

The Journal

Volume 18/2, June 2026

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

Contents

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

Contributed Research Articles

binspp: An R Package for Bayesian Inference for Neyman-Scott Point Processes with Complex Inhomogeneity Structure	5
rPAex: R Package for the Automatic Detection of Experimental Units in Precision Agriculture	23
Frame to frame interpolation for high-dimensional data visualisation using the woylier package.	41
highlightr: Creating Highlighted Text Based on Source Frequency	55
nimblewomble: An R package for Bayesian Wombling with ‘nimble’	71
exametrika: Test Data Engineering with R	85
DeepLearningCausal: R Package for Estimating Treatment Effects Using Deep Neural Networks and Ensemble Learning	103
SFDesign: An R Package for Space-Filling Designs	124
ProfileLadder: Functional-Based Reserving	139
blocking: An R Package for Blocking of Records for Record Linkage and Deduplication	162
NMAforest: An R Package for Generating Forest Plots in Network Meta-Analysis with Decomposition of Direct, Indirect, and Study-Level Evidence Contributions	176
Kerasnip: A Bridge Between Keras and Tidymodels	188
Conformal Prediction Tools in R	203
CovEsts: Nonparametric Autocovariance Estimation and Analysis	223
deseats: An R Package for Data-Driven Trend and Seasonality Estimation in Time Series.	246

News and Notes

Bioconductor Notes, July 2026	295
Changes on CRAN	299
News from the Forwards Taskforce	301
R Foundation News	303

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 2 of the R Journal. This issue features fifteen research articles, plus news from the R Foundation, Forwards Taskforce, Bioconductor and changes from CRAN.

This issue highlights the continued growth and diversity of the R ecosystem, bringing together contributions that advance statistical methodology, computational tools, and domain-specific applications. New packages introduce methods for Bayesian inference for complex spatial point processes ([binspp](#)) and spatial boundary analysis ([nimblewomble](#)), nonparametric autocovariance estimation ([CovEsts](#)), data-driven trend and seasonality estimation ([deseats](#)), space-filling designs for computer experiments ([SFDesign](#)), and functional-data-based insurance reserving ([ProfileLadder](#)). Developments in modern machine learning and uncertainty quantification are represented by [DeepLearningCausal](#), which combines deep neural networks, ensemble learning, and conformal prediction for causal inference, alongside a review of conformal prediction tools available in R. The issue also features software for transparent evidence synthesis in network meta-analysis ([NMAforest](#)), educational test data engineering ([exametrika](#)), precision agriculture using multispectral imagery ([rPAex](#)), scalable record linkage ([blocking](#)), and text comparison and visualisation ([highlightr](#)). Complementing these methodological advances are improvements to data visualisation through [woylier](#), which introduces a new interpolation approach for high-dimensional tours, and software interoperability through [kerasnip](#), which bridges [keras3](#) and the [tidy-models](#) ecosystem. Collectively, these articles demonstrate the continued strength of the R community in developing accessible, extensible, and practically useful software that supports modern statistical analysis across a wide range of disciplines.

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

2026 Rousseeuw Prize for Statistics Awarded to the R Core Team

The Rousseeuw Prize, named after statistician Peter Rousseeuw, recognises major contributions that have transformed the practice of statistics and its impact on society. The 2026 Rousseeuw Prize for Statistics has been awarded to five members of the R Core Team in recognition of their decades-long work developing and maintaining R, the open-source statistical computing language that has become foundational to modern data science.

The 2026 prize recipients are **Brian D. Ripley** (University of Oxford), **Martin Maechler** (ETH Zurich), **Kurt Hornik** (WU Vienna University of Economics and Business), **Peter Dalgaard** (Copenhagen Business School), and **Luke Tierney** (University of Iowa). They are recognised for their sustained contributions to the R project; half of the prize is awarded to these five individuals, with the remaining half shared among other members of the R Core Team in acknowledgement of the collective nature of the work.



Over nearly three decades, the R Core Team and contributors have built R into a global infrastructure for statistical computing, enabling reproducible research and lowering barriers to advanced analytics. Today, R underpins thousands of research projects and educational programs worldwide and is widely used by organisations including regulatory agencies, pharmaceutical companies, and central banks. Congratulations to the R Core Team on this well-deserved recognition!

Emi Tanaka
Australian National University

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

binspp: An R Package for Bayesian Inference for Neyman-Scott Point Processes with Complex Inhomogeneity Structure

by Jiří Dvořák, Radim Remeš, Ladislav Beránek and Tomáš Mrkvička

Abstract The Neyman-Scott point process is a widely used point process model which is easily interpretable and easily extendable to include various types of inhomogeneity. The inference for such complex models is then complicated and fast methods, such as minimum contrast method or composite likelihood approach do not provide accurate estimates or fail completely. Therefore, we introduce Bayesian MCMC approach for the inference of Neyman-Scott point process models with inhomogeneity in any or all of the following model components: process of cluster centers, mean number of points in a cluster, spread of the clusters. We also extend the Neyman-Scott point process to the case of overdispersed or underdispersed cluster sizes and provide a Bayesian MCMC algorithm for its inference. The R package **binspp** provides these estimation methods in an easy to handle implementation, with detailed graphical output including traceplots for all model parameters and further diagnostic plots. All inhomogeneities are modelled by spatial covariates and the Bayesian inference for the corresponding regression parameters is provided.

1 Introduction

The Neyman-Scott point process (Neyman and Scott, 1958) is a cluster point process model widely used in biology, astronomy, forestry, medicine, etc. Its main advantages are the straightforward interpretation of the model parameters and the closed form of moment properties, at least for the stationary version of the model.

The model can be constructed in two stages and it is often called a doubly stochastic process: first, cluster centers are randomly sampled from a given Poisson point process; second, conditionally on the positions of cluster centers, a random number of offspring points are randomly and independently placed around each cluster center. Thus, the stationary Neyman-Scott point process model is specified by the intensity of the Poisson process of cluster centers, the distribution of the number of points per cluster (hereafter called cluster size), and the distribution of the relative displacement of the offspring points around their respective cluster centers (hereafter called cluster spread).

In the most popular type of Neyman-Scott point process, called the (modified) Thomas process (Thomas, 1949), the cluster size is assumed to follow a Poisson distribution and the cluster spread is governed by a radially symmetric Gaussian distribution. The stationary Thomas process is then described by the following parameters: the intensity κ of the process of cluster centers, the mean number of points in a cluster α and the standard deviation ω of the Gaussian distribution determining cluster spread.

A natural way of introducing inhomogeneity into the Neyman-Scott process model is allowing some of the model components (intensity of cluster centers κ , cluster size α , cluster spread ω) to depend on a set of spatial covariates. Their significance then needs to be assessed. Several models of this type have been studied in the literature, as discussed in the following paragraphs.

The Neyman-Scott point process with inhomogeneous cluster centers, with the distribution of the clusters being the same in terms of size and spread, allows for varying the number of clusters in the space. The inference for this process was investigated in Mrkvička et al. (2014) and it was found that the Bayesian MCMC estimation procedure is more precise than composite likelihood or minimum contrast method. This model is not second-order inhomogeneity reweighted stationary (SOIRS) (Baddeley et al., 2000), but if instead the mean number of points in a cluster α is inhomogeneous, the resulting process is very close to SOIRS. The inference for SOIRS Neyman-Scott process (stationary Neyman-Scott point process thinned by a spatially varying function) can be performed by a two step method based on minimum contrast or composite likelihood (Waagepetersen and Guan, 2009), as implemented in the R package **spatstat** (Baddeley et al., 2015).

The inhomogeneity can be also introduced in the cluster spread ω , then the process will be locally-scaled Neyman-Scott point process (Hahn et al., 2003). It is also possible to introduce inhomogeneity simultaneously in α and ω , then the process can be called a Neyman-Scott point process with growing clusters (Mrkvička, 2014).

For the models with homogeneous population of cluster centers but inhomogeneity in the cluster properties we talk about cluster inhomogeneity. On the other hand, for the Neyman-Scott point process with inhomogeneous cluster centers we talk about inhomogeneity of centers. Combining the cluster inhomogeneity and inhomogeneity of centers leads to the notion of doubly inhomogeneous Neyman-Scott point process. The Bayesian MCMC inference for such a process was studied in [Mrkvička and Soubeyrand \(2017\)](#). Other kinds of inference were found to be useless for such a complex model.

Therefore, we build up the R package [binspp](#) which contains the Bayesian MCMC estimation procedure for the most general model with inhomogeneity in all three model components. This model contains all the previous models as special cases, including the stationary one. The radially symmetric Gaussian distribution is assumed to determine the cluster spread, which does not limit practical applicability of the models. We do not include SOIRS Neyman-Scott point process due to the different construction of the model and also due to availability of moment-based estimation methods. However, practically speaking the Neyman-Scott point process with inhomogeneous cluster size α can be used as an approximation to the SOIRS Neyman-Scott point process model.

The most important statistical problem here is to assess the dependence of the data on the given set of covariates. Our package handles the spatial covariates influencing any or all of the model components: the intensity of the cluster centers κ , the cluster size α and the cluster spread ω . The Bayesian MCMC procedure is time consuming, but its great benefit is that the significance of all covariates is provided from the estimated posterior distributions in a natural way. For example, in case of the inhomogeneity of cluster centers if a faster estimation method is used, it is necessary to perform parametric bootstrap in order to obtain the significance of the covariates. This is as time consuming as the Bayesian MCMC procedure ([Mrkvička et al., 2014](#)). Only in the case of SOIRS Neyman-Scott processes it is possible to use the fast estimation method and the significance of covariates can be assessed using the asymptotic normality result obtained in [Waagepetersen and Guan \(2009\)](#).

As mentioned above, the minimum contrast approach and the composite likelihood approach are available for SOIRS Neyman-Scott point process. They are also described in [Mrkvička et al. \(2014\)](#) for the Neyman-Scott point process with inhomogeneous cluster centers. To the best of our knowledge, they are not available for the other kinds of inhomogeneity discussed above.

In order to model inhomogeneous clustered point patterns log-Gaussian Cox process (LGCP) models are often used ([Møller et al., 1998](#)). The inference for SOIRS LGCP is well developed with integrated nested Laplace approximation (INLA) ([Rue et al., 2009](#)) available through the package [inlabru](#) ([Bachl and Lindgren, 2020](#)), with Bayesian MCMC inference available through the package [lgcp](#) ([Taylor et al., 2020](#)) or with the moment methods available through the package [spatstat](#). Nevertheless, none of these packages allows for inference for LGCP with more complex types of inhomogeneity. An attempt in this direction has been made in [Dvořák et al. \(2019\)](#).

With these considerations in mind, we have implemented the core functions of the [binspp](#) package using [Rcpp](#) to speed up the computation. As a result, short runs of the chain, useful for tuning up the hyperparameters of the prior distributions, are finished within minutes on a regular laptop. Long runs, used for the actual estimation, may be finished within a few hours, see the detailed example in Section 2.4.5. This makes our implementation easily applicable in practice, without the need to worry about the computational demands.

The inference for Neyman-Scott point processes is usually performed with the assumption of Poisson distribution of the number of points in a cluster. The same is assumed in all models discussed above, but the [binspp](#) package contains also the Bayesian MCMC estimation procedure for homogeneous generalised Neyman-Scott process. The method was described in [Andersson and Mrkvička \(2020\)](#). This model uses the generalised Poisson distribution (GPD) as a distribution of the number of points in a cluster. The GPD allows for modelling of under- or over-dispersion. This allows for more flexible modelling of the distribution of the number of points in a cluster.

