a$ and $d$ denote the counts of $(1,1)$ and $(0,0)$ matches in the $p_2$ binary variables, respectively, $\alpha_{ij}$ represents the number of agreements along the $p_3$ multiclass categorical variables, and $G_h$ corresponds to the range of the $h$-th quantitative variable. According to Gower (1971), the similarity matrix associated with this coefficient is positive semi-definite (except possibly in the presence of missing values), which guarantees a multidimensional Euclidean representation of the sample. And the well-known Gower's distance is obtained from equation (7) as $\delta_{ij} = \sqrt{1 - s_{ij}}$.

For mixed-type data, Gower's coefficient remains the most established approach. Its computation in R can be achieved through several functions: the `daisy()` function from

the **cluster** package (Maechler et al., 2026) supports multiple variable types but does not allow for variable weighting or parallelization. The **gower** package (van der Loo, 2024) offers `gower::gower_dist()`, which provides variable-wise weights, column matching, and multi-threaded computation. Finally, `StatMatch::gower.dist()` extends Gower's method by accommodating variable weights, robust options for outliers, and advanced treatment of ordinal and continuous variables following (Kaufman and Rousseeuw, 1990). Despite these improvements, none of these implementations include correlation-aware or robust variants, nor do they support individual weights or automatic Euclidean corrections, which are often required for techniques such as MDS or clustering.

Although Gower's distance is a relevant improvement for managing data of mixed-type, when dealing with modern datasets, there are some drawbacks that need to be mentioned. Gower's distance (in squared values) is the simple mean of three distinct dissimilarity measures: the range-normalized Manhattan distance for quantitative variables, a distance associated to Jaccard's similarity coefficient for binary variables, and the Hamming distance for categorical multiclass variables.

In computational terms, Gower's distance has a low computational cost, which is considered its main advantage. Nonetheless, it is also important to know some of the disadvantages. First, it neglects variable dependencies: The metric's reliance on Pythagorean addition assumes independence between variable groups, disregarding potential correlations among quantitative features (Krzanowski, 1994) and this may introduce redundancy in distance estimation. Second, the sensitivity to outliers in continuous variables: Empirical analyses demonstrate that MDS embeddings based on Gower's distance exhibit instability in the presence of atypical observations, compromising robustness (Gower and Legendre, 1986; Albarrán et al., 2015; Grané and Romera, 2018; Grané et al., 2021b; Boj and Grané, 2024). Finally, the imbalance weighting of categorical variables: The formulation disproportionately emphasizes categorical differences, which might dominate the distance measure in mixed-data applications (Podani, 1999).

### 2.3 A new family of hybrid distances via Related Metric Scaling

Cuadras and Fortiana (1998) introduced a general method called the Related Metric Scaling (RelMS), which constructs a joint metric from multiple distance matrices obtained from the same set of units $\Omega$. The RelMS approach was first used in Albarrán et al. (2015) for the case of three sources of information, corresponding to quantitative, binary, and multiclass categorical variables to address the limitations of classical Gower's distance.

Conceptually, the RelMS procedure can be understood as a strategy for constructing a hybrid distance from multiple distance matrices, each derived from distinct sources of information. The best point of this methodology is that this approach allows for scalable integration of multiple data types in tasks such as clustering (Grané et al., 2021b,a), MDS (Albarrán et al., 2015; Grané et al., 2026), or even DB prediction models (Boj et al., 2010, 2016; Boj and Grané, 2024; Boj et al., 2025).

Consider $m$ sources of information (or variables of $m$ different kinds), for each of which it is possible to compute a matrix of pairwise squared distances $\mathbf{\Delta}_\ell$, for $\ell = 1, \ldots, m$. These sources may include, for instance, numerical, binary, multiclass categorical data, functional data, compositional data, etc. The process is structured around the following key steps:

1. Commensurability check: Ensuring all $\mathbf{\Delta}_\ell$ share equal geometric variability (see formula 3), achieved by rescaling each matrix appropriately.

2. Gram matrix calculation: For each $\mathbf{\Delta}_\ell$ compute the associated weighted Gram matrix $\mathbf{G}_{\mathbf{w},\ell}$, and its standardized version $\mathbf{F}_{\mathbf{w},\ell}$ (see equations (1) and (2)), a key step for embedding distances in Euclidean space.

3. Euclideanarity check: Each $\mathbf{G}_{\mathbf{w},\ell}$ must be positive semi-definite. If this is not the case, the additive transformation is applied to $\mathbf{\Delta}_\ell$ and the associated Gram matrix (and its standardized version) are conveniently updated.

4. Final hybrid distance construction:

$$\mathbf{\Delta} = \mathbf{1} \cdot \mathbf{g}_{\mathbf{w}}^{J} + \left(\mathbf{g}_{\mathbf{w}}^{J}\right)^{\top} \cdot \mathbf{1}^{\top} - 2\mathbf{G}_{\mathbf{w}}^{J},$$

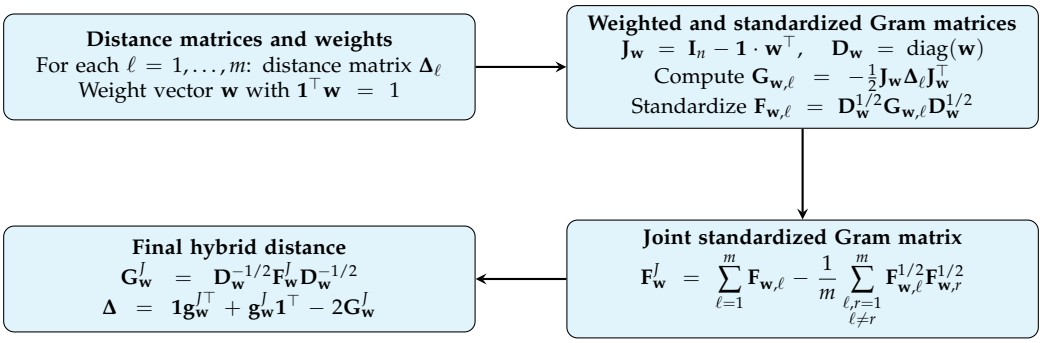
where $\mathbf{G}_{\mathbf{w}}^{J} = \mathbf{D}_{\mathbf{w}}^{-1/2} \cdot \mathbf{F}_{\mathbf{w}}^{J} \cdot \mathbf{D}_{\mathbf{w}}^{-1/2}$ is the joint weighted Gram matrix, $\mathbf{g}_{\mathbf{w}}^{J}$ is the row vector containing the diagonal elements of $\mathbf{G}_{\mathbf{w}}^{J}$, $\mathbf{1}$ is a column vector of ones, $\mathbf{D}_{\mathbf{w}} = \text{diag}(\mathbf{w})$ is a diagonal matrix whose diagonal entries are the individual weights in $\mathbf{w}$, and

$$\mathbf{F}_{\mathbf{w}}^{J} = \sum_{\ell=1}^{m} \mathbf{F}_{\mathbf{w},\ell} - \frac{1}{m} \sum_{\substack{\ell,r=1 \\ \ell \neq r}}^{m} \mathbf{F}_{\mathbf{w},\ell}^{1/2} \cdot \mathbf{F}_{\mathbf{w},r}^{1/2}, \tag{8}$$

where $\mathbf{F}_{\mathbf{w},\ell}^{1/2}$ denotes the square root of $\mathbf{F}_{\mathbf{w},\ell}$, which can be obtained through the singular value decomposition of $\mathbf{F}_{\mathbf{w},\ell}$. The first term in (8) accumulates all the relevant information from the individual sources, while the second term penalizes redundancy by subtracting all possible first-order interactions between different sources of information. In case two sources of information are not redundant, their Euclidean configurations generate orthogonal subspaces and their first-order interaction is null. See Albarrán et al. (2015), for the mathematical proofs.

The hybrid metric obtained via RelMS has the following properties: (i) Preservation of the individual metrics when others vanish or are identical; (ii) Orthogonality of contributions in the embedded space; (iii) Positive semi-definiteness guaranteed under Euclidean assumptions (Albarrán et al., 2015). To sum up, this joint metric construction offers a rigorous and flexible approach for combining heterogeneous distance measures. By addressing key limitations of simpler additive approaches, such as Gower's metric, it enhances the robustness and interpretability of multivariate analyses with mixed data types. The full workflow is illustrated in Figure 1.

**Figure 1:** Workflow of the general RelMS method




RelMS thus enables the integration of diverse data types into a unified metric framework suitable for visualization and clustering, though a higher computational cost. This is the reason why in Grané et al. (2021b) a simplified version of the above procedure was proposed, called generalized Gower distance (G-Gower), by considering only the first addend of formula (8), which mirrors Gower's formula of simply summing over the distance matrices (here in terms of the standardized Gram matrices). The construction of a hybrid distance is implemented in function `robust_distances()` (see Section 3.3 for details) which enables the methods `relms` or `ggower`, for the simplified version.

### Adding robustness to the hybrid metric

As outlined in Section "Distances for mixed-type data", Gower's distance is particularly useful for handling mixed-type data (quantitative, binary and multiclass categorical data). However, it also presents some notable limitations, most importantly, its inability to account

for correlations among continuous variables and its lack of robustness, which makes it highly sensitive to outliers.

To overcome these limitations, in Boj and Grané (2024), a new robust distance was proposed for quantitative variables, which can be incorporated into either G-Gower's distance or the RelMS approach, and is implemented in function `robust_distances()`. This new robust approach increases the resistance of the metric to outliers and enhances reliability when integrating mixed-type variables with observation-specific weights.

Specifically, the quantitative component of the hybrid metric is computed as a robust Mahalanobis distance by means of a DB-trimming estimator, introduced in Grané et al. (2021b) for outlier detection in complex datasets, and adapted for weighted data in Boj and Grané (2024). It should be noted that robustness in the current framework is introduced through the continuous component of the mixed-data dissimilarities, using robust Mahalanobis-type distances. Binary and categorical variables are handled through classical similarity and dissimilarity measures.

In particular, let $\Omega = \{1, \dots n\}$ be the set of units for which the values of $p$ variables are observed, $\mathbf{z}_1, \dots, \mathbf{z}_n \in \mathbb{R}^p$, and form the rows of the data matrix $\mathbf{Z}_{n\times p}$. Consider a distance measure $\delta$ and for all pairs of units in $\Omega$ compute the matrix of squared distances $\mathbf{\Delta} = \left(\delta^2\left(\mathbf{z}_i, \mathbf{z}_j\right)\right)_{1\leq i,j\leq n}$. Then, given a new unit $i_0 \notin \Omega$, with observed values $\mathbf{z}_0 \in \mathbb{R}^p$, the proximity of $i_0$ to the individuals in $\Omega$ is defined as:

$$\phi(\mathbf{z}_0) = \sum_{i=1}^{n} w_i \cdot \delta^2(\mathbf{z}_0, \mathbf{z}_i) - \mathbf{V}_{\mathbf{\Delta}}, \tag{9}$$

where $w_i$ denotes the weight associated with unit $i$, and $\mathbf{V}_{\mathbf{\Delta}}$ is the geometric variability of the distance matrix $\mathbf{\Delta}$, as defined in equation (3).

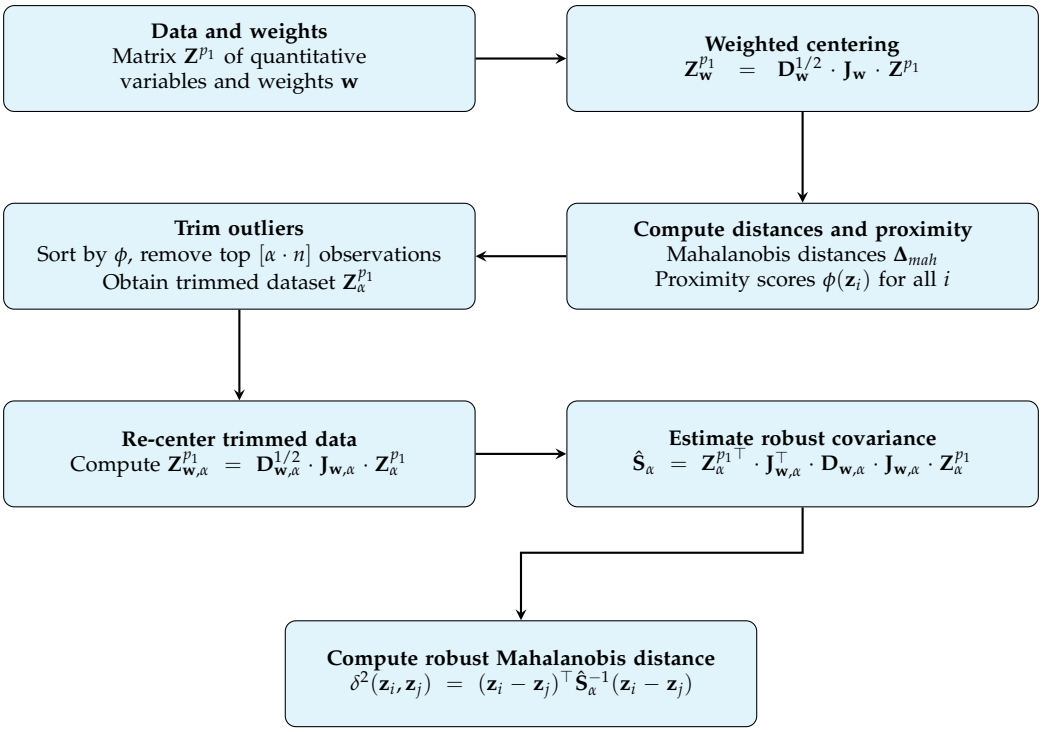
The proximity function (9) is used as a trimming estimator to exclude a fixed proportion $\alpha$ of the most outlying observations according to distance $\delta$. Although equation (9) is defined for a new observation $\mathbf{z}_0$, in practice the proximity function is also evaluated for observations belonging to the reference sample. In this case, no leave-one-out procedure is performed; the computation uses the previously obtained distance matrix and the corresponding geometric variability measures. Therefore, when $\mathbf{z}_0 = \mathbf{z}_i$ for some $i_0 \in \Omega$, the self-distance is equal to zero and is naturally included in the calculation.

With the trimmed sample, the covariance matrix for the quantitative variables is estimated and plugged into the Mahalanobis distance formula. The steps to compute the DB robust estimate of the covariance matrix for the quantitative variables are described in Figure 2, where $\mathbf{Z}^{p_1}$ denotes the $n \times p_1$ data matrix with the observations for the quantitative variables, and $\alpha$ is the trimming parameter.

The latest version of **dbrobust** includes the wMCD estimator proposed in Boj et al. (2026) as an alternative to the DB-trimming estimator for robust covariance estimation. The wMCD estimator extends the classical Minimum Covariance Determinant (MCD) methodology to the weighted setting and provides a robust covariance estimator that explicitly incorporates sampling weights. Its inclusion is mainly motivated by computational considerations, as it was designed to substantially reduce the computation time required to estimate the covariance matrix, particularly in high-dimensional and large-scale data sets. The robust covariance estimator used within `robust_distances()` is specified through the `cov_method` argument, where `"gv"` (the default option) corresponds to the DB-trimming estimator, whereas `"mcd"` corresponds to the wMCD estimator.

Furthermore, when only continuous variables are present, robust G-Gower reduces to a robust Mahalanobis distance, providing a robust weighted distance measure in its own right. This constitutes a novel contribution of **dbrobust**, since no other R implementation currently provides robust Mahalanobis distances based on robust covariance estimators that explicitly accommodate sampling weights. Both DB-trimming and wMCD allow the computation of robust weighted Mahalanobis distances, with wMCD offering improved scalability for big data applications.

**Figure 2:** Workflow for computing the robust Mahalanobis component for quantitative variables




## 3 Software implementation: Core functions

Although R is a powerful and flexible language for statistical computing, it presents notable limitations in DB multivariate analysis. In particular, when working with weighted mixed-type data, we identify several gaps: fragmented distance computation, lack of robust methods and Euclidean corrections, and underdeveloped visualization tools, each hindering effective multivariate analysis. These limitations highlight the need for a unified **dbrobust** package that consolidates and enhances existing tools.

### 3.1 Computing dissimilarities and similarities

One of the central functions in the **dbrobust** package is `calculate_distances()`, which provides a unified interface for computing pairwise distance or similarity matrices from a data frame or matrix. The function supports a wide range of data types, including binary, categorical, continuous, and mixed-type data, and offers compatibility with multiple well-known distance or similarity measures.

The primary objective of `calculate_distances()` is to simplify the computation of distances in datasets of mixed-type by providing a flexible and consistent function. Internally, it delegates computation to specialized functions depending on the selected method, while ensuring a standardized value structure. The internal and logical modular structure of `calculate_distances()` can be found in Section 1 of the Appendix.

Supported methods include:

- Binary data: Jaccard, Dice, Sokal-Michener, Russell-Rao, Sokal-Sneath, Kulczynski, and Hamming.
- Categorical data: Matching coefficient, Cramér-based.
- Continuous data: Euclidean, Standardized Euclidean, Manhattan, Minkowski, Canberra, Maximum, Cosine, Correlation, and Mahalanobis.
- Mixed-type data: Gower.

Note that Gower is a generalized (aggregated) distance that combines continuous, binary, and categorical variables into a single dissimilarity measure. Although conceptually different from purely numeric or purely binary distances, in `calculate_distances()` it is included alongside the other methods to provide a unified interface for the user.

A notable feature is the ability to convert distance matrices into similarity matrices (when applicable), and to optionally square the pairwise distances. The function is also designed with flexibility in its output: users can request either a condensed `dist` object, a full numeric distance matrix, or a similarity matrix. Moreover, users can select the Minkowski power only if the `minkowski` method is chosen, via the `p` argument.

The `calculate_distances()` function is a core component of the **dbrobust** package, providing essential support for subsequent operations. At the same time, its general-purpose nature makes it valuable beyond the scope of the package itself, allowing users to compute versatile distance or similarity matrices that can serve as input for a wide range of external analytical tasks.

The usage of the `calculate_distances()` function is:

```
calculate_distances(x, method = "gower", output_format = "dist",
                    squared = FALSE, q = NULL,
                    similarity_transform = "linear",
                    type = list(),weights = NULL)
```

where:

**x:** A matrix or data.frame. Each row represents an observation.

**method:** A string specifying the distance/similarity method. Supported options are:

- Binary: `"jaccard"`, `"dice"`, `"sokal_michener"`, `"russell_rao"`, `"sokal_sneath"`, `"kulczynski"`, `"hamming"`.
- Categorical: `"matching_coefficient"`, `"cramer"`.
- Continuous: `"euclidean"`, `"euclidean_standardized"`, `"manhattan"`, `"minkowski"`, `"canberra"`, `"maximum"`, `"cosine"`, `"correlation"`, `"mahalanobis"`.
- Mixed: `"gower"`.

**output_format:** Output format: `"dist"` (distance object), `"matrix"` (numeric matrix), or `"similarity"` (only for binary/categorical/mixed methods). squared Logical; if `TRUE`, returns squared distances (not applied to similarities).

**squared:** Logical; if `TRUE`, returns squared distances (not applied to similarities).

**q:** Numeric; the power parameter for the Minkowski distance (required if `method = "minkowski"`); ignored otherwise.

**similarity_transform:** Character string; if`output_format = "similarity"`, this specifies the formula to convert distances to similarity scores. Supported: `"linear"` (default), $s_{ij} = 1 - \delta_{ij}$, or `"sqrt"`, $s_{ij} = 1 - \delta_{ij}^2$.

**type** Optional list specifying variable types for Gower distance (passed directly to `cluster::daisy`). Arguments can be `symm`, `asymm`, `ordratio`, or `logratio`.

**weights** Optional numeric vector containing weights. Its interpretation depends on the chosen method:

**For Gower distance (`method = "gower"`):** A vector of length `ncol(x)` to weight the *variables* (columns).

**For Cramér's V distance (`method = "cramer"`):** A vector of length `nrow(x)` to weight the *observations* (rows), useful for handling class imbalance or stratified sampling.

While `calculate_distances()` computes distances for various data types, it does not guarantee that the resulting matrices fulfill the Euclidean property. The next section presents `make_euclidean()`, a function designed to verify and, if required, correct squared distance matrices to ensure their Euclidean validity.

### 3.2 Ensuring Euclideanarity of matrices of pairwise squared distances

The `make_euclidean()` function is designed to verify and, if necessary, transform a matrix of pairwise *squared* distances into a matrix of pairwise squared distances that fulfills the Euclidean property. This correction is important because some analytical methods, such as classical or metric MDS, require the distance values to satisfy this property. Numerical imprecision or certain data transformations may lead to small violations of this condition, which `make_euclidean()` can automatically detect and fix.

The function works by computing the weighted Gram matrix $\mathbf{G_w} = -\frac{1}{2}\mathbf{J_w} \cdot \mathbf{\Delta} \cdot \mathbf{J_w}^\top$, where $\mathbf{\Delta}$ is the matrix of pairwise squared distances, $\mathbf{w}$ is the user-supplied weight vector, and $\mathbf{J_w} = \mathbf{I_n} - \mathbf{1} \cdot \mathbf{w}^\top$ is the centering matrix defined by the normalized weights. It then checks the eigenvalues of $\mathbf{G_w}$. If the smallest eigenvalue $\lambda_{\min}$ is smaller than a negative tolerance, the function adds a constant shift to all pairwise distances following the method of Lingoes (1971) and Mardia (1978), ensuring positive semi-definiteness of the Gram matrix.

The usage of the `make_euclidean()` function is:

```
make_euclidean(D, w = NULL, tol = 1e-10)
```

where:

**D:** Square matrix of pairwise *squared* distances.

**w:** Numeric vector of weights, with length equal to the number of observations. Defaults to a vector of ones.

**tol:** Numeric tolerance for detecting negative eigenvalues.

The function returns a list of values containing four main components. The element `D_euc` contains the corrected squared distance values, guaranteed to satisfy Euclidean properties after the procedure. The elements `eigvals_before` and `eigvals_after` provide the eigenvalues of the weighted Gram matrix $\mathbf{G_w}$ before and after correction, respectively, allowing users to assess the magnitude and necessity of the adjustment. Finally, the element `transformed` is a logical value indicating whether any correction was applied, or if the matrix argument `D` already fulfilled the Euclidean conditions presented in Section "The Euclidean property". The internal and logical modular structure of function `make_euclidean()` can be found in Section 2 of the Appendix.

Using `make_euclidean()` offers several advantages. It ensures that squared distance values are compatible with algorithms relying on Euclidean geometry, thereby preventing failures or distortions in the resulting embeddings. The correction is deliberately minimal (applied only when violation is detected and using the smallest possible shift), thus preserving the original distance structure as much as possible. It is important to note that the function expects squared distances as argument; if only raw distances are available, they should be squared before calling the function. In this context, the `disttoD2()` function from the **dbstats** package can be used to conveniently convert a `dist` object into a matrix of pairwise squared distances $\mathbf{\Delta}$ before applying `make_euclidean()`.

### 3.3 Computing robust distances

The `robust_distances()` function is a core component of the **dbrobust** package, designed to compute robust squared (or simple) distance values between individuals in a weighted dataset containing quantitative, binary, and multiclass categorical variables. It supports two

robust distance measures, G-Gower and RelMs described in Section 2.3, and incorporates a DB-trimming procedure for estimating robust covariance matrices in the continuous subspace as was detailed in Figure 2.

The usage of the `robust_distances()` function is:

```
robust_distances(data = NULL, cont_vars = NULL, bin_vars = NULL,
                 cat_vars = NULL, w = NULL, p = NULL,
                 method = c("ggower", "relms"),
                 cov_method = c("gv", "mcd"),
                 robust_cov = NULL,
                 alpha = 0.1, niter = 40,
                 nstart = 20, return_dist = FALSE)
```

where:

**data:** Matrix or data frame containing the data.

**cont_vars, bin_vars, cat_vars:** Character vectors of variable names for continuous, binary and categorical respectively (if p is not provided).

**w:** Numeric vector of observation weights. If `NULL`, uniform weights are used.

**p:** Integer vector of length 3 with the number of continuous, binary, and multiclass categorical variables (only if columns are sorted in this order).

**method:** Aggregation method: `"ggower"` for robust G-Gower distance values and `"relms"` for robust RelMS distance values. Defaults to `"ggower"`.

**cov_method:** Character string specifying the robust covariance method for continuous variables: either `"gv"` (Geometric Variability) for the DB-trimming or `"mcd"` (Minimum Covariance Determinant) for the wMCD trimming algorithm. Default is "gv".

**robust_cov:** Optional precomputed robust covariance matrix for quantitative variables.

**alpha:** Trimming proportion for robust covariance estimation in quantitative variables. Defaults to `0.1`.

**niter:** Integer specifying the number of concentration steps (C-steps) per start. Only used if `cov_method = "mcd"`. Defaults to `40`.

**nstart:** Integer specifying the number of random initializations. Only used if `cov_method = "mcd"`. Defaults to `20`.

**return_dist:** Logical. If `TRUE`, returns the distance matrices $\mathbf{D}$ as `dist` objects; if `FALSE`, returns squared distance matrices $\mathbf{\Delta}$ as `D2` objects. Defaults to `FALSE`.

The internal and logical modular structure of function `robust_distances()` can be found in Section 3 of the Appendix.

This function is the most critical component of **dbrobust**, as it operationalizes the novel concepts introduced in Boj and Grané (2024) and Boj et al. (2026), which underpin the methodological framework of our current research line. By implementing these ideas within a robust and flexible mixed-data framework, `robust_distances()` bridges recent theoretical advances with practical computational tools, enabling their direct application.

### 3.4 Aggregating dissimilarities

The `merge_distances()` function is a core component of the **dbrobust** package, designed to combine multiple distance matrices into a single consensus distance matrix that fulfills the Euclidean property. The aggregation can be performed either through a direct weighted combination of distances based on a Pythagorean sum (Gower-like approach) or through

RelMS, which accounts for redundancy among sources (see Figure 1). The procedure accommodates both source-specific weights, controlling the contribution of each distance matrix to the consensus solution, and sampling weights associated with the observations. In all cases, input and output distance matrices are enforced to fulfill the Euclidean requirement through `make_euclidean()`.

The usage of the `merge_distances()` function is:

```
merge_distances(distance_list, matrix_weights = NULL,  w = NULL,
                squared = FALSE, RelMS = TRUE, tol = 1e-10 )
```

where:

**distance_list:** A list of distance matrices or `dist` objects. All distance matrices must be square ($n \times n$) and correspond to distances that fulfill the Euclidean property.

**matrix_weights:** Numeric vector of weights associated with the distance matrices in `distance_list`. Weights must be strictly positive and sum to one. If `NULL`, all matrices receive equal weight.

**w:** Optional numeric vector of observation weights. Weights must be strictly positive and are normalized to sum to one. If `NULL`, uniform weights are assumed.

**squared:** Logical value indicating whether the input matrices contain squared distances. If `FALSE` (default), distances are squared internally during the aggregation process. If `TRUE`, inputs are treated as already squared.

**RelMS:** Logical value indicating the aggregation method. If `TRUE` (default), aggregation is performed using RelMS, which penalizes redundancy among distance matrices. If `FALSE`, a direct weighted combination based on a Pythagorean sum is computed.

**tol:** Numeric tolerance used by `make_euclidean()` when enforcing the Euclidean property through eigenvalue decomposition. Defaults to `1e-10`.

In the latest version of **dbrobust**, the `merge_distances()` function has been added as a general framework for distance aggregation. Besides enabling the application of RelMS to the classical Gower approach for mixed data, it allows the combination of any set of distance matrices, whether derived from different variable blocks, different distance measures, or externally provided dissimilarity data, providing a flexible and widely applicable distance integration strategy.

The details of the modular structure can be found in Section 4 of the Appendix.

### 3.5 Visualizing distances and similarities

Visualizing relationships in distance matrices remains limited in R. Tools such as `qgraph()` from the **qgraph** package (Epskamp et al., 2012) and `ggpairs()` from the **GGally** package (Schloerke et al., 2025) offer partial solutions, but lack integration with MDS configurations, scale poorly with mixed-type data, and provide limited customization for grouped or weighted analyses. In particular, conditional visualizations, faceted MDS plots, and DB heatmaps or network representations are not natively supported in a cohesive and user-friendly manner.

The functions `cmdscale()` and `wcmdscale()`, which perform classical and weighted MDS respectively, do not internally verify whether the distance matrix passed as an argument fulfills the Euclidean requirement. In particular, they do not check the positive semi-definiteness of the associated Gram matrix constructed via double centering of the squared distances. If the Gram matrix is not positive semi-definite, the distance matrix does not fulfill the Euclidean property, and the MDS embedding may produce invalid or complex (imaginary) coordinates.

The `visualize_distances()` function in the **dbrobust** package provides a versatile interface to explore and display distances or similarities through various visualization techniques. It supports multiple plotting methods including classical and weighted MDS, heatmaps, and network graphs.

This function accepts distance matrices in either `dist` objects or symmetric square matrices `D2`, performing argument validation to ensure correctness. It offers flexibility in the number of dimensions for MDS (parameter `k`), optional weighting schemes, and grouping variables to color or annotate observations. Supported visualization methods include:

- Classic MDS: Classic MDS from `R` base function `cmdscale`,
- Weighted MDS: Weighted MDS variant implemented using the `wcmdscale()` function from the **vegan** package,
- Heatmap: Displays pairwise distances as a colored matrix with optional grouping and clustering, and
- Network graph: Visualizes relationships as a network using the **qgraph** package.

The usage of the `visualize_distances()` function is:

```
visualize_distances(dist_mat, method = c("mds_classic", "mds_weighted",
                    "heatmap", "qgraph"), k = 3, weights = NULL,
                    group = NULL, group_colors = NULL,
                    main_title = NULL, tol = 1e-10, ...)
```

where:

**dist_mat:** Square distance matrix (numeric) or a `dist` object containing pairwise distances between individuals.

**method:** Character string specifying the visualization method: `"mds_classic"`, `"mds_weighted"`, `"heatmap"`, or `"qgraph"`. Defaults to `"mds_classic"`.

**k:** Integer. Number of dimensions for MDS (default `3`). Must satisfy $1 \leq k \leq \min(4, n_{\text{obs}} - 1)$.

**weights:** Optional numeric vector of weights for weighted MDS. The dimension must match the number of observations. Defaults to a vector of ones.

**group:** Optional factor or vector indicating group membership for coloring or annotating observations.

**group_colors:** Optional character vector of custom colors for the groups. If `NULL`, a predefined custom palette is used.

**main_title:** Optional character string specifying the main plot title.

**tol:** Numeric tolerance for checking approximate symmetry and Euclidean property. Defaults to `1e-10`.

**...** Additional arguments passed to internal plotting functions (`plot_mds`, `plot_heatmap`, or `plot_qgraph`), such as clustering method, palette, max nodes, edge thresholds, etc.

The modular design makes `visualize_distances()` maintainable, extensible, and easy to understand. By delegating specific tasks to helper functions, each visualization method is implemented independently, which simplifies future extensions and maintenance. The details of the modular structure can be found in Section 5 of the Appendix.

The function `visualize_distances` does not automatically enforce Euclidean geometry. However, users who need a Euclidean embedding can preprocess their distance matrix using the function `make_euclidean`, which inspects the Gram matrix and optionally applies an eigenvalue correction (as detailed in Section 2.3). This ensures that the resulting MDS embedding is valid and interpretable in a Euclidean space.

### 3.6 Interoperability with the R ecosystem

Beyond its compatibility with the **dbstats** package, the dissimilarities computed by **dbrobust**, which can be represented as `dist` objects, can be directly employed by a broad range of DB procedures available in R. These include hierarchical and partitioning clustering methods such as `hclust()` from **stats** and `agnes()`, `diana()`, `pam()`, `fanny()`, and `clara()` from **cluster**. They can also be used in ordination techniques, including classical multidimensional scaling via `cmdscale()` from **stats**, non-metric multidimensional scaling via `isoMDS()` from **MASS** and `metaMDS()` from **vegan**, and principal coordinates analysis via `pcoa()` from **ape** (Paradis and Schliep, 2019). Furthermore, the resulting dissimilarities can serve as input for inferential procedures such as PERMANOVA through `adonis2()`, ANOSIM through `anosim()`, Mantel tests through `mantel()`, and DB redundancy analysis through `capscale()`, all available in **vegan**. Additional examples are summarized in Table 3, which highlights the breadth of DB methods that can directly operate on dissimilarities generated by **dbrobust**. Consequently, **dbrobust** may be viewed not only as a tool for robust distance computation but also as a general-purpose provider of dissimilarity measures that can be readily integrated into a wide variety of statistical learning, exploratory data analysis, and DB modelling workflows within the R ecosystem.

**Table 3:** Examples of downstream methods accepting `dist` objects generated from **dbrobust**.

| Method category | Package | Functions |
|---|---|---|
| Hierarchical clustering | **stats**, **cluster** | `hclust()`, `agnes()`, `diana()` |
| Partitioning clustering | **cluster** | `pam()`, `fanny()`, `clara()` |
| Ordination | **stats**, **MASS**, **vegan**, **ape** | `cmdscale()`, `isoMDS()`, `metaMDS()`, `pcoa()` |
| DB inference | **vegan** | `adonis2()`, `anosim()`, `mantel()`, `capscale()` |
| DB modelling | **dbstats** | `dblm()`, `dbglm()`, `dbplsr()`, `ldblm()`, `ldbglm()` |

## 4 Examples

In this section, we analyze two mixed-type datasets to illustrate different functionalities of the **dbrobust** package. The simulated dataset is used to highlight the behavior of the proposed methodologies under controlled contamination settings, whereas the real-data application serves as a practical vignette illustrating the package workflow and its main functions. In both cases, we start by computing three different metrics: classical Gower, robust G-Gower and robust RelMS, by means of functions `calculate_distances()` using `method = "gower"` (the default option) and `robust_distances()` with `method = "ggower"` or `method = "relms"`. For the robust metrics, the sample covariance matrix required in the Mahalanobis distance is estimated using DB-trimming (`cov_method = "gv"`, the default option) with a trimming parameter of `alpha = 0.1` (the default value). The next step is to assess the Euclideanarity condition, and for that purpose function `make_euclidean()` is applied to the three distance matrices. Finally, we proceed by comparing their Euclidean configurations with multiple MDS plots, as well as a direct comparison by means of network representations. For that purpose, the `visualize_distances()` function is used as a powerful tool for uncovering patterns and anomalous data through the visual inspection of dissimilarities between units.

### 4.1 Simulated data

The package `dbrobust` contains four sets of simulated data called `Data_HC_no_contamination`, `Data_HC_contamination`, `Data_MC_no_contamination` and `Data_MC_contamination`. All of them were generated from multivariate normal distributions with $p = 9$ components, and

different patterns of underlying correlation (high/moderate). The first four components were retained in the continuous form, while the remaining dimensions were later discretized into binary or multiclass categorical variables using percentile-based thresholds. Finally, a 5%-outlying contamination was added to some of the continuous components (see Boj and Grané, 2024 for the details).

In addition to the original variables, each dataset incorporates a weighting variable containing the weights for each observation, specifically constructed to illustrate the methodology with weights. To create the weighting variable, a proportional frequency-based approach derived from the joint distribution of the first multiclass and the first binary variables was adopted.

In this section, we focus on the dataset `Data_MC_contamination` to illustrate some of the functionalities of the `dbrobust` package. In particular, this dataset is composed of 525 observations and 9 variables of mixed type (4 continuous, 3 multiclass categorical, and 2 binary), with an underlying moderate correlation structure ($\rho = 0.6$) as well as the weights. The last 25 rows correspond to contaminated observations created by adding perturbations equal to three times the standard deviation of each quantitative variable to a subset of the 500 original units.

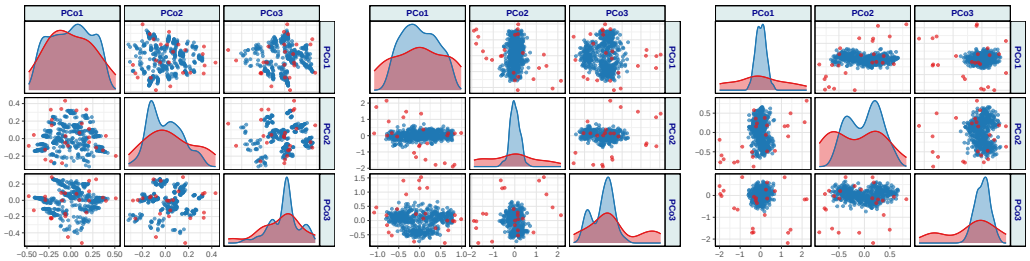
We present a comparative study of three distance metrics: the classical Gower and two robust alternatives, G-Gower and RelMS. The goal of this comparison is to evaluate the effectiveness of robust methods under realistic conditions involving data contamination.

The analysis begins with a qualitative exploration using classical MDS configurations to visualize how each metric captures the underlying data structure (see Figure 3, where option `"mds_classic"` was used to produce the conditional multiple MDS plot and outlying units were identified with the `"group"` option). The classical Gower metric (panel 3a) exhibits a pronounced artifact caused by binary and categorical variables, which severely degrades the visualization. This discretization leads to an artificial stratification of the data into three distinct bands, obscuring the underlying continuous relationships among observations. In contrast, both robust metrics, G-Gower (panel 3b) and RelMS (panel 3c), successfully separate the outlying observations from the main data cloud, placing them clearly apart from the bulk of the data. This effect results from the robust Mahalanobis distance used for continuous predictors in G-Gower and the combined treatment of variable types in RelMS, which integrates robust Mahalanobis, Jaccard and Hamming distances for continuous, binary, and multiclass predictors, respectively.

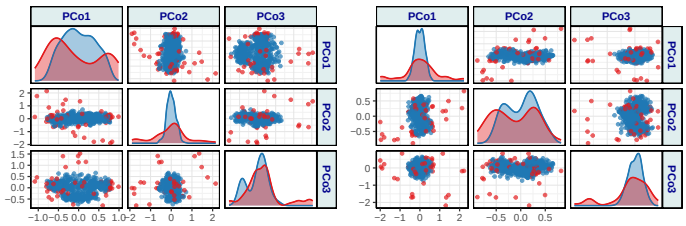
Panels 3b–3c show the true outliers (known from the data generation process) for comparison. The visual agreement between outliers detected in the trimming step (panels 3d–3e) and true outliers confirms the good sensitivity of the robust methods. Overall, the robust configurations reveal not only a better isolation of anomalous units but also a clearer depiction of the underlying clustered structure among regular observations, an effect that is masked under the classical Gower metric.

The analysis is followed by an examination of similarity networks to provide a more detailed view of the relationships among observations. Figure 4 displays the similarity networks (produced with `"qgraph"` option of function `visualize_distances()`), where each node represents an observation, and spatial proximity between nodes reflects pairwise similarity (obtained as $1 - \delta$, where $\delta$ is the distance measure): nodes positioned closer together correspond to more similar observations, while those farther apart are more dissimilar. For visualization purposes, the graphs show randomly selected stratified subsamples of 100 observations, preserving the target outlier proportions. Outlying units are colored red.

A clear visual pattern emerges: many red nodes are either clustered together or separated from the main cloud of blue nodes (non-outliers). Even when red nodes are not distinctly isolated, they tend to appear on the periphery of the blue cloud and are often connected to other red nodes. Furthermore, since the true outliers were placed as the last observations (indices $> 500$), visual inspection reveals that many of the trimmed nodes correspond to this upper-index region, confirming that the trimming procedure effectively captures a substantial proportion of the true outliers.

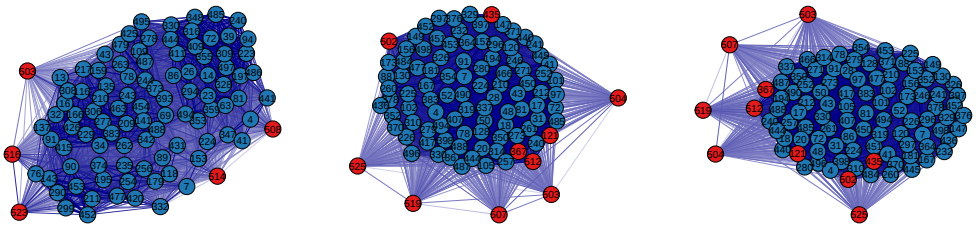



**(a)** Gower (outlying units) **(b)** G-Gower (outlying units) **(c)** RelMS (outlying units)

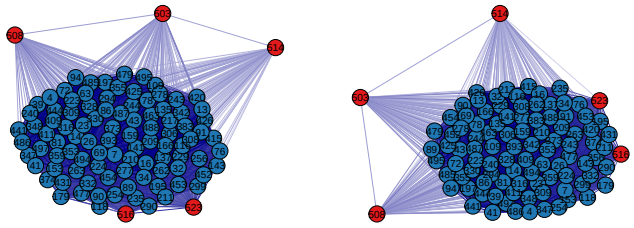



**(d)** G-Gower (trimmed units) **(e)** RelMS (trimmed units)

**Figure 3:** MDS configurations for the `Data_MC_contamination` dataset. Panels (a)–(c) display the configurations obtained using the Gower, G-Gower, and RelMS distances, respectively, where red points correspond to the true outliers introduced in the data generation process. In panels (d)–(e) highlighted points indicate the trimmed units to estimate the covariance matrix in Mahalanobis distance.



**(a)** Gower (outlying units) **(b)** G-Gower (trimmed units) **(c)** RelMS (trimmed units)



**(d)** G-Gower (outlying units) **(e)** RelMS (outlying units)

**Figure 4:** Similarity networks (qgraphs) for the `Data_MC_contamination` dataset. Panels (a)–(c) display the connectivity structure of the observations under the Gower, G-Gower, and RelMS distance measures, respectively. In panel (a), red nodes correspond to the true outliers introduced during data generation, whereas in panels (b) and (c), red nodes indicate the outliers detected by the trimming procedure. Panels (d)–(e) show the true outliers for G-Gower and RelMS, respectively, to facilitate comparison between detected and actual anomalous units.

It is worth emphasizing, however, that the trimming step implemented in the robust distances is not primarily intended as an outlier detection procedure. Its goal is to robustify the distance estimation by reducing the influence of observations that lie far from the bulk of the data, rather than to explicitly identify or label them as anomalous. The fact that many trimmed units coincide with the true outliers illustrates the effectiveness of this robustification process: by downweighting or excluding extreme observations, the robust distances recover a more faithful representation of the underlying data geometry.

Overall, the visual evidence provided by the MDS configurations and similarity networks generated using the `visualize_distances()` function from the `dbrobust` package suggests that the robust metrics are effective in isolating anomalous observations while preserving the structure of the regular data, thereby achieving a more reliable foundation for subsequent modeling or clustering tasks.

### 4.2 World development indicators

To illustrate the performance of the robust distance measures in a real-world setting, we analyze a dataset derived from the *World Development Indicators* (World Bank), available at https://www.kaggle.com/datasets/hn4ever/world-development-indicators-bycountries. The dataset comprises a set of socioeconomic indicators that capture diverse dimensions of national development with a sample size of 97 countries worldwide. Specifically, we use the observations corresponding to the year 2021, which is the latest year consistently available for the selected indicators. The numerical variables include the homicide rate (`Hom`, number of homicides per 100,000 inhabitants), the proportion of the population living on less than $3.20 per day adjusted for purchasing power parity (`B320`), life expectancy at birth (`LEx`), infant mortality rate (`IMo`, number of children dying before age one per 1,000 live births), the percentage of the population suffering from insufficient nutrition (`INu`), government expenditure on education as a percentage of GDP (`GEE`), and the proportion of arable land suitable for cultivation (`ArL`). Three binary variables describe key aspects of infrastructure and public services: the adequacy of the health system (`HSy`, with categories 1 = Adequate, 2 = Inadequate), access to electricity (`Ele`, with categories 1 = High, 2 = Low) and the poverty indicator (`Pov`, with two categories: 1 = High and 2 = Low) that classifies countries according to their general poverty status. In addition, two categorical variables provide contextual information: the continent of each country (`Con`, with five categories: Africa, Americas, Asia, Europe, and Oceania) and the pollution level (`Pol`, with three categories: 1 = High, 2 = Medium, 3 = Low).

It is important to note that the variables `Con` (Continent) and `Pov` (Poverty) were not included in the distance computation, as they were used exclusively for conditioning and interpretative purposes in the subsequent analyses.

Additionally, to account for the relative importance of each observation, a weighting variable was incorporated into the dataset based on national population size. Specifically, total population data for the most recent available year (2022) were retrieved from the DataHub repository (sourced from the World Bank). The resulting population figures were merged with the main dataset by country name, and normalized to create observation-specific weights. These weights reflect the demographic scale of each country and are employed in subsequent analyses to ensure that larger populations exert proportionally greater influence while preserving comparability among units.

To analyze this mixed-type dataset, we proceed as in the previous application. We start by computing the three distances, namely Gower, robust G-Gower and robust RelMS, and before their comparison, we include a heatmap representation for robust G-Gower metric (produced with `"heatmap"` option) for illustrative purposes (see Figure 5).

To illustrate the performance of the three distances, we present a comparative study whose main goal is to assess the behavior of the classical and robust approaches under realistic socioeconomic data conditions, where heterogeneity across countries and potential data irregularities (e.g., extreme values or nonlinear dependencies) may distort the analysis.

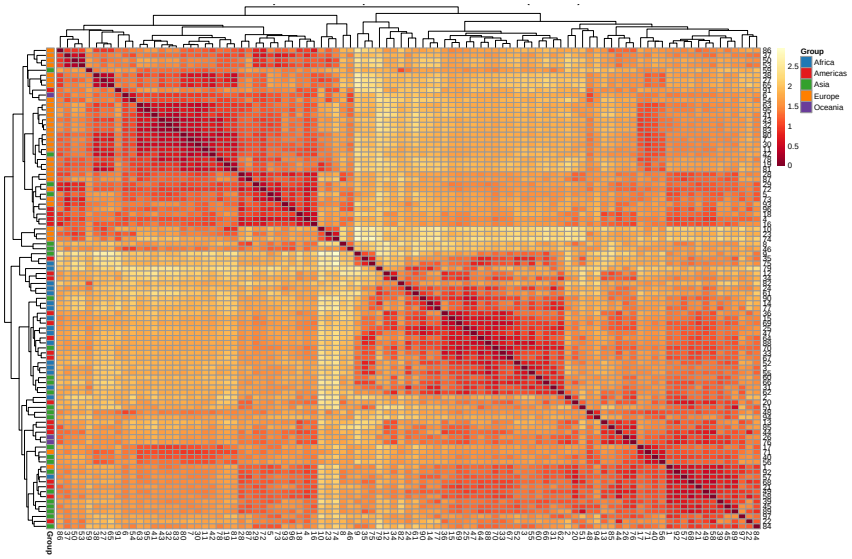
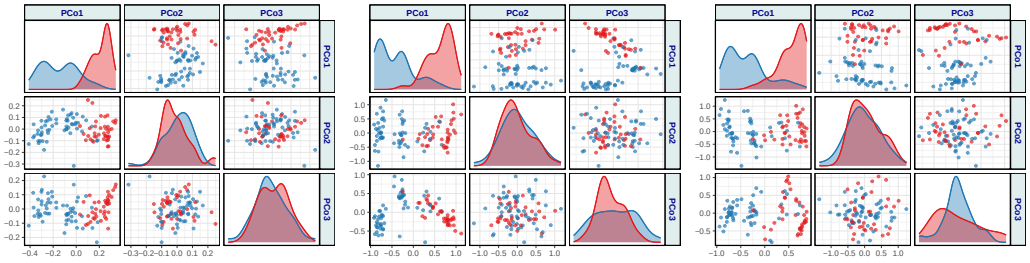



**Figure 5:** Heatmap representation for robust G-Gower pairwise distances. World Development Indicators.

To further illustrate these aspects, Figure 6 presents the multiple MDS plots with the classical MDS configurations obtained from the three distance matrices, with variable `Pov` (Poverty) as the conditioning factor.



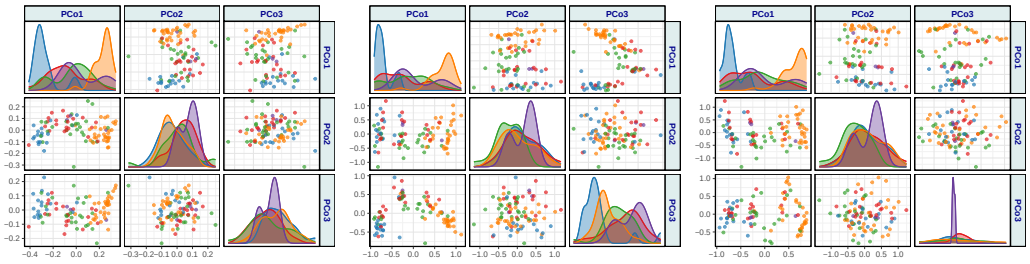

**(a)** Gower **(b)** G-Gower **(c)** RelMS

**Figure 6:** Multiple MDS configurations for the World Development Indicators dataset under three distance measures. Each panel displays the two-dimensional projection of 97 countries, colored by poverty level (`Pov`; Blue = High poverty level, Red = Low poverty level).