The package specifically considers the following models. The inhomogeneous Thomas point process allowing for modelling of cluster centers, cluster spread and cluster sizes through covariates. The covariates are incorporated in every model component via exponential model. Such a general process was not presented in the literature, even [Mrkvička and Soubeyrand \(2017\)](#) presents only a special case of this general model. Furthermore, the generalised Neyman-Scott point process which allows for modelling underdispersed or overdispersed cluster sizes is considered.

This paper is organized as follows. First, in Section 2.2 we describe the models considered here. Then we briefly describe the algorithms in Section 2.3. In Section 2.4 we show the use of our package for different types of models, with a detailed example in Section 2.4.5 considering a real dataset of infected oak trees. This is a part of a larger dataset studied in [Fernández-Habas et al. \(2019\)](#). The model for the observed point pattern includes inhomogeneity in all three components of the model and for illustration we provide the outputs of a long run of the MCMC chain. We also show in Section 2.4.7 the use of the estimation procedure for homogeneous generalised Neyman-Scott point process and the possibility of detecting over- or under-dispersion of cluster sizes. Section 2.5 is left for discussion.

2 Models

Let us first describe the model in its full generality, i.e. the doubly inhomogeneous Neyman-Scott point process. The process of cluster centers C follows an inhomogeneous Poisson point process with intensity function $\kappa f(\beta, u)$, $u \in W \subset \mathbb{R}^2$, where $\kappa > 0$ and $\beta \in \mathbb{R}^k$ are parameters. The clusters $X_c, c \in C$, are independently attached to every cluster center c . The Neyman-Scott point process is the superposition of the clusters $X = \cup_{c \in C} X_c$, where X_c are independent Poisson point processes with intensity function $\alpha(\mu, c)k(u - c, \omega(v, c))$, $u \in \mathbb{R}^2$, which depends on c . Here $\alpha(\mu, c)$ is the expected number of offspring points in the cluster corresponding to the parent point c and $\mu \in \mathbb{R}^{l+1}$ is a parameter. Furthermore, $k(\cdot, \omega(v, c))$ is the probability density function governing the relative displacement of the offspring points around the parent point c (in this paper we assume k is the density of the centered radially symmetric normal distribution with the standard deviation $\omega(v, c)$). The distribution of the number of points in the cluster with the cluster center c is assumed to be Poisson for all inhomogeneous models, with the probabilities being denoted $p(n, \alpha(\mu, c))$.

The parametric functions f , α and ω are the key ingredients of the model which describe the dependence on the spatial covariates. These functions are assumed to take the following parametric form:

$$\begin{aligned} f(\beta, u) &= \exp(\beta_1 z_1(u) + \dots + \beta_k z_k(u)), \\ \alpha(\mu, c) &= \exp(\beta_0^\alpha + \beta_1^\alpha z_1^\alpha(c) + \dots + \beta_l^\alpha z_l^\alpha(c)), \\ \omega(v, c) &= \exp(\beta_0^\omega + \beta_1^\omega z_1^\omega(c) + \dots + \beta_m^\omega z_m^\omega(c)). \end{aligned}$$

Here $z_1, \dots, z_k$ are the spatial covariates influencing the population of cluster centers, $z_1^\alpha, \dots, z_l^\alpha$ are the spatial covariates influencing the cluster size and $z_1^\omega, \dots, z_m^\omega$ are the spatial covariates influencing the cluster spread. All the β s are real-valued regression parameters. Note that the model for f does not contain the intercept since its role is taken by the parameter κ . We use this parametrization to be consistent with the earlier works describing the models and the corresponding Bayesian inference (Mrkvička, 2014; Kopecký and Mrkvička, 2016; Mrkvička and Soubeyrand, 2017).

Specific choices of k, l, m result in different special cases of the general model. For $k = 0, l = 0, m = 0$ the process is the stationary Neyman-Scott process. For $k > 0, l = 0, m = 0$ we obtain the inhomogeneous cluster centers. Similarly, $k = 0, l > 0, m = 0$ results in inhomogeneous cluster sizes, while $k = 0, l = 0, m > 0$ leads to locally scaled process (inhomogeneous cluster spread). For $k = 0, l > 0, m > 0$ we obtain the process with growing clusters (Mrkvička, 2014) and finally for $k > 0, l > 0, m > 0$ we obtain the most general, doubly inhomogeneous process (Mrkvička and Soubeyrand, 2017).

Our package allows for all the possible choices of k, l, m . However, the user must be aware of the possible identifiability issues which occur if the same covariate is used for z_i and z_j^α for some i and j , i.e. if the same covariate influences both f and α . It is a property of the two-step estimation algorithm described in the next section that the parameters β_i and β_j^α cannot be estimated correctly in this case. For example, if $\beta_i = 0$ and $\beta_j^\alpha \neq 0$ then the first step of the algorithm will estimate $\hat{\beta}_i$ to be approximately β_j^α .

The **binspp** package allows also for the estimation of the homogeneous generalised Neyman-Scott point process, where the distribution of the cluster sizes follows the generalised Poisson distribution, which is a popular model for count data (Wang and Famoye, 1997). The probability mass function of GPD is

$$p(n|\lambda, \theta) = \begin{cases} \frac{1}{n!} \theta (\theta + \lambda n)^{n-1} e^{-\theta - \lambda n}, & n = 0, 1, 2, \dots, \\ 0, & \text{if } n > \tilde{n} \text{ when } \lambda < 0, \end{cases}$$

where $\theta > 0$, $|\lambda| < 1$, and $\tilde{n}$ is the largest integer such that $\theta + \lambda \tilde{n} > 0$ when $\lambda < 0$. When $\lambda < 0$, $p(n|\lambda, \theta)$ does not sum to 1, and needs to be renormalized. The expectation is equal to $\alpha = \frac{\theta}{1-\lambda}$, and variance is equal to $\frac{\theta}{(1-\lambda)^3}$. The parameter $\lambda \in [-1, 1]$ in GPD models the over- or under-dispersion, $\lambda = 0$ corresponds to the Poisson case, $\lambda > 0$ to the over-dispersed and $\lambda < 0$ to the under-dispersed case.

3 Inference

The inference for the doubly inhomogeneous Neyman-Scott point process observed in a bounded observation window W is performed in two steps, similar to the approach of Waagepetersen and Guan (2009).

In the first step, the parameters $\bar{\beta} = (\log(\lambda), \beta_1, \dots, \beta_k)$ of the intensity function are estimated, based on the assumption that X is the Poisson process with the intensity function

$$\bar{f}_{\bar{\beta}}(u) = \exp(\bar{z}(u)\bar{\beta}^T), \quad u \in \mathbb{R}^2, \quad (1)$$

where $\bar{z}(u) = (1, z_1(u), \dots, z_k(u))$. This assumption is intuitively justified if the range of interaction among the points is small compared to the range of changes in the spatial covariates and if z_i is different from z_j^α for all combinations of i and j . Specifically, we maximize the log-likelihood of the assumed inhomogeneous Poisson process:

$$l(\bar{\beta}) = \sum_{x \in X \cap W} \bar{z}(x)\bar{\beta}^T - \int_W \exp(\bar{z}(u)\bar{\beta}^T) du \quad (2)$$

Here W is the observation window.

The second step consists of estimation of the interaction parameters μ and ν , conditionally on the estimate of $\bar{\beta}$.

Bayesian estimation for the homogeneous Neyman-Scott point processes was carried out with an MCMC algorithm e.g. in [Guttorp and Thorarinsdottir \(2012\)](#); [Møller and Waagepetersen \(2007\)](#); [Mrkvička \(2014\)](#); [Kopecký and Mrkvička \(2016\)](#). In this approach, the cluster centers and the model parameters are updated in each step of the MCMC algorithm. After reaching the equilibrium, posterior distributions of the parameters can be estimated. The cluster centers are generally viewed as nuisance parameters.

Considering the inhomogeneous clusters, the MCMC algorithm proceeds in the same way as in the homogeneous case, except that the likelihood is influenced by the parameters connected with the cluster inhomogeneity. We remark here that the estimation algorithm was not presented in such generality earlier, even [Mrkvička and Soubeyrand \(2017\)](#) considered only a special case of this model, without allowing for the general form with covariates. However, the generalisation is straightforward.

Let C denote the inhomogeneous Poisson point process of cluster centers with the intensity $\kappa f(\beta, u)$. Let $p(C|\kappa, \beta)$ denote the Poisson probability density function of the point process C , conditionally on κ and β , with respect to the distribution of the unit-rate homogeneous Poisson point process. Furthermore, let $p(X|C, \beta, \kappa, \mu, \nu)$ denote the Poisson probability density function of the point process X under the knowledge of C , β , κ , μ and ν . The joint posterior distribution of the process C and the parameters is then

$$p(C, \kappa, \mu, \nu|X) \propto p(X|C, \beta, \kappa, \mu, \nu) p(C|\kappa, \beta) p(\mu) p(\nu), \quad (3)$$

where $p(\mu)$ and $p(\nu)$ denote the prior probability density functions for the respective parameters. No prior for κ is required because it is, in our estimation procedure including a modification similar to the one proposed by [Kopecký and Mrkvička \(2016\)](#), a deterministic function of μ and β . Indeed, the expected number $\mathbb{E}M$ of the observed points in the observation window W is equal to

$$\kappa \int_W \alpha(\mu, u) \left[\int_{\mathbb{R}^2} k(u - c, \omega(\nu, c)) f(\beta, c) dc \right] du,$$

which can be approximated by

$$\mathbb{E}M \approx \kappa \int_W \alpha(\mu, u) f(\beta, u) du.$$

Thus, in each iteration of the MCMC algorithm, κ can be re-computed when μ is updated.

Our MCMC algorithm consists of updating the process of cluster centers C and updating the parameters μ and ν . For updating C we use the birth-death-move algorithm described in [Møller and Waagepetersen \(2004\)](#). For updating μ and ν we use the Metropolis-Hastings algorithm. In order to obtain better mixing properties μ and ν are updated separately. Full details about this algorithm can be found in [Mrkvička and Soubeyrand \(2017\)](#).

The inference for the interaction parameters μ and ν is performed from the estimated posterior distributions which are obtained from the MCMC samples after appropriate burn-in. The inference about the first-order inhomogeneity parameters β cannot be obtained from the first step where the Poisson distribution is assumed. Therefore, we base the inference about β on the posterior distribution of the cluster centers obtained in the second step. In every step the significance of $z_1, \dots, z_k$ with respect to the process of cluster centers C is assessed using the Poisson distribution of C assumed in this model. The median of respective p -values computed from the posterior distribution is taken to be the estimate of the p -value of the test of significance of the given covariate. We remark that it is also possible to perform the full Bayesian estimation by considering the inhomogeneity of cluster centers

in the MCMC procedure. However, this approach was found to be less efficient than the two-step approach due to identifiability issues in the full likelihood.

Considering the generalised Neyman-Scott process, the MCMC algorithm consists of one extra step in addition to the traditional Metropolis-Hastings update of the model parameters and the birth-death-move update of the cluster centers. It is the update of the connections between the points and the cluster centers, since by assuming the non-Poisson distribution these connections take part in the likelihood of the process. Full details about this algorithm can be found in [Andersson and Mrkvička \(2020\)](#).

4 Examples

In this section we provide a set of examples illustrating how different types of models can be fitted using the Bayesian MCMC approach implemented in the [binspp](#) package. The first few examples illustrate the use of the package, while the most complex example in Section 2.4.5 describes the outputs of the algorithm in full detail. Model parametrization is described in Section 2.2.

Below we assume that X is the observed point pattern in the ppp format used in the [spatstat](#) package. We further assume that X is observed through the observation window W , which is a union of aligned rectangles, aligned with the coordinate axes. x_left , x_right , y_bottom and y_top are vectors giving the coordinates of the extreme points of the rectangles whose union forms the observation window W . When the observation window is rectangular, it is possible to supply it using the argument W in the estimation function instead of the vectors x_left to y_top . Furthermore, W_dil is the dilated observation window used to accommodate cluster centers outside W to mitigate the edge effects. An easy way to obtain W_dil from the vectors x_left to y_top is shown in Section 2.4.5. All covariates such as $cov1$ are assumed to be pixel images (objects of type `im` from the [spatstat](#) package) defined over the W_dil domain.

The list `control` contains important tuning constants such as the required number of iterations to be run (`NStep`), the length of the initial part of the chain to be discarded before computing estimates (`BurnIn`) or the sampling frequency used to reduce autocorrelations in the values used for computing the estimates (`SamplingFreq`). Also, hyperparameters for prior distributions for different parameters can be specified in this list, as illustrated in the detailed example in Section 2.4.5. Providing hyperparameter values guided by the knowledge of the problem at hand is highly recommended! However, some default values are used if the user does not provide them.

All priors are normal distributions, priors for β_i^α and β_i^ω , $i > 0$, have expectation 0. We remark here that also the hyperparameters for β_0^α and β_0^ω must be given in log scale because all parameters, except κ , are estimated in the exponential form.

The estimation algorithm runs an MCMC chain of all the model parameters (and the process of cluster centers). The chain converges to the equilibrium state corresponding to the posterior distribution. The values of the MCMC chain can be accessed using the `rawMCMCoutput` function, providing samples from the posterior distribution of the model parameters. The tuning constants given in the `control` argument specify how the MCMC algorithm is run. `NStep` gives the required number of steps of the chain, and `BurnIn` gives the number of initial steps to be disregarded so that the equilibrium state is reached. The equilibrium state can be determined by various diagnostic plots provided by the package. E.g. the log-likelihood does not increase anymore, the number of centers is stabilized, the trace plots of the parameters show stationary behavior, and the histograms of posterior distributions are unimodal. The value `SamplingFreq` controls how many steps of the chain are made before recording a new sample. This helps control the dependence between the sampled values. The bigger the `SamplingFreq`, the lower the correlation, but fewer samples are obtained.

4.1 Homogeneous Thomas process

First, we simulate a point pattern using the [spatstat](#) function `rThomas` and specify the observation window. In the following examples, we do not explicitly include these commands for conciseness.

```
library(spatstat)
library(binspp)
W <- square(1)
W_dil <- dilation.owin(W, 0.1)
X <- rThomas(30, 0.02, 5, W)
```

Then we set up the control parameters. Based on our experience, for this homogeneous model we recommend running at least 50 000 iterations, with burn-in of at least 25 000 steps.

```
control <- list(NStep=50000, BurnIn=25000, SamplingFreq=10)
```

The following commands provide equivalent ways of specifying that all three components of the model are homogeneous (no covariates are provided):

```
Output <- estintp(X=X, control=control, W=W, W_dil=W_dil)
Output <- estintp(X=X, control=control, W=W, W_dil=W_dil,
                  z_beta=NULL, z_alpha=NULL, z_omega=NULL)
Output <- estintp(X=X, control=control, W=W, W_dil=W_dil,
                  z_beta=list(), z_alpha=list(), z_omega=list())
```

In this example the default hyperparameter values were used. These can be retrieved in the following way:

```
Output$priorParameters
```

The text outputs and graphical outputs are obtained as follows:

```
print(Output)
plot(Output)
```

4.2 Thomas process with inhomogeneous cluster centers

In this example the population of parent points is inhomogeneous, with intensity function depending on a covariate. More covariates can be included in the model, provided they are given in the `z_beta` list, see Section 2.4.5 for an illustration.

For this model with simple inhomogeneity we recommend running at least 100 000 iterations, with burn-in of at least 50 000 steps. Note that the covariates in the list `z_beta` must be named in order for the `ppm` function from the [spatstat](#) package to run properly. We first create a simple covariate describing the x -coordinate.

```
cov1 <- as.im(function(x,y){x}, W=W_dil)
control <- list(NStep=100000, BurnIn=50000, SamplingFreq=10)
Output <- estintp(X=X, control=control, W=W, W_dil=W_dil,
                  z_beta=list(Z1=cov1))
```

4.3 Thomas process with inhomogeneous mean number of points in a cluster

Now we assume that the mean number of points in a cluster depends on the position of the corresponding parent point c , with the function $\alpha(\mu, c)$ depending on a covariate. Again, more than one covariate may be used, provided they are given in the `z_alpha` list.

For this model with simple inhomogeneity we recommend running at least 100 000 iterations, with burn-in of at least 50 000 steps.

```
control <- list(NStep=100000, BurnIn=50000, SamplingFreq=10)
Output <- estintp(X=X, control=control, W=W, W_dil=W_dil,
                  z_alpha=list(cov1))
```

4.4 Thomas process with inhomogeneous cluster spread

In this case the spread of the clusters depends on the position of the corresponding parent point c , with the function $\omega(v, c)$ depending on a covariate. As before, more than one covariate may be given in the `z_omega` list.

For this model with simple inhomogeneity we recommend running at least 100 000 iterations, with burn-in of at least 50 000 steps.

```
control <- list(NStep=100000, BurnIn=50000, SamplingFreq=10)
Output <- estintp(X=X, control=control, W=W, W_dil=W_dil,
                  z_omega=list(cov1))
```

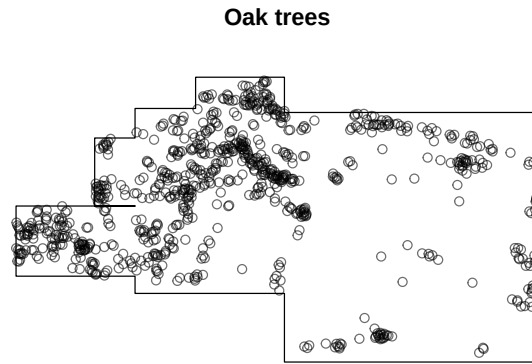



Figure 1: The oak trees dataset sampled in 2009 in a region consisting of 5 rectangles.

4.5 Thomas process with complex inhomogeneities in all model components

For this example we use the real dataset provided in the [binspp](#) package, see also [Fernández-Habas et al. \(2019\)](#). The dataset is shown in Figure 1 and it can be plotted using `plot` command as usual. The aim of this example is to show the possibility of modeling simultaneously the inhomogeneous centers of clusters, inhomogeneous cluster sizes and inhomogeneous cluster spread, using the lists of covariates `z_beta`, `z_alpha` and `z_omega`. The presented methodology is the only available methodology that allows for such a complexity.

```
X      <- trees_N4
x_left <- x_left_N4
x_right <- x_right_N4
y_bottom <- y_bottom_N4
y_top <- y_top_N4
```

Several covariates accompany the observed point pattern. In the following lists we specify which covariates are assumed to influence which model components. Note that each model component depends on two covariates. Due to identifiability reasons the lists `z_beta` and `z_alpha` must be disjoint (no covariate can appear in both lists).

```
z_beta <- list(refor=cov_refor, slope=cov_slope)
z_alpha <- list(tmi=cov_tmi, td=cov_tdensity)
z_omega <- list(slope=cov_slope, reserv=cov_reserv)
```

The observation window is given as the union of aligned rectangles, aligned with the coordinate axes.

```
W <- owin(c(x_left[1],x_right[1]),c(y_bottom[1],y_top[1]))
if(length(x_left)>=2){
  for(i in 2:length(x_left)){
    W2 <- owin(c(x_left[i],x_right[i]),c(y_bottom[i],y_top[i]))
    W <- union.owin(W,W2)
  }
}
```

The dilated observation window is obtained as follows:

```
W_dil <- dilation.owin(W,100)
```

The parameter 100 for the dilation specifies the width of the zone around `W` where centers having offsprings in `W` can occur. Since the Gaussian distribution for the offsprings is used, the width is infinite in theory, but for computational reasons we bound this region.

For this model with complex inhomogeneities we recommend running at least 250 000 iterations, with burn-in of at least 150 000 steps. Hyperparameter values for prior distributions can be specified as follows:

```
control <- list(NStep=250000, BurnIn=150000, SamplingFreq=10, Prior_alpha_mean=3,
               Prior_alpha_SD=2, Prior_omega_mean=5.5, Prior_omega_SD=5,
               Prior_alphavec_SD=c(4.25,0.012), Prior_omegavec_SD=c(0.18,0.009))
```

The following commands perform the MCMC estimation. With the required 250 000 steps of the algorithm the computation takes approx. 2.5 hours on a regular laptop due to the implementation of the core functions using the **Rcpp** package.

```
set.seed(12345)
Output <- estintp(X=X, control=control, x_left=x_left, x_right=x_right,
                 y_bottom=y_bottom, y_top=y_top, W_dil=W_dil,
                 z_beta=z_beta, z_alpha=z_alpha, z_omega=z_omega)
```

Text output, providing the parameter estimates (medians of the estimated posterior distributions) together with the corresponding 2.5% and 97.5% quantiles, is obtained by the command

```
print(Output)
```

These quantiles provide the 95% credible interval. Note that another credibility level may be specified as an argument of the function `estintp`.

Graphical output, given in Figures 2 to 7, is provided by the command

```
plot(Output)
```

First, estimated surfaces of the first-order intensity function, the $\alpha(c)$ function, describing the mean number of points in a cluster, and the $\omega(c)$ function, describing the spread of the clusters, are plotted, as illustrated in Figure 2. The estimated surfaces are plotted in the dilated window.

Then, histograms describing the estimated posterior distribution of the model parameters are plotted, see Figure 3.

The histograms for the first-order parameters are not plotted, because the point estimates of β_1 and β_2 are computed in the first step under the assumption of Poissonity. But these parameters govern the inhomogeneity of cluster centers, i.e. their significance should be computed from cluster centers only. In order to deal with this issue, we record the centers in every step of the Markov chain and also we record the significance of the covariates from the list `z_beta` with respect to the population of cluster centers in every step of the chain. Histograms of the corresponding p-values are plotted instead of the posterior histograms for these parameters, see the bottom part of Figure 3. This provides more precise inference about the significance of the covariates influencing the cluster centers than the outcomes of the first step of estimation, where the `ppm` function from the **spatstat** package is used and where the locations of the (observed) offsprings may confound the significance of the covariates with respect to the (unobserved) parent points.

Finally, traceplots for various quantities describing the state of the chain are plotted, including 1) the model parameters, 2) the p-values discussed in the previous paragraph, 3) the value of the log-likelihood itself, 4) the number of parent points in the dilated window, 5) acceptance probabilities for the proposed updates of parameters influencing $\alpha(c)$ or $\omega(c)$ (not very informative for long runs but useful when tuning the algorithm for a new dataset with shorter runs), and 6) fractions of accepted updates in the past 1000 steps of the algorithm (much more informative than plots of the acceptance probabilities). See Figures 4 to 7 for illustration. The traceplots for model parameters show also the estimated median of the posterior distribution (given by the solid red line) together with the bounds of the credible interval with the required credibility (red dashed lines). These bounds correspond to the empirical 2.5% and 97.5% quantiles of the posterior distribution when 95% credibility is chosen.