The MDS configurations in Figure 6 display the spatial arrangement of countries according to their socioeconomic similarity, grouped by poverty level (`Pov`). Under the classical Gower distance (a), two well-defined clusters emerge, corresponding to the high- and low-poverty groups, with only a few countries lying in intermediate positions. The same general structure is preserved under the robust G-Gower and RelMS distances (b)–(c), indicating that the dataset exhibits low-contamination conditions.

To gain further insight into the global development patterns, Figure 7 displays the multiple MDS plots with the classical MDS configurations obtained from the three distance matrices, using the variable `Con` (Continent) as the conditioning factor. Coloring the countries by continent allows us to assess whether geographic or regional factors are reflected in the latent similarity structure captured by each distance measure.

The MDS configurations in Figure 7 reveal a clear regional structure in the World Development Indicators dataset. Under the classical Gower distance (a), European and American countries form compact clusters, whereas African and Asian nations appear more dispersed, suggesting greater internal heterogeneity in their socioeconomic indicators. When the robust G-Gower and RelMS distances are applied (b)-(c), the global configuration becomes smoother and less dominated by extreme countries. The robust methods attenuate the influence of highly unbalanced indicators such as homicide rate or access to electricity,

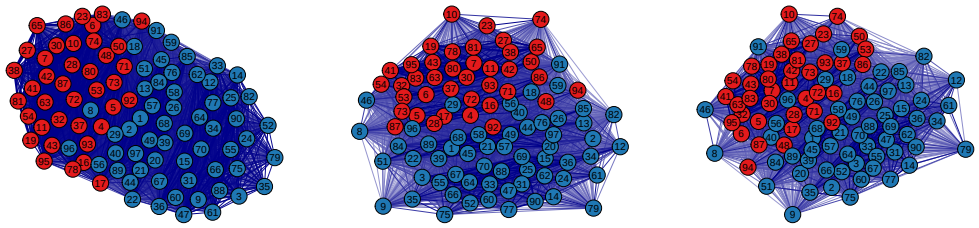



**(a)** Gower **(b)** G-Gower **(c)** RelMS

**Figure 7:** Multiple MDS configurations for the World Development Indicators dataset under three distance measures. Each panel displays the two-dimensional projection of 97 countries, colored by continent (`Con`; Blue = Africa, Red = America, Green = Asia, Orange = Europe, Purple = Oceania).

allowing a more balanced representation of intercontinental relationships. For example, countries from Asia and the Americas overlap more consistently, reflecting intermediate development profiles, while African countries remain more distinct but less isolated than in the classical solution.

To complement the MDS representations, Figure 8 displays the network structures derived from the three distance matrices using the `"qgraph"` visualization method, with countries grouped by their poverty level (`Pov`). In these networks, edges represent the strongest inter-country similarities, while node proximity and color reflect the degree of connectedness within and across poverty groups.



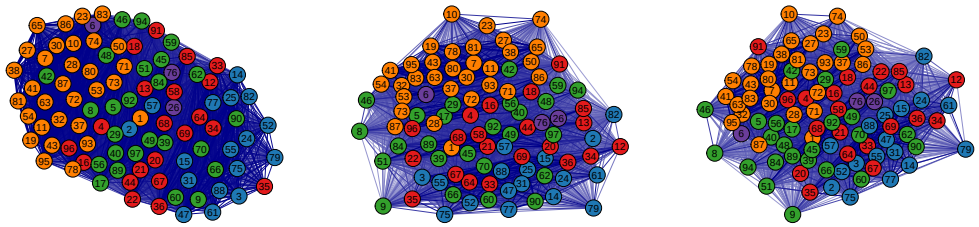
**(a)** Gower **(b)** G-Gower **(c)** RelMS

**Figure 8:** Network visualizations (`qgraph`) of the World Development Indicators dataset under three distance measures, with nodes colored by poverty level (`Pov`; Blue = High poverty level, Red = Low poverty level).

In the network representations of Figure 8, countries are grouped by poverty level. The classical Gower graph (a) displays two well-established clusters with a few overlapped observations, the same for G-Gower (b) and RelMS (c) demonstrating that the data present low-contamination conditions. When contamination is absent, robust methods perform comparably to classical ones.

Similarly, Figure 9 presents the `"qgraph"` representations for the same three distance measures, this time grouping countries by continent (`Con`). These network structures highlight regional cohesion and cross-continental linkages based on socioeconomic similarity.

The network visualizations in Figure 9 show that the three distance measures produce very similar connectivity patterns. In all cases Europe appears as a relatively compact and well-defined cluster, indicating homogeneous socioeconomic profiles among European countries in the dataset. The other continents (Africa, Asia and the Americas) display greater internal variability and, in some areas, partially overlapping connections; this overlapping is present across the Gower, G-Gower and RelMS graphs and is not visibly reduced by the robust alternatives. In short, the robust distances do not produce a markedly different network topology here, a finding consistent with a dataset that lacks strong influential outliers: when contamination is absent or weak, robustification yields configurations comparable to the classical measure rather than “improvements” in network structure.



**(a)** Gower **(b)** G-Gower **(c)** RelMS

**Figure 9:** Network visualizations (`qgraph`) of the World Development Indicators dataset under three distance measures, with nodes colored by continent (`Con`; Blue = Africa, Red = America, Green = Asia, Orange = Europe, Purple = Oceania).

Overall, the empirical results obtained in this application indicate that the robust distance measures, G-Gower and RelMS, do not outperform the classical Gower distance in either the MDS or network representations. This outcome is conceptually consistent with the theoretical rationale of robust methods: their purpose is not to enhance performance under regular data conditions, but to ensure stability and resilience in the presence of contamination, extreme observations, or structural imbalance. In this dataset, characterized by moderate variability and limited influence of atypical cases, the classical and robust approaches converge toward similar configurations. Rather than aiming to highlight the superiority of robust measures, this application was intended to demonstrate the practical implementation and versatility of the `dbrobust` package, illustrating how it can be applied to complex, real-world socioeconomic data.

## 5 Conclusion

This work introduces **dbrobust**, a comprehensive R package designed to address the methodological and practical challenges of robust DB analysis in datasets of mixed-type. Through a unified framework, the package enables the computation and visualization of classical and robust distances across mixed variable types, offering tools that are both accessible and extensible for applied researchers.

The proposed methodology demonstrates advantages in handling data contamination, preserving structural patterns, and enhancing interpretability. By integrating novel robust formulations and advanced visualization techniques, **dbrobust** facilitates exploratory and confirmatory analyses that are resilient to outliers and adaptable to diverse research contexts. In the current framework, robustness is primarily introduced through the continuous component of the mixed-data dissimilarities, whereas binary and categorical variables are handled through classical similarity and dissimilarity measures. Importantly, the package bridges the gap between theoretical innovation and practical implementation, supporting seamless interoperability with existing R tools and workflows. Its utility is particularly relevant in fields such as social sciences, quality of life research, and gender inequality studies, where weighted complex data structures and robustness requirements are common.

The development of **dbrobust** represents a significant contribution to the statistical ecosystem, empowering analysts to extract meaningful insights from complex, real-world datasets through robust and interpretable DB methods. Future research may extend the current framework by incorporating robust alternatives for binary and categorical variables, further broadening the scope of robust mixed-data analysis. Its seamless compatibility with the **dbstats** package for DB regression modeling, as well as with other widely used R packages that rely on distance inputs, such as those for clustering and classification, ensures its integration into existing analytical workflows and reinforces its practical value for applied research.

## CRediT authorship contribution statement

**Marcos Álvarez:** Software, Writing – Original draft preparation, Investigation. **Eva Boj:** Conceptualization, Methodology, Software, Writing – Original draft preparation, Reviewing and Editing, Investigation, Supervision. **Aurea Grané:** Conceptualization, Methodology, Software, Writing – Original draft preparation, Reviewing and Editing, Investigation, Supervision.

## Funding

The authors gratefully acknowledge the support of grants PID2021-123592OB-I00 and TED2021-129316B-I00 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe" and "European Union NextGenerationEU/PRTR".

## References

I. Albarrán, P. J. Alonso, and A. Grané. Profile identification via weighted related metric scaling: An application to dependent spanish children. *Journal of the Royal Statistical Society Series A: Statistics in Society*, 178(3):593–618, 2015. URL https://doi.org/10.1111/rssa.12084. [p8, 9]

E. Boj and A. Grané. The robustification of distance-based linear models: Some proposals. *Socio-Economic Planning Sciences*, 95:101992, 2024. URL https://doi.org/10.1016/j.seps.2024.101992. [p4, 8, 10, 14, 18]

E. Boj, P. Delicado, and J. Fortiana. Distance-based local linear regression for functional predictors. *Computational Statistics & Data Analysis*, 54(2):429–437, 2010. URL https://doi.org/10.1016/j.csda.2009.07.013. [p3, 8]

E. Boj, A. Caballé, P. Delicado, A. Esteve, and J. Fortiana. Global and local distance-based generalized linear models. *TEST*, 25(1):170–195, 2016. URL https://doi.org/10.1007/s11749-015-0447-1. [p8]

E. Boj, A. Caballé, P. Delicado, and J. Fortiana. dbstats: Distance-based statistics. R package version 2.0.3, 2025. URL https://doi.org/10.32614/CRAN.package.dbstats. [p1, 8]

E. Boj, A. Grané, and A. Mayo-Íscar. Robust distance-based generalized linear models: a new tool for classification. *Statistical Methods and Applications*, 2026. URL https://doi.org/10.1007/s10260-026-00860-1. [p4, 10, 14]

I. Borg and P. J. F. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005. URL https://doi.org/10.1007/0-387-28981-X. [p3]

C. M. Cuadras. Distance analysis in discrimination and classification using both continuous and categorical variables. In Y. Dodge, editor, *Statistical Data Analysis and Inference*, pages 459–473. North-Holland, Amsterdam, 1989. URL https://doi.org/10.1016/B978-0-444-88029-1.50047-2. [p5, 6]

C. M. Cuadras and J. Fortiana. A continuous metric scaling solution for a random variable. *Journal of Multivariate Analysis*, 52(1):1–14, 1995. URL https://doi.org/10.1006/jmva.1995.1001. [p3]

C. M. Cuadras and J. Fortiana. Visualizing categorical data with related metric scaling. In J. Blasius and M. Greenacre, editors, *Visualization of Categorical Data*, pages 365–376. Academic Press, Elsevier, 1998. [p8]

M. D'Orazio. Statmatch: Statistical matching or data fusion. R package version 1.4.3, 2025. URL https://doi.org/10.32614/CRAN.package.StatMatch. [p4]

S. Dray, A.-B. Dufour, and D. Chessel. The ade4 package – ii: Two-table and k-table methods. *R News*, 7(2):47–52, 2007. URL https://cran.r-project.org/doc/Rnews/. [p4]

S. Epskamp, A. O. J. Cramer, L. J. Waldorp, V. D. Schmittmann, and D. Borsboom. qgraph: Network visualizations of relationships in psychometric data. *Journal of Statistical Software*, 48(4):1–18, 2012. URL https://doi.org/10.18637/jss.v048.i04. [p1, 15]

G. F. Estabrook and D. J. Rogers. A general method of taxonomic description for a computed similarity measure. *BioScience*, 16(11):789–793, 1966. URL https://doi.org/10.2307/1293644. [p7]

J. C. Gower. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, 53(3–4):325–338, 1966. URL https://doi.org/10.1093/biomet/53.3-4.325. [p2]

J. C. Gower. A general coefficient of similarity and some of its properties. *Biometrics*, 27(4): 857–871, 1971. URL https://doi.org/10.2307/2528823. [p7]

J. C. Gower and P. Legendre. Metric and euclidean properties of dissimilarity coefficients. *Journal of Classification*, 3(1):5–48, 1986. URL https://doi.org/10.1007/BF01896809. [p2, 3, 5, 6, 7, 8]

A. Grané, S. Salini, and G. Infante. New distances for mixed-type data able to cope with redundant information. *AStA Advances in Statistical Analysis*, 2026. URL https://doi.org/10.1007/s10182-026-00565-6. [p7, 8]

A. Grané and R. Romera. On visualizing mixed-type data: A joint metric approach to profile construction and outlier detection. *Sociological Methods & Research*, 47(2):207–239, 2018. URL https://doi.org/10.1177/0049124115621334. [p8]

A. Grané, G. Manzi, and S. Salini. Smart visualization of mixed data. *Stats*, 4(2):472–485, 2021a. URL https://doi.org/10.3390/stats4020029. [p8]

A. Grané, S. Salini, and E. Verdolini. Robust multivariate analysis for mixed-type data: Novel algorithm and its practical application in socio-economic research. *Socio-Economic Planning Sciences*, 73:100907, 2021b. URL https://doi.org/10.1016/j.seps.2020.100907. [p8, 9, 10]

L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, New York, 1990. ISBN 0471878766. [p8]

W. J. Krzanowski. Ordination in the presence of group structure, for general multivariate data. *Journal of Classification*, 11:195–207, 1994. URL https://doi.org/10.1007/BF01195679. [p8]

J. C. Lingoes. Some boundary conditions for a monotone analysis of symmetric matrices. *Psychometrika*, 36(2):195–203, 1971. URL https://doi.org/10.1007/BF02291398. [p3, 13]

M. Maechler, P. Rousseeuw, A. Croux, C. Todorov, A. Ruckstuhl, M. Salibian-Barrera, T. Verbeke, M. Koller, E. L. T. Conceicao, and M. di Palma. *robustbase: Basic Robust Statistics*, 2025. URL https://doi.org/10.32614/CRAN.package.robustbase. R package version 0.99-4-1. [p5]

M. Maechler, P. Rousseeuw, A. Struyf, M. Hubert, and K. Hornik. cluster: Cluster analysis basics and extensions. R package version 2.1.8.2, 2026. URL https://doi.org/10.32614/CRAN.package.cluster. [p1, 4, 8]

K. V. Mardia. Some properties of classical multi-dimensional scaling. *Communications in Statistics – Theory and Methods*, 7(13):1233–1241, 1978. URL https://doi.org/10.1080/03610927808827707. [p3, 13]

D. Meyer and C. Buchta. proxy: Distance and similarity measures. R package version 0.4-29, 2025. URL https://doi.org/10.32614/CRAN.package.proxy. [p1]

J. Oksanen, G. L. Simpson, F. G. Blanchet, R. Kindt, P. Legendre, P. R. Minchin, R. B. O'Hara, P. Solymos, M. H. H. Stevens, E. Szoecs, H. Wagner, M. Barbour, M. Bedward, B. Bolker, D. Borcard, G. Carvalho, M. Chirico, M. De Caceres, S. Durand, H. B. A. Evangelista, R. FitzJohn, M. Friendly, B. Furneaux, G. Hannigan, M. O. Hill, L. Lahti, D. McGlinn, M.-H. Ouellette, E. Ribeiro Cunha, T. Smith, A. Stier, C. J. F. Ter Braak, J. Weedon, and T. Borman. vegan: Community ecology package. R package version 2.7-5, 2026. URL https://doi.org/10.32614/CRAN.package.vegan. [p1]

E. Paradis and K. Schliep. ape 5.0: an environment for modern phylogenetics and evolutionary analyses in R. *Bioinformatics*, 35:526–528, 2019. URL https://doi.org/10.1093/bioinformatics/bty633. [p17]

J. Podani. Extending gower's general coefficient of similarity to ordinal characters. *Taxon*, 48(2):331–340, 1999. URL https://doi.org/10.2307/1224438. [p8]

R Core Team. R: A language and environment for statistical computing, 2026. URL https://www.R-project.org/. [p1]

B. D. Ripley and W. N. Venables. *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]

B. Schloerke, D. Cook, J. Larmarange, F. Briatte, M. Marbach, E. Thoen, A. Elberg, and J. Crowley. Ggally: Extension to ggplot2. R package version 2.4.0, 2025. URL https://doi.org/10.32614/CRAN.package.GGally. [p15]

Z. Sulc, J. Cibulkova, H. Rezankova, and J. Hornicek. nomclust: Hierarchical cluster analysis of nominal data. R package version 2.8.1, 2025. URL https://doi.org/10.32614/CRAN.package.nomclust. [p7]

V. Todorov and P. Filzmoser. rrcov: Scalable robust estimators with high breakdown point. *R Journal*, 1(2):27–41, 2009. URL https://doi.org/10.18637/jss.v032.i03. [p5]

M. van der Loo. gower: Gower's distance. R package version 1.0.2, 2024. URL https://doi.org/10.32614/CRAN.package.gower. [p4, 8]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://doi.org/10.1007/978-3-319-24277-4. [p1]

M. Álvarez, E. Boj, and A. Grané. dbrobust: Robust distance-based visualization and analysis of mixed-type data. R package version 1.1.1, 2026. URL https://doi.org/10.32614/CRAN.package.dbrobust. [p1]

*Eva Boj*
*Department of Economic, Financial and Actuarial Mathematics*
*Universitat de Barcelona*
*Avinguda Diagonal 690, 08034 Barcelona*
*Spain*
*ORCID: 0000-0002-9487-0693*
evaboj@ub.edu

*Aurea Grané*
*Statistics Department*
*Universidad Carlos III de Madrid*
*Calle Madrid 126, 28903 Getafe*
*Spain*

*ORCID: 0000-0003-0980-6409*
aurea.grane@uc3m.es

*Marcos Álvarez*
*Statistics Department*
*Universidad Carlos III de Madrid*
*Calle Madrid 126, 28903 Getafe*
*Spain*
*ORCID: 0009-0006-6192-0152*
marcoalv@pa.uc3m.es

# Appendix: Internal and logical modular structure of functions in dbrobust

## 1 Internal and logical modular structure of function for computing dissimilarities and similarities

The `calculate_distances` function is designed following a modular architecture, which delegates specific tasks to helper functions depending on the data type and distance method. Table 4 summarizes each internal component and its role.

**Table 4:** Modular structure of function for computing dissimilarities and similarities.

| Function | Description |
|---|---|
| `calculate_distances` | Main entry point and wrapper function. Validates arguments, selects method category, delegates computation to specialized helper functions, and formats values. |
| `distBinary` | Computes pairwise distances for binary data using standard measures such as Jaccard, Dice, and Sokal-Michener. Uses **ade4** where appropriate. |
| `distCategorical` | Computes distances for categorical variables using the matching coefficient (proportion of attribute matches) or the Cramér-based distance which accounts for the pair-wise association between variables. |
| `distContinuous` | Handles numerical data using measures such as Euclidean, Minkowski, Mahalanobis, Cosine, and Correlation. Uses base R and **proxy** where appropriate. |
| `distMixed` | Implements Gower's distance for mixed-type data. Continuous variables are scaled by range; binary variables can be treated as symmetric or asymmetric (controlled by `binaryAsym`; only 1/1 counts as a match); categorical variables are compared by exact match. Users can optionally specify which columns are continuous, binary, or categorical using the arguments `continuousCols`, `binaryCols`, and `categoricalCols`. Similarities are converted to distances ($d = 1 - s$), yielding a symmetric matrix in [0,1]. |
| `formatOutput` | Converts distance matrix into the desired value format: `dist`, full matrix, or similarity matrix using selected transformation. |
| `convertToDist` | Utility function to coerce similarity matrices into `dist` objects. Supports linear and square-root transformations. |

Modularizing the `calculate_distances()` function is crucial for maintaining clarity, flexibility, and ease of maintenance. By breaking down the computation into specialized helper functions based on data type and distance measure, the function becomes more organized and easier to extend or debug. This approach also promotes code reuse and separation of concerns, allowing each component to focus on a specific task. Furthermore, leveraging established external libraries such as **proxy** or **ade4** whenever possible ensures efficient implementations of complex distance calculations, reduces development time, and benefits from community-tested and optimized algorithms, ultimately enhancing the reliability and performance of the values returned by the function.

The function `calculate_distances()` might be intricate and modular, yet its logic can be broken down into a series of conceptual steps, which are outlined in Table 5.

**Table 5:** Internal logical structure of function for computing dissimilarities and similarities and helper functions

| Step | Description |
|---|---|
| Argument validation | Check `x` type, `outputFormat`, and `p` for Minkowski. |
| Normalize method | Standardize method string (lowercase, underscores). |
| Select distance function | Choose appropriate function based on method category (binary, categorical, continuous, mixed). This acts as a wrapper for the specialized routines. |
| Compute distances | Call `distBinary`, `distCategorical`, `distContinuous`, or `distMixed`. |
| Optional transformations | Square distances if requested; convert to similarity using linear or sqrt transformation. |
| Return values | Return values as `dist` object, numeric matrix, or similarity matrix. |

## 2 Internal and logical modular structure of function for ensuring the Euclideanarity of matrices of pairwise squared distances

While `make_euclidean()` is implemented as a single function, its logic can be decomposed into conceptual steps, summarized in Table 6.

**Table 6:** Internal logical structure of function for ensuring Euclideanarity of matrices of pairwise squared distances

| Step | Description |
|---|---|
| Argument validation | Ensure the matrix argument `D` is a square matrix, and the weight vector `w` has matching length. |
| Weight normalization | Normalize `w` to sum to 1. |
| Compute weighted Gram matrix | Apply centering matrix $\mathbf{J}_w$ to $\mathbf{D}$ to obtain $\mathbf{G}_w$. |
| Eigenvalue analysis | Extract eigenvalues of $\mathbf{G}_w$ and check smallest value against `tol`. |
| Correction step | If $\lambda_{\min} < -\texttt{tol}$, shift $\mathbf{D}$ by $2\,\lvert\lambda_{\min}\rvert$ off-diagonal. |
| Recompute eigenvalues | Compute eigenvalues of corrected $\mathbf{G}_w$ to verify positive semi-definiteness. |
| Return values | Return a list containing the corrected $\mathbf{\Delta}$ matrix, eigenvalues before/after, and transformation flag. |

## 3 Internal and logical modular structure of function for computing robust distances

**Table 7:** Internal logical structure of function for computing robust distances

| Step | Description |
|---|---|
| Argument validation | Ensures that either `p` or all variable groups are provided, checks for missing variables, and forbids `NA` values in the selected columns. |
| Data preparation | Reorders variables, converts factors/characters to numeric codes, and assembles the data matrix in [cont, bin, cat] order. |
| Weight handling | Sets equal weights if `w` is `NULL`; otherwise, normalizes and validates the user-supplied weights argument. |
| Robust covariance estimation | If continuous variables are present and no covariance matrix is supplied by the user via the argument `robust_cov`, a robust covariance estimator is computed according to the argument `cov_method`. The option `"gv"` calls `robust_covariance_gv`, which estimates the covariance matrix using the geometric variability trimming approach controlled by the trimming proportion `alpha`. Alternatively, `"mcd"` calls `robust_covariance_mcd`, which computes a weighted Minimum Covariance Determinant (wMCD) estimator using an iterative C-step algorithm with multiple random initializations. Both estimators return the robust covariance matrix together with the corresponding diagnostic information (central observations, detected outliers, and trimming statistics). |
| Squared distance computation | Depending on `method`, calls either `robust_ggower` or `robust_RelMS` to combine distances from each variable type. |
| Attach diagnostics | If trimming was performed, attaches indices of central and outlier units, the $\phi$ proximity values, and trimming quantile `q` as attributes of the result. |
| Return values formatting | Returns either squared distances `D2` or converts to a `dist` object with `D2toDist` if `return_dist = TRUE`. |

The `robust_distances()` function is designed following a modular architecture as well as `calculate_distances()`, which delegates specific tasks to helper functions depending on the robust distance construction method. Tables 7–8 summarize each internal and modular component and its role.

The robust aggregated distances computed by `robust_distances()` provide a solid foundation for subsequent multivariate analyses in mixed-type data settings. These robust metrics can be seamlessly integrated, for instance, with the **dbstats** package, enabling robust DB regression models. This compatibility arises from the fact that `robust_distances()` can

**Table 8:** Modular structure of function for computing robust distances and helper functions

| Function | Description |
|---|---|
| `robust_covariance_gv` | Computes a robust covariance matrix for continuous variables using the weighted Mahalanobis distance approach summarized in Figure 2 (see Section 3. Identifies central observations via a trimming procedure controlled by `alpha`, excluding outliers from the covariance estimation. Returns the covariance matrix, indices of central and outlier observations, the proximity measure ($\phi$), and the trimming threshold (`q`). |
| `robust_covariance_mcd` | Computes a weighted Minimum Covariance Determinant (wMCD) estimator for the continuous variables. This estimator provides a highly robust covariance matrix by identifying a subset of observations with minimum covariance determinant, while allowing observation weights. The resulting covariance matrix can subsequently be used by both `robust_ggower` and `robust_RelMS`. |
| `robust_ggower` | Computes a robust G-Gower distance for mixed data by replacing the normalized city-block distance for continuous variables with a robust Mahalanobis distance using trimming to mitigate outliers. Keeps classical definitions for binary and categorical variables and combines all three types by simple weighted summation. Purpose: robustify distances in presence of continuous outliers without altering the classical fusion scheme. |
| `robust_RelMS` | Extends `robust_ggower` by integrating continuous, binary, and categorical sources through RelMS. Uses double-centering and interaction terms to reduce redundancy across variable types. Employs robust covariance and trimming for continuous data as before. Purpose: provide a more structured, robust, and less redundant joint metric for multitype data. |

return its results either as `D2` objects or as `dist` objects, two of the accepted argument formats for functions such as `dblm` or `dbglm` in **dbstats**.

## 4 Internal and logical modular structure of function for merging distances

The internal workflow of `merge_distances()`, including argument validation, Euclidean correction, weighted aggregation, and consensus distance reconstruction, is summarized in Tables 9–10. The modular design facilitates the integration of heterogeneous distance sources while ensuring computational consistency and preserving Euclidean properties throughout the merging process.

**Table 9:** Internal logical structure of function for merging distances

| Step | Description |
|---|---|
| Argument validation | Checks that at least two distance matrices are supplied, converts `dist` objects into matrices, verifies that all matrices are square with identical dimensions, and ensures that all distances are non-negative. |
| Weight handling | Validates and normalizes both the observation weights (`w`) and the matrix weights (`matrix_weights`). When either argument is omitted, equal weights are assigned before normalization. |
| Euclidean preprocessing | Converts the input distance matrices into squared distances when necessary and applies `make_euclidean()` to each matrix to guarantee Euclidean properties before fusion. |
| Gram matrix extraction | Computes the weighted Gram matrices and their square-root decompositions for each distance source using weighted double-centering. |
| Source weighting | Incorporates the user-defined matrix weights into both the Gram matrices and their square-root representations. |
| Consensus Gram matrix construction | Depending on the value of `RelMS`, either computes a direct weighted aggregation of the Gram matrices or subtracts cross-source interaction terms computed by `relms_compute_cross_terms()` to reduce redundancy among sources. |
| Distance reconstruction | Reconstructs the consensus squared distance matrix from the joint Gram matrix and removes possible numerical negative values. |
| Output formatting | Applies a final Euclidean correction using `make_euclidean()` and returns either the consensus squared distance matrix or a `dist` object according to the value of the argument `squared`. |

**Table 10:** Modular structure of function for merging distances and helper functions

| Function | Description |
|---|---|
| `make_euclidean` | Transforms non-Euclidean distance matrices into Euclidean ones using the Lingoes correction, ensuring that all input and output distance matrices satisfy Euclidean properties. |
| `relms_compute_cross_terms` | Computes the cross-interaction terms between weighted Gram matrix square roots using a C++ implementation based on **RcppArmadillo**. These interaction terms are used by RelMS to reduce redundancy among multiple distance sources. |
| `merge_distances` | Combines multiple distance matrices into a consensus distance matrix by either direct weighted aggregation or Related Metric Scaling (RelMS), returning the result either as a squared distance matrix or as a `dist` object. |

The `merge_distances()` function provides a flexible framework for integrating multiple distance matrices into a single consensus representation. By supporting both direct weighted aggregation and the Related Metric Scaling (RelMS) methodology, the function allows users to combine complementary sources of information while reducing redundancy among them when the RelMS option is selected. The resulting consensus distance matrix can be directly employed in subsequent visualization, clustering, multidimensional scaling, or distance-based statistical analyses available in **dbrobust** and compatible packages such as **dbstats**.

## 5 Internal and logical modular structure of function for visualizing distances and similarities

The internal workflow of `visualize_distances()`, including argument validation, method selection, data preparation, and execution of the plotting routines, is summarized in Tables 11–12. From version `1.1.0`, the visualization framework has been extended to support user-defined group color schemes through the `group_colors` argument. When no custom colors are provided, the internal helper function `get_custom_palette()` automatically generates a palette of visually distinct colors.

**Table 11:** Internal logical structure of function for visualizing distances and similarities

| Step | Description |
|---|---|
| Argument validation | Checks that `dist_mat` is square and symmetric, and validates the arguments `k`, `weights`, `group`, and the optional `group_colors`. |
| Method selection | Matches the `method` argument and directs execution to the corresponding plotting routine. |
| Data preparation | Converts distance matrices to the appropriate format, applies weights if relevant, assigns group labels, and prepares either user-defined or automatically generated color schemes for visualization. |
| Visualization execution | Calls the internal plotting functions (`plot_mds`, `plot_heatmap`, and `plot_qgraph`) according to the selected `method`. |
| Return values | Returns the corresponding plotting object or produces the graphical output, depending on the selected visualization method (`ggmatrix` for MDS, `pheatmap` for heatmaps, and direct plotting for `qgraph`). |

The `visualize_distances()` function provides a unified interface for exploring distance matrices through complementary graphical representations. By combining multidimensional scaling, heatmaps, and network visualizations within a common workflow, it facilitates the interpretation of complex distance structures while allowing users to incorporate grouping information through customizable visual annotations.

**Table 12:** Modular structure of function for visualizing distances and similarities and helper functions.

| Function | Description |
|---|---|
| `visualize_distances` | Main interface for visualizing distance matrices. Validates that the matrix is square, symmetric, and has zero diagonals. Handles optional arguments such as `k`, `weights`, `group`, `group_colors`, and `main_title`. Delegates plotting tasks to `plot_mds`, `plot_heatmap`, or `plot_qgraph` according to the selected `method`. |
| `plot_mds` | Performs classical (`cmdscale`) or weighted (`wcmdscale`) multidimensional scaling. Generates pairwise scatterplot matrices with density plots on the diagonal and supports group-based coloring to facilitate the interpretation of the projected observations. |
| `plot_heatmap` | Plots a heatmap from a distance matrix, with hierarchical clustering and optional group annotations. Supports subsampling, color palette customization, and annotation colors for categorical group variables. |
| `plot_qgraph` | Constructs a network graph from a distance matrix using **qgraph**. Converts distances to similarities, applies edge thresholding, supports group coloring, and limits the number of displayed nodes. |
| `get_custom_palette` | Generates a palette of visually distinct colors for group annotations. Returns a predefined set of colors or interpolates additional colors when more groups are present, providing the default color scheme whenever `group_colors` is not specified. |

# xplainfi: Feature Importance and Statistical Inference for Machine Learning in R

*by Lukas Burk, Fiona Katharina Meinzer, Giuseppe Casalicchio, Marvin N. Wright, and Bernd Bischl*

**Abstract** We introduce xplainfi, an R package built on top of the mlr3 ecosystem for global, loss-based feature importance methods for machine learning models. Various feature importance methods exist in R, but significant gaps remain, particularly regarding conditional importance methods and associated statistical inference procedures. The package implements permutation feature importance, conditional feature importance, relative feature importance, leave-one-covariate-out, and generalizations thereof, and both marginal and conditional Shapley additive global importance methods. It provides a modular conditional sampling architecture based on Gaussian distributions, adversarial random forests, conditional inference trees, and knockoff-based samplers, which enable conditional importance analysis for continuous and mixed data. Statistical inference is available through multiple approaches, including variance-corrected confidence intervals and the conditional predictive impact framework. We demonstrate that xplainfi produces importance scores consistent with existing implementations across multiple simulation settings and learner types, while offering competitive runtime performance. The package is available on CRAN and provides researchers and practitioners with a comprehensive toolkit for feature importance analysis and model interpretation in R.

## 1 Introduction

In machine learning (ML), understanding feature-target relationships is increasingly valued alongside predictive accuracy. Complex models can capture nonlinear patterns but are typically considered "black box" models due to their opaque internal mechanisms, which has motivated the development of interpretable machine learning (IML) methods (Molnar, 2020; Murdoch et al., 2019). Among these, feature importance (FI) methods aim to quantify the relevance of input features for a given prediction target, such as a fixed model, the learning algorithm that produced it, or the underlying data-generating process, thereby providing insight into what drives predictive behavior (Murdoch et al., 2019; Fisher et al., 2019). The applications range from increasing our understanding of a given data-generating process (DGP) to practical issues such as feature selection (Guyon and Elisseeff, 2003; Guidotti et al., 2018).

We present **xplainfi**, an R package that provides a unified interface for computing and comparing global, loss-based FI methods, which measure the change in predictive performance when features are removed, perturbed, or marginalized (Ewald et al., 2024). Alternative approaches out of scope include local FI methods (i.e., FI values for individual predictions such as Shapley values (see Shapley, 1953; Rozemberczki et al., 2022)) and variance-based sensitivity measures, such as Sobol indices (Sobol', 2001). When we refer to FI methods from here on, we mean global, loss-based FI methods unless otherwise stated. Various FI methods are implemented across packages in both the R and Python ecosystems, but in the R ecosystem, some methods are completely absent. **xplainfi** aims to fill this gap by providing a larger collection of FI methods than previously available, along with p-values and confidence intervals for FI uncertainty quantification, which are often needed in applications. The package is built on top of the **mlr3** ecosystem (Lang et al., 2019), which allows development to focus on the FI methods themselves without re-implementing common building blocks such as abstractions for learning algorithms, resampling, tuning, and pipelines (Binder et al., 2021).

Because many real-world datasets exhibit dependent features, marginal, perturbation-based FI methods suffer from two distinct issues. First, breaking the dependence structure forces the model to extrapolate to implausible feature combinations, potentially yielding unreliable predictions and hence FI values (Hooker et al., 2021). Second, importance is often distributed across dependent predictors (if they are reflected in the model), which is not necessarily a problem, but users need to be aware of it at the very least (Nicodemus et al., 2010; Debeer and Strobl, 2020). Conditional importance methods address this by assessing performance changes under interventions that preserve feature dependencies (Watson and Wright, 2021), yet remain underrepresented in available implementations. One challenge with these methods is the need for suitable conditional sampling methods, which is why **xplainfi** provides multiple methods for continuous and mixed data. Both conditional importance methods and the ability to handle mixed data are motivated by common issues arising in practical statistical work and data analysis, and they are also areas of active research (see also Blesch et al., 2024; Redelmeier et al., 2020). To that end, we offer **xplainfi** as a comprehensive, extensible tool to support these efforts.

### Contributions

**xplainfi** provides a unified framework for FI methods built on the **mlr3** ecosystem. Key contributions include: 1) Implementation of many standard FI methods, including permutation feature importance (PFI) (Breiman, 2001), conditional feature importance (CFI) (Strobl et al., 2008; Debeer and Strobl, 2020), leave-one-covariate-out (LOCO) (Lei et al., 2018), relative feature importance (RFI) (König et al., 2021), and both marginal and conditional SAGE (Shapley Additive Global importancE) (Covert et al., 2020); 2) Seamless integration with **mlr3**'s learners, tasks, measures, and resampling strategies; 3) A modular conditional sampling interface supporting Gaussian, adversarial random forests (ARF) (Watson et al., 2023), conditional inference trees (Hothorn et al., 2006), and knockoff-based samplers (Candès et al., 2018); 4) The option to target model, learner, or DGP importance by leveraging **mlr3**'s support for resampling, ensembling, and tuning; 5) Uncertainty quantification via variance-corrected confidence intervals (Nadeau and Bengio, 2003), observation-level LOCO inference (Lei et al., 2018), and the CPI testing framework (Watson and Wright, 2021).

The paper is structured as follows: First, we give a brief overview of the implemented global, loss-based FI methods and ways to quantify their uncertainty in Section 2. In Section 3, we present other R and Python packages that implement FI methods. Section 4 gives an introduction to **xplainfi** and showcases core functionality. Example scenarios, along with comparisons to existing implementations based on FI values and runtime, are provided in Section 5. We conclude with a discussion and outlook on **xplainfi** in Section 6.

Reproducibility and availability: **xplainfi** is on CRAN and maintained on GitHub. The GitHub repository contains all the code required to reproduce the results. The code and results are also included in the online supplement.

## 2 Implemented feature importance methods

We briefly review the FI methods implemented in **xplainfi**, drawing on the comprehensive overview in Ewald et al. (2024), to which we refer for a broader discussion and additional references. We also summarize the statistical inference procedures supported by the package. Rather than providing a full theoretical treatment, we focus on the estimands and quantities that are required for computation and implementation.

**Notation**: We assume the supervised learning setting with $n$ observations of $p$ features $X = (X_1, \ldots, X_p)$ and a target $Y$. A model $\hat{f}$ is trained to predict $Y$ from $X$, and its performance is evaluated using a loss function $L(Y, \hat{f}(X))$. The *feature of interest* (FOI) is denoted $X_j$, with $X_{-j}$ representing all remaining features. More generally, $X_S$ denotes a subset of features indexed by $S \subseteq \{1, \ldots, p\}$. Feature importance for feature $j$ is denoted $\text{FI}_j$, with specific methods indicated by subscripts (e.g., $\text{PFI}_j$, $\text{LOCO}_j$).

One central distinction between FI methods is whether they target *marginal* (unconditional) association between a feature and the target, $X_j \not\perp\!\!\!\perp Y$, or *conditional* association given a set of features $G$, $X_j \not\perp\!\!\!\perp Y \mid X_G$ (Strobl et al., 2008; Watson and Wright, 2021). Many well-established model-independent methods target marginal association (e.g., correlation- or information-theoretic measures) (Li et al., 2018; Bommert et al., 2020).

In the paper and **xplainfi**, we primarily focus on conditional association of $X_j$ with $Y$, often with the important special case of conditioning on all remaining features $X_{-j}$. This captures the *incremental* predictive value of a feature when the others are already available, and it entails a substantially harder estimation problem than marginal association, typically requiring predictive models and procedures that respect the joint structure of the features. Conditioning on arbitrary feature subsets is also supported by **xplainfi**, e.g., via relative feature importance (RFI) (see Section 2.2). Importantly, the loss-based FI methods considered below quantify *predictive* importance (or reliance) under a particular intervention scheme (e.g., permutation, conditional resampling, refitting, or marginalization) and are therefore not equivalent to generic measures of statistical dependence. For certain population-level (oracle) importance parameters—including refitting-based measures in the framework of Williamson et al. (2023) under suitable losses—a zero importance parameter can be equivalent to conditional independence, but FI does not imply a causal effect of $X_j$ on $Y$, and estimated FI values can depend on the chosen intervention distribution as well as the presence of correlated or redundant predictors.

### 2.1 Estimands and statistical inference

There are three possible targets of an FI analysis, corresponding to distinct estimands: a fitted *model*, a given *learner*, or the true prediction function regarding the DGP, also called population-level inference (see also Chiaburu et al., 2024; Molnar et al., 2023). *Model importance* analyzes a single, fixed model

and quantifies how it uses features for prediction. *Learner importance* targets the learning algorithm which produces the model when training data is sampled from the DGP of a given sample size. Unlike model importance, it captures how much models of this learner class rely on each feature on average, and requires refitting via resampling across multiple model instantiations on different test data. *DGP importance* aims to explain the true feature-target relationship by analyzing the population-level prediction function (or Bayes-optimal predictor), e.g., $f_0(x) = \mathbb{E}[Y \mid X = x]$ under $\ell_2$ loss. As we usually never have direct access to this Bayes-optimal predictor in practical applications, this function must then be approximated by a strong learner in an AutoML-like fashion, e.g., by optimizing over multiple model classes and hyperparameters (Thornton et al., 2013) or by constructing heterogeneous stacking ensembles (Erickson et al., 2020; van der Laan et al., 2007). In practice this is very similar to *learner importance*, while model building is usually more expensive.

Statistical inference can be performed at each of the three estimand levels. At the model level, hypothesis tests can be based on observation-wise losses on a single test set. At the learner level, variance-corrected tests can be applied to paired loss differences across resampling iterations. Since FI estimation inherently involves random components (e.g., from data resampling, permutation, or stochastic learners), repeated evaluation is generally needed to quantify this uncertainty, just as cross-validation is needed for reliable performance estimation. **xplainfi** provides dedicated inference methods at the model and learner levels, which therefore also include population-level inference.

When evaluating multiple FOIs simultaneously, any of these inference procedures introduces a multiple-testing problem. Depending on the use case, it is necessary to control either the family-wise error rate (FWER) via methods such as the Bonferroni-Holm correction, or to control the false-discovery rate (FDR) with methods such as the Benjamini-Hochberg or Benjamini–Yekutieli procedures, where the latter is valid under arbitrary dependence structures (Holm, 1979; Benjamini and Hochberg, 1995; Benjamini and Yekutieli, 2001). In an exploratory setting, it is often sufficient to control the FDR, which controls the expected proportion of false positives among all rejections (e.g., 1 out of 20 features deemed important), whereas controlling the FWER is more suitable for confirmatory settings in which avoiding false positives takes priority.

Regardless of the estimand, loss-based FI methods are usually constructed by "removing" the information about the FOI(s) and calculating the difference in expected loss (with vs. without this information). Following Ewald et al. (2024), we group the FI methods by the strategy used to remove information: feature perturbations, model refitting, or Shapley values.

### 2.2 Methods based on feature perturbations

The most well-known method in this category is *permutation feature importance* (PFI), originally introduced by Breiman (2001) for random forests. PFI targets the effect of breaking the link between $X_j$ and the rest of the joint distribution at evaluation time, holding $\hat{f}$ fixed. The model is fit once but evaluated twice: once with the original features and once with the FOI $X_j$ replaced by a perturbed version $\tilde{X}_j$. In PFI, $\tilde{X}_j$ is derived by simply shuffling $X_j$ in-place, which keeps the marginal distribution of $X_j$ intact. Importance is then measured as the difference in performance between the original model prediction and the model prediction using the uninformative replacement for the FOI. Formally,

$$\text{PFI}_j = \mathbb{E}\left[L\left(Y, \hat{f}(\tilde{X}_j, X_{-j})\right)\right] - \mathbb{E}\left[L\left(Y, \hat{f}(X)\right)\right]. \tag{1}$$

While PFI is computationally cheap, requiring only model predictions and simple shuffling of feature vectors, it breaks dependencies not only between the FOI and the target, but also between the FOI and all other features. This can yield implausible feature combinations and misleading attributions under feature dependence (see Hooker et al., 2021). *Conditional permutation feature importance* (CFI), introduced by Strobl et al. (2008), addresses this by perturbing the FOI conditional on a set of other features: $X_j$ is replaced by $\tilde{X}_j$ sampled from the conditional distribution of $X_j \mid X_{-j}$, thereby preserving (parts of) the dependence structure among the predictors. König et al. (2021) generalize PFI and CFI to *relative feature importance* (RFI) by specifying a conditioning set $G$ for which $\tilde{X}_j$ retains conditional dependencies, i.e., $\tilde{X}_j$ is sampled from the conditional distribution of $X_j \mid X_G$; this yields CFI for $G = -j$ and PFI for $G = \emptyset$.

Since the permutation (or sampling) step introduces randomness, multiple repetitions should be used to stabilize the estimate of the importance score. In **xplainfi**, this is controlled via the `n_repeats` parameter, with the final importance score averaged across repetitions. In terms of inference, **xplainfi** allows applying the corrected t-test approach proposed by Nadeau and Bengio (2003) to paired loss differences across resampling iterations, which can be used to quantify uncertainty of (learner-level) FI estimates based on repeated data splits. Molnar et al. (2023) evaluate and recommend this approach for PFI in combination with subsampling or bootstrapping with 10–15 iterations. Schulz-Kümpel et al. (2025) recommend for this test a train-test ratio of 0.9 and 25 subsampling iterations. **xplainfi** lets the

user apply the inference method to all FI methods, while warning the user if fewer than 10 bootstrap or subsampling iterations are used.

Beyond conditional permutation-based approaches, the conditional predictive impact (CPI) framework proposed by Watson and Wright (2021) provides a dedicated hypothesis test for conditional feature importance. **xplainfi** offers both the original version based on the knockoff framework (Candès et al., 2018) and an ARF-based version for mixed data (Blesch et al., 2025).

The conditional sampling required by CFI, RFI, and conditional SAGE (described below) can be performed using different approaches, each with distinct trade-offs. The Gaussian approach assumes multivariate normality, making it fast and straightforward but limited to continuous features. Conditional inference trees provide a nonparametric alternative that handles mixed feature types, as proposed by Redelmeier et al. (2020) for conditional distribution estimation (e.g., for conditional Shapley-style estimands) and implemented in **shapr**. Adversarial random forests offer flexible density estimation and conditional sampling for mixed data at a higher computational cost (Watson et al., 2023). Finally, knockoff sampling provides a specialized approach that enables valid inference through the CPI framework, but implementations for mixed data are less common (Candès et al., 2018; Blesch et al., 2024). Gaussian, conditional inference tree, and ARF samplers can be applied to all conditional importance methods in **xplainfi**, while knockoff sampling is only compatible with CFI.

### 2.3 Methods based on model refitting

A more intuitive approach to remove an FOI's information is to refit a model without the FOI (using the same learner) and then measure the performance difference between the reduced and the original (full) model. The best-known method following this approach is the *leave-one-covariate-out* (LOCO) (Lei et al., 2018), which analyzes a single FOI $X_j$. If one is interested in the importance of a set of features at once, this is known as *leave-one-group-out* (LOGO) (Au et al., 2022). Williamson et al. (2023) proposed a general framework for refitting-based methods, including a dedicated statistical inference method. In **xplainfi**, we use the terms LOCO and WVIM (*Williamson's variable importance measure*; Ewald et al. (2024) proposed the acronym).

This approach is conceptually simple: To evaluate an FOI's importance, the model in question is fit twice, once with and once without the FOI. Again, the resulting performance difference gives a straightforward indication of the predictive value "lost" by leaving out feature $X_j$:

$$\text{LOCO}_j = \mathbb{E}\left[L\left(Y, \hat{f}_{-j}(X_{-j})\right)\right] - \mathbb{E}\left[L\left(Y, \hat{f}(X)\right)\right], \tag{2}$$

where $\hat{f}_{-j}$ denotes the model fitted without feature $X_j$. Model refitting, of course, increases the computational cost in comparison to FI methods based on perturbations, where only one model needs to be fit as the basis for all subsequent calculations. This can be problematic when using LOCO (or a generalization thereof) on datasets with many features, especially when combined with expensive learners.

The inverse operation, *leave-one-covariate-in* (LOCI), trains models with only a single feature and compares performance against a featureless baseline to measure the increase in performance. While theoretically consistent, LOCI is rarely useful in practice as it essentially investigates univariate associations between the target and a single feature, for which more appropriate methods are available (see above). The group version of LOCI (also included in the WVIM framework) is more useful, where a subset of the features is "left in", chosen, for example, based on domain knowledge about the data.

For statistical inference on LOCO, Lei et al. (2018) suggest an observation-level nonparametric test using the $\ell_1$ loss differences on a test set, which is implemented in **xplainfi** in a generalized manner that also enables other losses and tests. The corrected t-test approach mentioned in the previous section is also available for LOCO, but it has not been explicitly investigated in this context.

### 2.4 Methods based on Shapley values

Shapley values originate in cooperative game theory (see Shapley, 1953) and, in ML, are often used for "local" explanations, i.e., for explaining individual model predictions for selected data instances. But several methods have also been proposed to apply the underlying principle to global FI, including SFIMP (*Shapley Feature IMPortance*) by Casalicchio et al. (2019) and SPVIM (*Shapley Population Variable Importance Measure*) by Williamson and Feng (2020), which is related to the more widely adopted SAGE (*Shapley Additive Global importancE*) by Covert et al. (2020). **xplainfi** implements SAGE, which distributes the overall model loss across the features based on their individual contributions, yielding importance scores that sum to the overall model loss. The SAGE value for feature $X_j$ is defined as the weighted average of its marginal contributions to feature subsets $S$, also called coalitions, based on a

**Table 1:** Overview of FI methods and capabilities in xplainfi and related R packages and selected Python packages. A checkmark indicates an available feature, and a checkmark in parentheses indicates a feature is not directly offered as such but available in either a limited or indirect fashion. Model importance refers to analyzing a pre-fit model; learner importance refers to refitting models via resampling. Statistical inference denotes the availability of confidence intervals or hypothesis tests for importance estimates.

| | xplainfi | vimp | vip | iml | hstats | flashlight | DALEX | fippy | sage |
|---|---|---|---|---|---|---|---|---|---|
| **Methods** | | | | | | | | | |
| PFI | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| CFI | ✓ | | | | | | | ✓ | |
| RFI | ✓ | | | | | | | ✓ | |
| LOCO | ✓ | ✓ | | | | | | | |
| WVIM | ✓ | ✓ | | | | | | | |
| mSAGE | ✓ | | | | | | | ✓ | ✓ |
| cSAGE | ✓ | | | | | | | ✓ | (✓) |
| SPVIM | | ✓ | | | | | | | |
| **Capabilities** | | | | | | | | | |
| Model importance | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Learner importance | ✓ | ✓ | | | | | | (✓) | |
| Statistical inference | ✓ | ✓ | | | | | | ✓ | |

value function $v$:

$$\text{SAGE}_j = \sum_{S \subseteq P \setminus \{j\}} \frac{|S|!(p-|S|-1)!}{p!} \left[ v(S \cup \{j\}) - v(S) \right]. \tag{3}$$

Here, $v(S)$ measures how much better the model performs when the information of the features in $S$ is used for prediction (while features outside $S$ are integrated out) compared to a featureless baseline. *Marginal SAGE* (mSAGE) uses the marginal distribution, resulting in

$$v_m(S) = \mathbb{E}\left[ L(Y, \hat{f}_\emptyset) \right] - \mathbb{E}\left[ L\left( Y, \mathbb{E}_{X_{-S}}\left[ \hat{f}(X_S, X_{-S}) \right] \right) \right], \tag{4}$$

and *conditional SAGE* (cSAGE) uses the conditional distribution, replacing $\mathbb{E}_{X_{-S}}$ with $\mathbb{E}_{X_{-S}|X_S}$ in Equation (4) to yield $v_c(S)$.