4.6 Simulation of the Thomas process with complex inhomogeneity

The Thomas process with any kind of assumed inhomogeneity can be simulated using the function `rThomasInhom`. This function generates the parent process using the `rpoispp` function from the **spatstat** package. The offspring points are simulated directly from the appropriate normal distributions.

```
W <- square(1)
W_dil <- dilation.owin(W,0.1)
cov1 <- as.im(function(x,y){x}, W=W_dil)
cov2 <- as.im(function(x,y){y}, W=W_dil)
cov3 <- as.im(function(x,y){1 - (y - 0.5) ^ 2}, W=W_dil)
Y=rThomasInhom(kappa=10, betavec=c(1), z_beta=list(cov1),
               alpha=log(10), alphavec = c(1), z_alpha=list(cov2),
               omega=log(0.01), omegavec=c(1), z_omega=list(cov3),
               W=W, W_dil=W_dil)
```

Simulation from the fitted model is also possible using the function `simulate`.

```
simulate(Output)
```

4.7 Generalised Neyman-Scott

In this subsection we specify the implementation of the Bayesian MCMC algorithm for estimation of the homogeneous generalised Thomas point process (GTPP) which was described in [Andersson and Mrkvička \(2020\)](#). In this subsection we assume that the point pattern X is an object of the format `ppp` from the `spatstat` package and that it is observed in a rectangular observation window W . The notation and priors are slightly different from those for the inhomogeneous models, due to the different notations used in the corresponding original papers.

The GTPP can be simulated using

```
kappa <- 10; omega <- .1; lambda <- .5; theta <- 10
X <- rgtp(kappa, omega, lambda, theta, win = owin(c(0, 1), c(0, 1)))
plot(X$X)
plot(X$C)
```

Here κ corresponds to the intensity of centers, ω to the standard deviation of the radially symmetric Gaussian distribution determining the spread of offsprings, and λ and θ correspond to the parameters of the GPD governing the cluster size.

The priors used in the estimation are lognormal for all parameters, except λ which has a uniform prior. The hyperparameters and control parameters must be specified in the estimation function `estgtp` itself. The function `estgtp` allows for plotting of all results and outputs of the MCMC chain.

The posterior distribution of the parameter λ can be used to determine the over- or under-dispersion. Specifically, if 0 lies in the 95% credible interval for λ , then the Poisson assumption cannot be rejected. See the example code where the result summarizes all posterior medians and credible intervals for the model parameters.

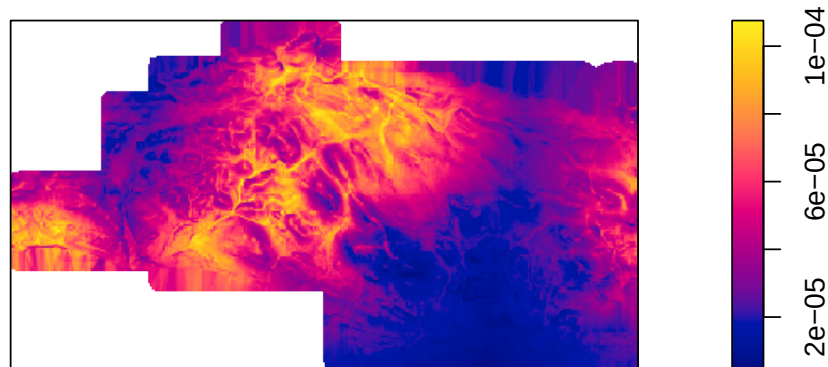
```
#Prior for parameter kappa
a_kappa <- 4
b_kappa <- 1
x <- seq(0, 100, length = 100)
hx <- dlnorm(x, a_kappa, b_kappa)
plot(x, hx, type = "l", lty = 1, xlab = "x value",
     ylab = "Density", main = "Prior")

#Prior for parameter omega
a_omega <- -3
b_omega <- 1
x <- seq(0, 1, length = 100)
hx <- dlnorm(x, a_omega, b_omega)
plot(x, hx, type = "l", lty = 1, xlab = "x value",
     ylab = "Density", main = "Prior")

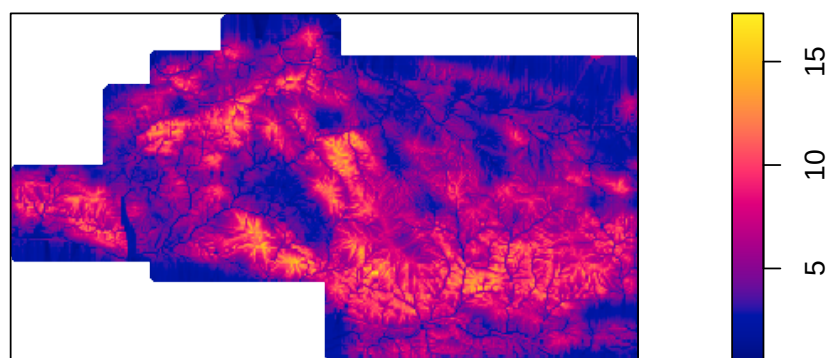
#Prior for parameter lambda
l_lambda <- -1
u_lambda <- 0.99
x <- seq(-1, 1, length = 100)
hx <- dunif(x, l_lambda, u_lambda)
plot(x, hx, type = "l", lty = 1, xlab = "x value",
     ylab = "Density", main = "Prior")

#Prior for parameter theta
a_theta <- 4
b_theta <- 1
x <- seq(0, 100, length = 100)
hx <- dlnorm(x, a_theta, b_theta)
plot(x, hx, type = "l", lty = 1, xlab = "x value",
     ylab = "Density", main = "Prior")
```

Plot of estimated first-order intensity function



Surface of $\alpha(c)$



Surface of $\omega(c)$

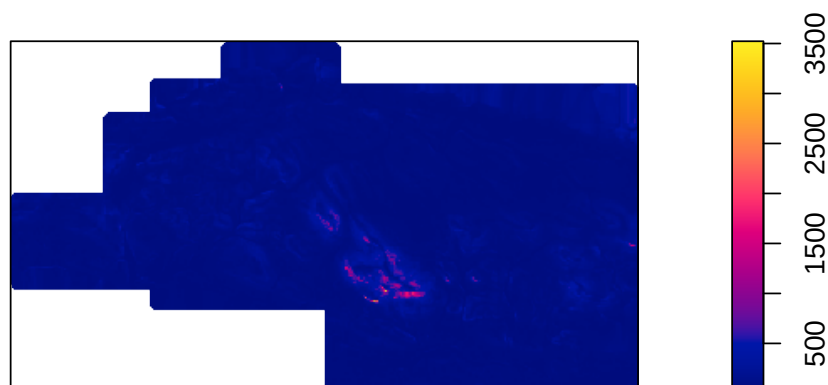


Figure 2: Plots of the estimated first-order intensity function (top), the estimated function $\alpha(c)$ describing the mean number of points in a cluster (middle) and the estimated function $\omega(c)$ describing the spread of the clusters (bottom).

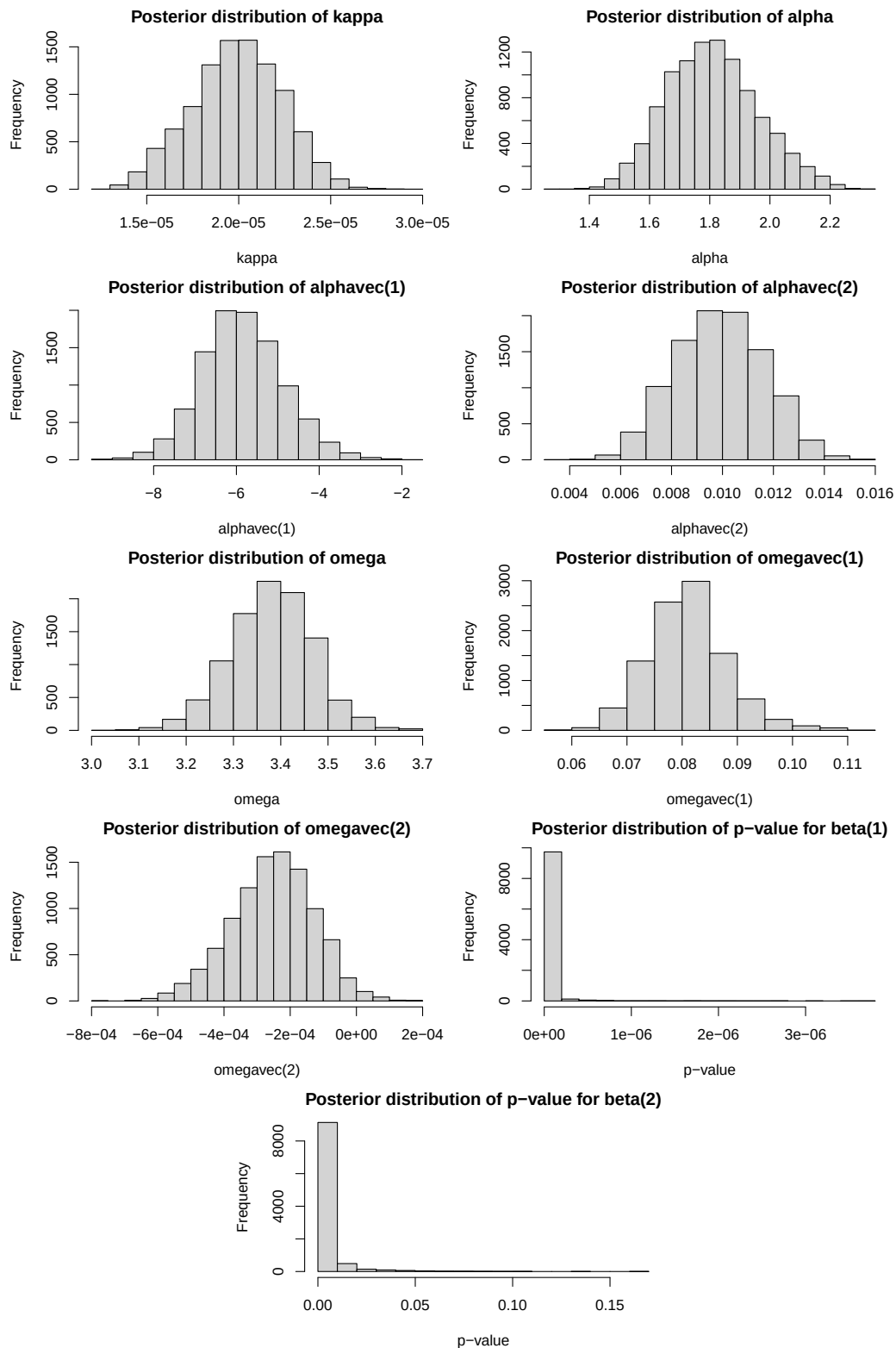


Figure 3: Histograms for estimated posterior distributions of various model parameters. Note that the parameter alpha in the histogram above corresponds to the parameters β_0^α , the parameter omega corresponds to β_0^ω and the parameters alphavec(i) corresponds to the parameters β_i^α from Section 2.2, omegavec(i) corresponds to β_i^ω and beta(i) corresponds to β_i .