Since exact computation requires evaluating all possible feature coalitions ($2^p$), implementations must rely on approximations. **xplainfi** implements the permutation estimator described by Covert et al. (2020), where the feature coalitions are built from one of `n_permutations` shuffles of the feature vector. The empty coalition is always evaluated, resulting in a total number of evaluated coalitions of $1 + p \cdot$ `n_permutations`. The variance of SAGE value estimates along the evaluated permutations is also used to define an "early stopping" criterion to avoid excessive computations. Covert et al. (2020) use these variances to construct confidence-like intervals, but we do not include them as statistical inference methods because their coverage is unclear. Additionally, the marginalization step of the estimation requires a subset of size `n_samples` for each data instance, and larger values yield more stable SAGE value estimates but incur additional time and memory costs for large datasets.

## 3 Related work

Various R packages on CRAN implement one or more FI methods. We compare them along two dimensions: (1) the *scope of FI methods* they provide, and (2) whether they support *model importance* (analyzing a single pre-fit model) or also *learner importance* via resampling and refitting (see Section 2). Table 1 gives a comparative overview.

**vip** (Greenwell and Boehmke, 2020) requires a pre-fit model and provides model-specific importance extraction, PFI, a variance-based method ("FIRM"), and a Shapley-based measure based on the mean absolute Shapley/SHAP values per feature, i.e., an aggregate of per-observation feature attributions of the model prediction. Unlike loss-based FI methods such as PFI and SAGE, both FIRM and the Shapley-based importance in **vip** summarize importance on the model-output (prediction) scale rather than via changes in loss. It supports a wide range of model classes and offers repeated permutations and optional subsampling of the evaluation data, but does not refit models; accordingly, it targets model importance. Similarly, **iml** (no longer actively developed) operates on pre-fit models

via a `Predictor` wrapper and provides PFI alongside other IML methods such as feature effects, interaction statistics, and local explanations. **DALEX** likewise wraps pre-fit models via an explainer object; its PFI implementation is provided by the companion package **ingredients**, complemented by local explanations and feature-effect visualizations. **hstats** provides PFI alongside H-statistics for interaction detection, also operating on pre-fit models. **flashlight** wraps pre-fit models in an explainer abstraction and provides model-agnostic PFI, interaction-strength measures based on Friedman's *H*-statistics for interaction detection, as well as Shapley-based importance via mean absolute (approximate) SHAP feature attributions aggregated across observations. It supports side-by-side comparison of multiple fitted models via the `multiflashlight` function, which can also be used to compare models that were manually refit on different resampling splits. However, it does not orchestrate refitting/resampling or aggregate fold-wise importance into a single learner importance estimate. The **mlr3** ecosystem includes **mlr3filters**, which provides PFI with resampling support within its feature-selection framework, but is designed for feature ranking rather than importance analysis with uncertainty quantification. Generally, PFI is also available in many model-specific implementations (e.g., in **ranger** and **randomForestSRC** for random forests).

To the best of our knowledge, there is no general-purpose, model-agnostic R implementation of CFI or RFI that mirrors the kind of model-wrapping interfaces available for PFI. **cpi** implements Conditional Predictive Impact (CPI), a dedicated conditional-importance testing framework based on knockoff sampling (Watson and Wright, 2021). It is tied to the knockoff construction rather than providing a general conditional-sampling interface for CFI/RFI and focuses on hypothesis testing (p-values and confidence intervals) rather than a broad FI computation framework. **permimp** implements a version of CFI but is restricted to tree-based methods and is therefore not model-agnostic.

**vimp** takes a different approach: rather than evaluating a pre-fit model, it refits learners for each evaluated feature subset using the **SuperLearner** framework, which automatically ensembles prediction models to approximate the Bayes-optimal predictor (Williamson et al., 2023). This targets population-level importance with valid confidence intervals and hypothesis tests. [1] It implements what it calls "conditional VIM" (equivalent to LOCO for individual FOIs), "marginal VIM" (equivalent to LOCI), and SPVIM (*Shapley Population Variable Importance Measure*; Williamson and Feng (2020)). SPVIM is a Shapley-based method that, like SAGE, distributes the overall model performance across features according to their marginal contributions. Unlike SAGE, it replaces the marginalization or conditional sampling step with model refitting for each feature coalition, which can be slow, but uses Kernel SHAP (Covert and Lee, 2021) to approximate the Shapley values. **vimp** also accepts pre-computed prediction vectors, which, in principle, allow a form of model importance, but with a notably different API compared to the model-wrapping approach of the other packages. It is tightly coupled with the **SuperLearner** framework and offers less flexibility in resampling strategies.

In the Python ecosystem, `scikit-learn` provides tree-based PFI and model-agnostic PFI, both operating on pre-fit estimators. The `ELI5` project similarly provides PFI as its only model-agnostic FI method (eli, 2017). `sage` offers the original (marginal) SAGE implementation with no dedicated interface for conditional sampling, but it allows supplying external surrogate models for the same purpose (Covert et al., 2020; Covert and Lee, 2021). The `fippy` package in Python offers the closest overlap with **xplainfi** in terms of perturbation-based methods, providing PFI, CFI, RFI, and both marginal and conditional SAGE using various sampling methods. It also supports simple Wald-type CIs for FI scores from the observation-based losses on test data if the chosen performance measure is decomposable. Overall, `fippy` is mainly centered around explaining pre-fit estimators on fixed evaluation data, but it also offers a (somewhat underdeveloped) `LearnerExplainer` which only supports CFI in combination with subsampling.

In summary, **xplainfi** offers the most comprehensive coverage of model-agnostic FI methods in R, including PFI, CFI, RFI, LOCO, WVIM, and both marginal and conditional SAGE, with flexible systems covering learners, metrics, resampling, and model/learner importance, including tuning and complex ensembles for the latter. Among the compared R packages, only **xplainfi** and **vimp** provide statistical inference methods for their importance estimates, including confidence intervals and hypothesis tests. In the Python ecosystem, `fippy` also provides inference, but limited to the model-importance setting.

## 4 An introduction to xplainfi

**xplainfi**'s architecture is heavily inspired by the **mlr3** framework and its extension packages, which also form the foundation of the underlying computational infrastructure. This means primarily two things for the user: 1. The package API is based on **R6** classes with the corresponding object-oriented design, using **mlr3** functions and concepts. 2. **xplainfi**'s basic ML capabilities are determined by

[1] But note that the **SuperLearner** interface also allows using only a single learner, which can then be analyzed for learner importance with **vimp**.

the **mlr3** ecosystem. We begin with a brief introduction to **mlr3** and then build from simple to more advanced applications of **xplainfi**. For a complete overview, we refer to Bischl et al. (2024), and in particular to Foss and Kotthoff (2024) for a full introduction.

### 4.1 A brief introduction to mlr3

**mlr3** is an R package and an associated ecosystem of extension packages born from methodological and applied ML research. Its core components are abstractions for the building blocks of ML pipelines. Most importantly for our purposes, these include the objects shown in this short application example:

```
#>     task_id     learner_id resampling_id iteration classif.ce
#>      <char>         <char>        <char>     <int>      <num>
#> 1: penguins classif.ranger            cv         1 0.00000000
#> 2: penguins classif.ranger            cv         2 0.02898551
#> 3: penguins classif.ranger            cv         3 0.00000000
#> 4: penguins classif.ranger            cv         4 0.00000000
#> 5: penguins classif.ranger            cv         5 0.02941176
#> Hidden columns: task, learner, resampling, prediction_test

#> classif.ce
#> 0.01167945
```

Here, we applied the random forest classification learner as implemented in **ranger** with 100 trees to the `penguins` dataset (Horst et al., 2020) in a 5-fold cross-validation procedure, evaluated each iteration with the classification error (CE), and finally calculated the average CE across iterations.

A `Task` (`tsk()`) encapsulates the data and the learning problem (regression, classification, etc.). Built-in tasks for common datasets are available, e.g., `tsk("penguins")`, and custom tasks can be created from any `data.frame`-like object, e.g., `as_task_regr(mtcars, target = "mpg")`. A `Learner` (`lrn()`) represents an ML algorithm with its hyperparameters. For example, `lrn("regr.ranger", num.trees = 100)` creates a random forest learner via **ranger** using 100 trees. Learners can also be extended into full ML pipelines with preprocessing, feature extraction, or hyperparameter optimization (see Thomas, 2024, Becker et al. (2024), Binder et al. (2021)). A `Measure` (`msr()`) quantifies prediction performance, e.g., `msr("regr.mse")` for MSE for a regression task. Some measures are decomposable into observation-wise losses, which is relevant for certain inference methods described below. A `Resampling` (`rsmp()`) defines train-test splitting strategies, such as holdout, k-fold CV, bootstrapping, or subsampling, e.g., `rsmp("cv", folds = 5)`.

The general **xplainfi** API relies on these components and applies them to all implemented feature importance methods. For each feature importance method, the user first defines the method object by specifying a `Task` as the target for the analysis, a `Learner` for training and predictions, a `Resampling` strategy, and a `Measure` for evaluating predictions. The `$compute()` method is then called to perform the actual computational steps required for the individual importance method. Finally, importance scores can be accessed at different levels of aggregation: `$importance()` returns importance values per feature, aggregated across resampling iterations and repetitions (e.g., permutation repetitions in `PFI`); `$scores()` returns importance values per feature *and* per resampling iteration and repetition, allowing for custom aggregation or visualization; and `$obs_loss()` returns observation-wise loss scores and importance values, if available for the current importance method and `Measure`.

In the following, we showcase **xplainfi** on an included DGP and refer to the package website for additional tutorials and descriptions of this and other illustrative simulation settings.

### 4.2 Example 1: PFI with xplainfi

We start by calculating PFI on a synthetic task with four normally distributed features `x1` through `x4`, two of which are correlated (`x1` and `x2`, $r = 0.8$) and two of which are independent (`x3` and `x4`). The DGP is $y = 2x_1 + x_3 + \varepsilon$, ($\varepsilon \sim \mathcal{N}(0, 0.04)$), generated by the `sim_dgp_correlated()` simulation utility function included in the package, which produces a `Task` object.

#### Model importance

The simplest use case is analyzing a single pre-trained model. We train a **ranger** random forest on a holdout split and compute PFI on the corresponding test set:

```
library(xplainfi)
task <- sim_dgp_correlated(n = 5000, r = 0.8)
lrn_ranger <- lrn("regr.ranger")
resampling_ho <- rsmp("holdout")$instantiate(task)
lrn_ranger$train(task, row_ids = resampling_ho$train_set(1))

pfi_model <- PFI$new(
  task = task, learner = lrn_ranger, measure = msr("regr.mse"),
  resampling = resampling_ho, n_repeats = 10)
pfi_model$compute()
pfi_model$importance()

#> Key: <feature>
#>    feature   importance
#>     <char>        <num>
#> 1:      x1 6.7588443770
#> 2:      x2 0.1570592707
#> 3:      x3 1.8315843234
#> 4:      x4 0.0001283878
```

When a pre-trained learner is passed, **xplainfi** detects this automatically and skips the training step, using the fitted model directly for prediction. The resampling must be instantiated with exactly one test set (i.e., holdout), as there is only one model to evaluate. This yields *model importance*: the importance scores reflect the behavior of this specific model on this specific test set.

### Learner importance

To capture importance at the *learner* level, we pass an untrained learner together with a resampling strategy. **xplainfi** then trains a new model in each resampling iteration, and importance scores are aggregated across iterations:

```
pfi <- PFI$new(
  task = task, learner = lrn("regr.ranger"), measure = msr("regr.mse"),
  resampling = rsmp("cv", folds = 3), n_repeats = 10)
pfi$compute()
pfi$importance()

#>    feature  importance
#> 1:      x1  6.50505612
#> 2:      x2  0.16087409
#> 3:      x3  1.80416657
#> 4:      x4 -0.00000201
```

The construction of the `PFI` object and the computation are separated: calling `$new()` defines the setup, while `$compute()` performs the actual work. To access individual importance scores for each permutation and resampling iteration, the `$scores()` method is available, which stores the corresponding feature, iteration indices, and the associated measure value of the original (baseline) model and the model post perturbation:

```
#>    feature iter_rsmp iter_repeat regr.mse_baseline regr.mse_post importance
#>     <char>     <int>       <int>             <num>         <num>      <num>
#> 1:      x1         1           1        0.06682468      6.394225   6.327400
#> 2:      x1         1           2        0.06682468      6.496952   6.430127
#> 3:      x1         1           3        0.06682468      6.801135   6.734310
```

Using resampling not only captures learner-level variation but also enables variance-corrected inference methods such as the Nadeau-Bengio correction (see Example 3), which require loss differences across resampling iterations and are therefore not available for model importance.

All other components are easily swappable. Here we rerun PFI with XGBoost using the mean absolute error and subsampling instead:

```
pfi <- PFI$new(
```

```
  task = task,
  learner = lrn("regr.xgboost", eta = 0.01, nrounds = 1000),
  measure = msr("regr.mae"),
  resampling = rsmp("subsampling", repeats = 10),
  n_repeats = 15)
pfi$compute()
pfi$importance()

#>    feature importance
#> 1:      x1   2.101450
#> 2:      x2   0.003902
#> 3:      x3   0.931480
#> 4:      x4   0.000172
```

A learner tuned by cross-validation integrates the same way. Because `auto_tuner()` wraps a learner into an ordinary **mlr3** learner, it can be passed directly to any importance method: its hyperparameters are tuned by an inner cross-validation within each resampling iteration, while importance is evaluated on the outer split. Here, we tune the random forest's `min.node.size` over a small grid with 10 iterations before computing PFI:

```
library(mlr3tuning)
lrn_tuned <- auto_tuner(
  learner = lrn("regr.ranger", min.node.size = to_tune(1, 50)),
  resampling = rsmp("cv", folds = 3), measure = msr("regr.mse"),
  tuner = tnr("grid_search", resolution = 10), term_evals = 10)

pfi <- PFI$new(
  task = task, learner = lrn_tuned, measure = msr("regr.mse"),
  resampling = rsmp("cv", folds = 3), n_repeats = 10)
pfi$compute()
pfi$importance()

#> Key: <feature>
#>    feature   importance
#>     <char>        <num>
#> 1:      x1  6.582438372
#> 2:      x2  0.151547198
#> 3:      x3  1.812646630
#> 4:      x4 -0.000139866
```

**xplainfi**'s capabilities are extendable by the **mlr3** ecosystem: any learner from **mlr3learners** or `mlr3extralearners` (Fischer et al., 2025) can be used, including integrated hyperparameter tuning via `to_tune()` tokens or `auto_tuner()` learners, preprocessing and other pipelines via **mlr3pipelines**, or full AutoML systems (Schneider and Becker, 2024; Binder et al., 2024). Furthermore, any metric from **mlr3measures** is available. The trained models and baseline results are retained via `$resample_result` for further analysis.

### 4.3 Example 2: CFI and conditional samplers

As discussed in Section 2, PFI can produce misleading results when features are correlated. CFI addresses this by sampling from the conditional distribution, which requires a *conditional sampler*. **xplainfi** provides a modular abstraction for this. Several conditional samplers are available, but here we focus on two: The `ConditionalGaussianSampler` assumes multivariate normality and is fast but limited to numeric features. The `ConditionalARFSampler` uses adversarial random forests (ARF) and handles mixed data, but is computationally more expensive (Watson et al., 2023).

Samplers are instantiated on a given task once, after which one can draw one or more observations, which `CFI` and related methods handle internally. Each conditional sampler allows the specification of an arbitrary conditioning set of features for sampling.

We continue with the correlated DGP from Example 1, where `x1` and `x2` are correlated ($r = 0.8$), but only `x1` affects the target. Using a Gaussian conditional sampler, we compute CFI and also request quantiles calculated across resampling iterations:

```
cfi <- CFI$new(
  task = task, learner = lrn("regr.ranger"), measure = msr("regr.mse"),
  resampling = rsmp("subsampling", repeats = 5),
  sampler = ConditionalGaussianSampler$new(task),
  n_repeats = 10)
cfi$compute()
cfi$importance(ci_method = "quantile", alternative = "two.sided")

#>    feature importance conf_lower conf_upper
#> 1:      x1   2.795434   2.776238   2.831308
#> 2:      x2   0.000642  -0.001292   0.002487
#> 3:      x3   1.815413   1.799239   1.831552
#> 4:      x4  -0.000175  -0.000566   0.000553
```

Compared to PFI, CFI identifies that `x2` is not associated with `y` conditional on the other features. The `sampler` argument distinguishes CFI from PFI: while PFI uses marginal permutation, CFI requires a conditional sampler. The `"quantile"` method shown above provides empirical quantiles from the distribution of importance scores across resampling iterations, which help to gauge the stability of the estimates. Note that via `"alternative"`, we explicitly requested the default of `"two-sided"` intervals, by analogy with two- or one-sided hypothesis tests, where `"alternative = 'greater'"` would have given us only the 5% quantile as lower bound.

### 4.4 Example 3: Inference

For principled inference beyond empirical quantiles, **xplainfi** supports two main approaches, depending on the chosen importance method.

#### Nadeau-Bengio correction for PFI

Molnar et al. (2023) recommend variance correction based on Nadeau and Bengio (2003) when using PFI with subsampling or bootstrapping. The correction accounts for dependence between resampling iterations that share training observations, yielding confidence intervals with improved (though still imperfect) coverage. The recommended setup uses approximately 15 subsampling iterations. Since our earlier PFI example already uses subsampling, we can request corrected confidence intervals via `ci_method = "nadeau_bengio"`:

```
head(pfi$importance(ci_method = "nadeau_bengio"), 3)

#>    feature importance      se statistic  p.value conf_lower conf_upper
#> 1:      x1      6.582 0.11098      59.3 0.000284      6.105      7.060
#> 2:      x2      0.152 0.00521      29.1 0.001179      0.129      0.174
#> 3:      x3      1.813 0.05624      32.2 0.000961      1.571      2.055
```

The output includes standard errors and adjusted confidence intervals. Note that this approach assumes normally distributed importance scores and was primarily evaluated for PFI; its use with other methods is experimental. Figure 1 compares the corrected confidence intervals with the empirical 95% quantiles and also unadjusted confidence intervals. We note that the latter are also available via `ci_method = "raw"` for comparison, but they are not valid for inference.

#### CPI for conditional importance

For CFI, inference is available through the CPI framework when using knockoff-based sampling, which leverages observation-wise losses to perform hypothesis tests for feature importance (Watson and Wright, 2021). Using a knockoff sampler with CFI enables `ci_method = "cpi"`, for example in conjunction with a t-test:

```
cfi_knockoff <- CFI$new(
  task = task, learner = lrn("regr.ranger"), measure = msr("regr.mse"),
  sampler = KnockoffGaussianSampler$new(task),
  resampling = rsmp("holdout"), n_repeats = 1)
cfi_knockoff$compute()
```

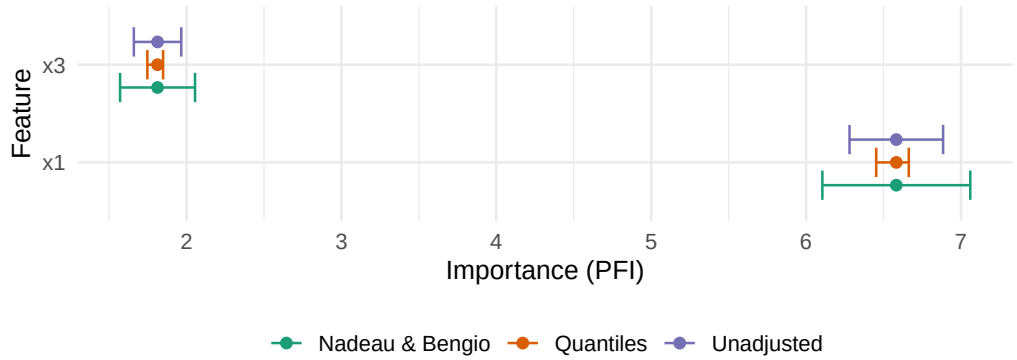



**Figure 1:** Comparison of uncertainty quantification methods using PFI on the correlated task (omitting noise features). Empirical 95% quantiles fall between the very narrow unadjusted confidence intervals and the wider Nadeau-Bengio-corrected intervals, which show the uncertainty masked by the unadjusted method.

```
cfi_knockoff$importance(
  ci_method = "cpi", alternative = "greater",
  p_adjust = "BH", test = "t")
```

```
#>    feature importance       se statistic   p.value conf_lower conf_upper
#> 1:      x1   3.013652 0.102096    29.518 4.35e-154    2.84563        Inf
#> 2:      x2   0.000600 0.002244     0.268  5.26e-01   -0.00309        Inf
#> 3:      x3   1.760204 0.060890    28.908 2.87e-149    1.65999        Inf
#> 4:      x4  -0.000501 0.000553    -0.907  8.18e-01   -0.00141        Inf
```

This CPI test provides p-values and one-sided confidence bounds for each feature. Here, we use the `p_adjust` argument to use R's `p.adjust()` function to correct for the FDR with the Benjamini-Hochberg procedure. We note that this only corrects the p-values, but not the confidence bounds, which would only be adjusted for `p_adjust = "bonferroni"`. Alternatively to the knockoff approach, `ConditionalARFSampler` can be used in place of knockoffs, enabling CPI-style inference for mixed data types as proposed by Blesch et al. (2025), though at a higher computational cost. Note that CPI was proposed in combination with a fixed model and a test set, whereas **xplainfi** also allows its use with cross-validation. Because the CPI test operates on observation-wise losses, the non-overlapping test sets from the resampling folds are pooled into a single set of observation-wise losses on which the test is computed (i.e., micro-averaging rather than averaging per-fold results).

### 4.5 Example 4: LOCO and WVIM

LOCO is implemented as a special case of the more general WVIM framework as described in Section 2. We apply it to the simulated task from Example 1 using a single refit iteration and show the same Bonferroni-adjusted p-values and confidence intervals based on the Wilcoxon test proposed by Lei et al. (2018):

```
loco <- LOCO$new(
  task = task, learner = lrn("regr.ranger"),
  measure = msr("regr.mse"), resampling = rsmp("holdout"))
loco$compute()
loco$importance(
  ci_method = "lei", test = "wilcox",
  p_adjust = "bonferroni")[, c(1:2, 5:7)]
```

```
#>    feature importance       p.value conf_lower conf_upper
#> 1:      x1    0.74831 0.00000000000    1.03087    1.26976
#> 2:      x2    0.00190 0.00000000339    0.00298    0.00767
#> 3:      x3    0.47690 0.00000000000    0.68225    0.83869
#> 4:      x4    0.00482 0.00000000000    0.00770    0.01198
```

`WVIM` supports arbitrary feature groups via the `groups` argument, which accepts a named list specifying features belonging to each group (i.e., LOGO):

```
groups <- list(correlated = c("x1", "x2"), independent = c("x3", "x4"))
wvim <- WVIM$new(
  task = task, learner = lrn("regr.ranger"),
  groups = groups, direction = "leave-out",
  measure = msr("regr.mse"), resampling = rsmp("holdout"))
wvim$compute()
wvim$importance()

#>        feature importance
#> 1:  correlated       4.40
#> 2: independent       1.11
```

Here, the `"correlated"` group contains features `x1` and `x2`, which are removed together. WVIM also supports a `"leave-in"` direction, which trains models with *only* the specified features and compares against a featureless baseline (i.e., LOCI or "LOGI"). The `groups` argument is also available for PFI, CFI, and RFI, where the specified groups of features are then always perturbed at once.

### 4.6 Conditional samplers

As described in Section 2, conditional methods like CFI and cSAGE require a mechanism to sample from the conditional distribution $P(X_j|X_{-j})$. **xplainfi** provides a modular sampler architecture that allows different sampling strategies to be instantiated and passed to the importance method. All samplers inherit from the `FeatureSampler` base class and share a common interface: Instantiate a sampler on a `Task`, then use the `$sample()` method to draw new values for specified features conditional on the remaining ones or a subset of features specified as `conditioning_set`.

We demonstrate the ARF sampler on the `penguins` task, which contains both numeric and categorical features. After instantiation, we sample new values for the `body_mass` feature for a few observations, conditional on all other features (including the categorical `island`):

```
task_penguins <- tsk("penguins")
sampler_arf <- ConditionalARFSampler$new(task_penguins)
# Original values for comparison
task_penguins$data(
  rows = c(1, 20, 40),
  cols = c("island", "bill_length", "body_mass"))

#>       island bill_length body_mass
#> 1: Torgersen        39.1      3750
#> 2: Torgersen        46.0      4200
#> 3:     Dream        39.8      4650

# Sample new body_mass values conditional on all other features
sampler_arf$sample(feature = "body_mass", row_ids = c(1, 20, 40))[, c(6, 3, 4)]

#>       island bill_length body_mass
#> 1: Torgersen        39.1      3774
#> 2: Torgersen        46.0      4467
#> 3:     Dream        39.8      3175
```

The sampler produces new values for `body_mass` drawn from the estimated conditional distribution $P(\texttt{body_mass}|X_{-\texttt{body_mass}})$. These sampled values replace the original feature when computing CFI or cSAGE, allowing importance to be assessed while preserving dependencies with other features.

The choice of sampler involves trade-offs between flexibility and computational cost. The ARF sampler handles mixed data types without making distributional assumptions but requires fitting an adversarial random forest, which increases computational time and memory usage.

## 5 Benchmark experiments

To evaluate **xplainfi**, we performed benchmark experiments in two categories: **Importance results**: Since **xplainfi** re-implements methods available in other packages, we verify that it produces equivalent importance scores across shared methods and DGPs, establishing that the implementations are faithful

and comparable across packages. **Runtime**: We assess whether **xplainfi**'s implementations are competitive in terms of computational cost compared to existing single-method packages, across varying task dimensionalities and method parameters.

We compare against packages that implement the same methods in a model-importance setting, as this is the common denominator across implementations (see Section 3). For PFI, we compare with **iml** and **vip**. For marginal SAGE, we compare with `fippy` and `sage`, the latter implementing kernel SAGE (Covert and Lee, 2021). For CFI and conditional SAGE, we compare with `fippy`, using a Gaussian conditional sampler as the lowest common denominator of available sampling options. **vimp** is not included as it only shares the refitting-based methods (LOCO, WVIM) with **xplainfi**, and differences in supported metrics, learner frameworks, and resampling strategies make a fair apples-to-apples comparison difficult. The MSE was used as the evaluation metric across all methods, except for **vip**, which only supports the root mean squared error. Since RMSE is a monotone transformation of MSE, feature rankings are preserved but scaled importance magnitudes differ. Both benchmarks are defined in R using **batchtools** (Lang et al., 2017) for cluster-based parallelization, and **reticulate** is used to run Python implementations.

Each method is evaluated using one of four learners to account for variability in learner capabilities: a linear model (**stats** / `scikit-learn` (Pedregosa et al., 2011)), a random forest (**ranger** / `scikit-learn`, 500 trees), `XGBoost` (Chen and Guestrin, 2016) (1000 rounds, early stopping after 50, $\eta = 0.1$), and an MLP (**mlr3torch** / `scikit-learn`, 1 hidden layer, 20 neurons, 500 epochs with early stopping). Of these, `XGBoost` is the only learner with an identical underlying implementation across R and Python. The linear model is expected to yield approximately identical results in either language. Learner configurations were chosen for simplicity and robustness rather than predictive performance, i.e., learners were not tuned beyond built-in regularization mechanisms.

### 5.1 Importance benchmark

We compare importance scores by scaling values for a given method and DGP to the unit interval to examine relative magnitudes. A rank-based comparison is available in the online supplement.

We compare five FI methods across one to four different implementations. Table 2 lists these methods and packages, alongside the parameters used for each method where applicable. To obtain reliable FI estimates, we used 50 repetitions for PFI and CFI, and similarly a high number of permutations for mSAGE and cSAGE, while enabling the "early stopping" options in both **xplainfi**'s and `fippy`'s implementations to avoid high computational cost at diminishing returns. Since we prioritize comparability, we used Gaussian conditional sampling for the conditional methods CFI and cSAGE throughout the experiment, as this is the only sampling mechanism implemented by both **xplainfi** *and* `fippy`. Both packages offer additional sampling mechanisms with greater flexibility, but would have introduced more variability into the comparison. The tasks selected for this experiment consist of simulation settings with DGPs designed to showcase differences and similarities among feature importance methods; a full listing is provided in the online supplement. Here, we focus on the "correlated" DGP generated by `xplainfi::sim_dgp_correlated()` as introduced in the previous section ($Y = 2X_1 + X_3 + \varepsilon$, with $\text{cor}(X_1, X_2) \in \{0.25, 0.75\}$, $p = 4$), and the "bike sharing" dataset introduced by Fanaee-T and Gama (2014) ($p = 12$), which is frequently used as an example for FI methods (e.g., Covert et al., 2020; Blesch et al., 2025). The bike sharing dataset's categorical features were converted to numeric values, as this was required to ensure comparability between **xplainfi** and `fippy` via the Gaussian conditional sampler. For simulated data, $n = 5000$ samples were generated and 2/3 of the observations were used for learner training and 1/3 for feature importance calculation as a test set. The bike sharing dataset was resampled in every replication with a different train-test split. The train and test sets were created consistently across the different implementations, i.e., the linear model for **xplainfi** was trained on the same data as the one for **vip** or `fippy` within a replication. Importance scores shown are aggregated from 25 replications. For additional details on the simulated settings, see the online supplement or **xplainfi**'s online documentation.

Due to the large number of factors in the experiment, we present a subset of results here and refer to the online supplement for a complete overview. We focus on the `linear` and `boosting` learners since they are the most comparable across R and Python.

Figure 2 shows almost identical scaled importance scores across all implementing packages for PFI and CFI. The exception is **vip**, which produces the same feature ranking but with differing importance scores, most likely due to the different evaluation metric. The feature ranking is equivalent between the two learner types. For mSAGE and cSAGE, similarly close agreement is visible between **xplainfi** and `fippy`, while `sage`'s kernel SAGE shows scores close to the other implementations with a slightly larger variance.

Next, we consider the bike sharing task. Figure 3 shows PFI and CFI using the linear model and boosting learner. For PFI, we see strong agreement between the methods similar to the previous

**Table 2:** Methods, implementations, and parameters for both benchmarks. Parameter names correspond to xplainfi's API and are set equivalently in other implementations. For the importance benchmark, early stopping was enabled for mSAGE and cSAGE in xplainfi and fippy; for the runtime benchmark, it was disabled to ensure identical workloads.

| Method | Packages | Parameters |
|---|---|---|
| PFI | xplainfi, fippy, iml, vip | Importance: n_repeats = 50<br>Runtime: n_repeats = {1, 50} |
| CFI | xplainfi, fippy | Importance: n_repeats = 50, sampler: Gaussian<br>Runtime: n_repeats = {1, 50}, sampler: Gaussian |
| mSAGE | xplainfi, fippy | Importance: n_permutations = 100, n_samples = 100, min_permutations = 20<br>Runtime: n_permutations = {10, 20, 50}, n_samples = {10, 50, 100} |
| mSAGE | sage | Importance: n_samples = 100<br>Runtime: n_samples = {10, 50, 100} |
| cSAGE | xplainfi, fippy | Importance: n_permutations = 100, n_samples = 100, min_permutations = 20, sampler: Gaussian<br>Runtime: n_permutations = {10, 20, 50}, n_samples = {10, 50, 100}, sampler: Gaussian |

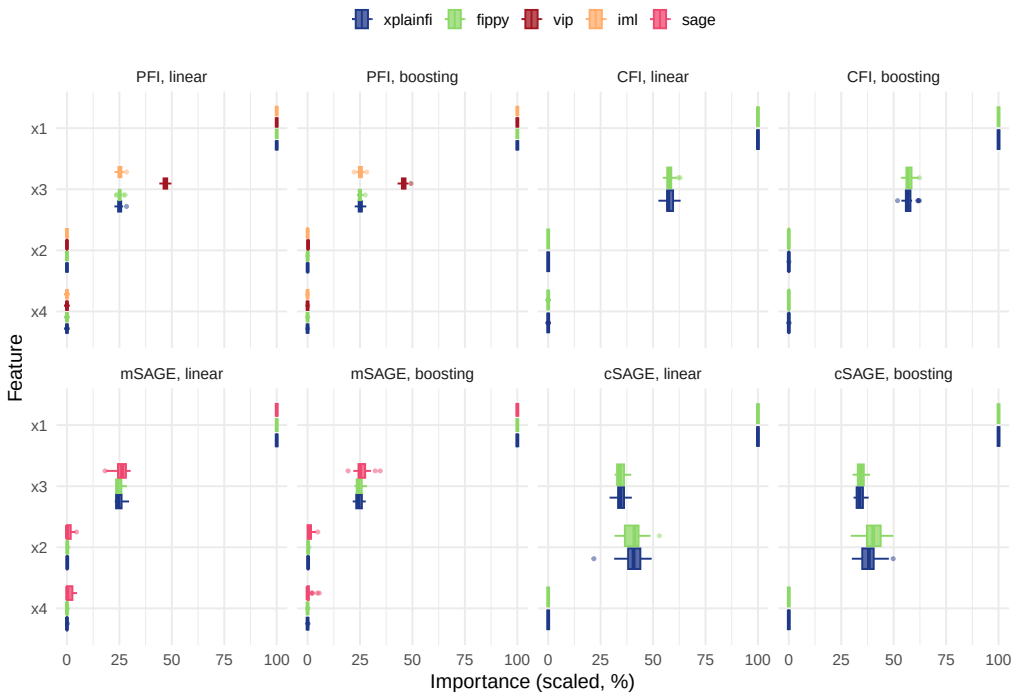



**Figure 2:** Importance scores (scaled to percentages) for PFI, CFI, mSAGE, and cSAGE across implementations on the correlated simulation setting with r = 0.75, based on either the linear model or the boosting learner.

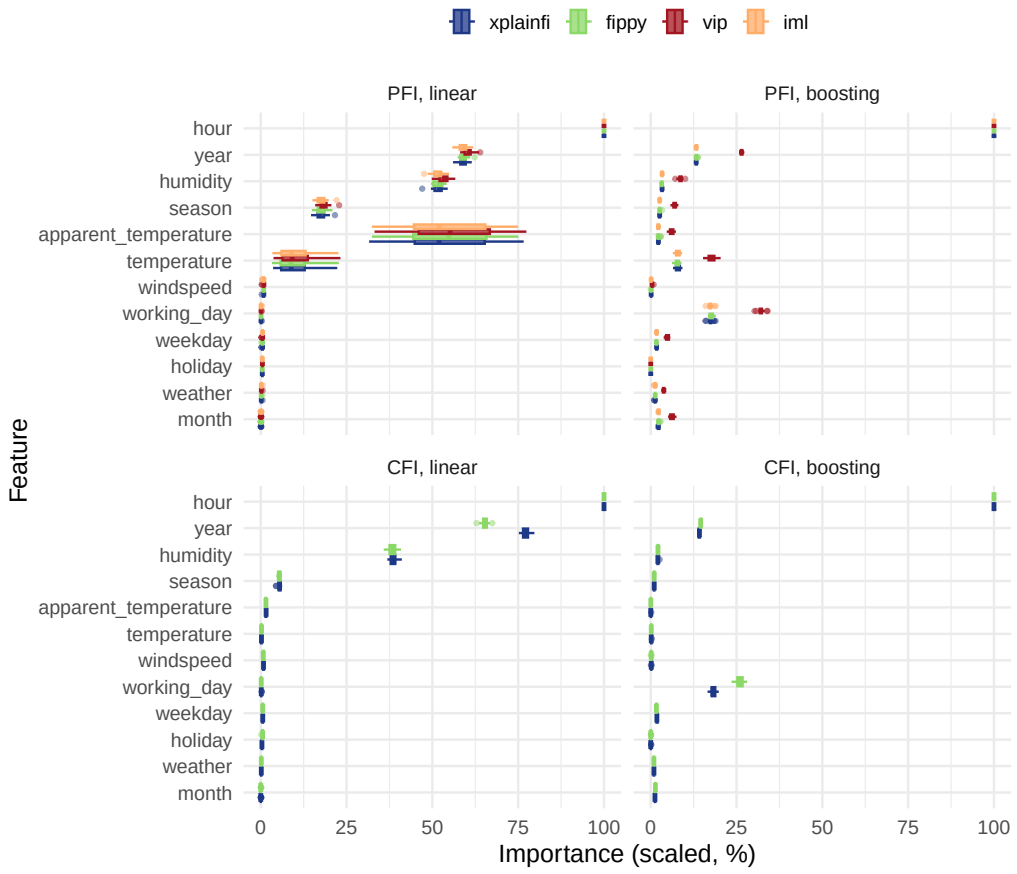



**Figure 3:** Importance scores (scaled to percentages) for PFI and CFI across implementations on the bike sharing dataset, based on either the linear model or the boosting learner.

setting. For CFI, **xplainfi** and `fippy` produce different (scaled) importance scores for the `year` and `working_day` features. This is most likely explained by the Gaussian conditional sampler being implemented slightly differently in both packages, combined with its use here on effectively categorical features encoded as integers, which is not the most appropriate choice. A similar pattern is visible in Figure 4 for cSAGE with the same underlying cause but good agreement otherwise. For mSAGE, `sage`'s importance scores produce a different pattern which does not appear to agree with the other methods. The linear model's importance scores additionally show comparatively large variance (e.g., for `apparent_temperature`), which reflects multicollinearity among the bike sharing features: `temperature` and `apparent_temperature` are strongly correlated. The coefficients of this unregularized, main-effects-only learner are therefore unstable across train-test splits, and the loss-based importance inherits this instability. Scores for the remaining simulation settings and learners are available in the online supplement. In all settings, **xplainfi**'s results agree with the reference implementations apart from minor deviations that did not affect the overall ranking, as also shown by the rank-based and paired-difference analyses in the supplement.

### 5.2 Runtime benchmark

Methods were run exclusively using linear models to isolate the computational cost of the FI method itself from that of the learner (see Table 2 for parameter ranges). We used only the `peak` task from **mlbench**, which provides a regression problem with user-definable sample and feature sizes. Experiments were run for 25 replications on the Intel Xeon Platinum 8380 compute nodes of the Leibniz Supercomputing Centre.

Figure 5 shows median runtimes with 25% and 75% quantiles for $n = 5000$ with varying $p$. For PFI, **iml** is clearly the slowest implementation after `fippy`. **vip** shows only slightly slower times than **xplainfi**. For mSAGE, `sage` notably becomes faster with more features, likely because the kernel SAGE implementation converges faster in that case. Across all methods, `fippy` is notably slower than **xplainfi**. Overall, **xplainfi** was consistently faster than or comparable to the reference implementations under close-to-equal settings. Additional results for all methods and parameter configurations are available in the online supplement.

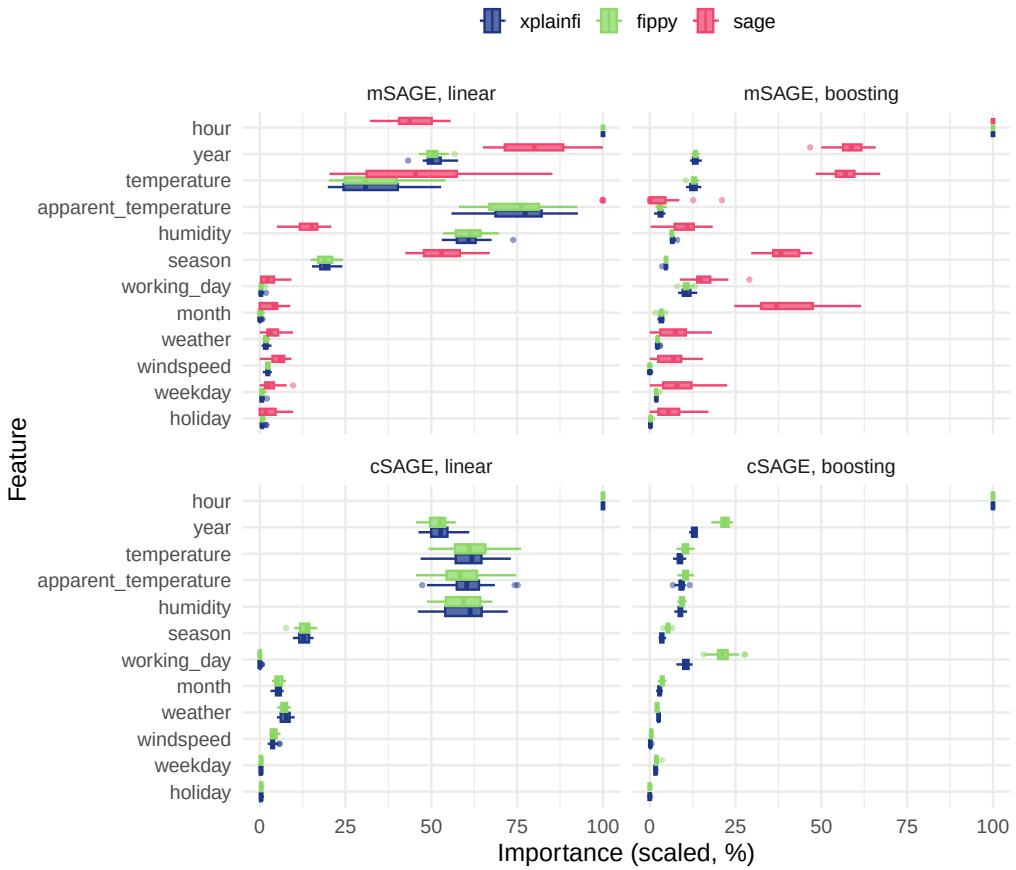



**Figure 4:** Importance scores (scaled to percentages) for mSAGE and cSAGE across implementations on the bike sharing dataset, based on either the linear model or the boosting learner.

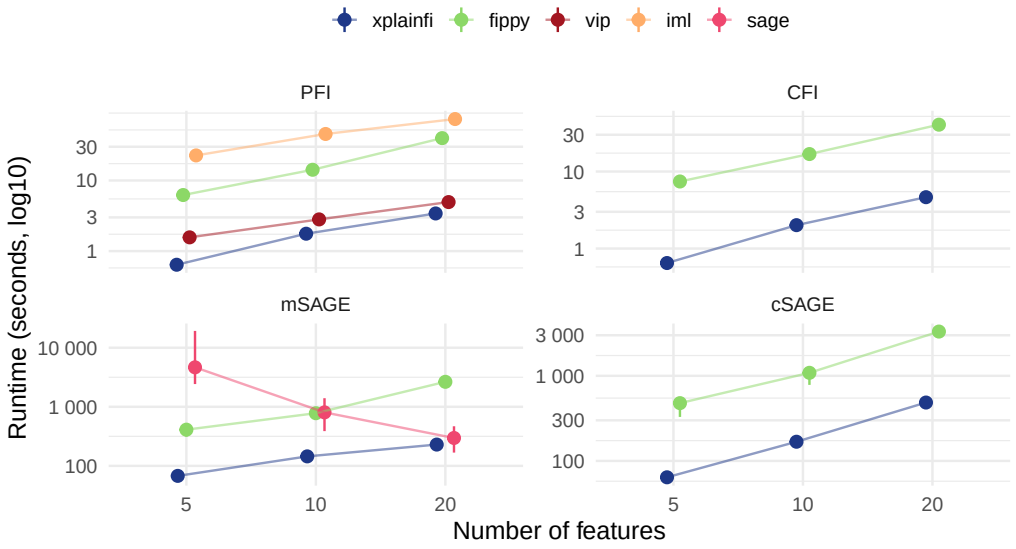



**Figure 5:** Median runtime in seconds with 25% and 75% quantiles for PFI and CFI with n_repeats = 50 and mSAGE and cSAGE with n_samples = 100 and n_permutations = 100 across implementations on the peak simulation setting with 5, 10, and 20 features and 5000 samples using a linear model across 25 replications.

## 6 Discussion and conclusion

We introduced **xplainfi**, a package that provides a comprehensive suite of global, loss-based FI methods within a unified framework built on the **mlr3** ecosystem, by implementing methods previously unavailable or scattered across packages, including conditional feature importance (CFI), relative feature importance (RFI), and both marginal and conditional SAGE. This allows **xplainfi** to focus primarily on FI methods without reimplementing common ML building blocks such as model interfaces, resampling, and performance measures. However, this also means **xplainfi** is not compatible with related frameworks such as **tidymodels** or arbitrary "unwrapped" models and learner implementations. We accept this compromise, as we could not otherwise maintain or expand the current feature set. Our benchmark experiments demonstrate that **xplainfi** produces importance scores consistent with reference implementations across multiple DGPs and learner types, while offering competitive runtime performance.

Choosing the "best" method depends on the research question, the type of association of interest (marginal vs. conditional), and the available computational budget, which can affect the choice between perturbation-based, refit-based, and Shapley-based methods. For detailed guidance on underlying estimands and interpretation, we refer to Ewald et al. (2024).

The modular feature sampler architecture is unique, enabling **xplainfi** to perform conditional FI analysis on continuous and mixed data alike via flexible samplers. This includes the ability to specify arbitrary conditioning sets for each sampler in the `Conditional` family, which enables conditional SAGE and RFI. Interest in conditional methods has been growing, and future work may benefit from the building blocks available with **xplainfi**. However, these methods come with trade-offs: the ARF sampler, while flexible, incurs higher computational cost and memory usage than Gaussian sampling, and SAGE methods can be slow due to the many feature coalitions that must be evaluated. Subsequent releases have reduced the computational cost of these methods, and a kernel-based SAGE estimator is in development.

The package could and should be extended to include additional confidence interval methods, or more generally, uncertainty quantification methods for FI. Although this is very desirable from a practical perspective, not enough (established) techniques and coverage studies exist, and we think exposing users to unvalidated techniques is not appropriate; such dedicated studies are out of scope for the current paper.

The package currently offers no visualization options, as we focused on the computational aspects. Visualizations for reports and publications often have specific requirements and are customized accordingly. Because we provide well-structured container types, the FI values (and other results) can be readily plotted using, e.g., **ggplot2**.

## 7 Acknowledgements

The authors gratefully acknowledge the computational and data resources provided by the Leibniz Supercomputing Centre (www.lrz.de).

Lukas Burk is supported by the Federal Ministry of Research, Technology and Space (BMFTR), grant number 01EQ2409E.

Marvin N. Wright is supported by the German Research Foundation (DFG), grant numbers 437611051 and 459360854, and the Federal Ministry of Research, Technology and Space (BMFTR), grant number 01EQ2409E.

Bernd Bischl is supported by the German Research Foundation (DFG), grant number 460135501.

## References

Eli5: Debug machine learning classifiers and explain their predictions, 2017. URL https://github.com/eli5-org/eli5. [p365]

Q. Au, J. Herbinger, C. Stachl, B. Bischl, and G. Casalicchio. Grouped feature importance and combined features effect plot. *Data Mining and Knowledge Discovery*, 36(4):1401–1450, 2022. doi: 10.1007/s10618-022-00840-5. [p363]

M. Becker, L. Schneider, and S. Fischer. Hyperparameter optimization. In B. Bischl, R. Sonabend, L. Kotthoff, and M. Lang, editors, *Applied Machine Learning Using mlr3 in R*. CRC Press, 2024. URL https://mlr3book.mlr-org.com/hyperparameter_optimization.html. [p366]

Y. Benjamini and Y. Hochberg. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society: Series B (Methodological)*, 57(1): 289–300, 1995. ISSN 0035-9246. doi: 10.1111/j.2517-6161.1995.tb02031.x. [p362]

Y. Benjamini and D. Yekutieli. The control of the false discovery rate in multiple testing under dependency. *The Annals of Statistics*, 29(4):1165–1188, 2001. ISSN 0090-5364, 2168-8966. doi: 10.1214/aos/1013699998. [p362]

M. Binder, F. Pfisterer, M. Lang, L. Schneider, L. Kotthoff, and B. Bischl. Mlr3pipelines - Flexible Machine Learning Pipelines in R. *Journal of Machine Learning Research*, 22(184):1–7, 2021. ISSN 1533-7928. URL http://jmlr.org/papers/v22/21-0281.html. [p360, 366]

M. Binder, F. Pfisterer, M. Becker, and M. N. Wright. Non-sequential pipelines and tuning. In B. Bischl, R. Sonabend, L. Kotthoff, and M. Lang, editors, *Applied Machine Learning Using mlr3 in R*. CRC Press, 2024. URL https://mlr3book.mlr-org.com/non-sequential_pipelines_and_tuning.html. [p368]

B. Bischl, R. Sonabend, L. Kotthoff, and M. Lang, editors. *Applied Machine Learning Using mlr3 in R*. CRC Press, 2024. ISBN 978-1-032-50754-5. URL https://mlr3book.mlr-org.com. [p366]

K. Blesch, D. S. Watson, and M. N. Wright. Conditional feature importance for mixed data. *AStA Advances in Statistical Analysis*, 108(2):259–278, 2024. ISSN 1863-818X. doi: 10.1007/s10182-023-00477-9. [p360, 363]

K. Blesch, N. Koenen, J. Kapar, P. Golchian, L. Burk, M. Loecher, and M. N. Wright. Conditional Feature Importance with Generative Modeling Using Adversarial Random Forests. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(15):15596–15604, 2025. ISSN 2374-3468. doi: 10.1609/aaai.v39i15.33712. [p363, 370, 372]

A. Bommert, X. Sun, B. Bischl, J. Rahnenführer, and M. Lang. Benchmark for filter methods for feature selection in high-dimensional classification data. *Computational Statistics & Data Analysis*, 143:106839, 2020. ISSN 0167-9473. doi: 10.1016/j.csda.2019.106839. [p361]

L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324. [p361, 362]

E. Candès, Y. Fan, L. Janson, and J. Lv. Panning for Gold: 'Model-X' Knockoffs for High Dimensional Controlled Variable Selection. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 80 (3):551–577, 2018. ISSN 1369-7412. doi: 10.1111/rssb.12265. [p361, 363]

G. Casalicchio, C. Molnar, and B. Bischl. Visualizing the Feature Importance for Black Box Models. In M. Berlingerio, F. Bonchi, T. Gärtner, N. Hurley, and G. Ifrim, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 655–670, Cham, 2019. Springer International Publishing. ISBN 978-3-030-10925-7. doi: 10.1007/978-3-030-10925-7_40. [p363]

T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. [p372]

T. Chiaburu, F. Haußer, and F. Bießmann. Uncertainty in xai: Human perception and modeling approaches. *Machine Learning and Knowledge Extraction*, 6(2):1170–1192, 2024. doi: 10.3390/make6020055. [p361]

I. Covert and S.-I. Lee. Improving KernelSHAP: Practical Shapley Value Estimation Using Linear Regression. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pages 3457–3465. PMLR, 2021. URL https://proceedings.mlr.press/v130/covert21a.html. [p365, 372]

I. Covert, S. M. Lundberg, and S.-I. Lee. Understanding Global Feature Contributions With Additive Importance Measures. In *Advances in Neural Information Processing Systems*, volume 33, pages 17212–17223. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/c7bf0b7c1a86d5eb3be2c722cf2cf746-Abstract.html. [p361, 363, 364, 365, 372]

D. Debeer and C. Strobl. Conditional permutation importance revisited. *BMC Bioinformatics*, 21(1):307, 2020. ISSN 1471-2105. doi: 10.1186/s12859-020-03622-2. [p360, 361]

N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola. AutoGluon-tabular: Robust and accurate AutoML for structured data. 2020. URL https://arxiv.org/abs/2003.06505. [p362]

F. K. Ewald, L. Bothmann, M. N. Wright, B. Bischl, G. Casalicchio, and G. König. A Guide to Feature Importance Methods for Scientific Inference. In L. Longo, S. Lapuschkin, and C. Seifert, editors, *Explainable Artificial Intelligence*, pages 440–464, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-63797-1. doi: 10.1007/978-3-031-63797-1_22. [p360, 361, 362, 363, 376]

H. Fanaee-T and J. Gama. Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence*, 2(2):113–127, 2014. ISSN 2192-6360. doi: 10.1007/s13748-013-0040-3. [p372]

S. Fischer, J. Zobolas, R. Sonabend, M. Becker, M. Lang, M. Binder, L. Schneider, L. Burk, P. Schratz, B. C. Jaeger, S. A. Lauer, L. A. Kapsner, M. Mücke, Z. Wang, D. Pulatov, K. Ganz, H. Funk, L. Harutyunyan, P. Camilleri, P. Kopper, A. Bender, B. Zhou, N. German, L. Koers, A. Nazarova, and B. Bischl. Mlr3extralearners: Expanding the mlr3 Ecosystem with Community-Driven Learner Integration. *Journal of Open Source Software*, 10(115):8331, 2025. ISSN 2475-9066. doi: 10.21105/joss.08331. [p368]

A. Fisher, C. Rudin, and F. Dominici. All Models are Wrong, but Many are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously. *Journal of machine learning research : JMLR*, 20:177, 2019. ISSN 1532-4435. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8323609/. [p360]

N. Foss and L. Kotthoff. Data and basic modeling. In B. Bischl, R. Sonabend, L. Kotthoff, and M. Lang, editors, *Applied Machine Learning Using mlr3 in R*. CRC Press, 2024. URL https://mlr3book.mlr-org.com/data_and_basic_modeling.html. [p366]

B. M. Greenwell and B. C. Boehmke. Variable importance plots—an introduction to the vip package. *The R Journal*, 12(1):343–366, 2020. doi: 10.32614/RJ-2020-013. [p364]

R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42, 2018. doi: 10.1145/3236009. [p360]

I. Guyon and A. Elisseeff. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3(Mar):1157–1182, 2003. ISSN ISSN 1533-7928. URL https://www.jmlr.org/papers/v3/guyon03a.html. [p360]

S. Holm. A Simple Sequentially Rejective Multiple Test Procedure. *Scandinavian Journal of Statistics*, 6 (2):65–70, 1979. ISSN 0303-6898. URL https://www.jstor.org/stable/4615733. [p362]

G. Hooker, L. Mentch, and S. Zhou. Unrestricted permutation forces extrapolation: Variable importance requires at least one more model, or there is no free variable importance. *Statistics and Computing*, 31 (6):82, Oct. 2021. ISSN 1573-1375. doi: 10.1007/s11222-021-10057-z. [p360, 362]

A. M. Horst, A. P. Hill, and K. B. Gorman. *Palmerpenguins: Palmer Archipelago (Antarctica) Penguin Data*, 2020. [p366]

T. Hothorn, K. Hornik, and A. Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, 15(3):651–674, 2006. ISSN 1061-8600, 1537-2715. doi: 10.1198/106186006X133933. [p361]

G. König, C. Molnar, B. Bischl, and M. Grosse-Wentrup. Relative Feature Importance. pages 9318–9325, 2021. doi: 10.1109/ICPR48806.2021.9413090. [p361, 362]

M. Lang, B. Bischl, and D. Surmann. Batchtools: Tools for R to work on batch systems. *Journal of Open Source Software*, 2(10):135, Feb. 2017. ISSN 2475-9066. doi: 10.21105/joss.00135. [p372]

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*, 4(44):1903, 2019. ISSN 2475-9066. doi: 10.21105/joss.01903. [p360]

J. Lei, G. Max, R. Alessandro, T. Ryan J., and L. Wasserman. Distribution-Free Predictive Inference for Regression. *Journal of the American Statistical Association*, 113(523):1094–1111, 2018. ISSN 0162-1459. doi: 10.1080/01621459.2017.1307116. [p361, 363, 370]

J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu. Feature Selection: A Data Perspective. *ACM Computing Surveys*, 50(6):1–45, Nov. 2018. ISSN 0360-0300, 1557-7341. doi: 10.1145/3136625. [p361]

C. Molnar. *Interpretable machine learning*. Lulu. com, 2020. URL https://christophm.github.io/interpretable-ml-book/. [p360]

C. Molnar, T. Freiesleben, G. König, J. Herbinger, T. Reisinger, G. Casalicchio, M. N. Wright, and B. Bischl. Relating the Partial Dependence Plot and Permutation Feature Importance to the Data Generating Process. In L. Longo, editor, *Explainable Artificial Intelligence*, pages 456–479, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-44064-9. doi: 10.1007/978-3-031-44064-9_24. [p361, 362, 369]

W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, and B. Yu. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080, 2019. doi: 10.1073/pnas.1900654116. [p360]

C. Nadeau and Y. Bengio. Inference for the Generalization Error. *Machine Learning*, 52(3):239–281, 2003. ISSN 1573-0565. doi: 10.1023/A:1024068626366. [p361, 362, 369]

K. K. Nicodemus, J. D. Malley, C. Strobl, and A. Ziegler. The behaviour of random forest permutation-based variable importance measures under predictor correlation. *BMC Bioinformatics*, 11(1):110, 2010. ISSN 1471-2105. doi: 10.1186/1471-2105-11-110. [p360]

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. URL https://jmlr.org/papers/v12/pedregosa11a.html. [p372]

A. Redelmeier, M. Jullum, and K. Aas. Explaining Predictive Models with Mixed Features Using Shapley Values and Conditional Inference Trees. In A. Holzinger, P. Kieseberg, A. M. Tjoa, and E. Weippl, editors, *Machine Learning and Knowledge Extraction*, pages 117–137, Cham, 2020. Springer International Publishing. ISBN 978-3-030-57321-8. doi: 10.1007/978-3-030-57321-8_7. [p360, 363]

B. Rozemberczki, L. Watson, P. Bayer, H.-T. Yang, O. Kiss, S. Nilsson, and R. Sarkar. The Shapley Value in Machine Learning. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pages 5572–5579, Vienna, Austria, 2022. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-1-956792-00-3. doi: 10.24963/ijcai.2022/778. [p360]

L. Schneider and M. Becker. Advanced tuning methods and black box optimization. In B. Bischl, R. Sonabend, L. Kotthoff, and M. Lang, editors, *Applied Machine Learning Using mlr3 in R*. CRC Press, 2024. URL https://mlr3book.mlr-org.com/advanced_tuning_methods_and_black_box_optimization.html. [p368]

H. Schulz-Kümpel, S. Fischer, R. Hornung, A.-L. Boulesteix, T. Nagler, and B. Bischl. Constructing Confidence Intervals for 'the' Generalization Error – a Comprehensive Benchmark Study, 2025. [p362]

L. S. Shapley. A Value for n-Person Games. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games (AM-28), Volume II*, pages 307–318. Princeton University Press, Dec. 1953. ISBN 978-1-4008-8197-0. doi: 10.1515/9781400881970-018. [p360, 363]

I. Sobol'. Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. *Mathematics and Computers in Simulation*, 55(1–3):271–280, 2001. ISSN 03784754. doi: 10.1016/S0378-4754(00)00270-6. [p360]

C. Strobl, A.-L. Boulesteix, T. Kneib, T. Augustin, and A. Zeileis. Conditional variable importance for random forests. *BMC Bioinformatics*, 9(1):307, 2008. ISSN 1471-2105. doi: 10.1186/1471-2105-9-307. [p361, 362]

J. Thomas. Preprocessing. In B. Bischl, R. Sonabend, L. Kotthoff, and M. Lang, editors, *Applied Machine Learning Using mlr3 in R*. CRC Press, 2024. URL https://mlr3book.mlr-org.com/preprocessing.html. [p366]

C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 847–855. Association for Computing Machinery, 2013. ISBN 978-1-4503-2174-7. doi: 10.1145/2487575.2487629. [p362]

M. J. van der Laan, E. C. Polley, and A. E. Hubbard. Super learner. *Statistical Applications in Genetics and Molecular Biology*, 6(1), 2007. doi: 10.2202/1544-6115.1309. [p362]

D. S. Watson and M. N. Wright. Testing conditional independence in supervised learning algorithms. *Machine Learning*, 110(8):2107–2129, 2021. ISSN 1573-0565. doi: 10.1007/s10994-021-06030-6. [p360, 361, 363, 365, 369]

D. S. Watson, K. Blesch, J. Kapar, and M. N. Wright. Adversarial Random Forests for Density Estimation and Generative Modeling. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, pages 5357–5375. PMLR, 2023. URL https://proceedings.mlr.press/v206/watson23a.html. [p361, 363, 368]

B. Williamson and J. Feng. Efficient nonparametric statistical inference on population feature importance using Shapley values. In *Proceedings of the 37th International Conference on Machine Learning*, pages 10282–10291. PMLR, Nov. 2020. URL https://proceedings.mlr.press/v119/williamson20a.html. [p363, 365]

B. D. Williamson, P. B. Gilbert, N. R. Simon, and M. Carone. A general framework for inference on algorithm-agnostic variable importance. *Journal of the American Statistical Association*, 118(543): 1645–1658, 2023. ISSN 0162-1459. doi: 10.1080/01621459.2021.2003200. [p361, 363, 365]

*Lukas Burk*
*1) Leibniz Institute for Prevention Research and Epidemiology - BIPS 2) LMU Munich, Department of Statistics 3) Munich Center for Machine Learning (MCML) 4) University of Bremen*
*Achterstraße 30, 28359 Bremen, Germany*
https://lukasburk.de
*ORCID: 0000-0001-7528-3795*
burk@leibniz-bips.de

*Fiona Katharina Meinzer*
*1) LMU Munich, Department of Statistics 2) Munich Center for Machine Learning (MCML)*
*Department of Statistics*
*Chair of Statistical Learning and Data Science*
*ORCID: 0009-0002-6372-3401*
fiona.ewald@lmu.de

*Giuseppe Casalicchio*
*1) LMU Munich, Department of Statistics 2) Munich Center for Machine Learning (MCML)*
*Department of Statistics*
*Chair of Statistical Learning and Data Science*
*ORCID: 0000-0001-5324-5966*

*Marvin N. Wright*
*1) Leibniz Institute for Prevention Research and Epidemiology - BIPS 2) University of Bremen*
*Achterstraße 30, 28359 Bremen, Germany*
*ORCID: 0000-0002-8542-6291*

*Bernd Bischl*
*1) LMU Munich, Department of Statistics 2) Munich Center for Machine Learning (MCML)*
*Department of Statistics*
*Chair of Statistical Learning and Data Science*
*ORCID: 0000-0001-6002-6980*

# LBBNN: An R Package for Sparse and Explainable Bayesian Deep Learning with Latent Binary Bayesian Neural Networks

*by Lars Skaaret-Lund, Eirik Høyheim and Aliaksandr Hubin*

**Abstract** The package LBBNN (latent binary Bayesian neural networks) provides a framework for doing sparse uncertainty aware and to a large degree interpretable Bayesian inference in neural network models, with potentially huge datasets, and highly overparametrized networks. Currently, most packages implementing Bayesian neural networks in R use Markov chain Monte Carlo based inference, without the possibility of GPU acceleration. Our package, using variational inference and the LibTorch backend (via the torch package), provides a more scalable and computationally efficient approach. The package integrates three different methodological works: Firstly, it takes into account the basic LBBNN model, where each weight in the network is associated with a Bernoulli latent inclusion indicator, allowing for incorporating model uncertainty and achieving substantial sparsification of the network while maintaining high predictive power. Secondly, using normalizing flows allows for modeling more complex and flexible variational distributions with statistical dependencies. And thirdly, by allowing the input variables to skip to any layer in the network, we can model different types of functions, e.g. we might only have linear connections. This allows for more explainable results. Lastly, local explanations at the prediction level with uncertainty are provided in the LBBNN package and are guaranteed to be exact for piecewise linear activations. To demonstrate how the package can be used in practice, we include experiments on both synthetic data and real-world datasets, covering both tabular and image based data.

## 1 Introduction

Modern deep learning architectures are typically parameterized by billions of parameters, in addition to being black box, and not offering uncertainty quantification, which makes them problematic to use in highly sensitive domains. Bayesian neural networks (BNN) alleviate the latter issue by having uncertainty quantification easily accessible by treating the model parameters as random variables. Further, the development of Latent Binary Bayesian Neural Networks (LBBNN) (Hubin and Storvik, 2024) allows for a scalable Bayesian approach to sparse neural network learning, by including a Bernoulli inclusion indicator variable for each weight. This allows using model uncertainty to reduce a network to a fraction of its initial size, while maintaining all the other advantages of doing Bayesian inference. In the frequentist setting, sparsification is usually obtained by more ad hoc procedures (see e.g. Frankle and Carbin (2018)).

Two limitations of LBBNNs as defined in Hubin and Storvik (2024) are addressed in Skaaret-Lund et al. (2024). The first is that while LBBNNs are sparser than BNNs during inference, they are still computationally expensive during training, since one has to sample *two* large matrices, one for the weights and one for the inclusion parameters for each layer in the network during forward propagation. The second issue is the mean-field posterior, which can only model the weights as independent random variables. The first issue is improved on by utilizing the local reparameterization trick (LRT) (Kingma et al., 2015), which makes it possible to sample the (approximate) Gaussian activations directly, rather than the weights and the inclusion variables. Secondly, by including normalizing flows (Rezende and Mohamed, 2015), the variational distribution can model more complex (and dependent) distributions instead of the simple mean-field Gaussian. However, this comes at the cost of having the additional flow parameters to optimize.

In addition to the aforementioned LBBNN models, the package also aims to include perhaps the most important recent development, namely input-skip LBBNNs (Høyheim et al., 2026). Input-skip refers to the design of a network such that the input is concatenated at each hidden layer. This, in addition to inclusion variables for these inputs, results in a network which could potentially include only linear functions, when the only non-zero inclusion variables will be from the inputs that have been concatenated at the last hidden layer, whereas standard networks must include all hidden layers. This is potentially very powerful, as traditional learning methods typically rely on selecting the model a priori, whereas in this case we can let the data decide the type of function to learn. In Høyheim et al. (2026), an experiment with synthetic data demonstrates the ability to select the correct function based on the data. The experiment shows that even though we start off dense (i.e. all connections from all layers are included), the network reduces to a simple linear function when the data is generated that way, or a more complicated non-linear one when that is required. In addition, input-skip LBBNNs

also generally give even sparser networks than standard LBBNNs on standard benchmark datasets such as MNIST. Finally, input-skip LBBNNs also introduce a novel approach to obtaining global and local explanations of predictions, important in domains where understanding the contributions of individual variables is desirable.

The latest stable release of the package can be installed from CRAN in an R session by running `install.packages("LBBNN")`.

## 2 Related work

Bayesian neural networks have been around since the early 1990s. They were popularized by Neal (1995) in his PhD-thesis, where inference was done using Hamiltonian Monte Carlo (HMC) (Duane et al., 1987) to sample from the posterior distribution of the neural network weights. HMC is considered an improvement on traditional MCMC techniques, as it uses Hamiltonian dynamics to explore the parameter space more efficiently. However, HMC does not scale well with large datasets, and very high-dimensional parameter spaces such as neural network posteriors, as we typically see in modern applications. In addition, deep and overparameterized neural network posteriors are typically highly multi-modal, another area where HMC struggles.

An alternative method that is more scalable is Variational Inference (VI) (Jordan et al., 1999). Rather than trying to sample from the true posterior distribution, VI defines a parameterized approximate posterior distribution, $q$, where the goal is to adjust the parameters of $q$ to minimize the discrepancy from the true posterior distribution, where the latter is typically measured by the KL-divergence. This optimization based approach is often preferred in practice (given finite computational budget) over HMC's asymptotically exact posterior samples, as one can utilize stochastic gradient descent and mini-batching of the data, using high performance parallel computing on modern GPU devices. Additionally, the objective function of VI is easy to monitor, avoiding the challenges of assessing HMC convergence in multi-modal settings.

Recent work on scalable VI such as Improved Variational Online Newton (IVON) (Shen et al., 2024) uses an implicit mean-field Gaussian approximation over the weights, where the mean and variance are updated with online second order curvature estimates of the loss, achieving a computational cost comparable to Adam, enabling scaling to large neural network architectures. While such methods improve the scalability of variational inference, recent work has also focused on improvements in sampling based inference, such as Microcanonical Langevin Ensembles (MILE) (Sommer et al., 2025), which uses deep ensembles to initialize short MCLMC (Robnik and Seljak, 2023) chains. This achieves substantial computational gains over the No-U-Turn Sampler (NUTS) (Hoffman et al., 2014) algorithm, at the cost of no longer guaranteeing asymptotic exactness of posterior sampling due to (deliberately) omitting the Metropolis step to correct the discretization bias.

Today, Python is the dominant programming language when implementing Bayesian (deep) neural networks. This is due to the availability of deep learning frameworks such as PyTorch (Paszke et al., 2019) and TensorFlow (Abadi et al., 2015), providing key features like high performance GPU computation and automatic differentiation. While it is possible to implement Bayesian neural networks directly using these frameworks, both ecosystems provide probabilistic programming libraries such as Pyro (Bingham et al., 2019) (for PyTorch) and TensorFlow Probability (Dillon et al., 2017) which facilitate Bayesian modeling by providing abstractions for probabilistic models and inference algorithms.

More recently, the Python ecosystem has grown to include `BlackJAX` (Cabezas et al., 2024), providing efficient, accelerator-agnostic implementations of MCMC algorithms including NUTS and HMC. In addition, the `bde` (Arvanitis et al., 2026) library combines scikit-learn compatibility with JAX-based training and uncertainty quantification for Bayesian deep ensembles, in particular an implementation of MILE. Another approach is the `posteriors` package (Duffield et al., 2024), providing scalable Bayesian deep learning built directly on top of PyTorch.

While the above mentioned Python based tools can be accessed in R via the `reticulate` (Ushey et al., 2026) package, this requires a properly configured Python/PyTorch environment alongside the R environment, which can introduce additional installation and dependency management considerations. In contrast, the R native `torch` (Falbel and Luraschi, 2026) package builds directly on libtorch, the backend C++ library that also powers PyTorch.

To the best of our knowledge, there is no R package that provides BNNs built on top of `torch`. The only R package we could find that offers GPU-accelerated variational inference for BNNs is `BayesFluxR` (Wegner, 2023) (itself a wrapper around a Julia backend), while the others — `bnns` (Chatterjee, 2025), `BNN` (Liang et al., 2018), `BLNN` (Sharaf et al., 2020), and `BoomSpikeSlab` (Steven L. Scott, 2025) rely on some form of MCMC. Finally, `selectnn` (McInerney and Burke, 2025) does a combinatorial search. For a detailed comparison of these packages, see Table 1.

Building on this gap, our `LBBNN` package offers a user-friendly implementation of sparse BNNs

(LBBNN) in R, built natively on the `torch` package, incorporating input-skip connections and supporting efficient optimization on modern high-performance hardware.

**Table 1:** Comparing our package LBBNN to other R packages implementing Bayesian neural networks.

| Package | Description | Backend | Inference | Hardware |
|---|---|---|---|---|
| `LBBNN` | BNNs with input-skip | LibTorch / C++ | VI + normalizing flows | CPU + GPU |
| `bnns` | Multi-layer feed forward BNN | Stan / rstan | MCMC (NUTS/HMC) | CPU |
| `BayesFluxR` | BNNs in Julia | Julia (BayesFlux.jl + Flux.jl) | MCMC, VI | CPU + GPU |
| `BNN` | One hidden layer BNN + shortcut connections | C | MCMC | CPU |
| `BLNN` | One hidden layer BNN | R | MCMC (HMC/NUTS) | CPU |
| `BoomSpikeSlab` | spike and slab regression | R / C++ | MCMC | CPU |
| `selectnn` | Neural network model selection | R | MLE + BIC | CPU |

## 3 Models

Given input variables, $\boldsymbol{x} \in \mathbb{R}^n$, and a response variable $\boldsymbol{y} \in \mathbb{R}^m$, a neural network models their relationship through a probability distribution $f$ parameterized by the vector $\boldsymbol{\eta}(\boldsymbol{x})$,

$$\boldsymbol{y} \sim f(\cdot; \boldsymbol{\eta}(\boldsymbol{x})),$$

with $f$ being a suitable distribution, e.g. for binary classification, $\boldsymbol{\eta}_i$ would be a vector of probabilities of success for the Bernoulli distribution. For LBBNNs, the vector of parameters $\boldsymbol{\eta}(\boldsymbol{x})$ is attained through a composition of semi-affine transformations:

$$u_j^{(l)} = \sigma^{(l)}\left(\sum_{i=1}^{n^{(l-1)}} u_i^{(l-1)}\gamma_{ij}^{(l)}w_{ij}^{(l)} + \gamma_j^{(l)}b_j^{(l)}\right), j = 1, \ldots, n^{(l)}, l = 1, \ldots, L, \tag{1}$$

where $\eta_j(\boldsymbol{x}) = u_j^{(L)}$. Additionally, $\boldsymbol{u}^{(l-1)}$ are the inputs from the previous layer (with $\boldsymbol{u}^0 = \boldsymbol{x}$). The $w_{ij}^{(l)}$'s are the weights and $b_j^{(l)}$'s – the bias terms. For each layer $l$ of a total $L$ layers, $n^{(l)}$ denotes the number of neurons, with $n^{(0)} = n$ the number of input variables. Furthermore, we have elementwise non-linear activation functions $\sigma^{(l)}$. Finally, $\gamma_{ij}^{(l)}$ and $\gamma_j^{(l)} \in \{0, 1\}$ are the binary inclusion variables for their corresponding weights and bias respectively. When all $\gamma_{ij}^{(l)} = 1$, equation (1) reduces to a standard BNN. In this case, we typically use independent Gaussians for the prior distribution of the weights and biases:

$$p(w_{ij}^{(l)}) = \mathcal{N}(w_{ij}; 0, \sigma_w^{2(l)}); \tag{2}$$

$$p(b_j^{(l)}) = \mathcal{N}(b_j^{(l)}; 0, \sigma_b^{2(l)}). \tag{3}$$

To take into account uncertainty in both weights, and the structure of the network, we use the spike-and-slab prior for our LBBNNs:

$$p(w_{ij}^{(l)}|\gamma_{ij}^{(l)}) = \gamma_{ij}^{(l)}\mathcal{N}(w_{ij}^{(l)}; 0, \sigma_w^{2(l)}) + (1 - \gamma_{ij}^{(l)})\delta(w_{ij}^{(l)});$$
$$p(\gamma_{ij}^{(l)}) = \text{Bernoulli}(\gamma_{ij}^{(l)}; \alpha_w^{(l)}),$$

and similarly for the biases. In the above, $\delta(\cdot)$ is the Dirac delta function, considered zero everywhere except at 0, where it has a spike. $\sigma^2$'s and $\alpha$'s denote the prior variances and inclusion probabilities for the weights (or biases). In our implementation, we allow for different values across layers, but within

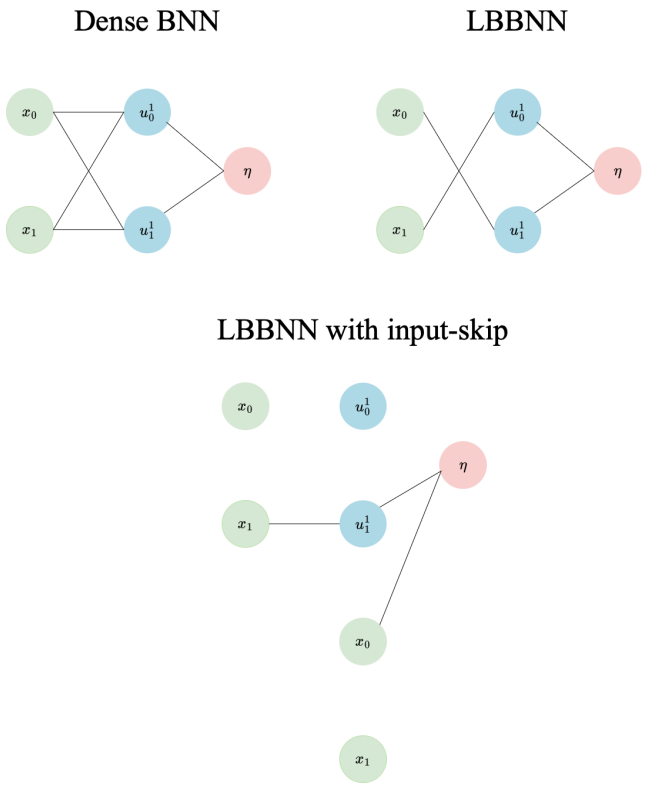



**Figure 1:** Top left, a dense BNN where all connections are included. Top right, LBBNN where some connections have been removed, resulting in a sparser representation. Bottom, LBBNN with input-skip allows both a sparse representation, and having a linear connection between $x_0$ and the output.

each layer they all have the same value. This is done for convenience but it is not strictly necessary. In principle each weight in the neural network could have a different prior inclusion and variance parameter.

Lastly, with input-skip connections enabled, we concatenate the input to the hidden layers as follows:

$$\boldsymbol{u}^{(l-1)} = \left[u_1^{(l-1)}, \ldots, u_{n^{(l-1)}}^{(l-1)}, x_0, \ldots, x_n\right], l = 2, \ldots, L. \tag{4}$$

For a graphical illustration of the differences between these methods, see Figure 1.

To implement the LBBNNs, we will follow Skaaret-Lund et al. (2024) and use LRT and normalizing flows. LRT allows for faster computations during training and inference, but assumes independent Gaussian distributions for the weights. Normalizing flows allow us to learn more complex variational posterior distributions as they account for dependencies between weights within a layer. Normalizing flows are implemented with RNVP (Dinh et al., 2016) with the numerically stable updates presented in Kingma et al. (2016). For a more detailed overview of computational costs related to both LRT and normalizing flows, please see Appendix A in Skaaret-Lund et al. (2024).

## 4 Package description

The main goal of LBBNN is to provide a framework to train Latent Binary Bayesian Neural Networks in R, using the `torch` package for high performance computation, while providing a user-friendly interface. For an overview of the workflow of the package, see Figure 2. We implement many of the generic R functions such as `summary()`, `coef()` and `plot()` to facilitate clear and understandable output. These utility functions support both standard LBBNN architectures, and LBBNN with input-skip. We hope to make LBBNNs available for both statisticians and applied researchers, who work on problems where data might be high dimensional and non-linear in nature, and where the domain dictates that reliable uncertainty calibration and explainability is desirable. Below we demonstrate five detailed examples of how to use the package in practice.

The first two deal with synthetic data, while the last three use real world data. The first tutorial is self-contained, whereas for the remaining tutorials we only report the most central details and refer the reader to the supplementary material for the full code needed to reproduce the examples.

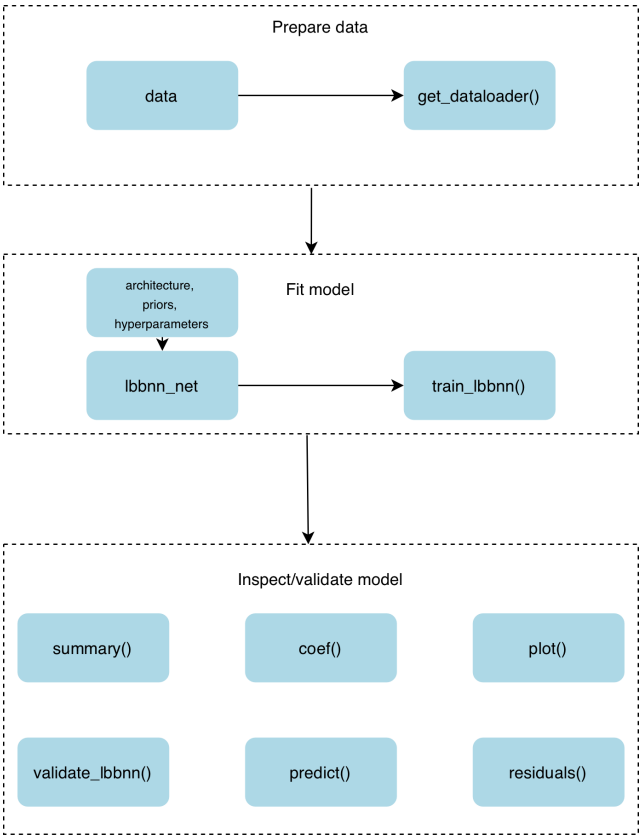



**Figure 2:** Schematic overview of the package.

### 4.1 Tutorial 1: Simulation study with linear effects

In the first example, we generate data in the following way:

$$x_{ij} \sim \mathcal{N}(0,1)$$
$$y_i = 0.6x_{i0} - 0.4x_{i1} + 0.5x_{i2} + \mathcal{N}(0,0.1),$$

with $i = 1,...,1000$ and $j = 0,...,15$. The goal is to check if the model is able to remove the non-relevant variables, in addition to recovering the true linear effects. The data is generated as follows:

```
library(LBBNN)
i <- 1000
j <- 15
set.seed(42)
torch::torch_manual_seed(42)
X <- matrix(rnorm(i * j, mean = 0, sd = 1), ncol = j)
y_base <-  0.6 * X[, 1] - 0.4 * X[, 2] + 0.5 * X[, 3] + rnorm(n = i, sd = 0.1)
sim_data <- as.data.frame(X)
sim_data <- cbind(sim_data, y_base)
```

**Data preprocessing.** In `torch`, it is common to use `torch::dataloader` objects to automatically handle mini-batching[1], shuffling and parallel loading of the data when training the model. To avoid users having to interface directly with this, we provide a wrapper, the `get_dataloaders()` function:

```
loaders <- get_dataloaders(sim_data, train_proportion = 0.9,
                           train_batch_size = 450, test_batch_size = 100,
                           standardize = FALSE)
train_loader <- loaders$train_loader
test_loader  <- loaders$test_loader
```

[1]We will not give here an introduction to `torch` but rather refer the reader to detailed tutorials here https://skeydan.github.io/Deep-Learning-and-Scientific-Computing-with-R-torch.

The function returns one loader with training data and one with test data. In addition, `sim_data` is a `data.frame` object with 1000 rows and 16 columns (the last being the target). In this example, we randomly select 900 samples for the training data, divided into two batches, and the remaining 100 as test samples.

**Model setup.** Further, we define the following hyperparameters:

```
problem <- "regression"
sizes <- c(j, 5, 5, 1) # 2 hidden layers, 5 neurons in each
incl_priors <- c(0.5, 0.5, 0.5) #prior inclusion probability
stds <- c(1, 1, 1) #prior for the standard deviation of the weights
incl_inits <- matrix(rep(c(-10, 10), 3), nrow = 2, ncol = 3) #inclusion inits
device <- "cpu" #can also be 'gpu' or 'mps'
```

In the above, `incl_inits` refers to the initialization of the parameters of the Bernoulli distribution governing weight inclusion, an important hyperparameter, as it controls the initial density of the network. In Tutorial 4, we go into more detail on different initialization strategies. The initial inclusion probabilities are computed with the logistic function, $\alpha_{\texttt{inits}} = \texttt{sigmoid(incl_inits)}$, which in the example above will give inclusion probabilities between $4.54 \cdot 10^{-5}$ and 0.99995. These hyperparameters are then used when defining an `lbbnn_net` object:

```
model_linear <- lbbnn_net(problem_type = problem, sizes = sizes,
                          prior = incl_priors, inclusion_inits = incl_inits,
                          std = stds, input_skip = TRUE, flow = FALSE,
                          num_transforms = 2, dims = c(10, 10, 10),
                          raw_output = FALSE, custom_act = NULL,
                          link = NULL, nll = NULL,
                          bias_inclusion_prob = FALSE, device = device)
```

The optional argument `input_skip` controls whether to use input_skip. Further, `flow` is for using normalizing flows with variational inference, where `num_transforms` and `dims` mean the number of transformations for the flow, and the dimensions of the hidden layers of the neural networks used in those transformations, respectively. `raw_output` gives the outputs of the last layer before any sigmoid/softmax transformation. Additionally, `link` and `nll` are for the user to supply their own link and likelihood function. `bias_inclusion_prob` determines whether the bias parameters should be associated with inclusion indicators. Lastly, `device` gives the option to move the training process to either `mps` or `gpu` devices.

**Statistical inference.** For training, we use the function `train_lbbnn`, where we pass the dataloader object defined earlier. In addition, `epochs` refers to the number of epochs to train for, where one epoch is a complete pass through the training dataset. `lr` defines the learning rate for the optimizer:

```
train_lbbnn(epochs = 300, LBBNN = model_linear,
            lr = 0.05, train_dl = train_loader, device = device)
```

The training process can be monitored on the console:

```
Epoch 1, training: loss = 827.28314, density = 0.47692
Epoch 2, training: loss = 711.41266, density = 0.46667
Epoch 3, training: loss = 618.56586, density = 0.46154
...
Epoch 299, training: loss = 29.64363, density = 0.04615
Epoch 300, training: loss = 29.75975, density = 0.04615
```

Showing the loss and density (proportion of weights with inclusion probabilities greater than 0.5) at each epoch. Using `verbose = FALSE` in the `train_lbbnn` function avoids printing to the console during training. Furthermore, we use `validate_lbbnn` to validate the results on unseen data:

```
validate_lbbnn(LBBNN = model_linear, num_samples = 10, test_dl = test_loader,
               device = device)
```

Here, `num_samples` refers to the number of samples from the variational posterior distribution of the parameters. The output is then averaged before making predictions. This typically improves performance compared to using the posterior mean of the parameters. `validate_lbbnn` returns the following output:

```
$validation_error
[1] 0.1127498

$validation_error_sparse
[1] 0.1108399

$density
[1] 0.04615385

$density_active_path
[1] 0.01538462
```

Where `validation_error` is the root mean squared error (RMSE) for the full model (no pruning). `validation_error_sparse` is the RMSE for the model where we have pruned away all the weights that are not included in active paths, where active paths are defined as paths that connect an input variable (directly, or via one or more hidden nodes) to the output node. See Høyheim et al. (2026) for a more formal definition of active paths. `density` refers to the proportion of weights in the whole network with a corresponding posterior inclusion probability larger than 0.5, whereas `density_active_path` only considers the weights included in active paths. The densities are quite low, which makes sense as we do not need many connections to represent the true data generative mechanism in this example.

**Results.** To further inspect the results, we can use `summary()`, which gives information about which variables are included in active paths, and from what layer. It also gives the average inclusion probabilities for each layer, and overall:

```
summary(model_linear)
```

Giving the following output:

```
Summary of lbbnn_net object:
------------------------------------
Shows the number of times each variable was included from each layer
------------------------------------
Then the average inclusion probability for each input from each layer
------------------------------------
The final column shows the average inclusion probability across all layers
------------------------------------
    L0 L1 L2    a0    a1    a2 a_avg
x0   0  0  1 0.404 0.391 0.999 0.452
x1   0  0  1 0.500 0.408 0.999 0.504
x2   0  0  1 0.405 0.430 1.000 0.470
x3   0  0  0 0.402 0.464 0.022 0.396
x4   0  0  0 0.496 0.451 0.020 0.432
x5   0  0  0 0.405 0.384 0.023 0.361
x6   0  0  0 0.139 0.404 0.025 0.249
x7   0  0  0 0.414 0.482 0.023 0.409
x8   0  0  0 0.500 0.500 0.028 0.457
x9   0  0  0 0.435 0.309 0.022 0.340
x10  0  0  0 0.500 0.311 0.027 0.371
x11  0  0  0 0.323 0.322 0.027 0.296
x12  0  0  0 0.486 0.366 0.023 0.390
x13  0  0  0 0.404 0.308 0.027 0.326
x14  0  0  0 0.501 0.405 0.025 0.414
The model took 9.9429999999702 seconds to train, using cpu
```

As we can see, only the first three variables are included in active paths (once each). This can also be visualized using the `plot()` function, with the argument `type = "global"`, to specify that we want the global explanation:

```
plot(model_linear, type = "global", vertex_size = 7,
     edge_width = 0.4, label_size = 0.4)
```

The plot can be seen in Figure 3. All connections are removed aside from the linear connections from the three data generating variables. To check if we recovered the true effects, we can obtain local explanations of predictions for some data samples of interest, using our version of `coef()` as follows:

```
coef(model_linear, dataset = train_loader, inds = NULL,
     output_neuron = 1, num_data = 5, num_samples = 10)
```

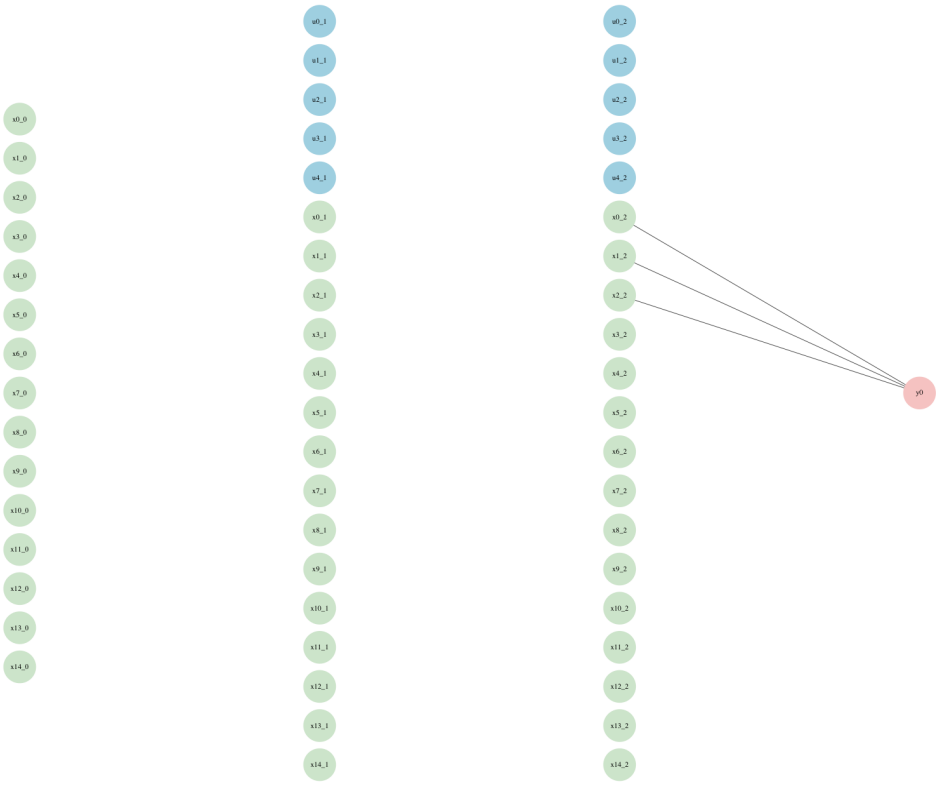


**Figure 3:** Global explanation for the network obtained in the simulation example with only linear effects.

The `dataset` argument refers to which dataset to use, it could be either a `torch::dataloader` object as in this case, or a user defined dataset. If the user wants to select specific samples from the dataset, then `inds` can be used to supply a vector of integers corresponding to the indices. If we have more than one output neuron (as in the case of multiclass classification), we get explanations for each output, `output_neuron` controls which one to select. Further, `num_data` refers to how many samples to select (randomly) from the given dataset, if no indices are provided. Lastly, `num_samples` refers to how many samples to use for model averaging over the weights in active paths. We report the mean, and 95% credible intervals for the explanations. If only one sample is used, the mean explanation and credible interval is given from model averaging over the weights. If multiple samples are selected, the overall mean explanation and confidence interval are derived from the individual mean explanations of each individual data sample.

For our example we obtain the following output:

```
     lower      mean        upper
x0   0.5947226  0.6046016  0.6130178
x1  -0.4096843 -0.4018597 -0.3905376
x2   0.4884034  0.4964451  0.5038068
x3   0.0000000  0.0000000  0.0000000
...
x14  0.0000000  0.0000000   0.0000000
```

The coefficients are very close to the true data generative ones, with only the first three variables included, as we wanted. If we only want to explain one specific sample, and also obtain the corresponding output, we can visualize this through the `plot()` function as follows:

```
x <- train_loader$dataset$tensors[[1]] #grab the dataset
y <- train_loader$dataset$tensors[[2]]
ind <- 42
data <- x[ind, ] #plot this specific data-point
output <- y[ind]
print(output$item()) #get the true y
[1] -0.8379451
plot(model_linear, type = "local", data = data)
```

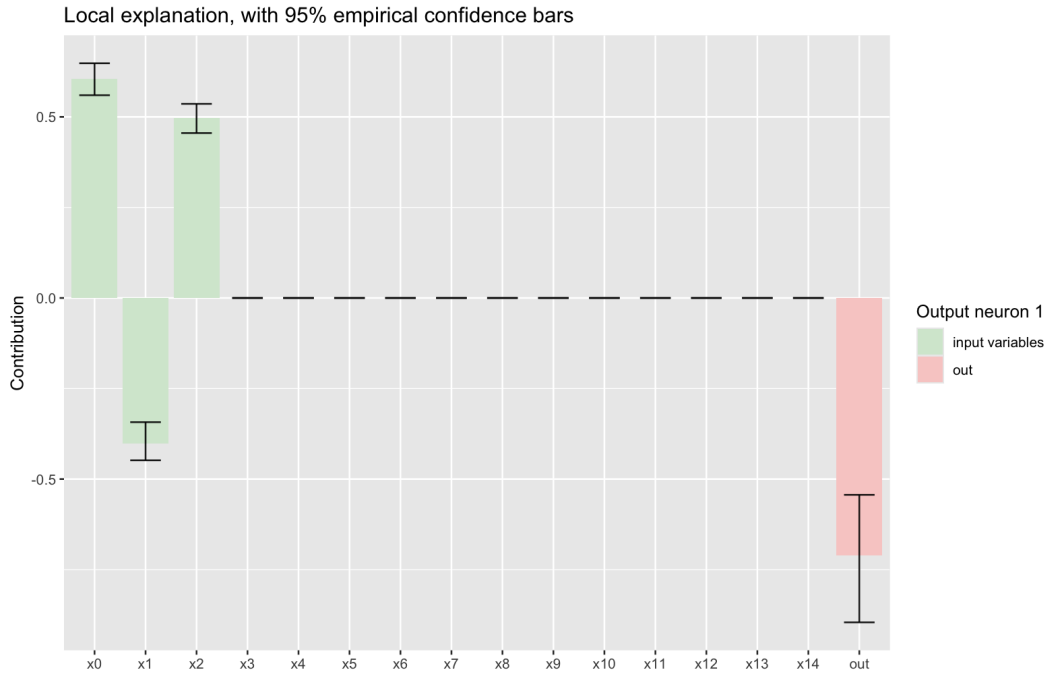



**Figure 4:** Local explanation with error bars for one sample in the simulation experiment with linear effects.

The resulting plot can be seen in Figure 4.

For a quick overview of the model and its parameters, `print()` is useful:

```
print(model_linear)
```

```
======================================
          LBBNN Model Summary
======================================

Module Overview:
  - An `nn_module` containing 607 parameters.

---------------- Submodules ----------------
  - layers               : nn_module_list  # 545 parameters
  - layers.0             : lbbnn_linear    # 235 parameters
  - layers.1             : lbbnn_linear    # 310 parameters
  - act                  : nn_leaky_relu   # 0 parameters
  - out_layer            : lbbnn_linear    # 62 parameters
  - out                  : nn_identity     # 0 parameters
  - loss_fn              : nn_mse_loss     # 0 parameters

Model Configuration:
  - LBBNN with input-skip
  - Optimized using variational inference without normalizing flows

Priors:
  - Prior inclusion probabilities per layer:  0.5, 0.5, 0.5
  - Prior std dev for weights per layer:     1, 1, 1

======================================================================
```

### 4.2 Tutorial 2: Simulation study with non-linear effects

The data is generated in the following way:

$$x_{ij} \sim U(0, 0.5)$$
$$\eta(\boldsymbol{x}_i) = -3 + 0.1\log(x_{i0}) + 3\cos(x_{i1}) + 2x_{i2}x_{i3} + x_{i4} - x_{i5}^2 + \mathcal{N}(0, 0.1),$$

With $i = 1, ..., 1000$ and $j = 0, ..., 15$. In this case, we further transform the output into a binary variable:

$$y_i = \begin{cases} 0, & \eta(x_i) < \text{median}(\eta(x_i)) \\ 1, & \eta(x_i) \geq \text{median}(\eta(x_i)). \end{cases}$$

Otherwise we use mostly the same hyperparameters as in the previous example, however since we now have a binary output, we specify:

```
problem <- "binary classification"
```

In addition, we use normalizing flows, setting the argument `flow = TRUE` when initializing the model object. Training, otherwise, is done in exactly the same way as in the previous example. To check the results we again start with `validate_lbbnn`:

```
validate_lbbnn(LBBNN = model_nl, num_samples = 100, test_dl = test_loader_nl,
               device = device)
$accuracy_full_model
[1] 0.87

$accuracy_sparse
[1] 0.86

$density
[1] 0.1384615

$density_active_path
[1] 0.08717949
```

We get around 86% accuracy with what appears to be a sparse network. To investigate the global structure of the network, we can use the `plot()` function again, specifying that we are interested in the global explanation:

```
plot(model_nl, type = "global", vertex_size = 9,
     edge_width = 0.4, label_size = 0.4)
```

The results are displayed in Figure 5. From this, we see that all weights related to the irrelevant variables $x_6 - x_{14}$ have been pruned away, leaving us with a very sparse representation of effects for data-generating covariates.