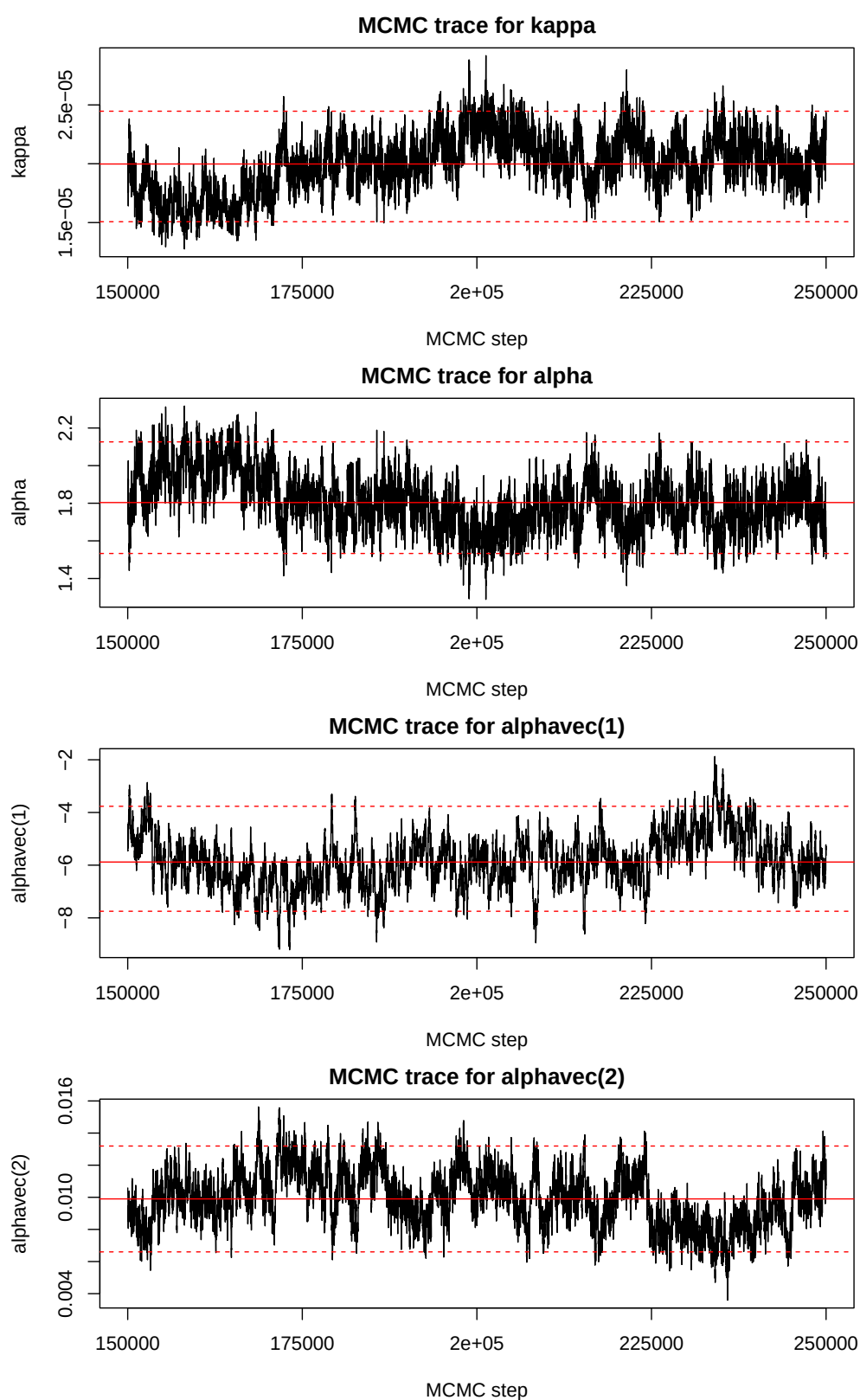


Figure 4: Traceplots for various quantities describing the state of the MCMC algorithm. Note that the parameter alpha in the traceplot above corresponds to the parameters β_0^α and the parameters alphavec(i) corresponds to the parameters β_i^α from Section 2.2

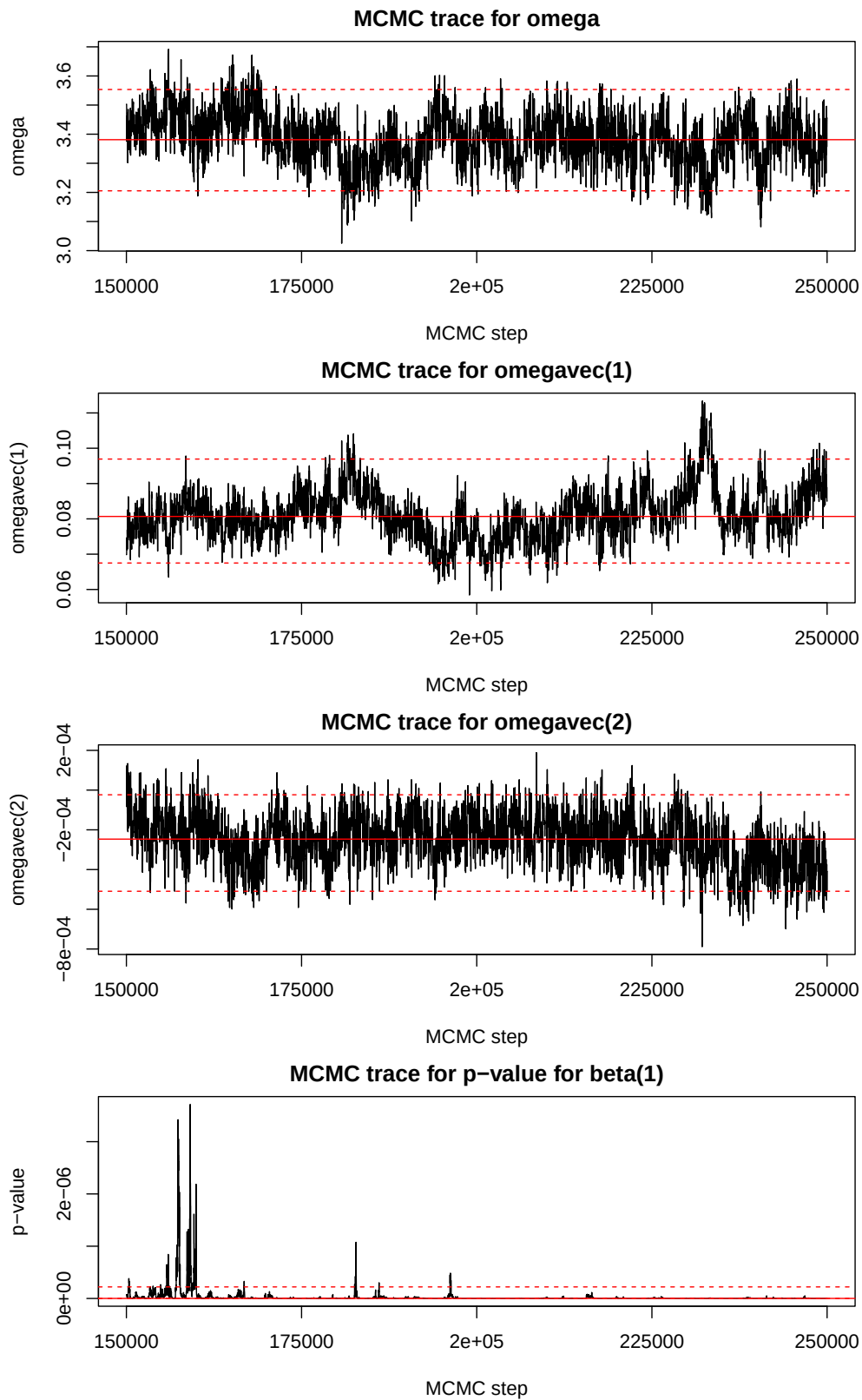


Figure 5: Traceplots for various quantities describing the state of the MCMC algorithm. Note that the parameter the parameter omega corresponds to β_0^ω and omegavec(i) corresponds to β_i^ω and beta(i) corresponds to β_i . The parameter beta(1) corresponds to the β_1 computed in the first step, nevertheless its p -value is computed in the Bayesian step.

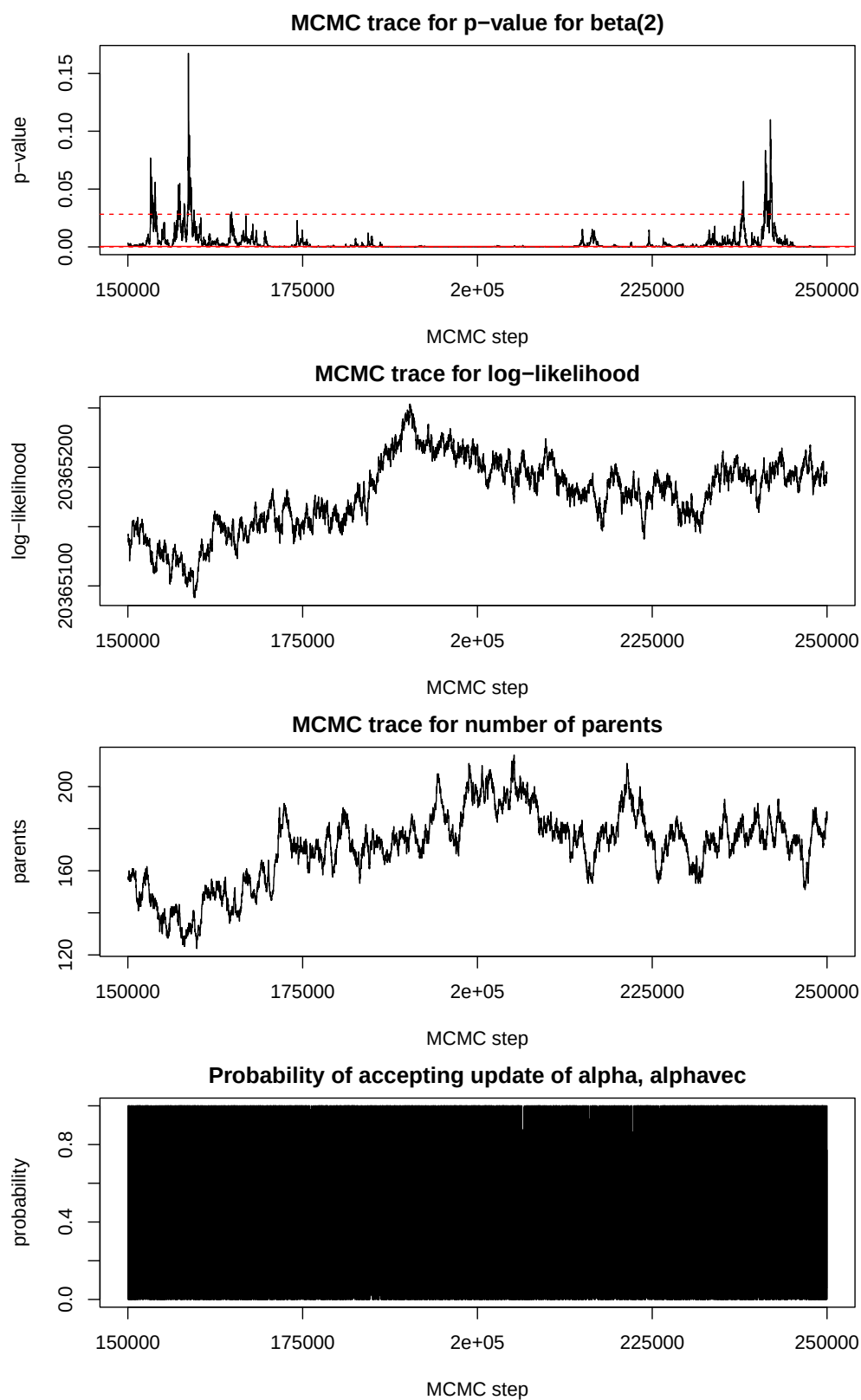


Figure 6: Traceplots for various quantities describing the state of the MCMC algorithm.