### 4.3 Tutorial 3: Real data classification experiment

For this experiment, we use the Gallstone dataset (Esen et al., 2024) from the UCI machine learning repository. It consists of data from 319 individuals, where 161 were diagnosed with gallstone disease. It contains a mix of demographic, bioimpedance and laboratory covariates. The workflow is the same as in the previous example, so we skip the detailed description.

For this problem, we use two hidden layers with three neurons in each, and train for 1000 epochs. We split the data 70/30 for training and validation, in line with the experiment described in Esen et al. (2024), although we do not know the exact indices for the split, so the results may vary. In Esen et al. (2024), the best result (85.42 %) is obtained using gradient boosting. Our sparse input-skip model obtained an accuracy of 83.33%, using only roughly 5% of the total weights.

The sparse structure is shown in Figure 6. We see that the entire first hidden layer has been pruned away, and only two of the three nodes in the last hidden layer are used. Additionally, many variables only have connections from the last layer.

Our package includes a few more standard R functions such as `predict()` and `residuals()` that we would like to demonstrate in this example. Below we show how to use the former:

```
predictions_gs <- predict(model_gs, newdata = test_loader_gs,
                          draws = 100, mpm = TRUE)
```

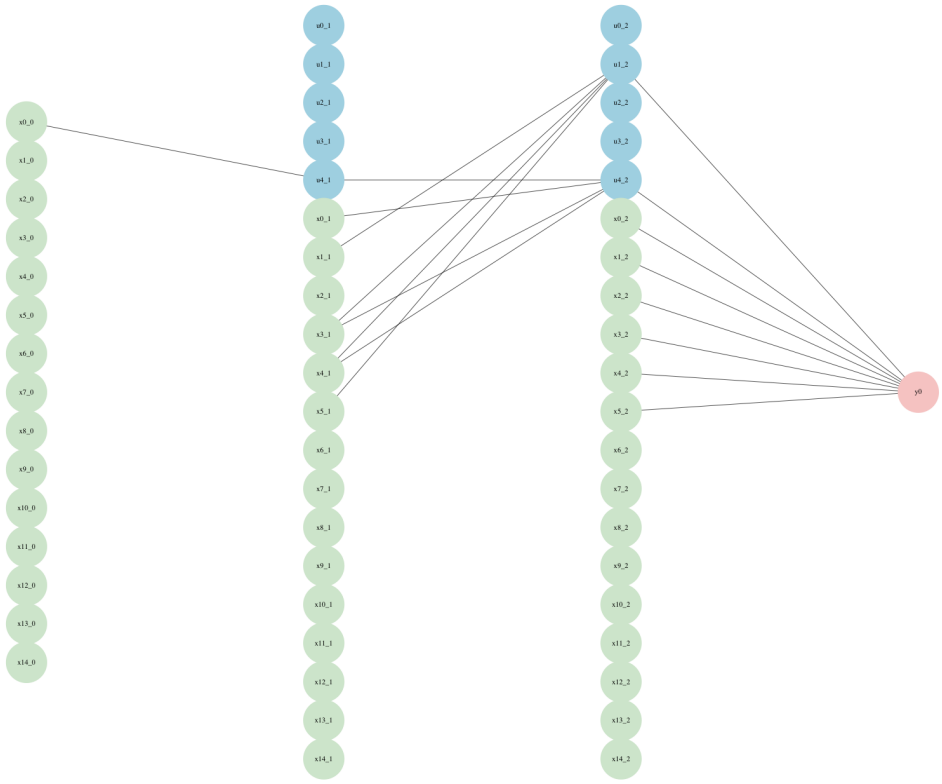


**Figure 5:** The structure of the network in the non-linear simulation example after removing weights that are not included in active paths.

Where `newdata` is the dataset to predict, `draws` refers to the number of posterior samples, and `mpm` refers to using the median probability model. The output of `predict` is the raw network outputs before sigmoid or softmax (or other link function) is applied. It is a three-dimensional tensor:

```
dim(predictions_gs)
[1] 100  96   1
```

where the axes correspond to draws, data points, and output units, respectively. For our case, the resulting tensor looks as:

```
print(predictions_gs)
torch_tensor
(1,.,.) =
-2.6755
0.2434
-3.2979
...
-1.3697
-0.3062
-1.5818
... [the output was truncated (use n=-1 to disable)]
[ CPUFloatType{100,96,1} ]
```

### 4.4 Tutorial 4: Varying initializations on the raisin dataset

Results (accuracy, final network structure) can be sensitive to some hyperparameters such as number of epochs, or the initializations of the probability of the inclusion parameters. More generally, LBBNNs (with or without input-skip) can present optimization challenges, as the joint optimization over weights and inclusion parameters may lead to unstable training dynamics. The commonly used Adam optimizer, which we employ, was not designed for this type of problem. In future work, we aim to develop optimization strategies specifically tailored to LBBNNs.

This example will go into more detail on initialization strategies for the inclusion probabilities, and

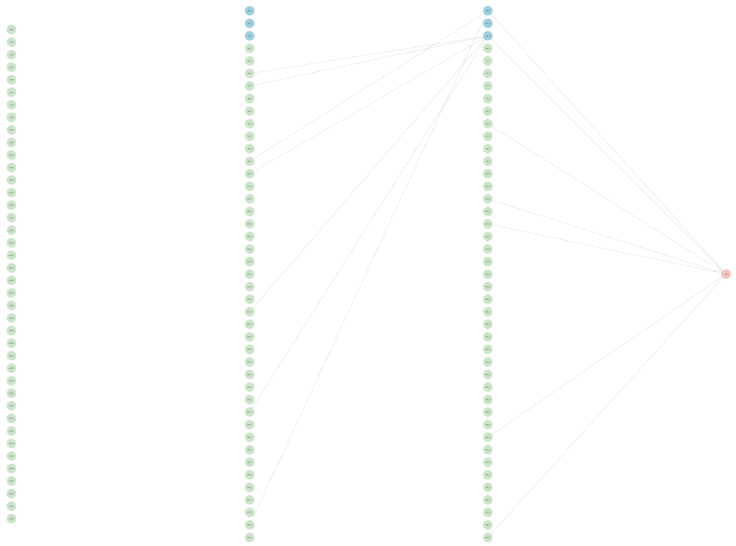


**Figure 6:** The structure of the network in the gallstone data classification example after removing weights that are not included in active paths.

will present the user with different options. As mentioned previously, they are initialized as follows:

$$U \sim \texttt{Uniform(a,b)}$$
$$\alpha_{\texttt{init}} = \texttt{sigmoid}(U).$$

As an alternative to providing the values for `a` and `b`, we allow the user to also submit the keywords found in Table 2. Figure 7 provides a visualization of the initial distribution of the resulting Bernoulli probabilities, $\alpha_{\texttt{init}}$.

**Table 2:** Initialization ranges for the logits of the Bernoulli probabilities.

| Keyword | balanced | dense | polarized | polarized_dense | polarized_mild | polarized_sparse | sparse |
|---|---|---|---|---|---|---|---|
| a | -1 | 1.5 | -10 | -5 | -3 | -10 | -2.5 |
| b | 1 | 2.5 | 10 | 10 | 3 | 5 | -1.5 |
| $a \rightarrow \alpha_{\texttt{lower}}$ | 0.27 | 0.82 | $\sim$0 | 0.01 | 0.05 | $\sim$0 | 0.08 |
| $b \rightarrow \alpha_{\texttt{upper}}$ | 0.73 | 0.92 | $\sim$1 | $\sim$1 | 0.95 | 0.99 | 0.18 |

In the following experiment, we use the raisin dataset (Çinar et al., 2020), consisting of 900 samples, where the goal is to classify two types of raisins based on 7 morphological features. We use a network with 3 hidden layers, each consisting of 10 neurons. Of the 900 samples, 720 are used for training and 180 for validation. We compare results with different initializations, where the model is now defined as:

```
model_raisins <- lbbnn_net(problem_type = problem,sizes = sizes,
                           prior = inclusion_priors,
                           inclusion_inits = 'balanced',input_skip = TRUE,
                           std = stds, flow = FALSE, device = device)
```

Figure 8 displays the densities during training for the different initializations. In the dense case, the density is 1 for around 300 epochs, before it quickly approaches something close to 0. We see that the polarized options take longer to converge to low sparsities, but in the end, all initializations lead to very sparse networks after training for 3000 epochs. As a very low density may not be desirable in all situations, we include a tuning parameter for the minimum allowed density, such that training is terminated if this value is reached. It can be done using the `min_density` keyword inside the `train_lbbnn` function:

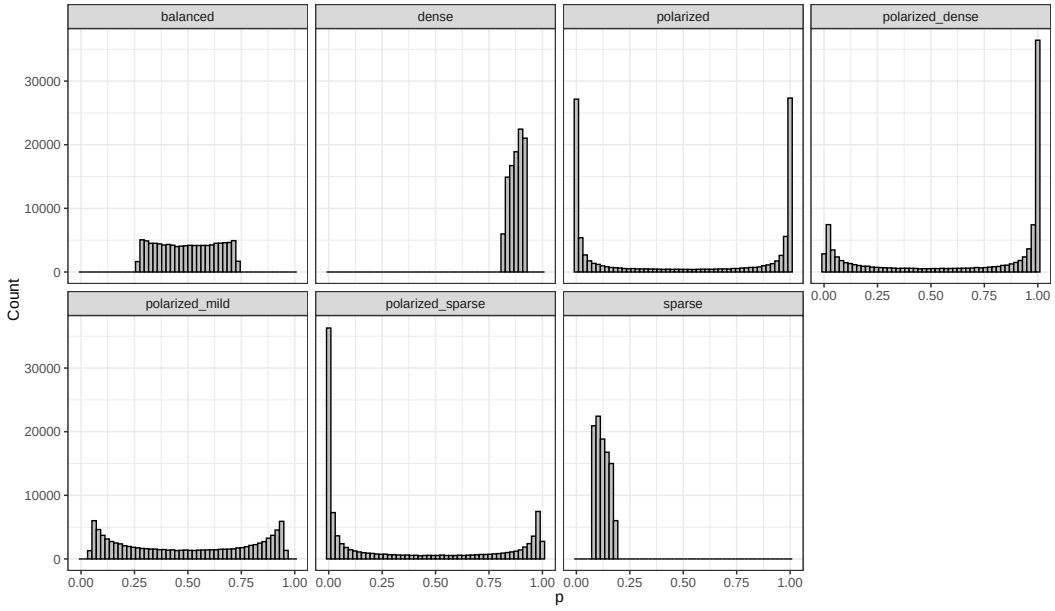



**Figure 7:** Empirical distributions of initial inclusion probabilities for different initialization schemes.

**Table 3:** Performance and sparsity metrics on validation data on the raisins dataset.

| Initialization | accuracy | sparse accuracy | density | density active paths |
|---|---|---|---|---|
| balanced | 0.8556 | 0.8556 | 0.0118 | 0.0118 |
| dense | 0.8611 | 0.8667 | 0.0141 | 0.0141 |
| polarized | 0.8667 | 0.8722 | 0.0351 | 0.0258 |
| polarized_dense | 0.8556 | 0.8556 | 0.0234 | 0.0211 |
| polarized_mild | 0.8611 | 0.8611 | 0.0258 | 0.0141 |
| polarized_sparse | 0.8556 | 0.8556 | 0.0281 | 0.0094 |
| sparse | 0.8722 | 0.8611 | 0.0141 | 0.0094 |

```
results_raisins <- train_lbbnn(epochs = 3000,LBBNN = model_raisins, lr = 0.005,
                               train_dl = train_loader_raisin, device = device,
                               min_density = NULL)
```

For this example, we did not use a minimum density, but trained for 3000 epochs, to ensure convergence. The results on the validation data can be seen in Table 3. We see that the resulting networks are very sparse, using around $1-2\%$ of the weights. Despite this, accuracy is good and comparable to the results reported in the original paper (Çinar et al., 2020), 0.8633 accuracy using a frequentist neural network. In Figure 9 the global structures with the different initializations are shown.

### 4.5 Tutorial 5: Extending the LBBNN to a convolutional architecture

In this tutorial, we demonstrate an example on how to extend LBBNN. The specific extension is a Bayesian convolutional neural network (CNN), using `lbbnn_linear` and `lbbnn_conv2d` for feed-forward and convolutional LBBNN layers respectively. We have previously used `lbbnn_linear` layers internally when defining `lbbnn_net` objects, but here we will go into more details about how to explicitly define the layers needed in a CNN architecture. We do not use input-skip in this example, for simplicity.

We use the KMNIST dataset (Clanuwat et al., 2018), consisting of 70,000 grayscale images (28x28 pixels) of handwritten Japanese Hiragana characters, with 10 different classes, providing a much larger and higher dimensional dataset than the previous examples. KMNIST is considered a more challenging alternative to the classical MNIST dataset, commonly used to benchmark novel machine learning methods. Some examples of what these images look like can be seen in Figure 10.

To begin, we download the dataset using the `torchvision` package, and create dataloaders, this time using `torch` directly:

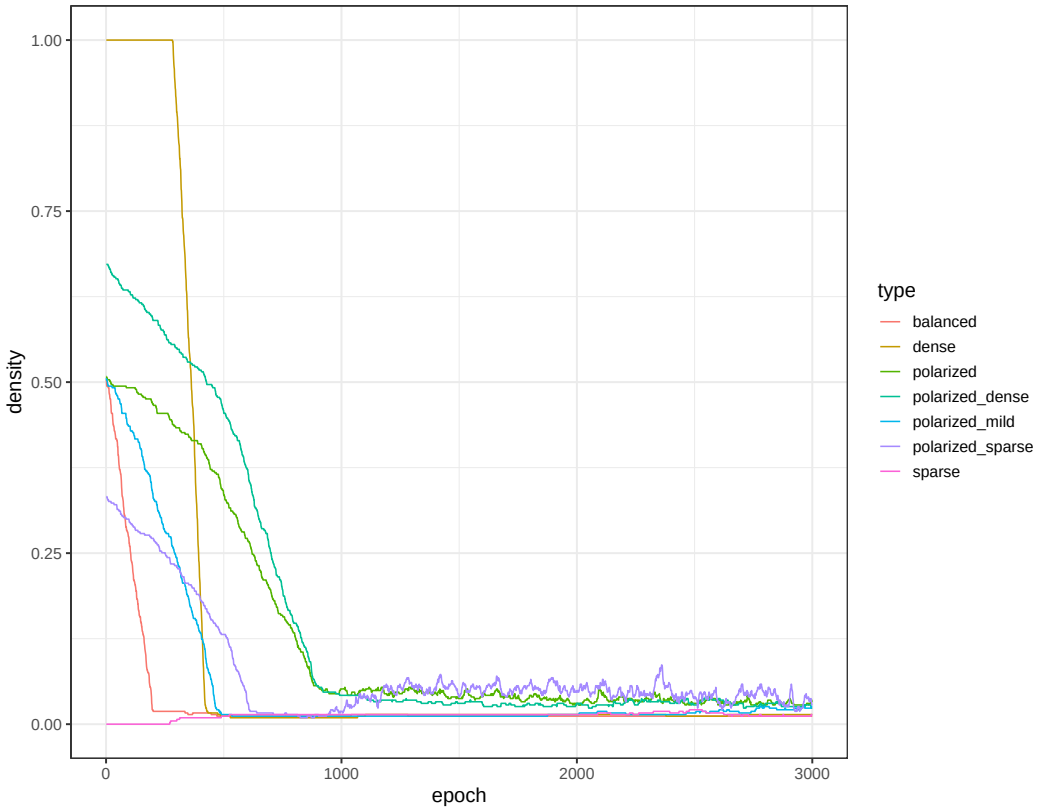



**Figure 8:** Density across epochs with different initializations.

```
torch::torch_manual_seed(42)

dir <- "./dataset/kmnist"
train_ds <- torchvision::kmnist_dataset(
   dir,
   download = TRUE,
   transform = torchvision::transform_to_tensor)

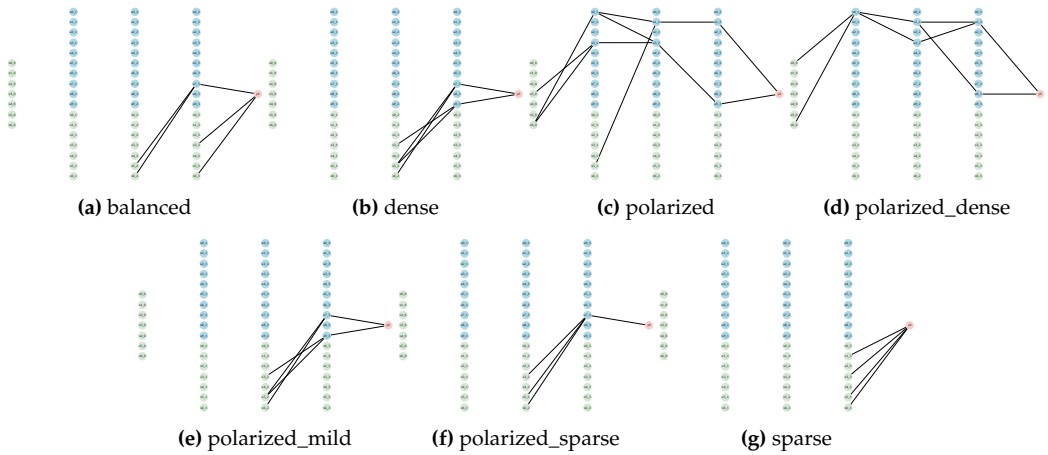
test_ds <- torchvision::kmnist_dataset(
  dir,
  train = FALSE,
  transform = torchvision::transform_to_tensor)

# define dataloaders
train_loader_kmnist <- torch::dataloader(train_ds, batch_size = 100, shuffle = TRUE)
test_loader_kmnist <- torch::dataloader(test_ds, batch_size = 100)
```

The dataset is split into 60,000 training samples and 10,000 test samples, loaded into the corresponding dataloaders. For the CNN architecture, we use two convolutional layers, and then two fully connected layers. The convolutional layers are defined as follows:

```
device <- "cpu"
conv_layer_1 <- lbbnn_conv2d(in_channels = 1, out_channels = 32, kernel_size = 5,
                             prior_inclusion = 0.5, standard_prior = 1,
                             density_init = c(-10, 10), num_transforms = 2,
                             flow = FALSE, hidden_dims = c(200, 200),
                             device = device)
conv_layer_2 <- lbbnn_conv2d(in_channels = 32, out_channels = 64, kernel_size = 5,
                             prior_inclusion = 0.5, standard_prior = 1,
                             density_init = c(-10, 15), num_transforms = 2,
                             flow = FALSE, hidden_dims = c(200, 200),
                             device = device)
```

Here, we used `'cpu'` device for compatibility across platforms, however for this example to run faster,



**(a)** balanced **(b)** dense **(c)** polarized **(d)** polarized_dense

**(e)** polarized_mild **(f)** polarized_sparse **(g)** sparse

**Figure 9:** Comparison of the global structure of the networks after training with different initializations on the raisin dataset.

we recommend using `'gpu'` or `'mps'` devices upon availability [2]. Further, `in_channels` refers to the number of input channels to the layer, which is 1 for grayscale images such as KMNIST. `out_channels` refers to the number of channels or feature maps produced by the convolutional layer. `kernel_size` specifies the dimensions of the convolutional filter applied to the input. For example, a `kernel_size` of 5 corresponds to a 5x5 filter. The rest of the arguments are the same as those used to define `lbbnn_net` objects in previous examples. Further, we define the following fully-connected layers:

```
linear_layer_1 <- lbbnn_linear(in_features = 1024, out_features = 300,
                         prior_inclusion = 0.5, standard_prior = 1,
                         density_init = c(-10, 10), num_transforms = 2,
                         flow = FALSE, hidden_dims = c(200, 200), device = device,
                         bias_inclusion_prob = FALSE, conv_net = TRUE)

linear_layer_2 <- lbbnn_linear(in_features = 300, out_features = 10,
                         prior_inclusion = 0.5, standard_prior = 1,
                         density_init = c(-5, 15),num_transforms = 2,
                         flow = FALSE, hidden_dims = c(200, 200), device = device,
                         bias_inclusion_prob = FALSE, conv_net = TRUE)
```

After pooling each convolutional layer and flattening the output of the final layer, we obtain 1,024 input features to the first layer, where `out_features` refers to the number of hidden neurons in that layer. For the final layer, we have 10 `out_features`, as there are 10 possible classes. The argument `conv_net` is used to skip the computation of active paths. The model class is defined as follows:

```
LBBNN_ConvNet <- torch::nn_module(
  "LBBNN_ConvNet",

  initialize = function(conv1, conv2, fc1 ,fc2 ,device = device) {
    self$problem_type <- "multiclass classification"
    self$input_skip <- FALSE
    self$conv1 <- conv1
    self$conv2 <- conv2
    self$fc1 <- fc1
    self$fc2 <- fc2

    self$pool <- torch::nn_max_pool2d(2)
    self$act <- torch::nn_leaky_relu()
    self$out <- torch::nn_log_softmax(dim = 2)
    self$pout <- torch::nn_softmax(dim = 2)
    self$loss_fn <- torch::nn_nll_loss(reduction = "sum")
  },
```

[2]For this specific example we achieved 1.7x speedup on a NVIDIA RTX 6000 Ada Generation 48GB GPU versus an Intel® Xeon® E5-2680 v4 @ 2.40GHz (56 threads) CPU, and when increasing batch size to 512 we got 2.1x speedup.

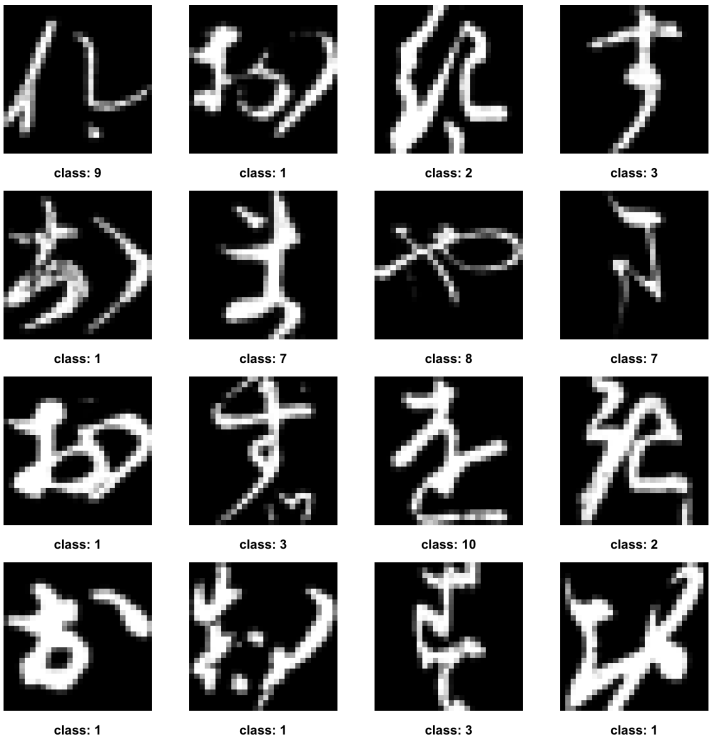



**Figure 10:** Random 4×4 grid of KMNIST images with class labels.

```
forward = function(x, MPM = FALSE, predict = FALSE) {
  x = self$act(self$conv1(x, MPM))
  x = self$pool(x)
  x = self$act(self$conv2(x, MPM))
  x = self$pool(x)
  x = torch::torch_flatten(x,start_dim = 2)
  x = self$act(self$fc1(x, MPM))
  if(!predict)
    x = self$out(self$fc2(x ,MPM))
  else
    x = self$pout(self$fc2(x ,MPM))
},
kl_div = function(){
  kl <- self$conv1$kl_div() + self$conv2$kl_div() +
    self$fc1$kl_div() + self$fc2$kl_div()
  return(kl)
},
density = function(){
  alphas <- NULL
  l1 <- self$conv1$lambda_l$clone()$detach()
  l2 <- self$conv1$lambda_l$clone()$detach()
  l3 <- self$fc1$lambda_l$clone()$detach()
  l4 <- self$fc2$lambda_l$clone()$detach()
  alphas <- c(as.numeric(torch::torch_sigmoid(l1)),
              as.numeric(torch::torch_sigmoid(l2)),
              as.numeric(torch::torch_sigmoid(l3)),
              as.numeric(torch::torch_sigmoid(l4)))
  return(mean(alphas > 0.5))
},
compute_paths = function(){
  NULL
},
density_active_path = function(){
  NA
}
```

```
)
```

The `initialize` method runs automatically each time an instance of `LBBNN_ConvNet` is created, setting up the network layers and additional parameters. As well as the layers mentioned, we also define the pooling layer, output layer, activation function and loss function here. The method `forward` applies two convolutional layers, each followed by a LeakyReLU activation function and max-pooling. The resulting feature maps are flattened and passed through a fully connected layer with LeakyReLU, and then through a final output layer. The argument `MPM` refers to the median probability model. When `TRUE`, only weights with a corresponding inclusion probability greater than 0.5 are used. `predict` refers to whether the forward pass is used in the prediction mode, returning class probabilities. Further, `kl_div` is used to compute the KL-divergence of the network, which is the sum of the individual KL-divergences between the prior and posterior (variational) distribution of the weights and inclusion parameters for each independent layer. This is added to the loss function of the network. To compute the density (proportion of weights in the network with inclusion probability larger than 0.5) we use the `density` method. Lastly, `compute_paths` and `density_active_path` need to be defined (but not do anything) so that the same training and validation functions as in previous examples can be used. We can now define an instance of the model and move it to the device:

```
model_kmnist <- LBBNN_ConvNet(conv_layer_1, conv_layer_2, linear_layer_1,
                      linear_layer_2, device)
model_kmnist$to(device = device)
```

We now use `train_lbbnn`, training for 20 epochs with a learning rate of 0.001:

```
train_lbbnn(epochs = 20, LBBNN = model_kmnist, lr = 0.001, train_dl = train_loader_kmnist,
            device = device)
```

During training, the loss, accuracy and density can be monitored just as in the standard LBBNN:

```
Epoch 1, training: loss = 2861.02612, acc = 0.84528, density = 0.47281
Epoch 2, training: loss = 2516.05835, acc = 0.95975, density = 0.44291
Epoch 3, training: loss = 2209.72510, acc = 0.97410, density = 0.40780
...
Epoch 20, training: loss = 233.06529, acc = 0.95875, density = 0.03716
```

Finally, we validate the network on the unseen test data, with 10 samples for model averaging:

```
validate_lbbnn(model_kmnist, num_samples = 10, test_dl = test_loader_kmnist, device = device)
```

Giving the following output:

```
$accuracy_full_model
[1] 0.9355

$accuracy_sparse
[1] 0.9259

$density
[1] 0.03715523

$density_active_path
[1] NA
```

The accuracy is lower than during training, as one can expect. It could be improved by training for longer, or using normalizing flows as shown in Skaaret-Lund et al. (2024), but the purpose here is not to obtain the best possible results, but rather demonstrate how a simple convolutional LBBNN can be created. We note that convolutional LBBNNs can also be very sparse without sacrificing too much predictive power in this case. Active paths are not defined for convolutional architectures, hence density within them is not available for this example.

As we now are working with a different object than the standard `lbbnn_net` object used earlier, our versions of generic R functions such as `summary()` and `coef()` will not work. We can still use `print()`, as this will use the `torch` version, and `LBBNN_ConvNet` is a `torch::nn_module` object. Below is the output:

```
print(model_kmnist)
An `nn_module` containing 1,087,412 parameters.
--Modules ---------------------------------
• conv1: <lbbnn_conv2d> #2,464 parameters
• conv2: <lbbnn_conv2d> #153,728 parameters
```

```
• fc1: <lbbnn_linear> #922,200 parameters
• fc2: <lbbnn_linear> #9,020 parameters
• pool: <nn_max_pool2d> #0 parameters
• act: <nn_leaky_relu> #0 parameters
• out: <nn_log_softmax> #0 parameters
• loss_fn: <nn_nll_loss> #0 parameters
```

The prediction function is not per default available for a convolutional network class, so we have to make predictions manually (or wrap it in a separate function). Below we show how to perform predictions manually:

```
draws <- 20 # how many samples from posterior to use
out_dim <- 10 # dimensionality of the output
mpm <- TRUE # if to use the MPM
model_kmnist$eval() # to avoid gradient computations
predictions_kmnist <- NULL
torch::with_no_grad({
  coro::loop(for (b in test_loader_kmnist)# go through all data
  {
    outputs <- torch::torch_zeros(draws,dim(b[[1]])[1],out_dim)$to(device=device)
    for(i in 1:draws)# go through all draws
    {
      data <- b[[1]]$to(device = device)
      outputs[i]<- model_kmnist(data, MPM = mpm, predict = TRUE)
    }
   predictions_kmnist <- torch::torch_cat(c(predictions_kmnist, outputs), dim = 2) #combine all

  })
})
```

Here we used 20 draws from the posterior per sample and we have 10 classes in the response. So, the dimension of the responses is

```
dim(predictions_kmnist)
[1]    20 10000    10
```

And we can take a look at the first 5 draws of the 255th[3] predicted image posteriors:

```
idx1 <- min(255, dim(predictions_kmnist)[2])
print(torch::torch_round(predictions_kmnist[1:5, idx1, ], 4))

torch_tensor
 0.0096  0.0183  0.6126  0.0018  0.0215  0.0228  0.0546  0.2543  0.0028  0.0016
 0.0006  0.0002  0.9166  0.0020  0.0218  0.0180  0.0367  0.0020  0.0017  0.0002
 0.0018  0.0108  0.2626  0.0058  0.2404  0.3693  0.0863  0.0215  0.0003  0.0014
 0.0056  0.0463  0.1627  0.0050  0.0088  0.2112  0.2470  0.3102  0.0029  0.0003
 0.0001  0.0003  0.6468  0.0008  0.0056  0.3341  0.0116  0.0003  0.0004  0.0000
[ CPUFloatType{5,10} ]
```

In this specific case we see some uncertainty between different classes. If we look at the first 5 draws of the 258th predicted image posteriors, we see significant confidence in class 5:

```
idx2 <- min(258, dim(predictions_kmnist)[2])
print(torch::torch_round(predictions_kmnist[1:5, idx2, ], 4))

torch_tensor
 0.0001  0.0001  0.0000  0.0000  0.9993  0.0000  0.0000  0.0003  0.0000  0.0001
 0.0017  0.0000  0.0001  0.0000  0.9947  0.0001  0.0003  0.0007  0.0002  0.0022
 0.0041  0.0035  0.0001  0.0005  0.7902  0.0021  0.0004  0.0136  0.0000  0.1854
 0.0001  0.0000  0.0000  0.0000  0.9987  0.0000  0.0010  0.0002  0.0000  0.0000
 0.0010  0.0005  0.0000  0.0001  0.9948  0.0001  0.0000  0.0024  0.0009  0.0002
[ CPUFloatType{5,10} ]
```

[3] indexes are shuffled in torch data_loaders, so results might differ on a different seed.

## 5 Summary

We have created a package that implements LBBNNs in R. It offers the possibility to use normalizing flows for a more flexible variational posterior distribution than the mean-field Gaussian. In addition, input-skip allows for a more flexible architecture where the concept of active paths can lead to very sparse representation, and thus obtaining global interpretability. Local explanations are also further available and proven to be exact for piecewise linear activations. Our package is the first R package implementing LBBNNs using `torch`, facilitating GPU acceleration using the `LibTorch` backend.

In our examples, we demonstrate how to use our package, both on toy data and real-world datasets. On the latter, our LBBNN model with input-skip obtains high accuracy with sparse networks, while also providing local explanations with uncertainty, easily accessible either through `coef()` (with the possibility of aggregating multiple samples) or for visualization of a single sample through `plot()`. Additional information about the model can be obtained through `summary()`, showing how many paths included each input variable from each layer, in addition to the average inclusion probability for each layer, and overall. The structure of the model and the number of total parameters can be seen through `print()`. We further provided an example on how to extend the package to work with convolutional architectures with image classification applications, demonstrating that our package can also handle high dimensional data. Further, the fundamental building blocks used in that example could also be used to create much deeper architectures, such as residual networks like ResNet-18, however, this would come at the cost of significantly more parameters compared to the roughly 1 million in our example.

Future work will focus on extending the LBBNN package to support deeper and more complex architectures like residual networks or transformers, explore alternative transformations for the normalizing flows, and, most importantly, develop novel LBBNN specific optimizers that can better handle model selection and the input-skip architecture in high-dimensional parameter settings.

## 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. [p382]

V. Arvanitis, A. Aslanidis, E. Sommer, and D. Rügamer. bde: A python package for Bayesian deep ensembles via MILE. *arXiv preprint arXiv:2605.14146*, 2026. URL https://arxiv.org/abs/2605.14146. [p382]

E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. A. Szerlip, P. Horsfall, and N. D. Goodman. Pyro: Deep universal probabilistic programming. *J. Mach. Learn. Res.*, 20:28:1–28:6, 2019. URL http://jmlr.org/papers/v20/18-403.html. [p382]

A. Cabezas, A. Corenflos, J. Lao, and R. Louf. Blackjax: Composable Bayesian inference in JAX, 2024. URL https://arxiv.org/abs/2402.10797. [p382]

S. Chatterjee. *bnns: Bayesian neural network with 'Stan'*, 2025. URL https://CRAN.R-project.org/package=bnns. R package version 0.1.2. [p382]

İ. Çinar, M. Koklu, and Ş. Taşdemir. Classification of raisin grains using machine vision and artificial intelligence methods. *Gazi Mühendislik Bilimleri Dergisi*, 6(3):200–209, 2020. URL https://dx.doi.org/10.30855/gmbd.2020.03.03. [p392, 393]

T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018. URL https://arxiv.org/abs/1812.01718. [p393]

J. V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M. Hoffman, and R. A. Saurous. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017. URL https://arxiv.org/abs/1711.10604. [p382]

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real NVP. *arXiv preprint arXiv:1605.08803*, 2016. URL https://arxiv.org/abs/1605.08803. [p384]

S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth. Hybrid Monte Carlo. *Physics letters B*, 195(2): 216–222, 1987. URL https://doi.org/10.1016/0370-2693(87)91197-X. [p382]

S. Duffield, K. Donatella, J. Chiu, P. Klett, and D. Simpson. Scalable Bayesian learning with posteriors. *arXiv preprint arXiv:2406.00104*, 2024. URL https://arxiv.org/abs/2406.00104. [p382]

I. Esen, H. Arslan, G. Aktürk, Selin, N. Mervenur, Kültekin, and O. Özdemir. Gallstone. UCI Machine Learning Repository, 2024. URL https://doi.org/10.1097/md.0000000000037258. [p390]

D. Falbel and J. Luraschi. *torch: Tensors and neural networks with 'GPU' acceleration*, 2026. URL https://torch.mlverse.org/docs. R package version 0.17.0. [p382]

J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018. URL https://arxiv.org/abs/1803.03635. [p381]

M. D. Hoffman, A. Gelman, et al. The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014. URL http://jmlr.org/papers/v15/hoffman14a.html. [p382]

E. Høyheim, L. Skaaret-Lund, S. Sæbø, and A. Hubin. Explainable Bayesian deep learning through input-skip Latent Binary Bayesian Neural Networks. *Journal of Artificial Intelligence Research*, 86(16), 2026. URL https://doi.org/10.1613/jair.1.20433. [p381, 387]

A. Hubin and G. Storvik. Sparse Bayesian neural networks: Bridging model and parameter uncertainty through scalable variational inference. *Mathematics*, 12(6):788, 2024. URL https://doi.org/10.3390/math12060788. [p381]

M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999. URL https://doi.org/10.1023/A:1007665907178. [p382]

D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28, 2015. URL https://papers.nips.cc/paper/2015/hash/bc7316929fe1545bf0b98d114ee3ecb8-Abstract.html. [p381]

D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29, 2016. URL https://proceedings.neurips.cc/paper/2016/hash/ddeebdeefdb7e7e7a697e1c3e3d8ef54-Abstract.html. [p384]

F. Liang, Q. Li, and L. Zhou. Bayesian neural networks for selection of drug sensitive genes. *Journal of the American Statistical Association*, 113(523):955–972, 2018. URL https://doi.org/10.1080/01621459.2017.1409122. [p382]

A. McInerney and K. Burke. A statistical modelling approach to feedforward neural network model selection. *Statistical Modelling*, 25(4):323–342, 2025. URL https://doi.org/10.1177/1471082X241258261. [p382]

R. M. Neal. *Bayesian Learning for Neural Networks*. PhD thesis, University of Toronto, Toronto, Canada, 1995. URL http://www.cs.toronto.edu/pub/radford/thesis.pdf. [p382]

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Jiang, S. Chintala, and A. Lerer. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html. [p382]

D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015. URL https://proceedings.mlr.press/v37/rezende15.html. [p381]

J. Robnik and U. Seljak. Fluctuation without dissipation: Microcanonical langevin Monte Carlo. *arXiv preprint arXiv:2303.18221*, 2023. URL https://arxiv.org/abs/2303.18221. [p382]

T. Sharaf, T. Williams, A. Chehade, and K. Pokhrel. BLNN: An R package for training neural networks using Bayesian inference. *SoftwareX*, 11:100432, 2020. URL https://doi.org/10.1016/j.softx.2020.100432. [p382]

Y. Shen, N. Daheim, B. Cong, P. Nickl, G. M. Marconi, C. Bazan, R. Yokota, I. Gurevych, D. Cremers, M. E. Khan, et al. Variational learning is effective for large deep networks. *arXiv preprint arXiv:2402.17641*, 2024. URL https://arxiv.org/abs/2402.17641. [p382]

L. Skaaret-Lund, G. Storvik, and A. Hubin. Sparsifying Bayesian neural networks with latent binary variables and normalizing flows. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=d6kqUKzG3V. [p381, 384, 397]

E. Sommer, J. Robnik, G. Nozadze, U. Seljak, and D. Rügamer. Microcanonical Langevin ensembles: Advancing the sampling of Bayesian neural networks, 2025. URL https://arxiv.org/abs/2502.06335. [p382]

Steven L. Scott. *BoomSpikeSlab: MCMC for Spike and Slab Regression*, 2025. URL https://CRAN.R-project.org/package=BoomSpikeSlab. R package version 1.2.7. [p382]

K. Ushey, J. Allaire, and Y. Tang. *reticulate: Interface to 'Python'*, 2026. URL https://rstudio.github.io/reticulate/. R package version 1.46.0. [p382]

E. Wegner. *BayesFluxR: Implementation of Bayesian neural networks*, 2023. URL https://CRAN.R-project.org/package=BayesFluxR. R package version 0.1.3. [p382]

*Lars Skaaret-Lund*
*KBM, Bioinformatics and applied statistics (BIAS), Norwegian University of Life Sciences*
*Elizabeth Stephansens v. 15, 1433 Ås*
*Norway*
lars.skaaret-lund@nmbu.no

*Eirik Høyheim*
*KBM, Bioinformatics and applied statistics (BIAS), Norwegian University of Life Sciences*
*Elizabeth Stephansens v. 15, 1433 Ås*
*Norway*
eirik.hoyheim@ffi.no

*Aliaksandr Hubin*
*KBM, Bioinformatics and applied statistics (BIAS), Norwegian University of Life Sciences*
*Elizabeth Stephansens v. 15, 1433 Ås*
*Norway*
aliaksandr.hubin@nmbu.no

# demofit: A Unified Framework for Mortality Modelling Integrating Parametric Laws and Stochastic Models

*by Jackie Li*

**Abstract** Mortality modelling and forecasting are widely used in actuarial science and demography, with numerous parametric mortality laws and stochastic mortality models proposed to describe age-specific patterns and temporal dynamics. Existing software implementations, however, typically focus on particular model classes and do not provide a unified framework for combining these approaches. We present the R package demofit, which integrates parametric mortality laws and stochastic mortality models within a coherent and user-friendly interface. The package includes a broad collection of mortality laws, together with widely used forecasting models such as the Lee-Carter, Cairns-Blake-Dowd, and spatial-temporal vector autoregressive frameworks. Model estimation is performed using specific optimisation routines to enhance robustness, and forecasts can be further regularised using parametric mortality curves to produce smooth and interpretable age profiles. We describe the underlying framework and illustrate the main functions of the package using real mortality data, demonstrating its application in model fitting, forecasting, and analysis.

## 1 Introduction

Over the past century, human life expectancy has trended up persistently. For instance, in Australia, (period) life expectancy rose from 60.97 years in 1921 to 83.47 years in 2021, and in New Zealand it increased from 68.81 years in 1948 to 82.30 years in 2021. These sustained gains, driven by continual improvements in public health, medical care, and living conditions, represent one of the most remarkable demographic shifts in modern history. However, longer lifespans also have substantial socioeconomic implications, including population ageing and increasing pressure on pension and healthcare systems. Consequently, the accurate modelling and forecasting of mortality and longevity are of utmost importance for policymakers, demographers, and actuaries in assessing the economic and social impact of longevity.

Since the influential contributions of Lee and Carter (1992) and Cairns et al. (2006), mortality modelling and forecasting have developed into a substantial and active field of research. The Lee-Carter (LC) family of models generally incorporates an age-specific component for capturing the age schedule and a time-varying component reflecting overall mortality improvements over time. By contrast, the Cairns-Blake-Dowd (CBD) family focuses on older ages and incorporates period effects linked to age in a more parsimonious structure. Their subsequent extensions integrate additional terms like cohort effects across birth cohorts. A wide range of these factor-based models has been proposed, with comprehensive reviews conducted by Cairns et al. (2009) and Haberman and Renshaw (2011). More recently, Li and Lu (2017) introduced the spatial-temporal vector autoregressive (STAR) model, imposing structured sparsity to achieve parsimony. Li et al. (2024) provided a review of its subsequent extensions, which further expand the methodological landscape of mortality forecasting.

In this paper, we introduce the **demofit** R package (Li (2026)), which provides a unified methodological and computational framework for mortality modelling by integrating parametric mortality laws and stochastic mortality models within a coherent interface. The package includes a collection of practical mortality curves with both constrained and unconstrained parameterisations, where the former offer greater interpretability and the latter provide increased flexibility in fitting complex mortality patterns. To enhance robustness, model estimation is performed using multiple optimisation routines, with the final selection based on minimising a least squares criterion. Moreover, the **demofit** package incorporates three major families of stochastic mortality models, namely the LC, CBD, and vector autoregressive (VAR) frameworks, and also ensemble and multi-population features, together with their estimation and forecasting procedures. A key feature of the package is that the forecasts produced from these stochastic models can further be guided by a selected mortality curve, which helps produce more realistic age patterns, particularly for small or noisy populations encountered in practical applications such as insurance pricing. The overall design emphasises a unified workflow and ease of use for empirical research, actuarial practice, and teaching purposes.

In the next section, we present the proposed framework and outline the list of parametric mortality laws and stochastic mortality models included. We then introduce the **demofit** package and compare it with a few related existing R packages. Afterwards, we apply the **demofit** package to specific datasets for illustration. Finally, we conclude with other remarks and a discussion of possible future

developments.

## 2 Mortality modelling

This section provides an overview of the parametric mortality laws and stochastic mortality models included in the **demofit** package. For the parametric mortality curves, a unified methodological and computational framework is adopted for model fitting, enabling consistent estimation and comparison across different specifications. These mortality curves are chosen to cover a broad range of modelling needs, including different age ranges, levels of flexibility, and practical applications, thereby providing users with a versatile set of tools for mortality analysis.

Let $m_x$ be the central death rate at age $x$. Users can choose one of the following fourteen mortality curves, with their indicative age ranges (or subsets thereof) stated in the following brackets (see Gompertz (1825), Makeham (1860), Oppermann (1870), Thiele (1871), Wittstein and Bumsted (1883), Perks (1932), Weibull (1951)[1], Beard (1971), Heligman and Pollard (1980), Rogers and Planck (1983), Siler (1983), Martinelle (1987), Thatcher (1999), and Tabeau (2001)[2]):

| | | |
|---|---|---|
| Gompertz 1825 | $m_x = Be^{Cx}$ | (30–90) |
| Makeham 1860 | $m_x = A + Be^{Cx}$ | (20–90) |
| Oppermann 1870 | $m_x = \frac{A}{\sqrt{x+1}} + B + C\sqrt{x+1}$ | (0–20) |
| Thiele 1871 | $m_x = A_1 e^{-B_1 x} + A_2 e^{-0.5B_2(x-C)^2} + A_3 e^{B_3 x}$ | (0–90) |
| Wittstein and Bumsted 1883 | $m_x = \frac{A^{-(Bx)^N}}{B} + A^{-(M-x)^N}$ | (0–90) |
| Perks 1932 | $m_x = \frac{A+BC^x}{1+DC^x}$ | (20–100+) |
| Weibull 1939 | $m_x = Bx^C$ | (50–100+) |
| Van der Maen 1943 | $m_x = A + Bx + Cx^2 + \frac{I}{N-x}$ | (80–100+) |
| Beard 1971 | $m_x = \frac{Ae^{Cx}}{1+Be^{Cx}}$ | (50–100+) |
| Heligman and Pollard 1980 | $m_x = A^{(x+B)^C} + De^{-E(\ln(x)-\ln(F))^2} + \frac{GH^x}{1+GH^x}$ | (0–100+) |
| Rogers and Planck 1983 | $m_x = A_0 + A_1 e^{-Ax} + A_2 e^{-B(x-U)-e^{-C(x-U)}} + A_3 e^{Dx}$ | (0–100+) |
| Siler 1983 | $m_x = A_1 e^{-B_1 x} + A_2 + A_3 e^{B_3 x}$ | (0–90) |
| Martinelle 1987 | $m_x = \frac{A+Be^{Cx}}{1+De^{Cx}} + Ee^{Cx}$ | (20–100+) |
| Thatcher 1999 | $m_x = A + \frac{Be^{Cx}}{1+De^{Cx}}$ | (20–100+) |

For consistency within the unified framework, all the mortality curves are expressed in terms of the central death rate $m_x$. Under the common assumption that the force of mortality is constant within each age interval (e.g. Li (2013)), the force of mortality $\mu_x$ coincides with the central death rate $m_x$, allowing those mortality curves originally formulated in terms of $\mu_x$ to be represented equivalently. From a practical perspective, $m_x$ can be readily estimated from observed annual data as the ratio of deaths to mid-year exposure (e.g. HMD (2026)), making it a natural choice for empirical implementation. In addition, empirically, for most ages, the one-year death probability $q_x$ satisfies $q_x \approx 1 - e^{-\mu_x} = 1 - e^{-m_x} \approx m_x$, which justifies expressing the mortality curves originally defined in terms of $q_x$ using $m_x$ instead. At very old ages like 100+, however, there is conflicting empirical evidence on how $\mu_x$ behaves with age (increasing, decreasing, or flattening), and at such extremely old ages, the numerical values of $q_x$ and $m_x$ can become quite different. For instance, at around age 100, $m_x$ often reaches 0.5 for many populations, for which the corresponding $q_x$ is only around 0.4. Consequently, the exact formulas $q_x = \int_0^1 {}_t p_x \, \mu_{x+t} \, dt$ and $m_x = \frac{q_x}{\int_0^1 {}_t p_x \, dt}$ need to be used. Nevertheless, those mortality curves originally formulated in terms of $q_x$ are generally much less suitable for capturing these centenarian mortality patterns anyway, and alternative formulations defined directly on $\mu_x$ or $m_x$, such as the logistic curve in Thatcher (1999), are more appropriate.

For the stochastic mortality models, a unified computational structure is similarly adopted to ensure consistency in model specification, estimation, and forecasting. These stochastic mortality models are selected to capture a wide range of dynamic mortality patterns, including different age structures, temporal behaviours, and cohort patterns, offering users a flexible and coherent set of tools for mortality forecasting. This coherent approach facilitates systematic comparison across different forecasting models and enables their integration with the parametric mortality curves within a consistent modelling framework.

Now let $m_{x,t}$ be the central death rate at age $x$ in year $t$, extended with the time dimension. Users

[1]The Weibull curve was popularised through Weibull (1951).
[2]The Van der Maen curve is extracted from Tabeau (2001).

can select one of the seven forecasting models below:

| | |
|---|---|
| Lee-Carter (LC) | $\ln(m_{x,t}) = \alpha_x + \beta_x\kappa_t + \epsilon_{x,t}$ |
| Renshaw-Haberman (RH) | $\ln(m_{x,t}) = \alpha_x + \beta_x\kappa_t + \gamma_{t-x} + \epsilon_{x,t}$ |
| Age-Period-Cohort (APC) | $\ln(m_{x,t}) = \alpha_x + \kappa_t + \gamma_{t-x} + \epsilon_{x,t}$ |
| CBD (M5) | $\ln(m_{x,t}) = \kappa_{1,t} + \kappa_{2,t}(x - \bar{x}) + \epsilon_{x,t}$ |
| CBD with Cohort (M6) | $\ln(m_{x,t}) = \kappa_{1,t} + \kappa_{2,t}(x - \bar{x}) + \gamma_{t-x} + \epsilon_{x,t}$ |
| CBD with Curvature and Cohort (M7) | $\ln(m_{x,t}) = \kappa_{1,t} + \kappa_{2,t}(x - \bar{x}) + \kappa_{3,t}((x - \bar{x})^2 - \sigma^2) + \gamma_{t-x} + \epsilon_{x,t}$ |
| Spatial-Temporal VAR (STAR) | $\ln(M_t) = \mu + R\ln(M_{t-1}) + E_t$ |

For consistency and similar reasons as above, all the forecasting models are also expressed in terms of the central death rate $m_{x,t}$, where $M_t$ is the vector of central death rates in year $t$. For the first three models, $\alpha_x$ refers to the age effect, $\kappa_t$ allows for the period effect with sensitivity $\beta_x$, and $\gamma_{t-x}$ represents the cohort effect. The constraints imposed include $\sum_x \beta_x = 1$, $\sum_t \kappa_t = 0$, and $\sum_c \gamma_c = 0$. For the CBD models, $\kappa_{1,t}$ is the intercept parameter, $\kappa_{2,t}$ is the slope parameter, $\kappa_{3,t}$ is the curvature parameter, $\bar{x}$ is the average age of the age range, $\sigma^2$ is the average value of $(x - \bar{x})^2$, and $\gamma_{t-x}$ is the cohort effect with $\sum_c c^i\gamma_c = 0$ for $i = 0, 1, 2$. For the STAR model, $R$ is a banded matrix with non-zero entries in the main diagonal and the two subdiagonals below that. The error terms are $\epsilon_{x,t}$ and $E_t$. Note that the LC, RH, APC, and STAR models can be applied to the whole age range or subsets of it, while the three CBD models are designed mainly for ages 50-90. More details of these forecasting models can be found in Lee and Carter (1992), Bray (2002), Cairns et al. (2006), Cairns et al. (2009), Haberman and Renshaw (2011), Li and Lu (2017), and Li et al. (2024).

Within our integrated framework, the forecasts generated from the stochastic mortality models can be further regularised flexibly through a selected parametric mortality curve. Although the stochastic models effectively capture temporal dynamics, the resulting forecasted age profiles may exhibit irregular, non-smooth, or demographically unexplainable patterns across ages, particularly for small or noisy populations, in the presence of cohort parameters, or when the forecasts are anchored to the observed values in the final year of the data. Such features can pose challenges for practical implementation, such as insurance pricing and communication with stakeholders. By further fitting a parametric curve to the forecasted rates for each projection year, the age patterns become smoother and more structured, while remaining consistent with the underlying temporal evolution. This pragmatic approach facilitates interpretation and implementation by expressing the forecasts in a functional form familiar to insurance practitioners and demographers, while maintaining coherence with the stochastic modelling framework. Note that the CBD (M5) model does not have this issue though, as it is by nature a simple linear curve fitted to each year. In this aspect, McNown and Rogers (1989) proposed a more natural approach of directly projecting the parameters of a mortality curve. As commented in Li and Liu (2019), however, estimating and forecasting too many parameters of a mortality curve (e.g. eight parameters in the Heligman-Pollard curve) can lead to a biologically unreasonable age schedule, despite the mathematical smoothness of the forecasts.

## 3 Package demofit

The **demofit** package provides a unified interface for fitting mortality curves and forecasting mortality rates within a coherent workflow. For mortality curve fitting, the core function `MC()` (MC denoting "mortality curve") fits a selected mortality curve to observed mortality rates by picking the solution with the smallest weighted sum of squares value on the log death rates. The package includes both constrained and unconstrained parameterisations of the mortality curves. The former is in line with the traditional designs and provides greater interpretability, while the latter offers increased flexibility for capturing more complex age patterns. To enhance robustness and numerical stability, multiple optimisation strategies are employed simultaneously, including the PORT routines, Nelder-Mead method, and Levenberg-Marquardt algorithm, with the final estimates chosen based on the best fit among the three methods. The fitted curve is returned as an S3 object with associated methods for obtaining parameter estimates (`coef`), fitted values (`fitted`), interpolated / extrapolated values (`predict`), weighted sum of squares (`deviance`), residuals (`residuals`), and graphical display (`plot`).

This multi-strategy approach differs from the existing **MortalityLaws** package (Pascariu (2025)), where optimisation is carried out using a single selected routine or loss function. By contrast, **demofit** applies several optimisation methods in parallel and selects the best-performing solution according to a common fitting criterion, thereby improving robustness to starting values, mitigating convergence issues, and producing more stable results for data of different calendar years and populations over a wide range of mortality curves. (See the next section for examples.) In addition, the function `CC()` is provided for evaluating and constructing a mortality curve given a set of parameters, allowing users to directly generate the fitted mortality schedule.

For stochastic mortality modelling, the **demofit** package provides the function `FCS()` (FC standing for "forecasting" and S for "smoothed") to fit a range of forecasting models within the same framework. All the models are formulated in terms of the log death rate, and estimation is generally carried out by minimising the sum of squared residuals (i.e. Gaussian likelihood). For computational efficiency, the LC model is estimated via singular value decomposition (SVD)[3] (e.g. Renshaw and Haberman (2006)), and the CBD (M5) and STAR models are fitted using regression-based methods (via OLS, SQP). For the other more complex specifications, parameter estimation is performed using an iterative Newton updating scheme (e.g. Renshaw and Haberman (2006), Li (2013)). After model fitting, the period and cohort parameters ($\kappa_t$, $\kappa_{1,t}$, $\kappa_{2,t}$, $\kappa_{3,t}$, $\gamma_c$) are forecasted using the function `auto.arima()` in the **forecast** package (Hyndman et al. (2026)). Moreover, there is a `jumpoff` argument where users can make the forecasts from the first six models anchored to the observed values in the final year of the data and so by nature more in line with the STAR forecasts. (As noted in Lee and Miller (2001), an adjustment to the correct jump-off level can give a more accurate life expectancy forecast and remove the bias from simply using the fitted values.) These settings ensure consistency in model fitting and facilitate comparison between different forecasts. The fitted model is returned as an S3 object with associated methods for extracting parameter estimates (`coef`), forecasts (`forecast::forecast`), standardised residuals (`residuals`), and graphical plots (`plot`).

In comparison, the **StMoMo** package (Villegas et al. (2025)) adopts a likelihood-based framework in which its mortality models are fitted assuming Poisson or Binomial distributions for deaths, with estimation carried out via generalised nonlinear models implemented in the gnm package (Turner et al. (2025)). The **demofit** package instead adopts a least squares formulation[4] directly on the log death rates, as a default, providing a unified, numerically stable, and computationally efficient estimation approach across different model specifications. This facilitates systematic comparison and the integration of mortality curves with stochastic mortality models. Moreover, two features further extend the scope of the **demofit** package. First, the inclusion of the STAR model, the ensemble and multi-population features, and the option to regularise the forecasts as a post-processing step using different mortality curves expand the range of modelling approaches available within a coherent computational framework. (See the next section for illustrations.) Second, the **demofit** package provides tailored functions, including `FCS()` (FC for "forecasting" and S for "smoothed"), `LCS()` (LC model), `RHS()` (RH model), `APCS()` (APC model), `CBDS()` (CBD model), `CBDCS()` (CBD with cohort model), `CBDQCS()` (CBD with curvature and cohort model), `STARS()` (STAR model), `ENS()` (ensemble), `CFMS()` (common factor model), and `CAES()` (common age effect model), which are designed for ease of use in practical applications without the need for extensive configuration.

Overall, the **demofit** package provides a useful addition to the relatively small list of packages for mortality modelling in R. While the two packages **MortalityLaws** and **StMoMo** offer well-established implementations for specific modelling approaches, the range of available R tools remains rather limited compared to other areas of statistical modelling. By integrating parametric mortality laws and stochastic mortality models within a coherent and user-friendly framework, **demofit** complements the existing packages and broadens the set of practical tools available for actuarial and demographic applications.

Besides these frequentist approaches, recent Bayesian packages such as **StanMoMo** (Barigou and Goffard (2023)) and **BayesMortalityPlus** (Figueiredo et al. (2025)) provide complementary frameworks for mortality modelling. The **StanMoMo** package implements Bayesian versions of several stochastic mortality models, including the LC, RH, APC, CBD, and M6 models, together with Bayesian model averaging and uncertainty quantification. The **BayesMortalityPlus** package focuses on Bayesian mortality graduation (smoothing) using the Heligman-Pollard and dynamic linear models, together with Bayesian LC (only) mortality forecasting. It also provides mortality closure methods for advanced ages. In contrast, the **demofit** package is designed as a unified methodological and computational framework spanning both parametric mortality curves and stochastic mortality models under a common interface, with an emphasis on practical implementation, systematic model comparison, and computational efficiency instead of Bayesian inference. It additionally provides multi-population modelling and forecast regularisation using mortality curves, thereby complementing the existing Bayesian packages with a wider methodological scope for practical applications.

[3] As noted in Renshaw and Haberman (2006), for the LC model, the maximum likelihood estimates from the iterative Newton updating scheme under the Gaussian error structure are numerically the same as the SVD estimates.

[4] For applications where exposure information is available and likelihood-based estimation is preferred, **demofit** also provides Poisson estimation for selected models.

**Table 1:** Average reductions (in \%) in sum of squares on log death rates of fitting mortality curves to Australian and New Zealand data across 1950–2021 using demofit compared to MortalityLaws

| Curve | Australia F | Australia M | New Zealand F | New Zealand M |
|---|---|---|---|---|
| Gompertz | 0.00 (0.00) | 0.00 (0.00) | 0.00 (0.00) | 0.00 (0.00) |
| Makeham | -0.00 (-0.00) | -0.00 (-0.00) | -0.00 (-0.00) | -0.00 (-0.00) |
| Oppermann | 0.00 (-0.00) | -0.00 (-0.00) | -0.00 (-0.00) | -0.00 (-0.00) |
| Thiele | 21.15 (24.17) | 8.91 (8.99) | 18.63 (19.66) | 12.91 (12.91) |
| Wittstein-Bumsted | 2.84 (2.84) | 0.42 (0.42) | 0.61 (0.61) | 0.24 (0.24) |
| Perks | -22.33 (8.35) | 0.01 (4.71) | -2.81 (3.07) | 0.20 (0.92) |
| Weibull | 0.00 (0.00) | 0.00 (0.00) | 0.00 (0.00) | 0.00 (0.00) |
| Van der Maen | 1.48 (11.57) | -21.45 (15.30) | 14.77 (17.27) | 17.98 (21.63) |
| Beard | -0.23 (45.69) | 0.00 (6.58) | 0.00 (9.86) | -0.00 (4.16) |
| Heligman-Pollard | 0.00 (0.00) | 0.00 (0.00) | 0.21 (0.23) | 0.00 (0.00) |
| Rogers-Planck | 24.46 (36.54) | 62.75 (64.16) | 21.19 (22.96) | 50.12 (49.95) |
| Siler | 0.64 (0.64) | 0.52 (0.58) | 0.29 (0.29) | -0.00 (0.09) |
| Martinelle | 2.42 (4.84) | 32.30 (54.44) | 2.59 (7.36) | 23.55 (35.84) |

## 4 Examples

In this section, we first illustrate the use of the function `MC()` and compare it numerically with the corresponding fitting function in **MortalityLaws**. The fitted curve under `MC()` is stored as an S3 object with associated methods including `coef()` for parameter estimates, `fitted()` for fitted values, `predict()` for interpolation / extrapolation, `deviance()` for weighted sum of squares, `residuals()` for residuals, and `plot()` for graphical display. We obtain Australian and New Zealand mortality data from the Human Mortality Database (HMD (2026)) for the period 1950-2021 and ages 0-100. We follow Lee and Miller (2001) and base the analysis on data since 1950 so as to avoid the major structural changes in the distant past. As an example, we use the following code to load the packages, fit the Rogers-Planck curve (with the usual parameter constraints) to the death rates of a particular calendar year, and compute the fitting errors. The same procedure applies to the other mortality curves in both packages.

```
library(demofit)
library(MortalityLaws)
x <- 0:100
fit1 <- MC(x=x, m=m, curve="rogersplanck2")
err1 <- deviance(fit1)
fit2 <- MortalityLaw(x=x, mx=m, law="rogersplanck")
err2 <- sum((log(m) - log(fitted(fit2)))^2)
```

Table 1 presents the average reductions (in %) in the errors of fitting thirteen mortality curves to the four populations across the data period. We do not include the Thatcher logistic curve (for centenarian mortality) as it is available in **demofit** only. Note that the values shown in parentheses correspond to the reductions obtained using unconstrained parameterisations instead. There are a number of interesting observations. First, for the mortality curves with only two or three parameters, both packages basically give rise to the same fitting errors. But for the other mortality curves with more than three parameters, **demofit** generates noticeable reductions in the fitting errors. These improvements are primarily driven by the multi-optimisation strategy, which reduces sensitivity to starting values and helps avoid local minima for more complex mortality curves, hence providing more robust estimation. (The only exception is the Heligman-Pollard curve, where MortalityLaws may have adopted special starting values or a tailored solution.) Second, using unconstrained parameterisations generally provides further reductions in the fitting errors, reflecting their greater flexibility in capturing detailed age patterns. Figure 1 illustrates the fitted Rogers-Planck curves for Australian female death rates in 2020 from different methods. It can be seen that **demofit**'s unconstrained parameterisation provides the best fit to the whole age range. The next is its constrained parameterisation version, which appears to miss some of the age patterns from around age 70 onward. It is then followed by the constrained version from **MortalityLaws**, where the age patterns both below age 20 and above age 70 are not adequately captured by comparison. (Their fitting errors are 7.97, 9.71, and 14.42, respectively.)

Furthermore, we demonstrate how the Thatcher curve (available in **demofit** only) is more suitable for centenarian mortality than the other traditional mortality curves. We collect high-quality old-age

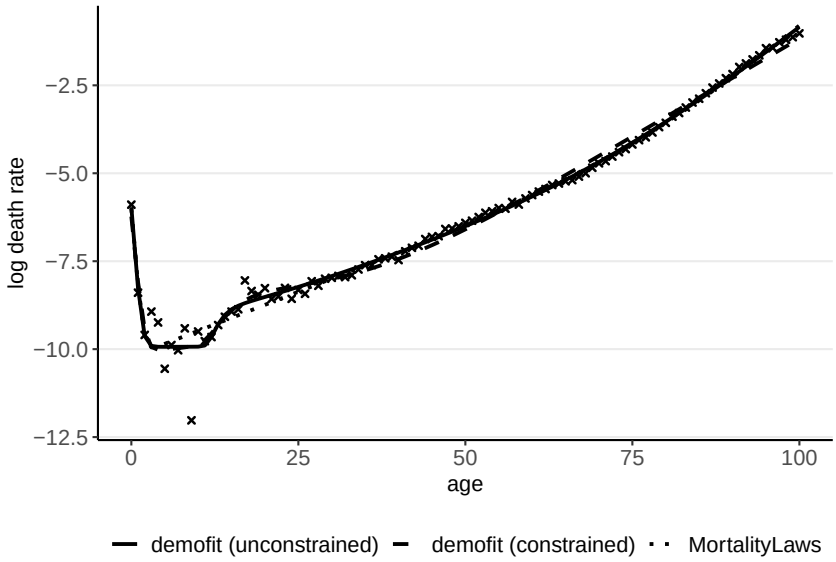



**Figure 1:** Fitted mortality curves (Rogers-Planck) from different methods for Australian female death rates in 2020. (Crosses are actual observations.)

data of Sweden from the Kannisto-Thatcher Database (KTD (2018)). Sweden lies in the best data quality group in that database as it has a very long history of national population registers. Since the data at extreme ages are sparse, we aggregate the deaths and exposures over 1991-2010 and calculate the corresponding death rates at ages 80-105. Figure 2 displays the fitted Thatcher, Weibull, and Gompertz curves (constrained) for Swedish female death rates. It is visually clear that the Thatcher curve is able to capture the flattening of the log death rates at extreme ages, while the traditional Weibull and Gompertz curves, having power and exponential growth, fail to do so. (Their fitting errors are 0.00, 0.17, and 0.30, respectively.) In practical applications, when old-age data are scarce, one may fit the Thatcher curve to the age range where data are relatively more reliable and then use the fitted curve for mortality closure by extrapolating the death rates to advanced ages. Figure 2 also illustrates the extrapolated death rates up to age 120 (via `predict()`). The flattening of the Thatcher curve here is consistent with the broader evidence reported by Thatcher (1999), where the conclusions were drawn from pooled mortality data across 13 industrialised countries, analysed both collectively and for different periods and birth cohorts, thereby reducing the uncertainty associated with sparse observations at the oldest ages.

We then demonstrate the use of the functions `LCS()`, `RHS()`, `APCS()`, `CBDS()`, `CBDCS()`, `CBDQCS()`, and `STARS()`[5] and examine their estimated parameters and forecasts. The fitted models are returned as S3 objects with associated methods including `coef()` for parameter estimates, `forecast::forecast()` for smoothed forecasts, `residuals()` for standardised residuals, and `plot()` for graphical plots. We continue to use the HMD data for the more recent period 1980-2021 and ages 0-90. As an illustration, we apply the code below to fit the LC, RH, APC, and STAR models to the death rates of a particular population, where the forecasts are regularised by the Heligman-Pollard curve (constrained), which is suitable for the whole age range. (Note that the raw forecasts, without adjustment, can still be obtained via, for instance, `f1$forecast`.) The same process applies to the CBD (M5, M6, M7) models, but the age range has to be restricted to ages 50-90 only. Figure 3 shows the forecasted age profiles in 2051 under the four models for New Zealand female death rates. First, the models differ in the projected shape of mortality at young adult ages, with the "accident hump" around age 20 being more pronounced under the LC and APC models. Second, noticeable differences emerge in the projected old-age mortality improvements, where the four models imply different rates of decline and hence different future age profiles. Third, while all four models preserve the overall age pattern of mortality, the flexibility introduced by the cohort parameters in the RH and APC models and the age-varying improvement structure in the STAR model leads to visibly different forecast trajectories compared with the classical LC model. Figure 4 compares the raw and smoothed forecasted death rates in 2051 under the APC model. For a small population like New Zealand, there is a relatively high level of noise in the data, which can be propagated into the forecasts, especially when the forecasts are anchored to the last observed year. The irregular and biologically unreasonable age patterns appear to be substantially alleviated by the selected Heligman-Pollard curve, which is familiar to actuaries and practitioners

[5]The last letter "S" refers to the feature that the mortality forecasts can be "smoothed".

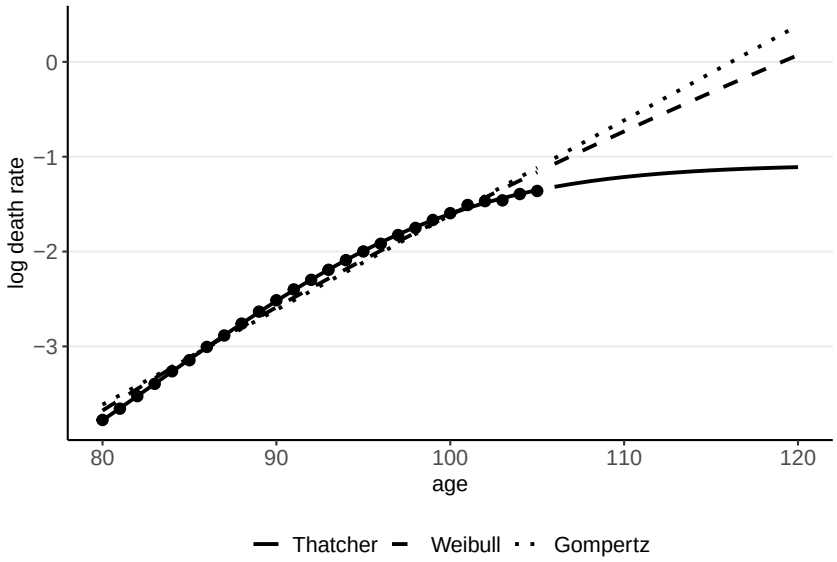



**Figure 2:** Fitted and extrapolated mortality curves (Thatcher, Weibull, Gompertz) for Swedish female death rates in 1991-2010. (Dots are actual observations.)

and is helpful for insurance product pricing and communication purposes. Note that the number of policyholders in a typical insurance portfolio is generally much smaller than the population size of a country and so the issue of forecasting irregular mortality schedules can be more severe in practical valuation work. More specifically, mortality schedules are often used directly in insurance pricing and valuation, where irregular age-to-age fluctuations can translate into unusual patterns in insurance premiums or other quantities derived from mortality rates. Such patterns may be difficult to justify from a demographic perspective and can also complicate the communication of insurance products to prospective customers. The regularised forecasts can therefore provide a more sensible and familiar mortality schedule for subsequent actuarial applications (Atkinson (2018)), without requiring the underlying stochastic mortality model to be replaced.

```
x <- 0:90
f1 <- LCS(x=x, M=M, curve="heligmanpollard2", h=30, jumpoff=2)
f2 <- RHS(x=x, M=M, curve="heligmanpollard2", h=30, jumpoff=2)
f3 <- APCS(x=x, M=M, curve="heligmanpollard2", h=30, jumpoff=2)
f7 <- STARS(x=x, M=M, curve="heligmanpollard2", h=30)
```

We now reduce the age range to 50-90 so as to fit all the seven forecasting models to Australian male data using both **demofit** and **StMoMo** (with the binomial assumption therein). Figure 5 plots the parameter estimates with respect to their corresponding dimensions under different models. The parameter estimates under the LC and APC models (first and third panels) are generally close between the two packages. The differences under the CBD (M5, M6, M7) models (last three panels) are relatively more apparent, though the overall age / temporal / cohort patterns are still broadly in line between the two packages. These differences can be attributed mainly to the distributional assumptions and estimation algorithms of the two packages. The major differences lie in the RH model outputs (second panel), where the classical linearity in overall mortality decline (i.e. $\kappa_t$) is preserved by **demofit** even after the inclusion of the cohort effect, while this overall temporal effect appears to be "absorbed" into the cohort parameters produced by **StMoMo**. It is well known that separating age, period, and cohort effects is inherently challenging due to identification and computational issues, and different modelling and estimation choices may lead to alternative but statistically equivalent representations. In this context, **demofit** maintains the cohort effect as a secondary component while preserving the classical Lee-Carter-style temporal structure[6], which has important implications for the stability, interpretability, and extrapolation behaviour of mortality forecasts. From a forecasting perspective, this decomposition avoids conflating long-term mortality improvement with cohort-specific effects, allowing the projected period trend to remain more stable and less sensitive to cohort fluctuations.

[6] **demofit** initialises the RH model from the classical LC decomposition before optimisation, which empirically tends to retain the principal period trend while allowing the cohort effect to capture residual cohort-specific variation.

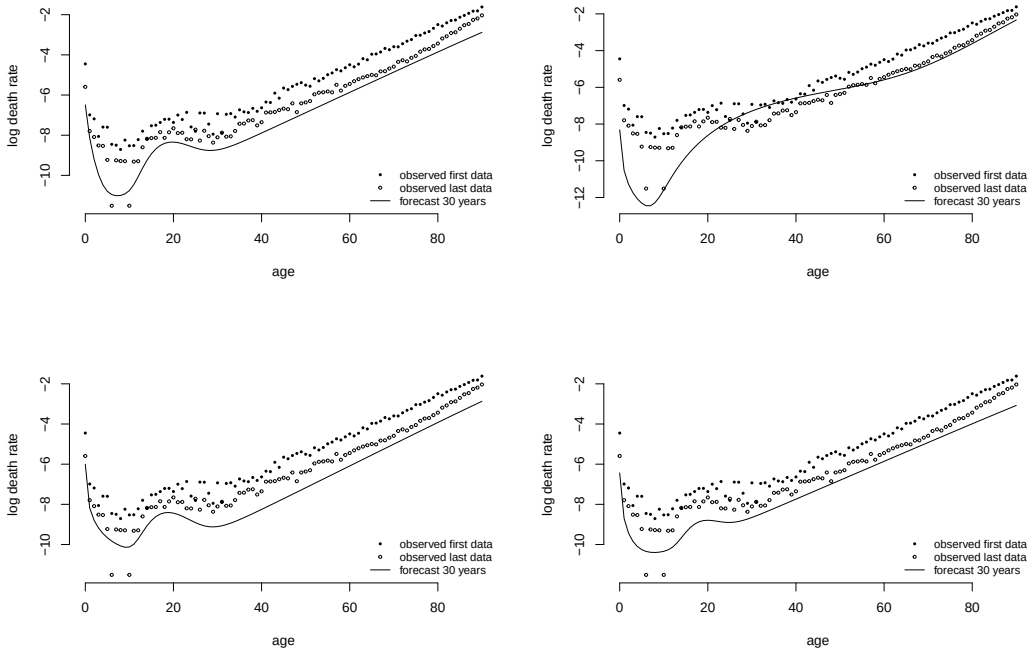



**Figure 3:** Mortality forecasts in 2051 (top left panel – LC; top right – RH; bottom left – APC; bottom right – STAR) using New Zealand female data of years 1980-2021 and ages 0-90.

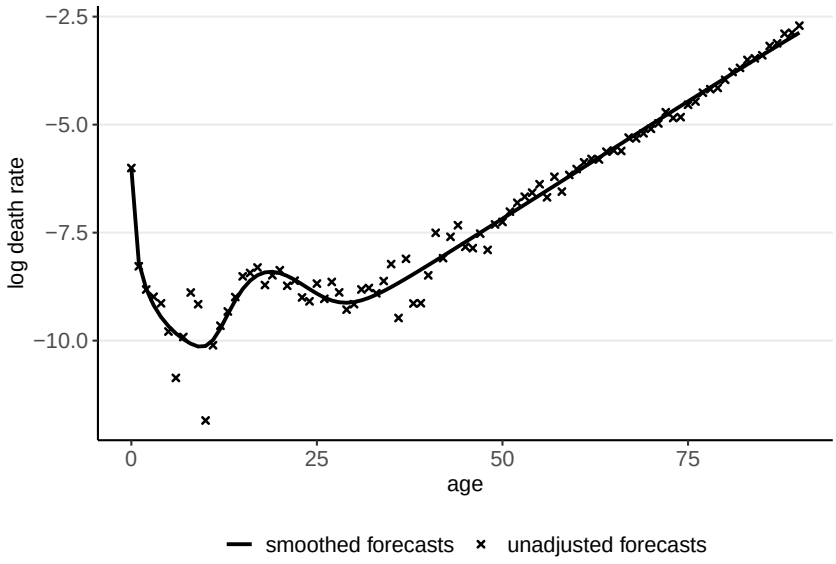



**Figure 4:** Smoothed and unadjusted forecasted death rates in 2051 for New Zealand females under APC model.

In addition, compared to the above six forecasting models, the STAR model (available in **demofit** only) has a distinctive feature in ensuring the age coherence property, whereby the ratio of forecasted death rates between any two ages converges to a constant over time. Figure 6 compares the ratios of forecasted death rates from 2022 to 2051 between the STAR and LC models. Under the STAR model, the forecasted ratios gradually converge, implying that the projected death rates across ages remain proportionally stable in the long term. By contrast, under the LC model, the forecasted ratios continue to increase or decrease over time, leading to persistent and growing divergence between ages. Such behaviour would be biologically unreasonable and inconsistent with the general expectation of stable relative mortality patterns across ages in the long term. Note that this long-run age coherence property is implied by the fitted STAR model specification, while the short- and medium-term forecast trajectories remain data-driven through the estimated model parameters and the latest trends in the data.

Moreover, we present the function `CFMS()`, which fits the common factor model proposed by Li and Lee (2005). The model specification is $\ln(m_{x,t,i}) = \alpha_{x,i} + B_x K_t + \beta_{x,i}\kappa_{t,i} + \epsilon_{x,t,i}$, where $B_x K_t$ is the common factor for two populations and $\beta_{x,i}\kappa_{t,i}$ is the specific factor for population $i$. The existing R packages for mortality modelling are primarily designed for single-population analysis and generally do not provide built-in support for multi-population modelling. We apply `CFMS()` to New Zealand female and male data (treated as population 1 and population 2, respectively) of years 1980-2021 and ages 50-90. As noted in Li (2013), under suitable time series assumptions, the ratio of forecasted death rates at each age between two related populations converges to a constant over time. Figure 7 shows how the ratios of forecasted death rates for both sexes at different ages converge under the common factor model. If the two populations are modelled and forecasted separately instead, this ratio is likely to increase or decrease indefinitely. The common factor model setting here ensures that there is a biologically sensible long-term relationship between female and male mortality. Similarly, this long-term convergence is a structural property of the common factor model under the assumed time series dynamics, whereas the estimated population-specific factors and the latest trends in the data determine the short-term forecasting behaviour.

```
f9 <- CFMS(x=x, M1=M1, M2=M2, curve="makeham2", h=30, jumpoff=2)
```

Finally, we briefly introduce the function `ENS()`, which performs weighted averaging over the forecasts from the seven models. Some recent mortality studies (e.g. Li (2023)) suggest that this kind of ensemble or stacking approach can lead to enhanced forecasting accuracy, particularly in applications where forecasting accuracy is prioritised over model interpretability (i.e. age schedule, overall trend, cohort pattern, etc). As a backtesting exercise for illustration, we apply this function to New Zealand female and male data of years 1980-2011 and obtain both the unadjusted and smoothed forecasts for years 2012-2021. Table 2 reports the mean absolute errors of the forecasts under different forecasting models and the ensemble approach (with equal weights). There are a number of interesting observations. First, for both populations, the ensemble approach yields the lowest errors when the forecasts are unadjusted, followed by the STAR model. But when the forecasts are smoothed, the improvement from the ensemble approach is much reduced. Second, the smoothed forecasts produce lower errors than the unadjusted forecasts for both populations, suggesting that the smoothing step does not necessarily jeopardise forecasting accuracy. Table 3 presents the mean absolute errors of the 10-year forecasts for a different data period of years 1975-2006. While the ensemble approach appears to perform even better in this case, the smoothed forecasts continue to have lower errors than the unadjusted forecasts for each model tested. It is also worth noting that allowing the ensemble weights to be determined based on in-sample validation may further enhance the forecasting performance of the ensemble approach.

```
f8 <- ENS(x=x, M=M, curve="makeham2", jumpoff=2)
```

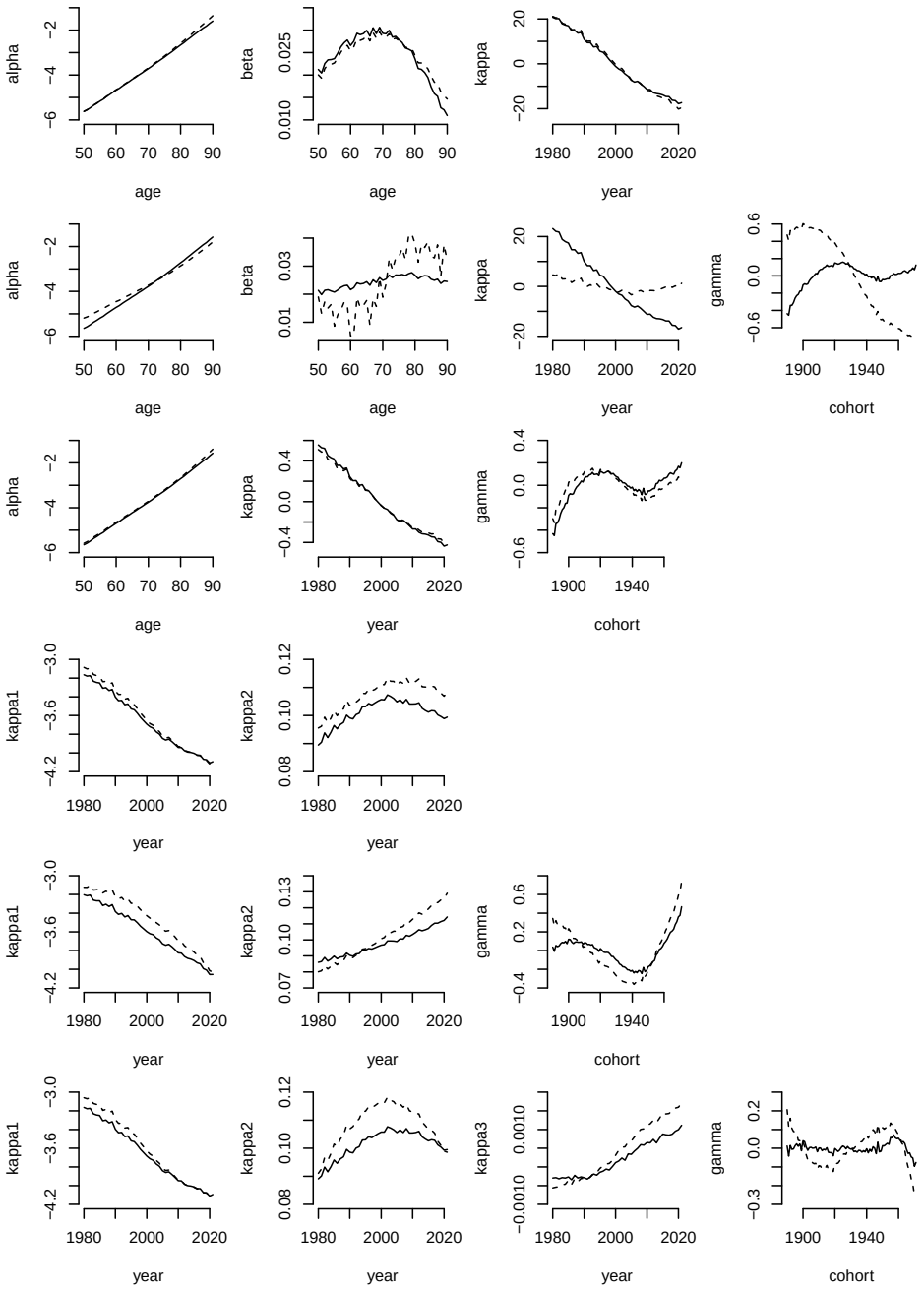



**Figure 5:** Parameter estimates (top to bottom panel – LC, RH, APC, M5, M6, M7) from demofit (solid lines) and StMoMo (dashed lines) using Australian male data of years 1980-2021 and ages 50-90.

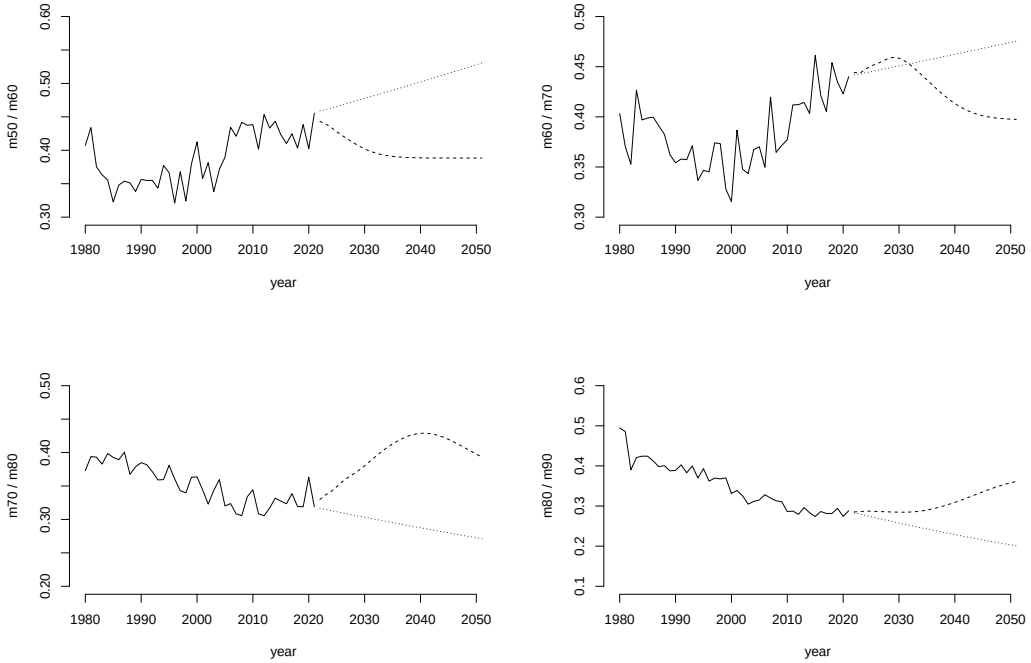



**Figure 6:** Ratios of death rates (top left – 50 vs 60; top right – 60 vs 70; bottom left – 70 vs 80; bottom right – 80 vs 90) from STAR (dashed lines) and LC (dotted lines) models using Australian male data of years 1980-2021 and ages 50-90.

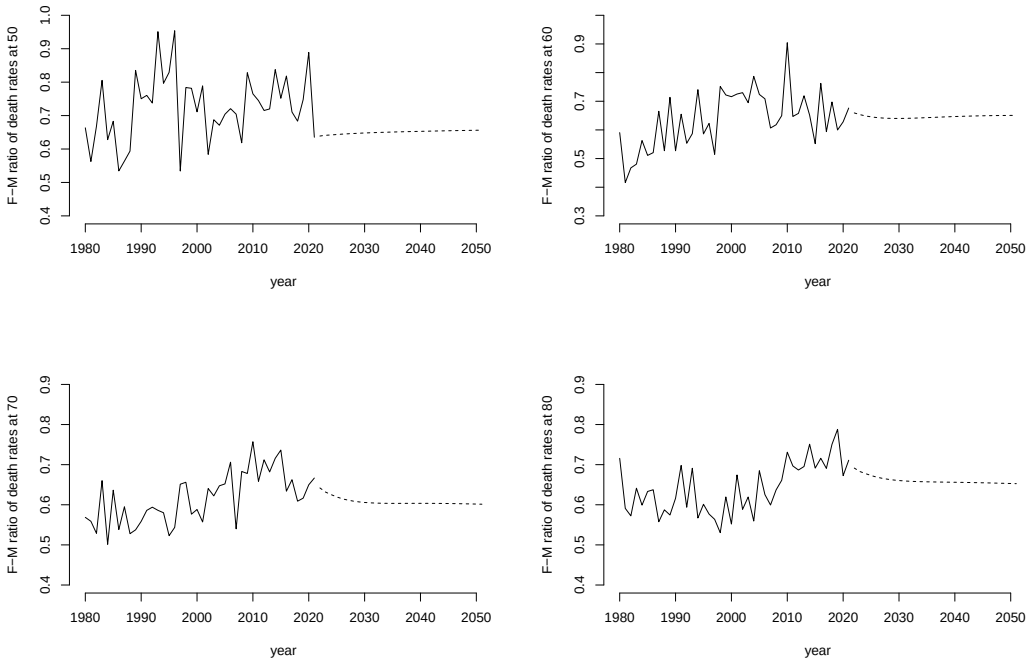



**Figure 7:** Female-to-male ratios of death rates at ages 50, 60, 70, and 80 from common factor model using New Zealand data of years 1980-2021 and ages 50-90.

**Table 2:** Mean absolute errors of 10-year forecasting performance under different stochastic mortality models using New Zealand data of years 1980-2011 and ages 50-90

| Model | NZ F (unadjusted) | NZ M (unadjusted) | NZ F (smoothed) | NZ M (smoothed) |
|---|---|---|---|---|
| LC | 0.1050 | 0.1070 | 0.0901 | 0.0921 |
| RH | 0.1183 | 0.1241 | 0.1039 | 0.1091 |
| APC | 0.1074 | 0.1203 | 0.0963 | 0.1067 |
| M5 | 0.1007 | 0.1042 | 0.0926 | 0.0926 |
| M6 | 0.1956 | 0.1039 | 0.1438 | 0.0902 |
| M7 | 0.1315 | 0.1066 | 0.1206 | 0.0918 |
| STAR | 0.1001 | 0.1038 | 0.0990 | 0.1005 |
| Ensemble | 0.0995 | 0.1010 | 0.0933 | 0.0929 |

**Table 3:** Mean absolute errors of 10-year forecasting performance under different stochastic mortality models using New Zealand data of years 1975-2006 and ages 50-90

| Model | NZ F (unadjusted) | NZ M (unadjusted) | NZ F (smoothed) | NZ M (smoothed) |
|---|---|---|---|---|
| LC | 0.0896 | 0.0796 | 0.0745 | 0.0690 |
| RH | 0.1081 | 0.0844 | 0.0946 | 0.0683 |
| APC | 0.0957 | 0.0876 | 0.0765 | 0.0696 |
| M5 | 0.0895 | 0.0802 | 0.0765 | 0.0714 |
| M6 | 0.1091 | 0.0885 | 0.0973 | 0.0691 |
| M7 | 0.1183 | 0.0869 | 0.1065 | 0.0716 |
| STAR | 0.0897 | 0.0851 | 0.0883 | 0.0839 |
| Ensemble | 0.0860 | 0.0730 | 0.0748 | 0.0641 |

As a final note regarding the differences between the raw and smoothed forecasts, we fit the LC, RH, APC, and STAR models to New Zealand data of years 1980-2011 and ages 0-90, where the forecasts are regularised by the Heligman-Pollard curve (constrained), being suitable for the entire age range. Compared to the results above, where the age range is 50-90 only, the age profile is more complex here and the results in Table 4 are mixed between the raw and smoothed forecasts. For the first three models, the smoothed forecasts yield lower errors, but the situation is reversed for the STAR model. Figure 8 compares the trajectories between the raw and smoothed STAR forecasts at ages 0 (birth), 20 (accident hump), and 65 (retirement). Despite their higher errors under the STAR model in this example, the smoothed forecasts still largely preserve the declining trends of the unadjusted forecasts. It appears that while the smoothing step improves age-profile coherence, its effect on forecasting accuracy generally depends on the data, model, and age range involved.

**Table 4:** Mean absolute errors of 10-year forecasting performance under LC, RH, APC, and STAR models using New Zealand data of years 1980-2011 and ages 0-90

| Model | NZ F (unadjusted) | NZ M (unadjusted) | NZ F (smoothed) | NZ M (smoothed) |
|---|---|---|---|---|
| LC | 0.3380 | 0.2673 | 0.3174 | 0.2535 |
| RH | 0.4055 | 0.3046 | 0.3702 | 0.2881 |
| APC | 0.3198 | 0.2544 | 0.3118 | 0.2371 |
| STAR | 0.2853 | 0.2144 | 0.3111 | 0.2377 |

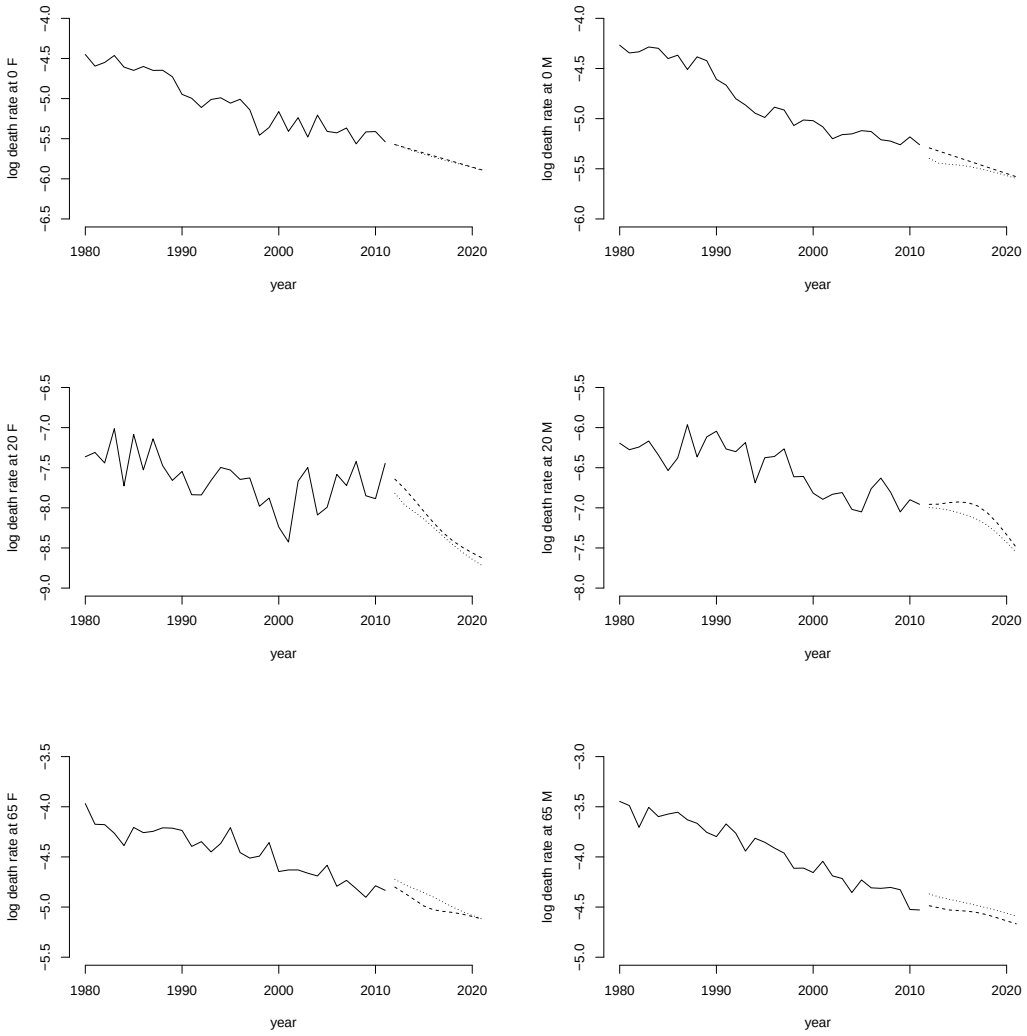



**Figure 8:** Observed log death rates (solid) in 1980-2011 and smoothed (dotted) and unadjusted (dashed) forecasted log death rates in 2012-2021 for New Zealand under STAR model.

## 5 Concluding remarks

In this paper, we have introduced the R package **demofit**, which provides a unified methodological and computational framework for mortality modelling by integrating parametric mortality laws and stochastic mortality models within a coherent interface. The package offers a comprehensive collection of mortality curves alongside widely used forecasting models. Parameter estimation for the mortality curves is carried out using multiple optimisation routines to enhance robustness, while computation remains efficient for typical mortality datasets. In addition, the ability to regularise mortality forecasts using parametric curves provides a practical mechanism for obtaining smooth and interpretable age profiles, particularly in insurance applications involving small or noisy policyholder populations. While similar functionality may be obtained by combining the few existing packages in a certain way, this requires substantial manual integration, inconsistent model specifications, and separate estimation frameworks. The demofit package instead provides a unified workflow with consistent estimation, model comparison, and forecast regularisation within a single interface. The package also includes ensemble forecasting functionality, allowing users to combine multiple models to improve predictive performance. In addition, it implements the common factor model for multi-population mortality modelling, enabling coherent forecasting across related populations.

We have described the underlying methodological and computational framework and illustrated the main functionalities of the package using real mortality data. Through a series of examples, we have demonstrated its use in mortality curve fitting, model comparison, forecasting, and the analysis of age-specific and time-varying mortality patterns. The results highlight the flexibility of the package in accommodating different modelling approaches within a single workflow, making it suitable for empirical research, actuarial applications, demographic studies, and teaching.

These examples also highlight that forecast regularisation / smoothing / graduation should be viewed as an *optional* post-processing step[7]. The additional backtesting results show that smoothing can reduce forecast errors in some settings, while producing higher errors in others. While this procedure does not necessarily alter the underlying temporal mortality trends, its effect depends on the selected mortality curve and the characteristics of the stochastic model. Users should therefore assess the effect of regularisation for their particular application by comparing the raw and smoothed forecasts and performing diagnostic checks[8]. The choice of mortality curve may also introduce systematic bias if its age profile is inappropriate for the population under study, and users are advised to fit and compare several mortality curves. Moreover, since the smoothed forecasts are obtained by transforming the original forecasts, the uncertainty measures generated by the underlying stochastic model should be interpreted with care rather than automatically regarded as uncertainty intervals for the smoothed forecasts.

There are several directions for future development. These include extending the range of models available within the package, such as incorporating other mortality and fertility curves and forecasting models, embedding life expectancy calculations, developing bootstrapping tools for uncertainty quantification, and building more multi-population models. In addition, further enhancements may include automated comparison and selection of models to assist users in identifying suitable specifications for different applications. We will continue to expand the functionality of the package to support a broader range of demographic and actuarial analyses.

## 6 Acknowledgements

The author thanks the editors and reviewers for their valuable comments and suggestions, which have greatly enhanced the presentation of the paper. The author also thanks Dr Emi Tanaka and Professor Rob Hyndman for their inspiration and insightful discussions at the initial stage of this R project. The author acknowledges support from the SMU Internal Research Grant (24-SOE-SMU-117).

## References

D. B. Atkinson. Table development. Technical report, Society of Actuaries, 2018. [p408, 415]

[7] In the latest **demofit** version 0.1.5, if the `curve` argument is missing, no smoothing will be performed.

[8] Note that the post-model regularisation adopted here is deliberately a pragmatic (and rather subjective) actuarial device rather than a replacement for statistically integrated smoothing approaches. In insurance practice, graduation may also be applied to modelled mortality rates when the resulting schedules require further smoothing, with the choice of graduation method depending on the data quality and intended use of the resulting mortality table (Atkinson, 2018). Overall, this package function offers a simple and flexible practical option that can be evaluated empirically for a particular application. Further investigation of its statistical properties is left for future research.