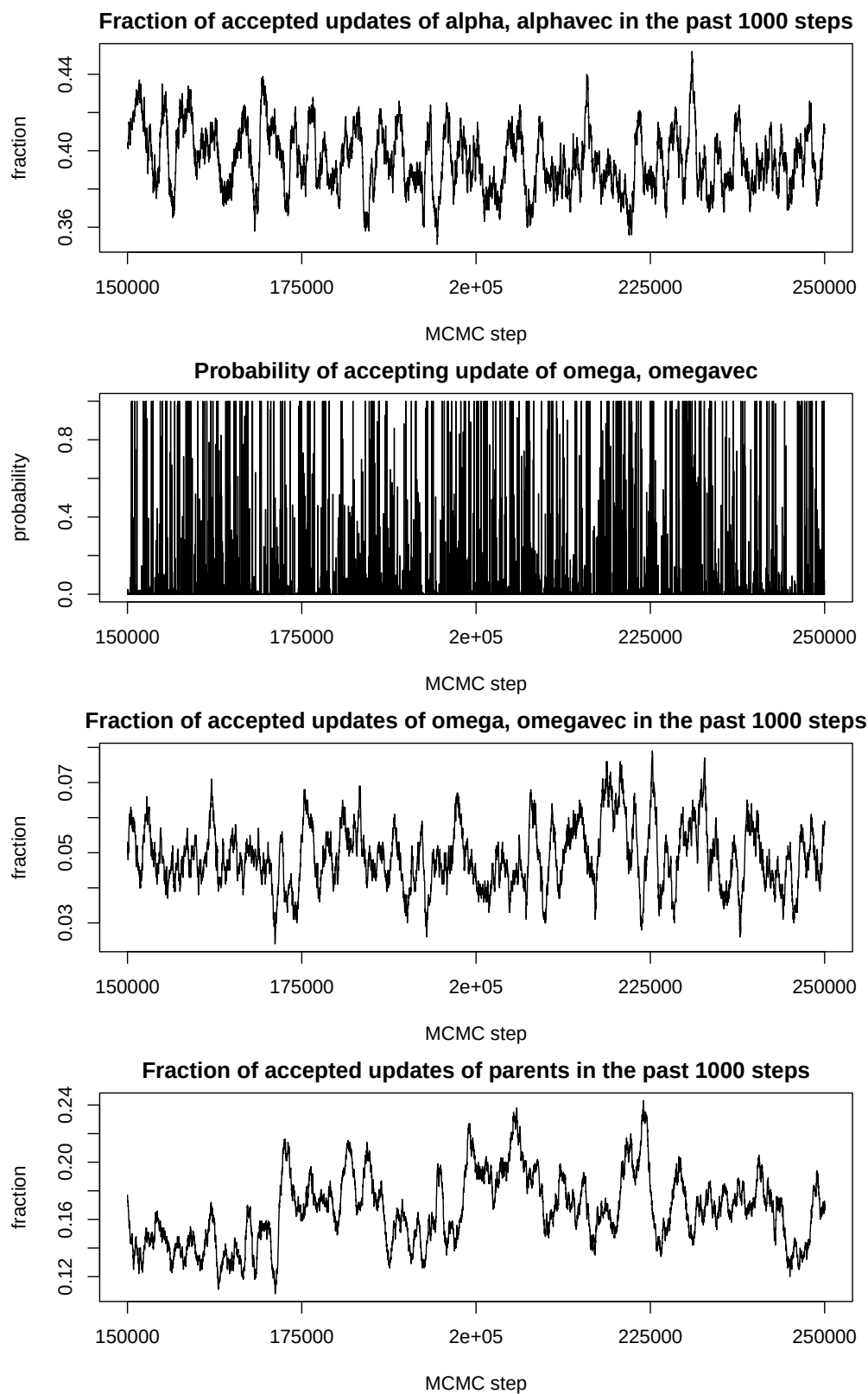


Figure 7: Traceplots for various quantities describing the state of the MCMC algorithm.

```
#estimation procedure with default values for standard deviations of updates for the parameters
est <- estgtp(X$X,
  skappa = exp(a_kappa + ((b_kappa ^ 2) / 2)) / 100,
  somega = exp(a_omega + ((b_omega ^ 2) / 2)) / 100, dlambd = 0.01,
  stheta = exp(a_theta + ((b_theta ^ 2) / 2)) / 100, smove = 0.1,
  a_kappa = a_kappa, b_kappa = b_kappa,
  a_omega = a_omega, b_omega = b_omega,
  l_lambda = l_lambda, u_lambda = u_lambda,
  a_theta = a_theta, b_theta = b_theta,
  iter = 1000, plot.step = 1000, save.step = 1e9,
  filename = "")

#plotting the results and estimation of the parameters from refined MCMC chain
discard <- 100
step <- 10

result <- estgtp(est, discard, step)
result
```

5 Summary

This paper presents a model and an implementation of the Bayesian MCMC algorithm for estimating parameters of inhomogeneous Neyman-Scott point process with inhomogeneity in cluster centers, cluster spread and/or cluster sizes. The simulation studies assessing the performance of the presented algorithm were published in [Kopecký and Mrkvička \(2016\)](#) for the homogeneous process, in [Mrkvička et al. \(2014\)](#) for inhomogeneous cluster centers and in [Mrkvička and Soubeyrand \(2017\)](#) for inhomogeneous cluster centers and cluster spread.

The presented package [binspp](#) allows also for identification of over- or under-dispersion of cluster sizes in a homogeneous model through the generalised Thomas point process. Since the algorithm is rather flexible, the package can be further extended by estimation methods for models where some cluster centers are given and fixed. It can also be extended in the direction of the spatio-temporal Neyman-Scott point processes.

A user of this package should be aware of the following issues, connected with the Bayesian MCMC algorithms. First, priors should be carefully chosen, even though we provide a sensible default. For example, the range of the ω prior must be in coherence with the size of the observation window; similarly, the range of the priors for the regression parameters connected with the covariates must reflect the range of the covariate values. Second, when slow mixing is observed via low fractions of accepted updates, one might think of changing the standard deviation for proposal distributions. These standard deviations are also provided in the package by their default value. Third, if many covariates are considered, longer chains are needed to get reasonably close to the equilibrium. The equilibrium state can be determined by the log-likelihood not increasing anymore, stabilized number of centers, stationary behaviour of the trace plots of the parameters, or by unimodal histograms of posterior distributions.

Acknowledgements

The project has been financially supported by the Grant Agency of Czech Republic (Project No. 19-04412S) and by the ERC CZ grant LL2407 of the Ministry of Education, Youth and Sport of the Czech Republic. The authors wish to express their gratitude to the editor and the anonymous referees for their insightful comments both on the text of the manuscript and the implementation of the package itself. The authors are also grateful to Begona Abellanas who initiated this project by asking for the implementation of the method and who provided the real data set.

References

- C. Andersson and T. Mrkvička. Inference for cluster point processes with over- or under-dispersed cluster sizes. *Statistics and Computing*, 30:1573–1590, 2020. URL <https://doi.org/10.1007/s11222-020-09960-8>. [p6, 9, 13]

- F. E. Bachl and F. Lindgren. *inlabru: Spatial Inference using Integrated Nested Laplace Approximation*, 2020. URL <https://CRAN.R-project.org/package=inlabru>. R package version 2.1.13. [p6]
- A. Baddeley, E. Rubak, and R. Turner. *Spatial Point Patterns: Methodology and Applications with R*. Chapman & Hall, New York, 2015. URL <http://www.crcpress.com/Spatial-Point-Patterns-Methodology-and-Applications-with-R/Baddeley-Rubak-Turner/9781482210200/>. [p5]
- A. J. Baddeley, J. Møller, and R. Waagepetersen. Non- and semi-parametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica*, 54:329–350, 2000. URL <https://doi.org/10.1111/1467-9574.00144>. [p5]
- J. Dvořák, J. Møller, T. Mrkvička, and S. Soubeyrand. Quick inference for log gaussian Cox processes with non-stationary underlying random fields. *Spatial Statistics*, 33:100388, 2019. ISSN 2211-6753. doi: <https://doi.org/10.1016/j.spasta.2019.100388>. URL <https://www.sciencedirect.com/science/article/pii/S2211675319301393>. [p6]
- J. Fernández-Habas, P. Fernández-Rebollo, M. R. Casado, A. M. García Moreno, and B. Abellanas. Spatio-temporal analysis of oak decline process in open woodlands: A case study in SW Spain. *Journal of Environmental Management*, 248:109308, 2019. URL <https://doi.org/10.1016/j.jenvman.2019.109308>. [p6, 11]
- P. Guttorp and T. L. Thorarinsdottir. Bayesian inference for non-Markovian point processes. In E. Porcu, J. M. Montero, and M. Schlather, editors, *Advances and Challenges in Space-time Modelling of Natural Events*, pages 79–102. Springer, 2012. [p8]
- U. Hahn, E. B. V. Jensen, M. N. M. van Lieshout, and L. S. Nielsen. Inhomogeneous spatial point processes by location-dependent scaling. *Advances in Applied Probability*, 35(2):319–336, 06 2003. doi: 10.1239/aap/1051201648. URL <http://dx.doi.org/10.1239/aap/1051201648>. [p5]
- J. Kopecký and T. Mrkvička. On Bayesian estimation for Neyman-Scott point processes. *Applications of Mathematics*, 61(4):503–514, 2016. [p7, 8, 20]
- J. Møller and R. P. Waagepetersen. *Statistical Inference and Simulation for Spatial Point Processes*. Number 100 in Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, 1 edition, Sept. 2004. ISBN 978-1-58488-265-7. [p8]
- J. Møller and R. P. Waagepetersen. Modern statistics for spatial point processes. *Scandinavian Journal of Statistics*, 34(4):643–684, Dec. 2007. [p8]
- J. Møller, A. R. Syversveen, and R. P. Waagepetersen. Log gaussian Cox processes. *Scandinavian Journal of Statistics*, 25(3):451–482, 1998. ISSN 03036898, 14679469. URL <http://www.jstor.org/stable/4616515>. [p6]
- T. Mrkvička. Distinguishing different types of inhomogeneity in Neyman-Scott point processes. *Methodology and Computing in Applied Probability*, 16:385–395, 2014. URL <https://doi.org/10.1007/s11009-013-9365-4>. [p5, 7, 8]
- T. Mrkvička and S. Soubeyrand. On parameter estimation for doubly inhomogeneous cluster point processes. *Spatial Statistics*, 20:191–205, 2017. URL <https://doi.org/10.1016/j.spasta.2017.03.005>. [p6, 7, 8, 20]
- T. Mrkvička, M. Muška, and J. Kubečka. Two step estimation for Neyman-Scott point process with inhomogeneous cluster centers. *Statistics and Computing*, 24:91–100, 2014. URL <https://doi.org/10.1007/s11222-012-9355-3>. [p5, 6, 20]
- J. Neyman and E. L. Scott. Statistical approach to problems of cosmology. *J Roy Stat Soc B*, 20:1–43, 1958. [p5]
- H. Rue, S. Martino, and N. Chopin. Approximate Bayesian inference for latent gaussian models by using integrated nested laplace approximations. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(2):319–392, 2009. doi: <https://doi.org/10.1111/j.1467-9868.2008.00700.x>. URL <https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9868.2008.00700.x>. [p6]
- B. M. Taylor, T. M. Davies, B. S. Rowlingson, P. J. D. A. code contributions from Edzer Pebesma, and D. Schumacher. *lgcp: Log-Gaussian Cox Process*, 2020. URL <https://CRAN.R-project.org/package=lgcp>. R package version 1.6. [p6]
- M. Thomas. A generalization of Poisson’s binomial limit for use in ecology. *Biometrika*, 36:18–25, 1949. [p5]

- R. Waagepetersen and Y. Guan. Two-step estimation for inhomogeneous spatial point processes. *Journal of the Royal Statistical Society, Series B*, 71:685–702, 2009. URL <https://doi.org/10.1111/j.1467-9868.2008.00702.x>. [p5, 6, 7]
- W. Wang and F. Famoye. Modeling household fertility decisions with generalized poisson regression. *Journal of Population Economics*, 10(3):273–283, 1997. URL <https://doi.org/10.1007/s001480050043>. [p7]

Jiří Dvořák
Charles university, Faculty of Mathematics and Physics
Sokolovská 83, 186 75 Prague
Czech Republic
(ORCID: 0000-0003-3290-8518)
dvorak@karlin.mff.cuni.cz

Radim Remeš
University of South Bohemia, Faculty of Agriculture and Technology
Studentská 1668
Czech Republic
(ORCID: 0000-0002-3306-0347)
inrem@ef.jcu.cz

Ladislav Beránek
University of South Bohemia, Faculty of Agriculture and Technology
Studentská 1668
Czech Republic
(ORCID: 0000-0001-5004-0164)
beranek@ef.jcu.cz

Tomáš Mrkvíčka
University of South Bohemia, Faculty of Agriculture and Technology
Studentská 1668
Czech Republic
(ORCID: 0000-0003-1613-2780)
mrkvicka.toma@gmail.com

rPAex: R Package for the Automatic Detection of Experimental Units in Precision Agriculture

by Felipe de Mendiburu, Augusto Choy, Rodrigo Morales, Roberto Quiroz, and David Mauricio

Abstract Experimental research in agriculture uses designs to control experimental error, analyze the applicable agronomic measures, and determine the factors of interest. Multispectral cameras in crewless vehicles have expanded the sources of experimental information, providing crop images during the different stages of crop development. We propose rPAex, a package developed in R, to facilitate the use of multispectral images in experimental designs. rPAex includes functions that allow us to (1) automate the distribution of experimental units in shape, size, and uniform spacing; (2) link the spectral information with the experimental design in the field; (3) rectify the plot coordinates in the future images of the same field; (4) provide the edge pixels of the EUs with spectral information; and (5) generate subplots in a particular EU. To demonstrate the ease of use of rPAex in precision agriculture, a case study on the spectral trend in the development of cassava, maize, and sweet potato crops, conducted by the International Potato Center, is described herein.

1 Introduction

The image processing of an agricultural field forms the basis for precision agriculture (PA) (Srinivasan, 2006), and georegistration and data acquisition through images over time are necessary in any experimental study during crop development. One of the fundamentals of image processing is the automatic detection of field status, e.g., furrow detection for efficient and robust operation of robotic equipment (Utstumo et al., 2018) and crop-weed separation (Montalvo et al., 2012). Data collected from the trials are recorded in field books, which are documents used by researchers to report analysis and hypothesis statements (Tripp, 1982). The experimental unit (EU) is the minimum unit of the material to which a treatment is applied (Bailey, 2008). So, it is necessary to identify them in the multispectral image of the experiment to relate the spectral data over time with the EUs in the field book.

Identifying the EUs is a fundamental activity in any experimental design in PA. It is essential because it allows us to discriminate spectral responses such as vegetation indices (Bendig et al., 2015), differentiate healthy potato plants from diseased ones (Griffel et al., 2018), count cotton bolls (Jung et al., 2018), etc. Nonetheless, EU identification is vulnerable to errors because its coverage may exceed the EU when the crop expands, generating unreliable information. Image processing in R can be conducted through various functions and tools. The X Window System enables the creation of graphical interfaces for image visualization and analysis. The rast function (Hijmans, 2023) allows reading images in multiple formats, while the locator function captures georeferenced coordinates directly from images. This capability remains time-invariant for .tif formats, facilitating the analysis of crop development over time (Lee, 2020).

Once the images have been processed and the geospatial data extracted, the **agricolae** package can be used to support the experimental design phase. Its design functions automatically generate field sketches and field books for different layout configurations, ensuring consistency between the analytical and experimental stages. However, the detection of EUs is not automated (Bendig et al., 2015), possibly causing confusion in the assignment of pixels to the EUs, which, in turn, produces errors in the data and results and the information integrated with the field book and the study treatments.

In this study, we introduce a tool named **rPAex** designed to automatically determine the shape and size of EUs, along with their specifications, such as edges, furrows, and spacing within an experimental area. This involves generating a spatial field book that incorporates images and assigns treatments to the experimental units based on the experimental design. The spatial field book integrates spectral information and coordinates with the experimental design, along with the variables measured in the field. rPAex has been implemented in R and is seamlessly integrated with the **terra** and **agricolae** packages.

This registration method facilitates regular flights once the unit is located, streamlining measurements over time and enabling the collection of spectral and physical data. By the end of the trial, researchers will have comprehensive information for experimental analysis. Furthermore, this approach enhances image data organization and minimizes errors that may arise in handling EU images.

The rest of this article is organized as follows: Section 2 provides a background of EU detection

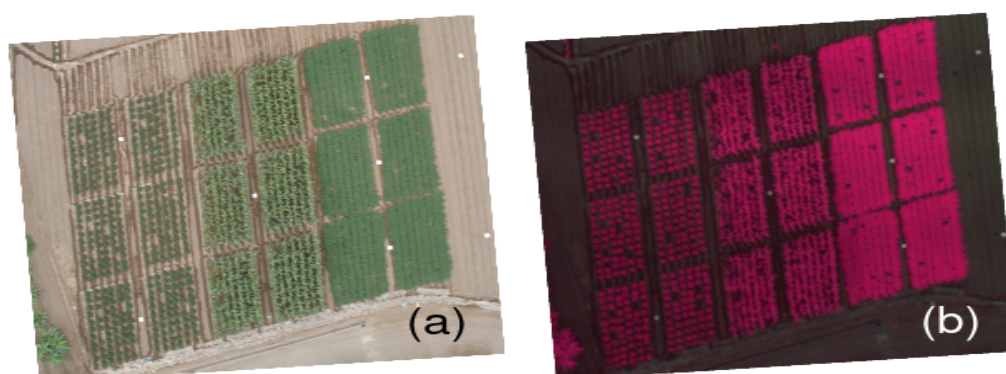


Figure 1: Experimental units(EUs): a) visible image; b) spectral image

in PA. Section 3 details the structure of the package in image processing and its use in experimental designs in multispectral images. Section 4 describes the development of the cassava crop through the data recorded in rPAex.

The following sections provide a discussion of the use of rPAex, a summary of its structure and importance, and acknowledgments about the data that were useful for the functionality of rPAex.

2 Background

Many packages are available in R to deal with the field map and its utility for an agricultural experiment; *agricolae* package performs the experimental design and provides the distribution of treatments for different experiment models but does not relate the space, size, and coordinates of the field. The *desplot* package (Wright, 2021) shows a field map in which different parts of the design are colored to map the experiment. Likewise, other R packages do not use the spectral information of the EU for analyzing the experiment.

Experiments with crops require an experimental design still in use under the current basic principles of experimentation (LeClerc et al., 1963). The EUs form the basis of the experiment in these designs. These designs have evolved with technology and currently use the geographic information system and remote sensing in what is now known as PA.

In the agronomic experiments, data are recorded on the plot dimension, corresponding to the shape, size, and number of the EUs, and the information of interest for the study, such as the number of plants, plant cover, and yield (1a). Additionally, remote sensing images containing spectral information obtained from remote sensors are added in PA. The spectral band values are recorded by each pixel of the image (Figure 1b) (Loayza et al., 2018).

Figure 1a shows the distribution of 18 plots grouped for three crops (cassava, maize, and sweet potato), covering a total area of 35 m × 45 mm. 1b shows the corresponding spectral image, with information on three bands: near-infrared (NIR) and red and green spectral bands in 18 plots with an average of 17135 pixels in 5 cm each. These bands contain information for identifying the EUs, their coordinates, and their spectral value.

Remote sensors and computer programming are widely applied in PA. One important package in R is *FIELDImageR* (Matias et al., 2020). The functionalities of the *FIELDImageR* package are aimed at treating plants in the agricultural field with spectral images. These functions allow the extraction of crop information, such as the number of plants and leaves on the plant and the quantification of the area and agricultural production.

Very few studies have explored the automation of EU detection in PA. Some studies explored the automation of EU identification, for example, the study of Willers et al. (2008), which proposes a method to define EUs in a commercial agricultural field, and the works of Montalvo et al. (2012), Guerrero et al. (2013), Pérez-Ortiz et al. (2015), Basso and Pignaton de Freitas (2020) and Zhang et al. (2018) for the automatic detection of furrows.

The objectives of the rPAex package are as follows:

- Automate the distribution and spacing of the EUs while maintaining the shape and size of each EU constant, integrating spectral data and pixel coordinates to obtain a flat file with spectral information and EU identification.
- Generate an experimental design from the field design in an image and link the spectral information with the field book.

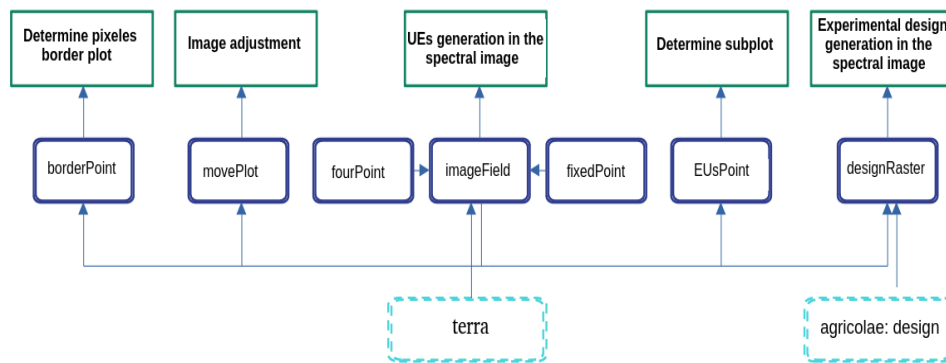


Figure 2: Package structure of rPAex

- Rectify the position of the EUs in the images captured over time; this correction is necessary when the EUs show a small deviation due to continuous flights.
- Determine the pixels of the plot edge using spectral information.
- Locate an EU and generate subplots.

To fulfill all its purposes, **rPAex** contains seven functions: `imageField`, `designRaster`, `borderPoint`, `EUsPoint`, and `movePlot` supported by `fourPoint` and `fixedPoint`. The package is organized into three layers (Figure 2). The first one corresponds to the five aims of the package, the second one to the seven functions, and the third one to the **terra** and **agricolae** packages, which are required by **rPAex** to complete its functionality.

3 Description of functions

Each function of **rPAex**, as well as their purpose, rationale, and syntax are described as follows. For a better understanding, we provide an example for each function.

Example Case

In the experimental station of the International Potato Center, three crops were studied in a 35 m × 45 m area. To illustrate the purposes of using **rPAex**, the cassava crop was considered. All crops were planted in December 2014 with different biological cycles—the cassava crop lasted eight and a half months. In all, 38 spectral images of cassava are available in the repository in three bands, namely, the near-infrared (NIR), red, and green spectral bands (Loayza et al., 2018). The vegetation indices were built based on the spectral bands; thus, the normalized difference vegetation index (NDVI), defined as the ratio of the difference between near-infrared and red reflectance to their sum $(\text{NIR} - \text{Red}) / (\text{NIR} + \text{Red})$, was used to evaluate vegetation vigor (Wójtowicz et al., 2016).

The following statements in R allow the initialization of variables and constants for the execution of the examples presented in the article.

```
library(rPAex)
library(ggplot2)
data(cassava)
vi <- with(cassava, (L1 - L2)/(L1 + L2))
vi <- data.frame(cassava[, 1:2], vi)
ndvi <- terra::rast(vi, type = 'xyz')
e <- terra::ext(287688, 287708, 8664196, 8664216)
r1 <- terra::crop(ndvi, e)
ndvi_1 <- r1
```

The Cassava database contains the variables x , y , l_1 , l_2 , and l_3 .

where x and y are the pixel coordinates.

and L_1 , L_2 , and L_3 are the spectral bands for NIR, red, and green, respectively.