K. Barigou and P. O. Goffard. *StanMoMo: Bayesian Mortality Modelling with Stan*, 2023. R Package version 1.2.0. [p405]

R. E. Beard. Some aspects of theories of mortality, cause of death analysis, forecasting and stochastic processes. *Biological Aspects of Demography*, pages 57–68, 1971. [p403]

I. Bray. Application of Markov chain Monte Carlo methods to projecting cancer incidence and mortality. *Journal of the Royal Statistical Society Series C*, 51(2):151–164, 2002. [p404]

A. J. G. Cairns, D. Blake, and K. Dowd. A two-factor model for stochastic mortality with parameter uncertainty: Theory and calibration. *Journal of Risk and Insurance*, 73(4):687–718, 2006. [p402, 404]

A. J. G. Cairns, D. Blake, K. Dowd, G. D. Coughlan, D. Epstein, A. Ong, and I. Balevich. A quantitative comparison of stochastic mortality models using data from England and Wales and the United States. *North American Actuarial Journal*, 13(1):1–35, 2009. [p402, 404]

L. F. V. Figueiredo, L. M. F. Silva, V. G. R. Lobo, T. C. O. Fonseca, and M. B. Alves. *BayesMortalityPlus: Bayesian Mortality Modelling*, 2025. R Package version 1.0.0. [p405]

B. Gompertz. On the nature of the function expressive of the law of human mortality, and on a new mode of determining the value of life contingencies. *Philosophical Transactions of the Royal Society of London*, 115(1825):513–583, 1825. [p403]

S. Haberman and A. Renshaw. A comparative study of parametric mortality projection models. *Insurance: Mathematics and Economics*, 48(1):35–55, 2011. [p402, 404]

L. Heligman and J. H. Pollard. The age pattern of mortality. *Journal of the Institute of Actuaries*, 107(1): 49–80, 1980. [p403]

HMD. Human Mortality Database. University of California, Berkeley (USA) and Max Planck Institute for Demographic Research (Germany). 2026. [p403, 406]

R. Hyndman, G. Athanasopoulos, C. Bergmeir, G. Caceres, L. Chhay, M. O'Hara-Wild, F. Petropoulos, S. Razbash, E. Wang, and F. Yasmeen. *forecast: Forecasting functions for time series and linear models*, 2026. R Package version 9.0.1. [p405]

KTD. Kannisto-Thatcher Database. Max Planck Institute for Demographic Research. 2018. [p407]

R. Lee and T. Miller. Evaluating the performance of the Lee-Carter method for forecasting mortality. *Demography*, 38(4):537–549, 2001. [p405, 406]

R. D. Lee and L. R. Carter. Modeling and forecasting US mortality. *Journal of the American Statistical Association*, 87(419):659–671, 1992. [p402, 404]

H. Li and Y. Lu. Coherent forecasting of mortality rates: A sparse vector-autoregression approach. *ASTIN Bulletin*, 47(2):563–600, 2017. [p402, 404]

J. Li. A Poisson common factor model for projecting mortality and life expectancy jointly for females and males. *Population Studies*, 67(1):111–126, 2013. [p403, 405, 410]

J. Li. A model stacking approach for forecasting mortality. *North American Actuarial Journal*, 27(3): 530–545, 2023. [p410]

J. Li. *demofit: Parametric mortality curve fitting and mortality forecasting tools*, 2026. R Package version 0.1.5. [p402]

J. Li and J. Liu. A logistic two-population mortality projection model for modelling mortality at advanced ages for both sexes. *Scandinavian Actuarial Journal*, 2019(2):97–112, 2019. [p404]

J. Li, J. Liu, and A. Butt. A systematic vector autoregressive framework for modelling and forecasting mortality. *Journal of Forecasting*, 43(6):2279–2297, 2024. [p402, 404]

N. Li and R. Lee. Coherent mortality forecasts for a group of populations: An extension of the Lee-Carter method. *Demography*, 42(3):575–594, 2005. [p410]

W. M. Makeham. On the law of mortality and the construction of annuity tables. *Journal of the Institute of Actuaries*, 8(6):301–310, 1860. [p403]

S. Martinelle. A generalized perks formula for old-age mortality. *Statistiska Centralbyran*, 1987. [p403]

R. McNown and A. Rogers. Forecasting mortality: A parameterized time series approach. *Demography*, 26(4):645–660, 1989. [p404]

L. H. F. Oppermann. On the graduation of life tables, with special application to the rate of mortality in infancy and childhood. *The Institute of Actuaries*, 1870. [p403]

M. Pascariu. *MortalityLaws: Parametric mortality models, life tables and HMD*, 2025. R Package version 2.1.3. [p404]

W. Perks. On some experiments in the graduation of mortality statistics. *Journal of the Institute of Actuaries*, 63(1):12–57, 1932. [p403]

A. E. Renshaw and S. Haberman. A cohort-based extension to the Lee-Carter model for mortality reduction factors. *Insurance: Mathematics and Economics*, 38(3):556–570, 2006. [p405]

A. Rogers and F. Planck. Model: A general program for estimating parametrized model schedules of fertility, mortality, migration, and marital and labor force status transitions. *IIASA Working Paper WP-83-102*, 1983. [p403]

W. Siler. Parameters of mortality in human populations with widely varying life spans. *Statistics in Medicine*, 2(3):373–380, 1983. [p403]

E. Tabeau. A review of demographic forecasting models for mortality. Forecasting Mortality in Developed Countries. *European Studies of Population*, 9, 2001. [p403]

A. R. Thatcher. The long-term pattern of adult mortality and the highest attained age. *Journal of the Royal Statistical Society Series A*, 162(1):5–43, 1999. [p403, 407]

T. N. Thiele. On a mathematical formula to express the rate of mortality throughout the whole of life, tested by a series of observations made use of by the Danish Life Insurance Company of 1871. *Journal of the Institute of Actuaries and Assurance Magazine*, 16(5):313–329, 1871. [p403]

H. Turner, D. Firth, B. Ripley, B. Venables, D. M. Bates, and M. Maechler. *gnm: Generalized nonlinear models*, 2025. R Package version 1.1-5. [p405]

A. Villegas, P. Millossovich, and V. Kaishev. *StMoMo: Stochastic mortality modelling*, 2025. R Package version 0.4.1. [p405]

W. Weibull. A statistical distribution function of wide applicability. *Journal of Applied Mechanics*, 18(3): 293–297, 1951. [p403]

T. Wittstein and D. A. Bumsted. The mathematical law of mortality. *Journal of the Institute of Actuaries and Assurance Magazine*, 24(3):153–173, 1883. [p403]

*Jackie Li*
*School of Economics, Singapore Management University*

jackieli@smu.edu.sg

# metafrontier: Unified Metafrontier Analysis for Efficiency and Productivity in R

*by Erik Enstad*

**Abstract** Comparing the technical efficiency of groups of firms that face different restrictions of a common production technology requires a metafrontier framework that envelops the group-specific frontiers. Existing R packages cover parts of this workflow, but none combines SFA- and DEA-based metafrontier estimation with inference for technology gap ratios and the metafrontier Malmquist decomposition. We introduce metafrontier, a package that provides a unified interface for three complementary approaches: the deterministic metafrontier of Battese, Rao, and O'Donnell (2004) under both of its identification criteria, the stochastic metafrontier of Huang, Huang, and Liu (2014), and DEA-based metafrontier models with radial, directional, hyperbolic, and FDH technologies. The package estimates group-specific frontiers, constructs the metafrontier envelope, and decomposes efficiency into group technical efficiency and the technology gap ratio. Additional features include the metafrontier Malmquist productivity index with three-way decomposition, bootstrap confidence intervals for technology gap ratios, Murphy–Topel corrected standard errors, poolability tests for both SFA and DEA fits, convergence diagnostics, latent class metafrontier estimation via the EM algorithm, panel stochastic frontier models with time-varying inefficiency on balanced or unbalanced panels, and ggplot2 visualisation methods. Pre-fitted group frontiers from sfaR and frontier are accepted directly, and estimation can be delegated to sfaR, frontier, or Benchmarking as backends. We illustrate the package with simulated and real data and a Monte Carlo study covering parameter and efficiency recovery, small samples, and unbalanced group sizes. All examples and simulations are fully reproducible and complete in under ten minutes on a standard laptop.

## 1 Introduction

Firms within the same industry often face different restrictions on the set of production plans available to them. In microeconomic terms the industry technology set contains all technically feasible input-output combinations, but regulation, environmental conditions, resource endowments, and the cost of switching production techniques can prevent particular groups of firms from accessing every point in it. Steel producers, for example, operate either the capital-intensive blast furnace–basic oxygen furnace (BF-BOF) route or the scrap-based electric arc furnace (EAF) route; both belong to the same industry technology set, but capital vintage and scrap availability restrict which points an individual producer can reach. Similarly, dairy farms in different countries face distinct climatic conditions and input markets that restrict their feasible production plans (Moreira and Bravo-Ureta, 2010). When groups of producers face different restricted technology sets, pooling all observations into a single stochastic frontier or data envelopment analysis model conflates within-group inefficiency with between-group differences in the accessible technology, potentially yielding misleading efficiency rankings.

The metafrontier framework, introduced by Battese et al. (2004) and extended by O'Donnell et al. (2008) and Huang et al. (2014), addresses this problem by defining an overarching production frontier, the *metafrontier*, that envelops the group-specific frontiers. This framework decomposes a firm's distance from the metafrontier into two components:

$$\mathrm{TE}_i^* = \mathrm{TE}_i \times \mathrm{TGR}_i \tag{1}$$

where $\mathrm{TE}_i$ is the firm's *technical efficiency* relative to its own group frontier, measuring managerial performance within the available technology, and $\mathrm{TGR}_i$ is the *technology gap ratio*, measuring how close the group's frontier lies to the best-practice metafrontier. The product $\mathrm{TE}_i^*$ is the firm's *metafrontier technical efficiency*, that is, its performance measured against the industry-wide best practice. Figure 1 in Section 2.2 illustrates this decomposition for the single-input case.

The package is designed for applied economists, regulators, and operations researchers who need to compare technical efficiency across groups operating under different production technologies.

Several R (R Core Team, 2025) packages address parts of the efficiency analysis workflow. The **frontier** package (Coelli and Henningsen, 2020), the **sfaR** package (Dakpo et al., 2023), and the newer **sfa** package (Bernstein et al., 2026) provide stochastic frontier estimation across a wide set of specifications but include no metafrontier layer. The **Benchmarking** package (Bogetoft and Otto, 2011) implements a wide range of DEA models; its `XREF`/`YREF` arguments score observations against an

externally supplied reference technology, so pooled-frontier DEA scores can be assembled manually, but the user must orchestrate the per-group runs, form and summarise the technology gap ratios, and forgo inference on them: its bootstrap targets Farrell scores with respect to a single technology, and its Malmquist functions compute the standard index rather than the metafrontier decomposition. The **deaR** package (Coll-Serrano et al., 2023) offers a general DEA toolkit, including directional distance functions via `model_dir()`, but no metafrontier workflow. The **smfa** package (Owili, 2026), the CRAN release of the earlier GitHub package **metafrontieR** (Olieko, 2026), wraps **sfaR** to estimate deterministic (LP and QP), stochastic, latent class, and sample selection (Greene, 2010) metafrontiers, but it is SFA-only: it provides no DEA-based metafrontier, no productivity decomposition, no bootstrap or Murphy–Topel inference for technology gap ratios, and no panel or visualisation support. No package, on CRAN or elsewhere, combines SFA- and DEA-based metafrontier estimation with inference for technology gap ratios and the metafrontier Malmquist decomposition of O'Donnell et al. (2008).

The **metafrontier** package fills this gap. Its principal contributions are:

- **Unified estimation framework.** A single function, `metafrontier()`, dispatches across SFA and DEA methods with deterministic or stochastic metafrontier formulations through the `method` and `meta_type` arguments.
- **Malmquist productivity decomposition.** The `malmquist_meta()` function computes the metafrontier Malmquist total factor productivity index with the three-way decomposition of O'Donnell et al. (2008).
- **Bootstrap confidence intervals.** The `boot_tgr()` function provides parametric and nonparametric bootstrap inference for technology gap ratios, with parallel computation support.
- **Murphy–Topel corrected standard errors.** The stochastic metafrontier accounts for the generated-regressor problem using the variance correction of Murphy and Topel (1985).
- **Latent class metafrontier.** The `latent_class_metafrontier()` function estimates group membership endogenously when technology classes are unobserved.
- **Interoperability.** Pre-fitted group frontiers from **sfaR** and **frontier** are accepted directly via the `models` argument, and the `engine` argument delegates group-frontier estimation to **sfaR**, **frontier**, or (for DEA) **Benchmarking**.
- **Diagnostics and user safety.** Every estimation stage records its convergence status, inspectable via `check_convergence()` and `summary()`; poolability of the group frontiers is testable for both SFA (likelihood ratio) and DEA (permutation) fits.

The remainder of this article is organised as follows. Section 2.2 reviews the metafrontier methodology. Section 2.3 describes the package design. Section 2.4 demonstrates basic usage with simulated data. Section 2.5 covers advanced features. Section 2.6 presents Monte Carlo evidence on estimator performance. Section 2.7 compares **metafrontier** with related packages, and Section 2.8 concludes.

## 2 Metafrontier methodology

Readers already familiar with stochastic frontier analysis and metafrontier models may wish to proceed directly to Section 2.3.

Throughout, we use the term *technology* in the sense of the metafrontier literature. The industry *metatechnology* $T^*$ is the technology set of microeconomic production theory: the set of all input-output combinations that are technically feasible given the current state of knowledge in the industry (O'Donnell et al., 2008). Group-specific *technologies* $T_j \subset T^*$ are restricted subsets of the metatechnology, arising because regulation, environmental conditions, resource endowments, or the cost of switching production techniques prevent the firms in group $j$ from choosing every point of $T^*$. Following the convention of Battese et al. (2004) and O'Donnell et al. (2008), we refer to these restricted sets simply as group technologies; the metafrontier is the boundary of $T^*$, and the group frontiers are the boundaries of the sets $T_j$.

### 2.1 Group-specific stochastic frontiers

Consider $J$ groups of firms, where group $j$ ($j = 1, \ldots, J$) has access to a potentially restricted technology set $T_j$. The stochastic frontier production function for group $j$ is

$$y_{ij} = \mathbf{x}'_{ij}\boldsymbol{\beta}_j + v_{ij} - u_{ij}, \quad i = 1, \ldots, N_j \tag{2}$$

where $y_{ij}$ is the log output of firm $i$ in group $j$, $\mathbf{x}_{ij}$ is a vector of log inputs (including a constant), $\boldsymbol{\beta}_j$ is the group-specific parameter vector, $v_{ij} \sim N(0, \sigma^2_{v,j})$ is a symmetric noise term, and $u_{ij} \geq 0$ is a one-sided inefficiency term (Aigner et al., 1977). The composed error is $\varepsilon_{ij} = v_{ij} - u_{ij}$.

The **metafrontier** package supports three distributional assumptions for $u_{ij}$: the half-normal ($u_{ij} \sim |N(0, \sigma^2_{u,j})|$), the truncated normal ($u_{ij} \sim N^+(\mu_j, \sigma^2_{u,j})$), and the exponential ($u_{ij} \sim \text{Exp}(\lambda_j)$). Two estimators of technical efficiency from the composed error are in common use, and the package computes both. The default is the conditional expectation of Battese and Coelli (1988) (BC88),

$$\text{TE}_{ij} = E\Big[\exp(-u_{ij}) \mid \varepsilon_{ij}\Big] = \frac{\Phi\Big(\mu^*_{ij}/\sigma^*_j - \sigma^*_j\Big)}{\Phi\Big(\mu^*_{ij}/\sigma^*_j\Big)} \exp\Big(-\mu^*_{ij} + \tfrac{1}{2}\sigma^{*2}_j\Big) \tag{3}$$

where $\mu^*_{ij}$ and $\sigma^*_j$ are the mean and standard deviation of the conditional distribution of $u_{ij}$ given $\varepsilon_{ij}$, and $\Phi(\cdot)$ is the standard normal distribution function. BC88 provides consistent estimates of technical efficiency scores. The alternative estimator $\exp(-\widehat{u}_{ij})$ with $\widehat{u}_{ij} = E[u_{ij} \mid \varepsilon_{ij}]$ (Jondrow et al., 1982, JLMS) does not account for Jensen's inequality; the difference is negligible when $\sigma_u$ is small relative to $\sigma_v$ but grows for highly inefficient samples. The `estimator` argument of `metafrontier()` selects between the two (`"bc88"`, the default, or `"jlms"`); both are stored on the fitted object, and `efficiencies()` can switch between them without refitting. For a book-length treatment of stochastic frontier analysis, see Kumbhakar and Lovell (2000).

### 2.2 Deterministic metafrontier

Battese et al. (2004; building on Battese and Rao, 2002) define the metafrontier as the deterministic envelope of the group-specific frontiers. The metafrontier production function satisfies $f^*(\mathbf{x}) \geq f_j(\mathbf{x})$ for all $\mathbf{x}$ and all $j$, meaning that it represents the output that would be attainable if all firms had access to the most productive technology available in the industry.

Let $\widehat{\boldsymbol{\beta}}_j$ denote the maximum likelihood estimate from group $j$, and let $\widehat{f}_{ij} = \mathbf{x}'_{ij}\widehat{\boldsymbol{\beta}}_j$ denote the fitted group frontier value for observation $i$ in group $j$. The **metafrontier** package obtains $\boldsymbol{\beta}^*$ by solving the linear program

$$\min_{\boldsymbol{\beta}^*} \sum_{i=1}^{N} \mathbf{x}'_i\boldsymbol{\beta}^* \quad \text{s.t.} \quad \mathbf{x}'_{ij}\boldsymbol{\beta}^* \geq \widehat{f}_{ij} \quad \forall\, i, j \tag{4}$$

which minimises the total metafrontier output subject to the envelope constraint, producing the tightest feasible metafrontier. Battese et al. (2004) propose two criteria for identifying $\boldsymbol{\beta}^*$ under the same envelope constraints: minimising the sum of absolute deviations, which reduces to the linear program in Equation (4) because the constraints force every deviation to be non-negative (O'Donnell et al., 2008, Eqs. 23–25), and minimising the sum of squared deviations, a convex quadratic program. The two share the same constraint set but differ in objective and can yield different $\boldsymbol{\beta}^*$ when not all constraints are binding; Section 2.6 reports their maximum absolute TGR difference on the simulation designs. The package implements both: `objective = "lp"` (the default) is solved via **lpSolveAPI** (Konis and Schwendinger, 2024), and `objective = "qp"` is solved exactly via **quadprog** (Turlach and Weingessel, 2019), with an adaptive-barrier fallback via `constrOptim()` when **quadprog** is unavailable. The technology gap ratio under the deterministic metafrontier is

$$\text{TGR}_{ij} = \frac{\exp(\mathbf{x}'_{ij}\widehat{\boldsymbol{\beta}}_j)}{\exp(\mathbf{x}'_{ij}\widehat{\boldsymbol{\beta}}^*)} \tag{5}$$

which lies in $[0, 1]$ by construction. Because the metafrontier is deterministic, no standard errors are available for $\boldsymbol{\beta}^*$ without resampling methods.

### 2.3 Stochastic metafrontier

Huang et al. (2014) propose an alternative in which the metafrontier itself is stochastic. Define the fitted group frontier value $\widehat{y}_{ij} = \mathbf{x}'_{ij}\widehat{\boldsymbol{\beta}}_j$. The stochastic metafrontier is estimated as a second-stage SFA:

$$\widehat{y}_{ij} = \mathbf{x}'_{ij}\boldsymbol{\beta}^* + v^*_{ij} - u^*_{ij} \tag{6}$$

where $v^*_{ij}$ captures approximation noise and $u^*_{ij} \geq 0$ represents the technology gap. This formulation provides standard errors and hypothesis tests for $\boldsymbol{\beta}^*$, overcoming a key limitation of the deterministic approach. Note that because the stochastic metafrontier need not envelop the group frontiers at every point (due to the noise term $v^*_{ij}$), the resulting TGR can exceed 1 for some observations: at those points the group frontier locally surpasses the estimated metafrontier. How often this occurs is

sample-dependent: in the Monte Carlo design used below no observation reaches TGR $> 1$, whereas on the `sfaR::utility` example used later around 40% of observations do, with values up to about 1.25. Such excess values reflect noise rather than superior-to-best-practice technology. For group-level means or ranking applications we recommend leaving them untruncated so the noise averages out; for firm-level interpretation we recommend truncating at one (`pmin(tgr, 1)`) and reporting the truncated share as a diagnostic.

However, because the dependent variable $\widehat{y}_{ij}$ is itself estimated from the first stage, the standard errors of $\boldsymbol{\beta}^*$ are understated unless corrected for the generated-regressor problem. The **metafrontier** package addresses this with the variance correction of Murphy and Topel (1985), as described in Section 2.5.

### 2.4 DEA-based metafrontier

The metafrontier framework extends naturally to nonparametric methods (O'Donnell et al., 2008; Charnes et al., 1978). For each group $j$, a data envelopment analysis (DEA) frontier is estimated using only the observations in that group, yielding group-specific efficiency scores $\text{TE}_{ij}^{\text{group}}$. A second DEA problem pools all observations across groups, producing metafrontier scores $\text{TE}_{ij}^{\text{pool}}$. In the output orientation (the default), the score of observation $(\mathbf{x}_i, \mathbf{y}_i)$ against a reference set $R$ is $\text{TE}_i = 1/\phi_i^*$, where $\phi_i^*$ solves the envelopment linear program

$$\max_{\phi_i, \boldsymbol{\lambda}} \phi_i \quad \text{s.t.} \quad \sum_{j \in R} \lambda_j \mathbf{x}_j \leq \mathbf{x}_i, \quad \sum_{j \in R} \lambda_j \mathbf{y}_j \geq \phi_i \mathbf{y}_i, \quad \boldsymbol{\lambda} \in \Lambda, \tag{7}$$

and in the input orientation the score is $\theta_i^*$ from $\min \theta_i$ subject to $\sum_j \lambda_j \mathbf{x}_j \leq \theta_i \mathbf{x}_i$, $\sum_j \lambda_j \mathbf{y}_j \geq \mathbf{y}_i$, $\boldsymbol{\lambda} \in \Lambda$. The set $\Lambda$ encodes the returns to scale: $\lambda_j \geq 0$ unrestricted (CRS), adding $\sum_j \lambda_j = 1$ (VRS), $\sum_j \lambda_j \leq 1$ (DRS), $\sum_j \lambda_j \geq 1$ (IRS), or $\lambda_j \in \{0, 1\}$ with $\sum_j \lambda_j = 1$ (FDH, dropping convexity). The group-frontier problem restricts the reference set $R$ to the observations of group $j$; the metafrontier problem pools all $N$ observations into $R$. This distinction is the entire metafrontier construction: the same LP is solved twice per observation against nested reference sets. The technology gap ratio is then

$$\text{TGR}_{ij} = \frac{\text{TE}_{ij}^{\text{pool}}}{\text{TE}_{ij}^{\text{group}}} \tag{8}$$

which lies in $(0, 1]$ because the pooled reference set contains the group reference set. Beyond radial measures, the DEA path exposes directional distance functions (DDF) via `type = "directional"`, with `direction` either a preset (`"proportional"`, `"output"`, `"input"`), a common numeric direction vector, or a matrix of firm-specific directions, and hyperbolic (graph) efficiency via `type = "hyperbolic"`, which contracts inputs and expands outputs simultaneously and remains feasible in cross-period comparisons where radial cross-period programs may not be. Second-stage slack maximisation is available via `slack = TRUE`, flagging observations that are radially efficient but not Koopmans-efficient. The resulting object carries the same `metafrontier` class and supports the standard extractor and plotting methods. Because DEA is nonparametric, it avoids functional form assumptions, but it does not account for statistical noise: positive noise realisations shift the estimated frontier upwards, while negative realisations are attributed to inefficiency. SFA, by contrast, accounts for noise through the composed error, although neither approach can separate the unobserved noise and inefficiency realisations for an individual observation; SFA efficiency scores are conditional expectations (Section 2.2.1).

### 2.5 Efficiency decomposition

Regardless of the estimation method, the fundamental identity holds:

$$\text{TE}_{ij}^* = \text{TE}_{ij} \times \text{TGR}_{ij} \tag{9}$$

where $\text{TE}_{ij}$ measures how efficiently firm $i$ uses its own group's technology, $\text{TGR}_{ij}$ measures how close that group's technology is to the best-practice metafrontier, and $\text{TE}_{ij}^*$ measures overall performance relative to the industry-wide frontier.

Figure 1 illustrates this decomposition for a single-input production function. The vertical distance from a firm's observed output to its group frontier measures technical inefficiency, while the distance from the group frontier to the metafrontier measures the technology gap. A firm can be technically efficient within its group ($\text{TE}_i = 1$) yet still fall short of the metafrontier if its group's technology is

inferior ($\text{TGR}_i < 1$).

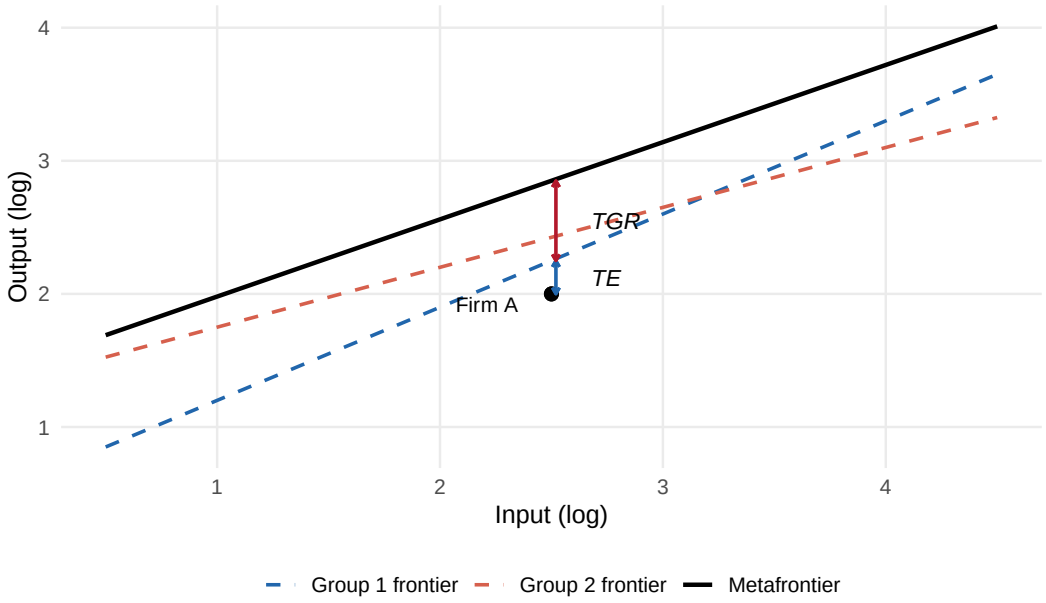



**Figure 1:** Metafrontier decomposition for a single-input production function (diagram drawn with ggplot2; code in the supplementary materials). For Firm A, technical efficiency (TE) is the distance from observed output to the group frontier, the technology gap ratio (TGR) the distance from the group frontier to the metafrontier (solid black), and TE* = TE x TGR.

Table 1 summarises the key properties of the three metafrontier approaches.

**Table 1:** Comparison of metafrontier approaches.

| Property | Deterministic MF | Stochastic MF | DEA |
|---|---|---|---|
| Functional form | Parametric | Parametric | Nonparametric |
| Statistical noise | First stage only | Both stages | None |
| Inference for metafrontier | None (resample) | Yes (with M–T correction) | None (bootstrap) |
| TGR bounds | [0, 1] | Can exceed 1 | [0, 1] |
| Key reference | BRO (2004) | HHL (2014) | ORB (2008) |

### 2.6 Malmquist productivity decomposition

For panel data, the metafrontier Malmquist total factor productivity (TFP) index extends the standard Malmquist index to incorporate technology gap dynamics (O'Donnell et al., 2008). The index between periods $t$ and $t+1$ decomposes as

$$M^*_{i,t,t+1} = \text{TEC}_{i,t,t+1} \times \text{TGC}_{i,t,t+1} \times \text{TC}^*_{i,t,t+1} \tag{10}$$

where TEC is the *technical efficiency change* (catching up to the group frontier), TGC is the *technology gap change* (movement of the group frontier relative to the metafrontier), and $\text{TC}^*$ is the *metafrontier technical change* (shift in the metafrontier itself). This three-way decomposition allows researchers to distinguish productivity growth driven by improved managerial practice from gains attributable to technology adoption or frontier innovation.

One interpretive caveat applies. The Malmquist index is not a *proper* productivity index in the sense of O'Donnell (2012): except under restrictive conditions on the technology, it is not multiplicatively complete and fails the transitivity axiom of index theory, so chained multi-period or cross-firm comparisons of index levels are not generally valid (O'Donnell, 2018). Interpretation should be confined to binary period-to-period comparisons, which is how the package reports the index. Multiplicatively complete alternatives such as the Färe-Primont and Lowe indexes are planned extensions (Section 2.8).

# 3 Package design and architecture

## 3.1 Design principles and S3 interface

The **metafrontier** package is designed around two principles: a unified interface and a consistent S3 method suite. The package exports 13 standalone functions and registers 38 S3 methods across five object classes: `metafrontier` (the primary estimation result), `malmquist_meta` (productivity decomposition), `boot_tgr` (bootstrap inference), `lc_metafrontier` (latent class models), and `metafrontier_convergence` (diagnostics). Table 2 lists all 16 user-facing functions grouped by category, including 3 `autoplot` methods that are registered as S3 methods of the **ggplot2** generic rather than standalone exports.

**Table 2:** User-facing functions in the metafrontier package.

| Category | Function | Description |
|---|---|---|
| Estimation | 'metafrontier()' | Main metafrontier estimation (SFA or DEA) |
| Estimation | 'malmquist_meta()' | Metafrontier Malmquist TFP index |
| Estimation | 'latent_class_metafrontier()' | EM algorithm for latent class metafrontier |
| Analysis | 'efficiencies()' | Extract TE, TGR, or TE* scores |
| Analysis | 'technology_gap_ratio()' | Technology gap ratios by group |
| Analysis | 'tgr_summary()' | Summary statistics for TGR |
| Analysis | 'poolability_test()' | Poolability test (LR for SFA, permutation for DEA) |
| Analysis | 'check_convergence()' | Convergence diagnostics for all estimation stages |
| Inference | 'boot_tgr()' | Bootstrap CIs for technology gap ratios |
| Inference | 'select_n_classes()' | BIC-based selection of latent classes |
| Inference | 'as_metafrontier_model()' | Convert external models to metafrontier format |
| Visualisation | 'autoplot.metafrontier()' | TGR density, efficiency, decomposition, frontier plots |
| Visualisation | 'autoplot.malmquist_meta()' | Malmquist decomposition, TGR evolution, MPI trend plots |
| Visualisation | 'autoplot.boot_tgr()' | Bootstrap distribution and CI plots |
| Simulation | 'simulate_metafrontier()' | Cross-sectional simulation with known DGP |
| Simulation | 'simulate_panel_metafrontier()' | Panel simulation with time-varying inefficiency |

A single `metafrontier()` entry point was chosen over separate functions for each method–meta-type combination because users frequently compare estimators on the same data; a unified interface guarantees that the returned objects share the same S3 class and methods regardless of the estimation path. S3 was preferred over S4 and R6 because it provides the lightest-weight dispatch mechanism, requires no additional dependencies, and matches the class system used by the upstream packages (**frontier**, **Benchmarking**) with which **metafrontier** interoperates.

All primary estimation is accessed through `metafrontier()`, which dispatches via its `method` argument (`"sfa"` or `"dea"`) and `meta_type` argument (`"deterministic"` or `"stochastic"`). The returned `metafrontier` object supports the standard R model interface: `summary()`, `coef()`, `vcov()`, `confint()`, `fitted()`, `residuals()`, `logLik()`, `nobs()`, `predict()`, and `plot()`, so fitted objects work with the standard model-handling idiom without adaptation.

## 3.2 Estimation workflow

The package supports three paths to metafrontier estimation. In the **formula path**, the user supplies raw data, a formula, and a group identifier; the package estimates all group-specific frontiers internally before constructing the metafrontier. In the **pre-fitted model path**, the user first estimates group frontiers using external packages (**sfaR** or **frontier**) and passes the fitted objects directly to `metafrontier()` via the `models` argument; conversion to the internal format happens automatically. In the **engine path**, the user stays in the formula interface but delegates the group-frontier estimation to an external backend via the `engine` argument: `engine = "sfaR"` or `engine = "frontier"` for SFA group frontiers, and `engine = "Benchmarking"` for the DEA path, where both the group frontiers and the pooled metafrontier are computed by `Benchmarking::dea()` using its `XREF/YREF` external-reference facility.

The formula path is the simplest workflow. A single call to `metafrontier()` handles all estimation steps: parsing the formula, splitting data by group, estimating group-specific frontiers, constructing the metafrontier envelope, and computing the TGR decomposition. The pre-fitted model and engine paths are useful when the user requires fine-grained control over individual group frontier specifications or needs features of the specialised packages that the internal estimators lack. The metafrontier stage of the SFA methods is always estimated internally, because the Murphy–Topel variance correction (Section 2.5) differentiates the internal stage-2 likelihood.

### 3.3 Interoperability

Fitted model objects from **sfaR** (class `sfacross`) and **frontier** (class `frontier`) can be passed directly to the `models` argument of `metafrontier()`; the conversion to the internal format is performed automatically by the `as_metafrontier_model()` generic, which users only need to call explicitly for hand-built list models (the conversion is idempotent, so pre-converting is harmless). Farrell objects from **Benchmarking** store neither coefficients nor the underlying data, so they cannot be used with `models`; the `engine = "Benchmarking"` path covers that use case instead. The following example illustrates the pre-fitted model path (not evaluated):

```
library(sfaR)
# Estimate group frontiers externally
mod1 <- sfacross(log_y ~ log_x1 + log_x2,
                 data = subset(dat, group == "G1"))
mod2 <- sfacross(log_y ~ log_x1 + log_x2,
                 data = subset(dat, group == "G2"))

# Pass the fitted objects directly; conversion is automatic
fit <- metafrontier(models = list(G1 = mod1, G2 = mod2))
```

### 3.4 Computational details

The package implements its own maximum likelihood estimator for group-specific stochastic frontiers, using BFGS optimisation with Nelder–Mead fallback for robustness. Variance parameters are log-transformed to ensure unconstrained optimisation over the positive real line. The deterministic metafrontier of Equation (4) is solved via linear programming using the **lpSolveAPI** package (Konis and Schwendinger, 2024), with LP object reuse across observations for improved performance on large datasets. The squared-deviations variant (`objective = "qp"`) is solved exactly by the dual method of Goldfarb and Idnani as implemented in **quadprog** (Turlach and Weingessel, 2019), which is listed in `Suggests` to keep the import footprint unchanged. Numerical stability is maintained through a safe Mills ratio computation (`.safe_mills()`) that prevents `NaN` values in extreme tail regions of the normal distribution, relevant for both panel SFA and latent class models.

The package keeps its dependency footprint minimal. The seven `Imports` (`stats`, `graphics`, `grDevices`, and `methods` from base R; `Formula` for multi-part formula parsing; `numDeriv` for Hessian computation; and `lpSolveAPI` for LP solving) are each exercised by core functionality. **lpSolveAPI** was chosen over alternatives such as **Rglpk** because it is a self-contained C library with no external system dependencies, so it installs without additional system libraries on all supported platforms. Packages used only in optional features (**ggplot2** for `autoplot()` methods, **parallel** for bootstrap parallelisation, **quadprog** for the QP objective, and the three upstream SFA/DEA engines for interoperability and backend delegation) are listed as `Suggests` and loaded conditionally.

Correctness is supported by a test suite of 22 **testthat** files (over 125 test blocks) covering core SFA/DEA estimation, Monte Carlo recovery, bootstrap, Murphy–Topel correction, panel SFA, latent class, DDF, and ggplot2 `autoplot` methods, plus snapshot tests of S3 outputs. Continuous integration runs the suite on Ubuntu, Windows, and macOS under R-devel, release, and oldrel. The custom MLE is additionally benchmarked against **frontier** (`test-frontier-equivalence.R`): on a 3-group dataset from `simulate_metafrontier()` with 150 firms per group and technology gaps of 0, 0.3, and 0.5 (seed 42), similar to the simulation introduced in Section 2.4.1, the maximum absolute coefficient difference against `frontier::sfa()` is $1.2 \times 10^{-5}$ for group 2 and $5.0 \times 10^{-6}$ for group 3. The two estimators use different internal parameterisations but deliver numerically equivalent slope and intercept estimates.

**Computational environment.** All timings reported in this article were obtained on a laptop with an Intel Core Ultra 5 125U processor and 32 GB of RAM running Windows 11 x64, using R version 4.6.1 (2026-06-24) and **metafrontier** 0.3.1. The complete article code, including the Monte Carlo study of Section 2.6, runs in under ten minutes in this environment.

## 4 Basic usage

All examples in this section use simulated data for full reproducibility. The **metafrontier** package includes data-generating functions that produce data with known true parameters, so estimator accuracy can be assessed directly.

### 4.1 Simulating metafrontier data

The `simulate_metafrontier()` function creates cross-sectional data with `n_groups` technology groups, each containing `n_per_group` firms. The data-generating process uses a Cobb–Douglas production function with group-specific intercepts that induce known technology gaps.

```
sim <- simulate_metafrontier(n_groups = 3, n_per_group = 200, seed = 42)
dat <- sim$data
head(dat[, c("log_y", "log_x1", "log_x2", "group",
             "true_te", "true_tgr")])
#>      log_y   log_x1    log_x2 group   true_te true_tgr
#> 1 3.371473 4.574030 4.4255884    G1 0.6700025        1
#> 2 3.665773 4.685377 2.5855553    G1 0.7704492        1
#> 3 2.784899 1.430698 4.2596549    G1 0.9834917        1
#> 4 3.916103 4.152238 2.2139813    G1 0.9853877        1
#> 5 2.313365 3.208728 0.7894005    G1 0.8407115        1
#> 6 2.210272 2.595480 2.2116232    G1 0.7410986        1
```

The returned list contains `$data` (the simulated data frame) and `$params` (the true parameter values). The data frame includes the true technical efficiency (`true_te`), true technology gap ratio (`true_tgr`), and true metafrontier efficiency (`true_te_star`), so estimates can be compared directly with estimated values.

### 4.2 Deterministic metafrontier

The deterministic metafrontier of Battese et al. (2004) is estimated by specifying `meta_type = "deterministic"`:

```
fit_det <- metafrontier(log_y ~ log_x1 + log_x2, data = dat,
                        group = "group", method = "sfa",
                        meta_type = "deterministic", dist = "hnormal")
summary(fit_det)
#>
#> Metafrontier Model Summary
#> ==========================
#>
#> Call:
#> metafrontier(formula = log_y ~ log_x1 + log_x2, data = dat, group = "group",
#>     method = "sfa", meta_type = "deterministic", dist = "hnormal")
#>
#> Method:        sfa
#> Metafrontier:  deterministic
#>
#> --- Group: G1 (n = 200) ---
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  1.04540    0.07899  13.234  < 2e-16 ***
#> log_x1       0.49023    0.01351  36.295  < 2e-16 ***
#> log_x2       0.19249    0.01378  13.972  < 2e-16 ***
#> log_sigma_v -1.51266    0.16168  -9.356  < 2e-16 ***
#> log_sigma_u -1.18597    0.24943  -4.755 1.99e-06 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#> Log-likelihood: -33.087
#>
#> --- Group: G2 (n = 200) ---
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  0.77078    0.10811   7.130 1.01e-12 ***
#> log_x1       0.47875    0.01450  33.022  < 2e-16 ***
#> log_x2       0.19243    0.01413  13.623  < 2e-16 ***
#> log_sigma_v -1.38894    0.16414  -8.462  < 2e-16 ***
#> log_sigma_u -1.53414    0.59564  -2.576     0.01 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#> Log-likelihood: -29.889
```

```
#>
#> --- Group: G3 (n = 200) ---
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  0.55554    0.06185   8.982   <2e-16 ***
#> log_x1       0.50554    0.01265  39.956   <2e-16 ***
#> log_x2       0.18212    0.01221  14.916   <2e-16 ***
#> log_sigma_v -1.77273    0.14227 -12.460   <2e-16 ***
#> log_sigma_u -1.05299    0.12265  -8.585   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#> Log-likelihood: -18.492
#>
#> --- Metafrontier ---
#>             Estimate
#> (Intercept)   1.0454
#> log_x1        0.4902
#> log_x2        0.1925
#>
#> --- Efficiency Decomposition ---
#>  Group Mean_TE Mean_TGR Mean_TE_star
#>     G1  0.7962   1.0000       0.7962
#>     G2  0.8488   0.7385       0.6269
#>     G3  0.7739   0.6223       0.4816
#>
#> --- Technology Gap Ratio Summary ---
#>  Group   N   Mean     SD    Min     Q1 Median     Q3    Max
#>     G1 200 1.0000 0.0000 1.0000 1.0000 1.0000 1.0000 1.0000
#>     G2 200 0.7385 0.0118 0.7178 0.7284 0.7380 0.7481 0.7598
#>     G3 200 0.6223 0.0169 0.5838 0.6111 0.6218 0.6346 0.6580
#>
#> --- Convergence ---
#> All estimation stages converged.
```

The `summary()` method reports the group-specific frontier coefficients, the metafrontier coefficients obtained from the constrained optimisation of Equation (4), and summary statistics for the technology gap ratios by group.

### 4.3 Stochastic metafrontier

The stochastic metafrontier of Huang et al. (2014) replaces the deterministic envelope with a second-stage SFA:

```
fit_sto <- metafrontier(log_y ~ log_x1 + log_x2, data = dat,
                        group = "group", method = "sfa",
                        meta_type = "stochastic", dist = "hnormal")
coef(fit_sto)
#> (Intercept)      log_x1      log_x2
#>   0.7975464   0.4928374   0.1867363
confint(fit_sto)
#>                 2.5 %    97.5 %
#> (Intercept) 0.5578845 1.0372084
#> log_x1      0.4817555 0.5039193
#> log_x2      0.1758149 0.1976577
```

Unlike the deterministic approach, the stochastic metafrontier provides standard errors, $z$-statistics, and confidence intervals for the metafrontier coefficients directly.

### 4.4 DEA-based metafrontier

The DEA-based metafrontier computes group-specific and pooled DEA frontiers without parametric assumptions. Because DEA requires strictly positive inputs and outputs, we use level-scale variables rather than the log-transformed variables used for SFA:

```
dat_lev <- within(dat, {
  y  <- exp(log_y)
  x1 <- exp(log_x1)
  x2 <- exp(log_x2)
})
fit_dea <- metafrontier(y ~ x1 + x2, data = dat_lev,
                        group = "group", method = "dea", rts = "vrs")
tgr_summary(fit_dea)
#>   Group   N      Mean         SD       Min        Q1    Median
#> 1    G1 200 0.9760509 0.05474776 0.5343945 0.9755096 1.0000000
#> 2    G2 200 0.8426304 0.09659937 0.6513259 0.7672661 0.8480490
#> 3    G3 200 0.5905357 0.11012920 0.4058857 0.5332346 0.5655475
#>          Q3 Max
#> 1 1.0000000   1
#> 2 0.9155486   1
#> 3 0.6110953   1
```

### 4.5 Extracting results and diagnostics

The `efficiencies()` function extracts efficiency scores. Three types are available: group technical efficiency (`"group"`), metafrontier efficiency (`"meta"`), and technology gap ratios (`"tgr"`). The fundamental identity of Equation (9) can be verified numerically:

```
te <- efficiencies(fit_det, type = "group")
tgr <- efficiencies(fit_det, type = "tgr")
te_star <- efficiencies(fit_det, type = "meta")
all.equal(te_star, te * tgr, tolerance = 1e-10)
#> [1] TRUE
```

The `tgr_summary()` function provides descriptive statistics for the TGR by group:

```
tgr_summary(fit_det)
#>   Group   N      Mean           SD       Min        Q1    Median
#> 1    G1 200 1.0000000 1.084493e-13 1.0000000 1.0000000 1.0000000
#> 2    G2 200 0.7385079 1.184913e-02 0.7177953 0.7284223 0.7379621
#> 3    G3 200 0.6223039 1.691828e-02 0.5838489 0.6110806 0.6218278
#>          Q3       Max
#> 1 1.0000000 1.0000000
#> 2 0.7481009 0.7597913
#> 3 0.6345533 0.6579921
```

The `poolability_test()` function tests the null hypothesis that all groups share a common frontier against the alternative of group-specific frontiers. For SFA fits it conducts a likelihood ratio test; for DEA fits, where no likelihood exists, it dispatches to a permutation test that exploits the exchangeability of group labels under the pooled-technology null (Simar and Zelenyuk, 2007; Kneip et al., 2016):

```
poolability_test(fit_det)
#>
#>  Likelihood Ratio Test for Poolability of Group Frontiers
#>
#> data:  fit_det
#> LR = 275.4, df = 10, p-value < 2.2e-16
```

Convergence of every estimation stage can be verified with `check_convergence()`, which reports one row per group frontier plus the metafrontier stage. The same information is printed by `summary()`, and a warning is issued at fit time whenever an optimiser stops at a non-zero code. Estimation aborts with an error only when no usable estimate exists; otherwise the codes are recorded so that users can inspect them before interpreting TGRs, confidence intervals, or productivity decompositions:

```
check_convergence(fit_det)
#>
#> Convergence of estimation stages
#> --------------------------------
```

```
#>         stage method code converged note
#>     group: G1    MLE    0      TRUE
#>     group: G2    MLE    0      TRUE
#>     group: G3    MLE    0      TRUE
#>  metafrontier     LP    0      TRUE
```

The **ggplot2** (Wickham, 2016)-based `autoplot()` method provides four plot types. Figure 2 shows the distribution of technology gap ratios by group, and Figure 3 displays the efficiency decomposition.

```
autoplot(fit_det, which = "tgr")
```




**Figure 2:** Distribution of technology gap ratios by group from the deterministic metafrontier (generated with the package autoplot() method, view: TGR density). Group G1 defines the metafrontier (TGR = 1.000 for all observations, annotated separately); groups G2 and G3 show increasing technology gaps.

```
autoplot(fit_det, which = "decomposition")
```

### 4.6 Real-data illustration

The examples above use simulated data with known parameters. To demonstrate the package on empirical data, we use the `utility` dataset from **sfaR**, which contains 791 observations on US fossil-fuel steam electric generating plants over 1986–1996. The binary variable `regu` equals one for plants in states that enacted retail-access (restructuring) legislation during the sample period and zero for plants that remained under traditional regulation. This gives a natural grouping of plants. The production function relates output (`y`) to capital (`k`), labour (`labor`), and fuel (`fuel`).

```
data("utility", package = "sfaR")
utility$group <- ifelse(utility$regu == 1, "Retail access", "Regulated")
utility$log_y     <- log(utility$y)
utility$log_k     <- log(utility$k)
utility$log_labor <- log(utility$labor)
utility$log_fuel  <- log(utility$fuel)

fit_util <- metafrontier(log_y ~ log_k + log_labor + log_fuel,
                         data = utility, group = "group",
                         method = "sfa", meta_type = "deterministic")
```

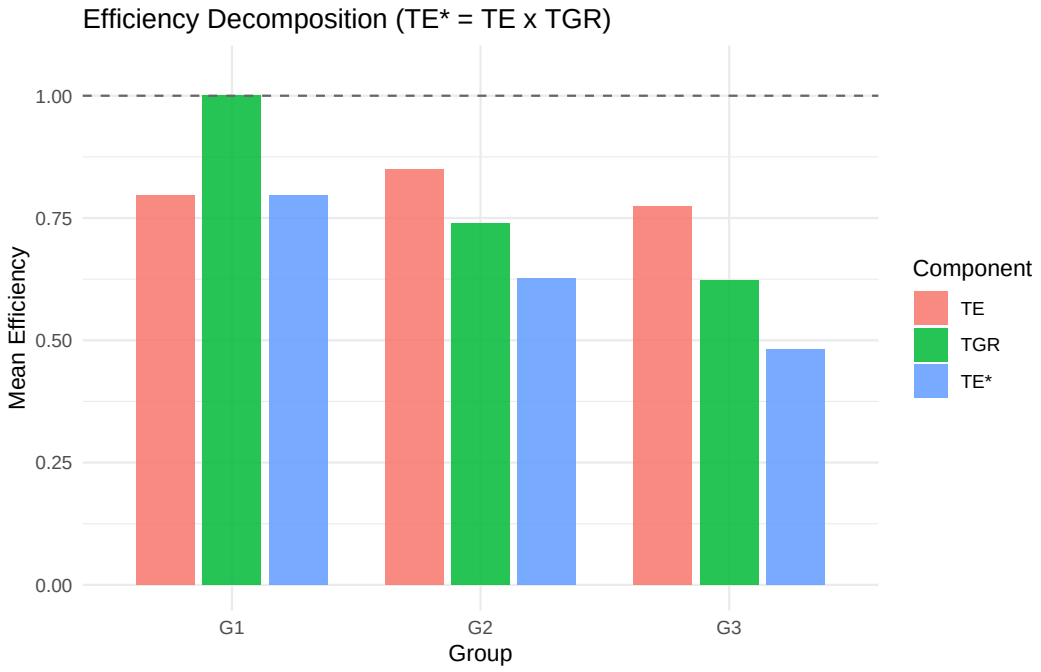



**Figure 3:** Mean efficiency decomposition by group (generated with the package autoplot() method, view: decomposition). The three bars per group show mean technical efficiency (TE), technology gap ratio (TGR), and metafrontier efficiency TE* = TE x TGR.

```
tgr_summary(fit_util)
#>           Group   N      Mean         SD       Min        Q1    Median
#> 1     Regulated 297 0.9457963 0.02197346 0.8755055 0.9318836 0.9423030
#> 2 Retail access 494 0.8920070 0.05876240 0.6279105 0.8614020 0.8973676
#>          Q3 Max
#> 1 0.9584353   1
#> 2 0.9327646   1
```

The traditionally regulated group exhibits a slightly higher mean TGR (0.946) than the group of plants in retail-access states (0.892); the regulated plants lie closer to the pooled metafrontier on average in this sample. We attach no structural interpretation to this illustrative comparison. In the metafrontier framework, a technology gap reflects restrictions on the feasible input-output set; regulatory constraints that only affect the choice of input mix would manifest as allocative rather than technical inefficiency and would not, by themselves, generate a technology gap.

Because all three estimators share the same interface and object class, cross-method robustness checks require no additional plumbing. The following chunk re-estimates the same model with the stochastic metafrontier and with DEA, then compares group-mean TGRs and the Spearman rank correlations of the plant-level TGRs across estimators:

```
fit_util_st  <- metafrontier(log_y ~ log_k + log_labor + log_fuel,
                             data = utility, group = "group",
                             method = "sfa", meta_type = "stochastic")
fit_util_dea <- metafrontier(log_y ~ log_k + log_labor + log_fuel,
                             data = utility, group = "group",
                             method = "dea", rts = "vrs")

tgr_by <- function(fit) tapply(efficiencies(fit, type = "tgr"),
                               utility$group, mean)
round(rbind(Deterministic = tgr_by(fit_util),
            Stochastic    = tgr_by(fit_util_st),
            DEA           = tgr_by(fit_util_dea)), 3)
#>               Regulated Retail access
#> Deterministic     0.946         0.892
#> Stochastic        1.041         0.975
#> DEA               0.996            NA
```

```
round(c(det_vs_stoch = cor(fit_util$tgr, fit_util_st$tgr,
                           method = "spearman"),
        det_vs_dea   = cor(fit_util$tgr, fit_util_dea$tgr,
                           method = "spearman")), 3)
#> det_vs_stoch   det_vs_dea
#>        0.858           NA
```

Agreement in the group ranking across a deterministic envelope, a distributional second stage, and a nonparametric pooled frontier indicates that the technology gap is a feature of the data rather than an artefact of any single method; this kind of triangulation is the main practical payoff of a unified interface.

## 5 Advanced features

### 5.1 Bootstrap confidence intervals

Because the deterministic metafrontier provides no standard errors for the technology gap ratios, the `boot_tgr()` function implements bootstrap inference. Both parametric (residual resampling) and nonparametric (case resampling) approaches are supported, with percentile and bias-corrected accelerated (BCa) interval types.

```
boot <- boot_tgr(fit_det, R = 199, type = "parametric",
                 ci_type = "percentile", seed = 1, progress = FALSE)
boot_ci <- confint(boot)
head(boot_ci, 5)
#>           2.5% 97.5%
#> [1,] 0.9589637     1
#> [2,] 0.9751020     1
#> [3,] 0.9349823     1
#> [4,] 0.9784612     1
#> [5,] 0.9782767     1
```

We use $R = 199$ here to keep the article reproducible in minutes; in applied work, $R \geq 999$ is recommended for reliable percentile intervals, and $R \geq 1999$ for BCa intervals.

The `confint()` method returns observation-level confidence intervals; Figure 4 displays the bootstrap distribution of the group-level means of the TGRs, and Table 3 summarises the intervals by group. Parallel computation is available via the `ncores` argument, which uses the **parallel** package for distributing replications across cores. Table 3 is produced by the following (echoed) chunk.

```
autoplot(boot)
```

```
ci_df <- data.frame(
  Group = dat$group,
  Lower = boot_ci[, 1],
  Upper = boot_ci[, 2],
  TGR = efficiencies(fit_det, type = "tgr")
)
ci_summary <- aggregate(cbind(TGR, Lower, Upper) ~ Group,
                        data = ci_df, FUN = mean)
names(ci_summary) <- c("Group", "Mean TGR", "Mean lower bound",
                       "Mean upper bound")
cap <- paste("Bootstrap 95\\% confidence intervals for TGR by group",
             "(group-level means of observation-level intervals).")
knitr::kable(ci_summary, caption = cap, digits = 4,
             align = "lccc", booktabs = TRUE, linesep = "") |>
  ktbl()
```

### 5.2 Murphy–Topel variance correction

The stochastic metafrontier of Huang et al. (2014) uses fitted values from the first-stage group SFA models as the dependent variable in a second-stage SFA. This two-step procedure understates the

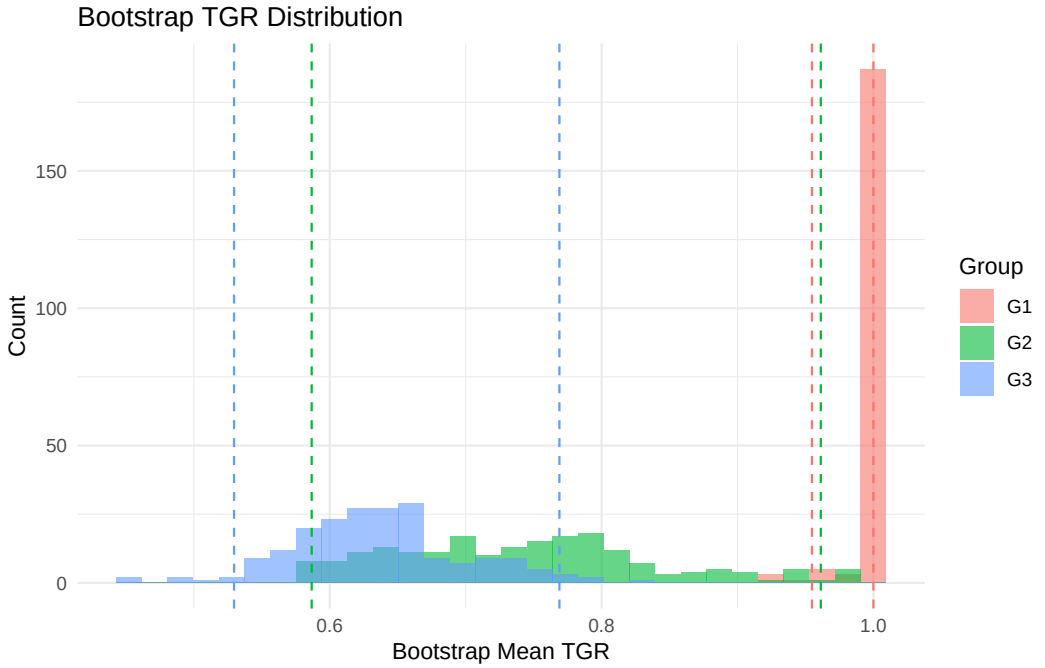



**Figure 4:** Bootstrap distribution of the group-level mean technology gap ratios, from 199 parametric replications (generated with the package autoplot() method for bootstrap objects, default view). Dashed vertical lines mark the 95 percent percentile confidence intervals for the group means.

**Table 3:** Bootstrap 95\% confidence intervals for TGR by group (group-level means of observation-level intervals).

| Group | Mean TGR | Mean lower bound | Mean upper bound |
|---|---|---|---|
| G1 | 1.0000 | 0.9584 | 1.0000 |
| G2 | 0.7385 | 0.5828 | 0.9680 |
| G3 | 0.6223 | 0.5195 | 0.7794 |

standard errors of the metafrontier coefficients because it ignores the sampling variability from the first stage (the generated-regressor problem). The `vcov()` method implements the correction of Murphy and Topel (1985), which inflates the second-stage variance–covariance matrix to account for first-stage estimation uncertainty.

```
se_uncorrected <- sqrt(diag(vcov(fit_sto)))
se_corrected <- sqrt(diag(vcov(fit_sto, correction = "murphy-topel")))
comparison <- data.frame(
  Uncorrected = se_uncorrected,
  `Murphy-Topel` = se_corrected,
  `Ratio` = se_corrected / se_uncorrected,
  check.names = FALSE
)
knitr::kable(comparison, digits = 4, booktabs = TRUE, linesep = "",
  caption = paste("Comparison of uncorrected and",
    "Murphy--Topel corrected standard errors",
    "for the stochastic metafrontier coefficients.")
) |> ktbl()
```

The corrected standard errors in Table 4 are uniformly larger; the increase is the sampling uncertainty contributed by the generated regressors. Confidence intervals computed with `confint(fit_sto, correction = "murphy-topel")` use the corrected standard errors. The correction is computed via numerical differentiation of the second-stage log-likelihood with respect to the first-stage parameters, requiring $O(p_1 \times p_2)$ likelihood evaluations per group, where $p_1$ is the number of first-stage parameters of that group's frontier (frontier coefficients plus variance parameters) and $p_2$ is the number of second-stage metafrontier parameters. For typical metafrontier specifications (3–5 regressors) this com-

**Table 4:** Comparison of uncorrected and Murphy–Topel corrected standard errors for the stochastic metafrontier coefficients.

| | Uncorrected | Murphy-Topel | Ratio |
|---|---|---|---|
| (Intercept) | 0.1223 | 0.1328 | 1.0861 |
| log_x1 | 0.0057 | 0.0095 | 1.6759 |
| log_x2 | 0.0056 | 0.0099 | 1.7703 |

pletes in seconds, but the cost scales quadratically and may become noticeable with high-dimensional translog models.

### 5.3 Heteroscedastic SFA

The package supports heteroscedastic group frontiers through the formula interface. A two-part formula `y ~ x1 + x2 | z1` models the mean of $\sigma_{u,i}$ (for half-normal and exponential) or $\mu_i$ (for truncated-normal) as a function of environmental variables:

```
fit_het <- metafrontier(log_y ~ log_x1 + log_x2 | log_x1, data = dat,
                        group = "group", method = "sfa",
                        meta_type = "deterministic", dist = "hnormal")
coef(fit_het)
#> (Intercept)      log_x1      log_x2
#>   1.0465785   0.4897621   0.1924461
```

This specification allows the degree of inefficiency to vary with observable firm characteristics within each group, which is relevant when the variance of inefficiency is not constant across producers. Heteroscedasticity enters only the group TE scores: the metafrontier envelope depends on $\mathbf{x}'\hat{\boldsymbol{\beta}}_j$, not on variance parameters, so TGRs are unaffected while TE and TE* reflect firm-specific inefficiency.

### 5.4 Latent class metafrontier

When group membership is unobserved, the `latent_class_metafrontier()` function discovers technology classes endogenously and estimates the metafrontier simultaneously using an EM algorithm. Unlike `metafrontier()`, this function does not accept a `group` argument: class membership is determined entirely by the data. The number of latent classes can be selected using BIC via `select_n_classes()`.

```
lc <- latent_class_metafrontier(log_y ~ log_x1 + log_x2, data = dat,
                                n_classes = 3, n_starts = 5, seed = 42)
coef(lc)
#> (Intercept)      log_x1      log_x2
#>   0.8829379   0.5897582   0.2895326
```

As with all latent class stochastic frontier models, estimation may be sensitive to starting values and local optima. The EM algorithm terminates when the relative change in marginal log-likelihood between iterations falls below a user-controllable tolerance (`tol`, default $10^{-6}$) or after `max_iter` iterations (default 200). The package therefore supports multiple random starts via the `n_starts` argument (default 10) and information-criteria-based model selection via `select_n_classes()`, which fits models for each class count in `n_classes_range` and returns the BIC-optimal specification. For applied work we recommend at least $n_starts = 10$ for two or three classes, increased to 20–30 for four or more classes or when the likelihood surface is known to be multimodal.

### 5.5 Panel SFA with time-varying inefficiency

For panel data, the package supports the time-varying inefficiency specifications of Battese and Coelli (1992) (BC92) and Battese and Coelli (1995) (BC95). The BC92 model parameterises inefficiency as $u_{it} = u_i \cdot \exp[-\eta(t - T)]$, where $u_i$ is a firm-specific truncated-normal draw, $T$ is the final period of the panel, $\eta > 0$ implies declining inefficiency over time, and $\eta < 0$ implies increasing inefficiency. Unbalanced panels are supported: each firm contributes its own set of periods $\mathcal{T}_i$ to the likelihood,

and the decay term is still measured from the panel-wide final period $T$, following the unbalanced formulation of Battese and Coelli (1992).

The BC95 model instead specifies observation-specific inefficiency $u_{it} \sim N^+(\mathbf{z}_{it}'\delta, \sigma_u^2)$, where $\mathbf{z}_{it}$ is a vector of inefficiency determinants and $\delta$ is estimated jointly with the frontier in a single stage by maximum likelihood. Unlike BC92, the BC95 model treats observations as independent draws (no within-firm time dependence). It is selected with `panel_dist = "bc95"` and requires the two-part formula syntax `y ~ x1 + x2 | z1 + z2` introduced for the heteroscedastic models above.

```
psim <- simulate_panel_metafrontier(
  n_groups = 2, n_firms_per_group = 30, n_periods = 5,
  eta = 0.05, seed = 42
)
pdat <- psim$data

fit_panel <- metafrontier(log_y ~ log_x1 + log_x2, data = pdat,
                          group = "group", method = "sfa",
                          meta_type = "deterministic",
                          panel = list(id = "firm", time = "year"),
                          panel_dist = "bc92")
coef(fit_panel)
#> (Intercept)      log_x1      log_x2
#>   1.0308915   0.5246135   0.2988751
tgr_summary(fit_panel)
#>   Group   N      Mean           SD       Min        Q1    Median
#> 1    G1 150 1.0000000 1.439400e-13 1.0000000 1.0000000 1.0000000
#> 2    G2 150 0.6095756 1.122649e-02 0.5705535 0.6009255 0.6098201
#>          Q3       Max
#> 1 1.0000000 1.0000000
#> 2 0.6169438 0.6365206
```

All estimated parameters, including the decay parameter $\eta$ and the variance parameters, are available with `coef(fit_panel, which = "group", extraPar = TRUE)`, and the group tables printed by `summary()` report $\eta$ with its standard error.

Panel SFA can be combined with either `meta_type = "deterministic"` (as above) or `meta_type = "stochastic"`; in the latter case the group-specific time-varying fits are fed into the second-stage stochastic metafrontier exactly as in the cross-sectional case.

### 5.6 Malmquist productivity decomposition

The `malmquist_meta()` function computes the metafrontier Malmquist TFP index with the three-way decomposition of Equation (10). Firms are matched across periods by the identifier supplied through the `id` argument; observations without a match in the adjacent period (unbalanced panels, entry and exit, or firms that switch groups) are dropped from the affected transition with a warning that reports the count. When `id` is omitted the function falls back to matching by row position within each group, which is only valid for balanced panels sorted identically in every period, and says so in a message. Under variable, decreasing, or increasing returns to scale, cross-period DEA programs can be genuinely infeasible when an observation lies outside the comparison technology; such cases yield `NA` (never infinite values), are excluded from reported means, and are counted in a consolidated warning and in the printed output. Constant returns to scale (used below) avoids the issue. The SFA path (`method = "sfa"`) approximates metafrontier distances by the pointwise maximum over the estimated group frontiers and announces this in a message; the DEA path computes exact distances to the pooled frontier and is the default. The identity $M^* = \text{TEC} \times \text{TGC} \times \text{TC}^*$ can be verified numerically.

```
pdat_lev <- within(pdat, {
  y  <- exp(log_y)
  x1 <- exp(log_x1)
  x2 <- exp(log_x2)
})
malm <- malmquist_meta(y ~ x1 + x2, data = pdat_lev,
                       group = "group", time = "year", id = "firm",
                       method = "dea", rts = "crs")
print(malm)
#>
```

```
#> Metafrontier Malmquist TFP Index
#> ================================
#> Method:       dea
#> Orientation:  output
#> RTS:          crs
#> Groups:       G1, G2
#> Periods:      1 -> 2 -> 3 -> 4 -> 5
#> Observations: 240
#>
#> Mean decomposition (M* = TEC x TGC x TC*):
#>   MPI  = 1.084
#>   TEC  = 1.029
#>   TGC  = 1.055
#>   TC*  = 1.028
```

Figure 5 displays the Malmquist decomposition over time. Each point is plotted at the end period of its transition, so the first point, at period 2, is the change from period 1 to period 2.

```
autoplot(malm, which = "mpi_trend")
```

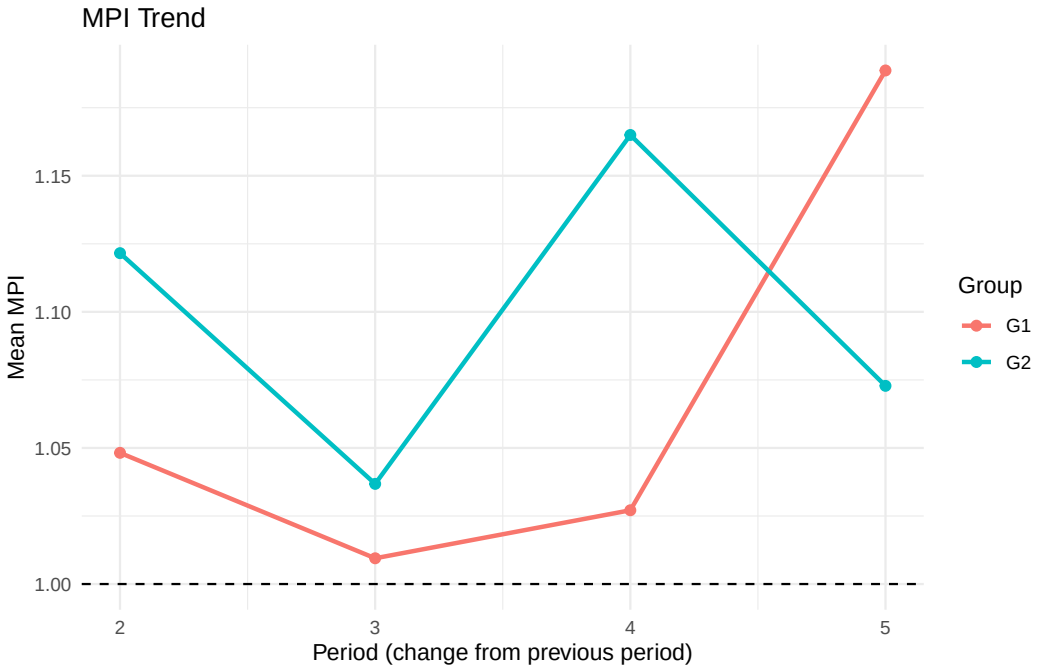



**Figure 5:** Mean metafrontier Malmquist productivity index (MPI) by period and group (generated with the package autoplot() method, view: MPI trend). Each point sits at the end period of its transition; values above the dashed line indicate improvement relative to the previous period.

## 6 Monte Carlo evidence

A simulation study evaluates the finite-sample performance of the deterministic, stochastic, and DEA metafrontier estimators.

### 6.1 Design

We study two data-generating processes. **Scenario A** follows the group SFA specification of Equation (2) with two groups ($J = 2$), 100 firms per group, and known true parameters: $\boldsymbol{\beta}^* = (1.0, 0.5, 0.3)'$ (intercept, $\beta_1$, $\beta_2$), technology gap $\delta = (0, 0.4)$, inefficiency standard deviations $\sigma_u = (0.2, 0.3)$, and noise standard deviation $\sigma_v = 0.15$. Log inputs are drawn independently and uniformly on $[0, 5]$, identically across groups, and the groups differ only in their frontier intercepts, so a common log-linear

metafrontier exists and coefficient recovery is well defined. **Scenario B** relaxes the features of Scenario A that are unrealistic in applications: the three groups have group-specific *slope* coefficients (not merely intercept shifts), log inputs are correlated within firm (correlation 0.6, a firm-size effect), and the group input distributions differ in location (one group is intensive in each input). Under group-specific slopes no single log-linear frontier tightly envelops all groups; the true envelope is the pointwise maximum $\max_j \mathbf{x}_i'\boldsymbol{\beta}_j$, the true TGR $\exp(\mathbf{x}_i'\boldsymbol{\beta}_g - \max_j \mathbf{x}_i'\boldsymbol{\beta}_j)$ varies across observations within a group, and the estimated metafrontier coefficients converge to a pseudo-true tightest linear envelope that depends on the input distribution. Scenario B is therefore evaluated on the recovery of the efficiency quantities (TE, TE*, TGR) rather than on coefficient bias. Each scenario is replicated 500 times, and in each replication we estimate the deterministic SFA, stochastic SFA, and DEA metafrontiers. The complete simulation code for this section, including both scenario loops and the table construction, is provided in the supplementary replication script (Targets 3 and 7 to 10).

### 6.2 Parameter recovery

Table 5 reports the bias, Monte Carlo standard error (MC SE), standard deviation, and root mean squared error (RMSE) of the metafrontier coefficient estimates across the 500 replications.

**Table 5:** Monte Carlo results for metafrontier coefficient estimation (500 successful replications out of 500). MC SE is the Monte Carlo standard error of the mean estimate; RMSE is the root mean squared error.

| Parameter | Method | True | Bias | MC SE | SD | RMSE |
|---|---|---|---|---|---|---|
| beta_1 | Deterministic | 0.5 | 0.0005 | 0.0006 | 0.0135 | 0.0135 |
| beta_1 | Stochastic | 0.5 | 0.0008 | 0.0006 | 0.0145 | 0.0145 |
| beta_2 | Deterministic | 0.3 | 0.0000 | 0.0006 | 0.0143 | 0.0143 |
| beta_2 | Stochastic | 0.3 | -0.0002 | 0.0006 | 0.0143 | 0.0143 |
| Intercept | Deterministic | 1.0 | -0.0269 | 0.0042 | 0.0933 | 0.0971 |
| Intercept | Stochastic | 1.0 | -0.1679 | 0.0052 | 0.1159 | 0.2040 |

All 500 replications complete without error and with optimiser convergence code zero at every stage, which is expected given the well-behaved Cobb–Douglas DGP with moderate noise levels; convergence failures are more likely with empirical data featuring collinear regressors or very small samples. In applied work, the `check_convergence()` function and the convergence block printed by `summary()` let users verify all estimation stages before interpreting TGRs, confidence intervals, or productivity decompositions. Figure 6 displays the distribution of coefficient bias across replications for both estimators.

Because the `objective` argument offers both identification criteria of Battese et al. (2004), we also refit the deterministic metafrontier under the squared-deviations QP objective on the first 50 Scenario A designs. The maximum absolute difference between LP- and QP-based TGRs is $2.5 \times 10^{-12}$: on this design the binding-constraint pattern coincides and the two criteria are numerically indistinguishable, although they can diverge at extreme input mixes.

Both estimators recover the true slope coefficients ($\beta_1$, $\beta_2$) with negligible bias. The stochastic metafrontier intercept shows a persistent downward bias because the composed error $v^* - u^*$ has non-zero mean $-\sigma^*\sqrt{2/\pi}$ under a half-normal specification, so the second-stage SFA absorbs part of the intercept into the inefficiency term. This is a structural feature of the two-step stochastic metafrontier (Huang et al., 2014): slope coefficients remain consistent as $n \to \infty$, but the intercept does not. Applied users interested in level-unit intercept interpretation should add back $\hat{\sigma}^*\sqrt{2/\pi}$ or rely on the deterministic metafrontier, whose intercept is unbiased in our Monte Carlo.

### 6.3 Efficiency recovery

Table 6 reports, for each of the three estimators under Scenario A, the mean estimated technical efficiency against the own-group frontier (TE), the mean metafrontier efficiency (TE*), and the mean technology gap ratio (TGR), together with the corresponding true means and the Spearman rank correlations between estimated and true values.

Two features of Table 6 require care in interpretation. First, the DEA rows should be read as robustness of the framework under a noisy DGP: with $\sigma_v > 0$, observations with favourable noise draws define the DEA frontiers, so DEA efficiency levels are contaminated by construction and only their rankings are informative. Second, under the intercept-only Scenario A the true TGR takes only

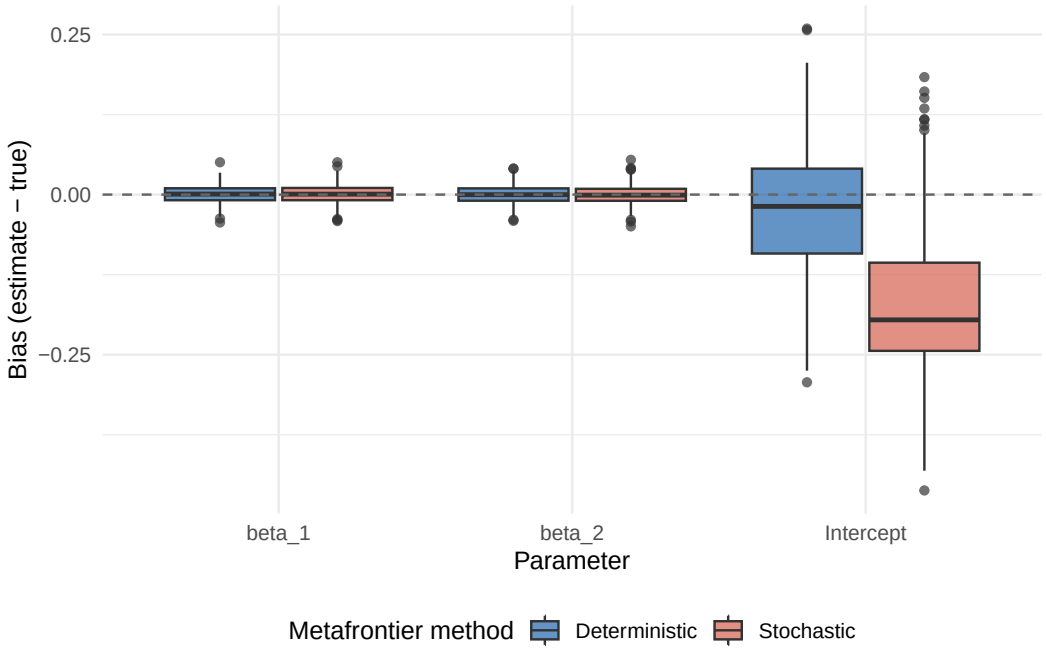



**Figure 6:** Distribution of metafrontier coefficient bias across 500 Monte Carlo replications for the deterministic and stochastic estimators (drawn with ggplot2; code in the supplementary materials). Boxes show the interquartile range and median; whiskers extend to 1.5 times the IQR.

**Table 6:** Efficiency recovery under Scenario A (means over 500 replications). TE is efficiency against the own-group frontier, TE* against the metafrontier, and TGR the technology gap ratio; Corr denotes the Spearman rank correlation between estimated and true values within a replication.

| Estimator | TE (true) | TE (est) | Corr(TE) | TE* (true) | TE* (est) | Corr(TE*) | TGR (true) | TGR (est) | Corr(TGR) |
|---|---|---|---|---|---|---|---|---|---|
| Deterministic SFA | 0.829 | 0.850 | 0.575 | 0.697 | 0.717 | 0.897 | 0.835 | 0.840 | 0.883 |
| Stochastic SFA | 0.829 | 0.850 | 0.575 | 0.697 | 0.827 | 0.880 | 0.835 | 0.969 | 0.866 |
| DEA (VRS) | 0.829 | 0.891 | 0.502 | 0.697 | 0.830 | 0.730 | 0.835 | 0.929 | 0.833 |

two distinct values (one per group), so the TGR rank correlation mostly reflects the between-group ordering; Scenario B below provides within-group TGR variation and hence a sharper test.

Table 7 reports the same statistics under Scenario B (three groups with group-specific slopes, correlated inputs, and group-specific input distributions), where the true TGR varies across observations within every group. Of the 500 replications, 500 complete successfully for all three estimators.

**Table 7:** Efficiency recovery under Scenario B: three groups with group-specific slopes, correlated log inputs, and group-specific input means (means over successful replications). The true TGR is computed against the pointwise maximum over the group frontiers; column definitions follow the Scenario A table.

| Estimator | TE (true) | TE (est) | Corr(TE) | TE* (true) | TE* (est) | Corr(TE*) | TGR (true) | TGR (est) | Corr(TGR) |
|---|---|---|---|---|---|---|---|---|---|
| Deterministic SFA | 0.828 | 0.845 | 0.547 | 0.766 | 0.513 | 0.644 | 0.925 | 0.608 | 0.761 |
| Stochastic SFA | 0.828 | 0.845 | 0.547 | 0.766 | 0.852 | 0.661 | 0.925 | 1.010 | 0.794 |
| DEA (VRS) | 0.828 | 0.898 | 0.502 | 0.766 | 0.810 | 0.566 | 0.925 | 0.902 | 0.731 |

Under Scenario B the estimated metafrontier coefficients converge to a pseudo-true tightest linear envelope rather than to a data-generating parameter vector, so no coefficient-bias table is reported. The efficiency rows in Table 7 show what this implies in practice. A single log-linear envelope cannot bend around three crossing group frontiers, so it lies above the pointwise-maximum envelope over much of the input space, and the deterministic TGR levels are understated by construction (mean 0.608 against a true mean of 0.925); the residual-based stochastic TGR centres near one regardless of the truth (mean 1.010). Rankings survive in both cases, with TGR rank correlations between 0.76 and 0.79. DEA imposes no functional form and recovers the TGR levels almost without bias (mean 0.902) at similar rank correlations. This contrast is the practical content of Scenario B: when group frontiers differ in slope and not merely in level, the DEA-based metafrontier is the safer choice for TGR levels,

while the SFA-based variants still order firms and groups reliably.

### 6.4 Small-sample and unbalanced-group robustness

To assess performance under more challenging conditions, we repeat the Scenario A simulation with 30 firms per group (vs. 100 in the baseline), keeping all other DGP parameters unchanged.

With only 30 observations per group, 500 of 500 replications complete successfully (here and below, "successfully" means the estimation finished without error; non-zero optimiser codes are recorded separately and can be inspected with `check_convergence()`). The deterministic estimator shows slope biases of 0 ($\beta_1$) and -0.002 ($\beta_2$), modestly larger than the baseline. The TGR rank correlation remains high at 0.869. The framework ranks group technologies reliably even with small samples.

Empirical metafrontier applications frequently involve groups of markedly different sizes, and the group frontiers are estimated separately, so a small group's frontier (and hence its TGR) can be unstable even when the total sample is large. To assess this, we simulate a strongly unbalanced design with 200 firms in the frontier-defining group and only 20 firms in the lagging group (a 10:1 ratio, with the technology gap on the small group), keeping all other Scenario A parameters unchanged.

Of the 500 replications, 500 complete without error and 500 additionally have optimiser convergence code zero at every stage. The deterministic estimator's slope biases (0 for $\beta_1$, -0.001 for $\beta_2$) remain modest, and the overall TGR rank correlation is 0.598. The instability concentrates where expected: the mean TGR bias in the 20-firm group is -0.019. The small group's own frontier estimate drives this bias, since the pooled metafrontier is estimated from all 220 observations. The large group shows no comparable bias. Applied users with severely unbalanced groups should therefore interpret the small groups' TGRs with caution and lean on the bootstrap intervals of `boot_tgr()`, which reflect the group-specific sample sizes.

## 7 Comparison with related packages

Table 8 provides a feature-level comparison of **metafrontier** with related R packages for efficiency analysis. All compared packages are on CRAN; **smfa** is the CRAN release (April 2026) of the GitHub package **metafrontieR** cited in the introduction. The table deliberately includes rows on which **metafrontier** offers nothing: the standard (single-technology) Malmquist index, the smoothed DEA score bootstrap of Simar and Wilson, and the sample selection metafrontier are all better served elsewhere.

**Table 8:** Feature comparison across R packages for efficiency and metafrontier analysis. N/A means not applicable (the package is a model engine, not a metafrontier framework); notes (a) and (b) are explained in the text below.

| Feature | metafrontier | smfa | deaR | Benchmarking | frontier | sfaR | sfa |
|---|---|---|---|---|---|---|---|
| Deterministic metafrontier | LP + QP | LP + QP | No | No | No | No | No |
| Stochastic metafrontier | Yes | Yes | No | No | No | No | No |
| DEA-based metafrontier | Yes | No | Manual (a) | Manual (a) | No | No | No |
| Latent class metafrontier | Yes | Yes | No | No | No | No | No |
| Metafrontier Malmquist (O'Donnell et al. 2008) | Yes | No | No | No | No | No | No |
| Standard Malmquist index | No | No | Yes | Yes | No | No | No |
| Bootstrap TGR inference | Yes | No | No | No | No | No | No |
| DEA score bootstrap (Simar–Wilson) | No | No | Yes | Yes | No | No | No |
| Murphy–Topel correction | Yes | No | No | No | No | No | No |
| Panel SFA (time-varying) | Yes | No | No | No | Yes | No | Yes |
| Unbalanced panel SFA | Yes | No | No | No | Yes | No | Yes |
| Sample selection MF | No | Yes | No | No | No | No | No |
| Directional distance functions | Yes (b) | No | Yes | Yes | No | No | No |
| Hyperbolic (graph) efficiency | Yes | No | No | Yes | No | No | No |
| FDH technology | Yes | No | Yes | Yes | No | No | No |
| Slack analysis | Yes | No | Yes | Yes | No | No | No |
| ggplot2 visualisation | Yes | No | Yes | No | No | No | No |
| External model import | SFA fits | sfaR only | No | No | N/A | N/A | N/A |
| S3 model methods | 38 | 10 | Yes | Yes | Yes | Yes | Basic |

Two cells of Table 8 carry notes. Note (a): pooled-reference scoring can be assembled manually in **deaR** and **Benchmarking** by scoring against a user-supplied reference set, but neither package forms, summarises, or provides inference for technology gap ratios. Note (b): custom direction vectors and firm-specific direction matrices are supported; **deaR** (`model_dir()`) and **Benchmarking** (`DIRECT`) also accept custom directions.

The **metafrontier** package is, to our knowledge, the only R package that combines SFA-based and DEA-based metafrontier analysis with the metafrontier Malmquist decomposition and TGR-specific

inference in a single framework. Several complementary strengths of the other packages deserve acknowledgement. The **smfa** package (Owili, 2026) offers both deterministic criteria and two stochastic formulations, implements latent class group frontiers via **sfaR**'s `sfalcmcross()`, and uniquely provides the sample selection metafrontier of Greene (2010) for non-random group composition. The **deaR** package (Coll-Serrano et al., 2023) provides a broad DEA toolkit (including fuzzy DEA and the Simar–Wilson bootstrap) that extends well beyond metafrontier analysis. The **sfaR** (Dakpo et al., 2023) and **frontier** (Coelli and Henningsen, 2020) packages remain leading engines for stochastic frontier estimation, with **sfaR** in particular supporting ten inefficiency distributions against the three implemented here; the newer **sfa** package (Bernstein et al., 2026) provides a recent alternative with flexible panel specifications. The **metafrontier** package's interoperability and engine layers are designed to complement rather than replace these estimators.

What does unification add beyond convenience? Two things. First, the three estimators return identical S3 objects, so cross-method robustness checks, as demonstrated in the real-data illustration of Section 2.4.6, require no glue code; discrepancies between a deterministic envelope, a stochastic second stage, and a nonparametric pooled frontier are themselves diagnostic. Second, and more fundamentally, the technology gap ratio is a *generated* quantity: it compares two estimated frontiers, so its sampling uncertainty is not available from any single-frontier package. The inference layer built around it (the TGR bootstrap of `boot_tgr()` and the Murphy–Topel correction for the stochastic metafrontier's generated-regressor problem) exists, to our knowledge, in no other R package, and is the part of the workflow that cannot be assembled from the engines in Table 8.

## 8 Summary and future directions

The **metafrontier** package provides a unified R framework for metafrontier production function analysis. Its principal contribution is the integration of three metafrontier approaches (deterministic, stochastic, and DEA) with the metafrontier Malmquist decomposition, bootstrap inference, Murphy–Topel correction, and latent class estimation under a consistent S3 interface. All simulations complete in under ten minutes in the computational environment described in Section 2.3.4, and Section 2.4 demonstrates the package on real data from **sfaR**; additional empirical examples are provided in the package vignette (`vignette("metafrontier")`).

Three limitations should be noted. First, the SFA-based metafrontiers require the researcher to commit to a parametric functional form (for example Cobb–Douglas or translog) through the model formula, and the same form is imposed on every group frontier and on the metafrontier envelope; the package performs no specification search, so a misspecified form propagates from the group frontiers into the estimated technology gap ratios. The DEA-based estimators avoid this choice, at the cost of ignoring statistical noise. Second, the package estimates production frontiers only ($\varepsilon = v - u$); cost frontier support ($\varepsilon = v + u$) is planned. Third, in the SFA-based Malmquist decomposition (`method = "sfa"` in `malmquist_meta()`), the distance of each observation to the metafrontier is approximated by evaluating all estimated group frontiers at that observation's inputs and taking the largest predicted output, rather than by re-estimating the enveloping metafrontier of Equation (4) in each period. This pointwise maximum coincides with the metafrontier of O'Donnell et al. (2008) wherever one group frontier dominates all others, but can understate it where group frontiers cross, so the resulting TGC and $\text{TC}^*$ components are approximate; the DEA-based Malmquist (the default) computes exact distances to the pooled frontier and should be preferred when an exact decomposition is required. The function documents this approximation and announces it at run time. Planned extensions include a sample selection metafrontier, spatial SFA metafrontiers, a Bayesian estimator propagating first-stage uncertainty, and multiplicatively complete, transitive (proper) TFP indexes such as the Färe–Primont and Lowe indexes (O'Donnell, 2012, 2018).

The package is available on CRAN (`https://CRAN.R-project.org/package=metafrontier`) and is developed on GitHub at `https://github.com/iik1/metafrontier`.

## Acknowledgements and declaration of tool use

The author thanks the editor and three anonymous reviewers for comments that substantially improved both the package and this article. Large language model tools (Anthropic Claude) were used to assist with drafting portions of the manuscript text and package documentation and with code review. All methods, code, results, and the final wording were written and verified by the author and remain the author's sole responsibility.

## References


D. Aigner, C. A. K. Lovell, and P. Schmidt. Formulation and estimation of stochastic frontier production function models. *Journal of Econometrics*, 6(1):21–37, 1977. doi: 10.1016/0304-4076(77)90052-5. [p419]

G. E. Battese and T. J. Coelli. Prediction of firm-level technical efficiencies with a generalized frontier production function and panel data. *Journal of Econometrics*, 38(3):387–399, 1988. doi: 10.1016/0304-4076(88)90053-X. [p420]

G. E. Battese and T. J. Coelli. Frontier production functions, technical efficiency and panel data: With application to paddy farmers in India. *Journal of Productivity Analysis*, 3(1–2):153–169, 1992. doi: 10.1007/BF00158774. [p432, 433]

G. E. Battese and T. J. Coelli. A model for technical inefficiency effects in a stochastic frontier production function for panel data. *Empirical Economics*, 20(2):325–332, 1995. doi: 10.1007/BF01205442. [p432]

G. E. Battese and D. S. P. Rao. Technology gap, efficiency, and a stochastic metafrontier function. *International Journal of Business and Economics*, 1(2):87–93, 2002. URL https://ideas.repec.org/a/ijb/journl/v1y2002i2p87-93.html. [p420]

G. E. Battese, D. S. P. Rao, and C. J. O'Donnell. A metafrontier production function for estimation of technical efficiencies and technology gaps for firms operating under different technologies. *Journal of Productivity Analysis*, 21(1):91–103, 2004. doi: 10.1023/B:PROD.0000012454.06094.29. [p418, 419, 420, 425, 435]

D. H. Bernstein, C. F. Parmeter, and A. Stead. *sfa: Stochastic Frontier Analysis*, 2026. URL https://CRAN.R-project.org/package=sfa. R package version 1.0.4. [p418, 438]

P. Bogetoft and L. Otto. *Benchmarking with DEA, SFA, and R*. Springer, New York, 2011. doi: 10.1007/978-1-4419-7961-2. [p418]

A. Charnes, W. W. Cooper, and E. Rhodes. Measuring the efficiency of decision making units. *European Journal of Operational Research*, 2(6):429–444, 1978. doi: 10.1016/0377-2217(78)90138-8. [p421]

T. Coelli and A. Henningsen. *frontier: Stochastic Frontier Analysis*, 2020. URL https://CRAN.R-project.org/package=frontier. R package version 1.1-8. [p418, 438]

V. Coll-Serrano, V. Bolós, and R. Benítez Suárez. *deaR: Conventional and Fuzzy Data Envelopment Analysis*, 2023. URL https://CRAN.R-project.org/package=deaR. R package version 1.5.4. [p419, 438]

K. H. Dakpo, Y. Desjeux, and L. Latruffe. *sfaR: Stochastic Frontier Analysis Using R*, 2023. URL https://CRAN.R-project.org/package=sfaR. R package version 1.0.0. [p418, 438]

W. H. Greene. A stochastic frontier model with correction for sample selection. *Journal of Productivity Analysis*, 34(1):15–24, 2010. doi: 10.1007/s11123-009-0159-1. [p419, 438]

C. J. Huang, T.-H. Huang, and N.-H. Liu. A new approach to estimating the metafrontier production function based on a stochastic frontier framework. *Journal of Productivity Analysis*, 42(3):241–254, 2014. doi: 10.1007/s11123-014-0402-2. [p418, 420, 426, 430, 435]

J. Jondrow, C. A. K. Lovell, I. S. Materov, and P. Schmidt. On the estimation of technical inefficiency in the stochastic frontier production function model. *Journal of Econometrics*, 19(2–3):233–238, 1982. doi: 10.1016/0304-4076(82)90004-5. [p420]

A. Kneip, L. Simar, and P. W. Wilson. Testing hypotheses in nonparametric models of production. *Journal of Business & Economic Statistics*, 34(3):435–456, 2016. doi: 10.1080/07350015.2015.1049747. [p427]

K. Konis and F. Schwendinger. *lpSolveAPI: R Interface to lp_solve*, 2024. URL https://CRAN.R-project.org/package=lpSolveAPI. R package version 5.5.2.0-17.12. [p420, 424]

S. C. Kumbhakar and C. A. K. Lovell. *Stochastic Frontier Analysis*. Cambridge University Press, Cambridge, 2000. doi: 10.1017/CBO9781139174411. [p420]

V. H. Moreira and B. E. Bravo-Ureta. Technical efficiency and metatechnology ratios for dairy farms in three southern cone countries: A stochastic meta-frontier model. *Journal of Productivity Analysis*, 33 (1):33–45, 2010. doi: 10.1007/s11123-009-0144-8. [p418]

K. M. Murphy and R. H. Topel. Estimation and inference in two-step econometric models. *Journal of Business & Economic Statistics*, 3(4):370–379, 1985. doi: 10.1080/07350015.1985.10509471. [p419, 421, 431]

C. J. O'Donnell. An aggregate quantity framework for measuring and decomposing productivity change. *Journal of Productivity Analysis*, 38(3):255–272, 2012. doi: 10.1007/s11123-012-0275-1. [p422, 438]

C. J. O'Donnell. *Productivity and Efficiency Analysis: An Economic Approach to Measuring and Explaining Managerial Performance*. Springer, Singapore, 2018. doi: 10.1007/978-981-13-2984-5. [p422, 438]

C. J. O'Donnell, D. S. P. Rao, and G. E. Battese. Metafrontier frameworks for the study of firm-level efficiencies and technology ratios. *Empirical Economics*, 34(2):231–255, 2008. doi: 10.1007/s00181-007-0119-4. [p418, 419, 420, 421, 422, 438]

S. Olieko. *metafrontieR: Stochastic and Deterministic Metafrontier Analysis*, 2026. URL https://github.com/SulmanOlieko/metafrontieR. R package version 1.0.0. [p419]

S. O. Owili. *smfa: Stochastic Metafrontier Analysis*, 2026. URL https://CRAN.R-project.org/package=smfa. R package version 1.0.0. [p419, 438]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2025. URL https://www.R-project.org/. [p418]

L. Simar and V. Zelenyuk. Statistical inference for aggregates of Farrell-type efficiencies. *Journal of Applied Econometrics*, 22(7):1367–1394, 2007. doi: 10.1002/jae.991. [p427]

B. A. Turlach and A. Weingessel. *quadprog: Functions to Solve Quadratic Programming Problems*, 2019. URL https://CRAN.R-project.org/package=quadprog. R package version 1.5-8. [p420, 424]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, New York, 2016. doi: 10.1007/978-3-319-24277-4. [p428]

*Erik Enstad*
*NHH Norwegian School of Economics*
*Department of Business and Management Science*
*Bergen, Norway*
*ORCID: 0009-0006-1053-3849*
erik.enstad@nhh.no

# Bioconductor Notes, September 2026

*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-sixth year, with funding for core development and infrastructure maintenance continuing through current NIH NHGRI support (NIH NHGRI 2U24HG004059). Additional support is provided by NIH NCI, Chan-Zuckerberg Initiative, National Science Foundation, Microsoft, and Amazon. In this news report, we highlight recent core team and project activities.

## 2 Software

The current Bioconductor release remains version 3.23. Development is ongoing towards the next Bioconductor release, scheduled for October 2026.

## 3 Community and Impact

### 3.1 Community Team Updates

The Bioconductor Community Team continues to expand developer and community engagement activities.

The monthly Developer Forum continues to bring together package developers and community members to discuss technical topics, share expertise, and identify opportunities for collaboration across the Bioconductor ecosystem. Recent sessions have included Aaron Lun discussing the future of the Orchestrating Single-Cell Analysis (OSCA) books and Sebastian Fischer introducing the {anvl} package. Upcoming sessions will address topics including R package dependency management, performance optimisation, and integration of R and Nextflow workflows.

Developer engagement activities also continued through community hackathons. Following BioC2026 in Seattle, a two-day post-conference hackathon brought together developers and contributors to collaborate on package development, documentation, infrastructure, and community projects.

The Bioconductor Africa seminar series, led by Laurah Nyasita Ondari, continues to provide regular online seminars for researchers across Africa. Recent sessions have covered genome-wide association studies (GWAS) and multi-omics analysis, while upcoming seminars focus on single-cell data analysis. Recordings and registration are available through the Bioconductor Training website.

Laurah Nyasita Ondari has also presented recent Bioconductor community-building activities in Africa at international Research Software Engineering events, including RSAfrica26 and RSECon26.

### 3.2 Regional Community Activities

The BiocAsia working group continues preparations for BioCAsia 2026 while developing regional engagement activities. Details of upcoming seminars and community events are available through the Bioconductor events calendar.

## 4 Conferences and Workshops

### 4.1 Upcoming

- **Bioconductor Microbiome Analysis Workshop (Addis Ababa):** A Bioconductor workshop on microbiome data analysis will be held in Addis Ababa, Ethiopia, in November 2026. The event is being organized in collaboration with local and international partners and will provide training in reproducible microbiome analysis workflows using Bioconductor. More information is available on the workshop webpage.
- **BioCAsia 2026:** BioCAsia 2026 will be held as part of the ABACBS conference in Melbourne, 19-20 November 2026. For more information, visit the conference website.
- **EuroBioC2027:** The 2027 European Bioconductor Conference will take place in Basel, Switzerland, 8-10 September 2027.
- **BioC2027:** The 2027 North American Bioconductor Conference will take place in Boston, USA, in Summer 2027. Dates will be announced through Bioconductor social media and Zulip when available.

### 4.2 Recap

- **BioC2026:** The North American Bioconductor Conference was held in August 2026 in Seattle, USA. The conference brought together researchers, package developers, educators, and infrastructure contributors from across the Bioconductor community. In addition to scientific presentations and workshops, the event included a post-conference hackathon focused on collaborative software development, technical infrastructure, training materials, and community initiatives. Recordings of talks are available on the Bioconductor YouTube channel.
- **BioFAIR 2026 Sprint:** During 15-17 September 2026, Bioconductor contributors participated in the BioFAIR Workflow Interoperability Sprint in Milton Keynes, UK, alongside collaborators from the Galaxy and Nextflow communities. The sprint focused on identifying opportunities for collaboration, improving interoperability between tools and workflows, and developing plans for future joint activities across the open-source bioinformatics ecosystem.

## 5 Boards and Working Groups Updates

The annual call for new members of the Technical Advisory Board (TAB) and Community Advisory Board (CAB) closed on 31 August 2026. New members will be selected and onboarded over the coming months.

## 6 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 Zenodo Bioconductor community contains materials from conferences, such as slides and posters, and other community events.
- 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 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*