The open database CIP DATAVERSE contains the images used in this article by Loayza et al. (2018). The file “TC_0559_georeferenced.tif” corresponds to week 11 of the three crops (See Figure 1a). The image of the cassava crop is shown (See Figure 3).

An image is plotted by using the following instructions in R:

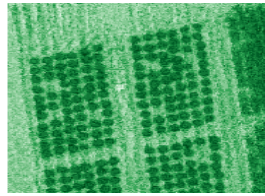


Figure 3: NDVI image Cassava crop

The CIP DATAVERSE open database contains the images used in this article [Loayza et al. \(2018\)](#), the file "TC_0559_georeferenced.tif" corresponds to week 11 of the 3 crops (See Figure 1a), where the image of the cassava is given (See Figure 3).

An image is plotted by using the following instructions in R:

```
op <- par(mar = c(0, 0, 0, 0))
terra::image(ndvi_1, col = hcl.colors(12, "Greens 3", rev = TRUE), axes = FALSE)
par(op)
```

3.1 fourPoint()

This function aims to determine the four points defining a study area delimited by a parallelogram. For this, in the x11 environment, it is sufficient to hover over three of the four extreme points (coordinates) of the image. The first point is the upper left corner of the area to be delimited, while the second and third points correspond to the other corners in the clockwise direction, and the fourth point is determined by the fourPoint(). Once the points are fixed, the locator() obtains the coordinates.

Syntax:

fourPoint(*P*)

where *P* is a list of three points obtained by the locator(3).

The fourth point is constructed as follows. Given the three points (x_1, y_1) , (x_2, y_2) and (x_3, y_3) , the fourth point (x_4, y_4) is determined by using the slopes (m_1, m_2) . See equations (1), (2), (3) and (4):

$$m_1 = (y_2 - y_1) / (x_2 - x_1) \quad (1)$$

$$m_2 = (y_3 - y_2) / (x_3 - x_2) \quad (2)$$

$$x_4 = (y_3 - y_2 - m_1 x_3 + m_2 x_1) / (m_2 - m_1) \quad (3)$$

$$y_4 = y_3 - m_1 (x_3 - x_4) \quad (4)$$

Example

Considering the three points of the study area, taken clockwise (Figure 4, interactive), we wish to determine the fourth point. To this end, point $Q = \text{fourPoint}(P)$ is taken as shown in the following instructions, corresponding to the location of the study area.

In R:

```
P <- list(x = c(287689.5, 287702.8, 287706.2), y = c(8664210.3, 8664214.2, 8664179))
Q <- fourPoint(P)
print(Q)
```

```
      x      y
[1,] 287689.5 8664210
[2,] 287702.8 8664214
[3,] 287706.2 8664179
[4,] 287692.9 8664175
```

Q matrix includes the spatial coordinates (x, y) of the points used to delineate the experimental plot presented in Figure 4. These coordinates were obtained from the georeferenced image and define the plot boundaries used in subsequent analyses.

The generated Q points can be applied to terra::image(r) as follows:

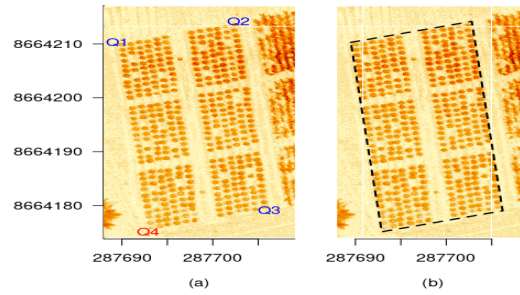


Figure 4: Image processed with the fourPoint(): a) image with 4 points and b) plot area.

```
op <- par(mfrow = c(1, 2), mar = c(4, 5, 0, 1))
r <- terra::rast(cassava, type = "xyz")
terra::image(r, main = "", xlab = "(a)", ylab = "", cex = 2.5, bty = "n", las = 1)
text(Q[1:3, ], paste("Q", 1:3, sep = ""), col = "blue")
text(Q[4, 1], Q[4, 2], "Q4", col = "red")
par(mar = c(4, 1, 0, 5))
terra::image(r, axes = FALSE, main = "", xlab = "(b)", ylab = "", cex = 2.5)
axis(1)
rownames(Q) <- paste("Q", 1:4, sep = "")
polygon(Q, lty = 2, lwd = 2)
par(op)
```

3.2 fixedPoint()

This function divides a single side segment into n segments for n individual EUs, where each segment is of length w .

Syntax:

`fixedPoint(( $x_1, y_1$ ), ( $x_n, y_n$ ),  $n, w$ )`

where (x_1, y_1) and (x_n, y_n) are the extreme points of one side of the field, n is the number of segments, and w is their length.

let n be the number of segments and P is the matrix of points (x, y) of the segments of length w corresponding to one side of the plot. The endpoints correspond to (x_1, y_1) and (x_n, y_n).

Equation (5) shows the length L of the side of the plot:

$$L = \sqrt{(x_n - x_1)^2 + (y_n - y_1)^2} \quad (5)$$

Further, s is the distance between the starting points of each segment:

$$s = (L - nw) / (n - 1)$$

The difference between endpoints is as shown in equation:

$$d = (x_n - x_1, y_n - y_1)$$

Further, d is a vector of elements (d_1, d_2) and the distance between the initial points of the segments is given by:

$$\delta = (w + s)d / \|d\|$$

where $\|d\| = \sqrt{d_1^2 + d_2^2}$

and then the points of each segment is determined by:

$$(x_k, y_k) = (x_{k-2}, y_{k-2}) + \delta, \text{ for } k = 3, 5, \dots, n - 1$$

$$(x_k, y_k) = (x_{k+2}, y_{k+2}) - \delta, \text{ for } k = n - 2, n - 4, \dots, 2$$

Example:

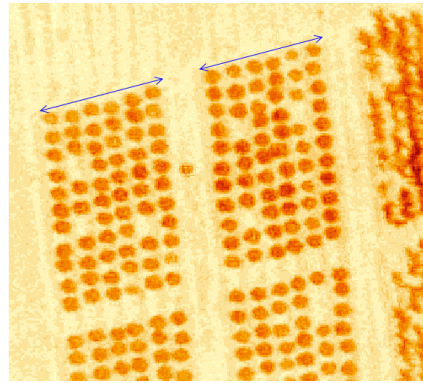


Figure 5: Image processed with the `fixedPoint()`

Using points Q of the plot in Figure 5, we want to determine the coordinate matrix of two segments of the side of the plot given by points $Q[1]$ and $Q[2]$, of length 6 m. Therefore, the following statement is run:

```
op <- par(mar = c(0, 0, 0, 0), cex = 0.8)
r1 <- terra::crop(r, e)
terra::image(r1, main = " ", axes = FALSE)
s <- fixedPoint(Q[1, ], Q[2, ], 2, 6)
arrows(s[1, 1], s[1, 2], s[2, 1], s[2, 2], col = "blue", code = 3, length = 0.1)
arrows(s[3, 1], s[3, 2], s[4, 1], s[4, 2], col = "blue", code = 3, length = 0.1)
par(op)
```

3.3 `imageField()`

The purpose of this function is to distribute the EUs in an orthomosaic image according to the shape and size of the EUs under the experiment considerations defined in the field. The georeferenced pixels of the image are related to the EUs to generate a table with correct information to identify the EU, as well as the pixel coordinates and multispectral registration ($L_1, L_2, \dots$), regardless of the number of spectral bands considered.

The image coordinates are essential if the image contains the actual georegistration of the image.

Syntax:

`imageField(r, Q, ny, nx, dy, dx, plotting = TRUE, ...)`

Where:

- `r`: orthomosaic image
- `Q`: coordinate matrix of dimension (4, 2) defining the plot.
- `ny`: number of EUs in the direction $Q[2,]$ and $Q[3,]$
- `nx`: number of EUs in the direction $Q[1,]$ and $Q[2,]$
- `dy`: length of EU in direction $Q[2,]$ and $Q[3,]$
- `dx`: length of EU in direction $Q[1,]$ and $Q[2,]$
- `plotting`: = TRUE for the distribution of the EUs on the image
- `...`: graph arguments of the R plot function.

The `imageField` process can be summarized in four steps. In step 1, the sides of the parallelogram are fixed in terms of $Q = (Q[1,], Q[2,], Q[3,], Q[4,])$. In step 2, the endpoints of ny segments on the sides parallel to the straight line are determined $Q[2,] \sim Q[3,]$, indicated in red in Figure 6. In step 3, the endpoints of nx segments in the direction $Q[1,], Q[2,]$ are determined for each pair of points generated in the previous step, i.e., a total of $ny \times nx$ points are generated (blue color). In step 4, using the points generated in step 3, the EUs are defined, and their pixels are assigned. Thus, a table of spectral information for the EUs is generated. The result is shown in the right panel of Figure 6.

The following code complements figure 6.

```
op <- par(mar = c(0, 0, 2, 0), cex = 0.8)
terra::image(r, axes = FALSE, main = "Result with imageField", cex.main = 1.5)
Rbook <- imageField(r, Q, ny = 3, nx = 2, dy = 11, dx = 6, plotting = TRUE)
```

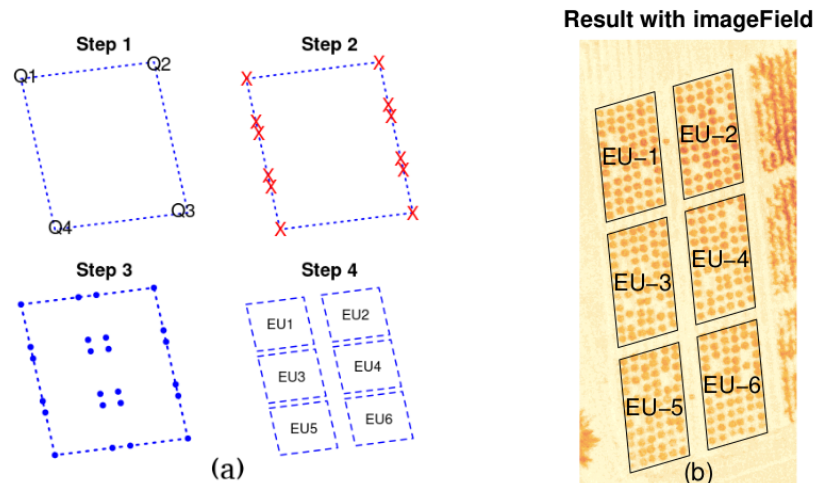


Figure 6: a) Sequence of the imageField process: (Step 1) Define plot; (Step 2) Determine edge points; (Step 3) Determine interior points; (Step 4) Determine EUs. b) Output in 6 plots (EUs) with imageField()

Table 1: First 6 records of Plot 1

EU	x	y	L1	L2	L3
1	287695.1	8664212	48	19	27
1	287695.1	8664212	56	16	29
1	287695.2	8664212	48	18	31
1	287695.2	8664212	42	18	34
1	287694.9	8664212	55	21	36
1	287695.0	8664212	55	25	29

```
center <- agricolae::tapply.stat(Rbook$Qbase[, 2:3], Rbook$Qbase[, 1])
edge <- Rbook$coordinates.EU
text(center$x, center$y, paste("EU", center[, 1], sep = "-"), cex = 1.5)
text(287699.5, 8664175, "(b)", cex = 2.2)
par(op)
```

Example:

Consider the image of the six cassava plots (3×2), with ends of plot $Q = (Q[1,], Q[2,], Q[3,], Q[4,])$. We want to determine the three plots between points $Q[2,]$ and $Q[3,]$ and two plots between points $Q[1,]$ and $Q[2,]$, with sides of 11 m and 6 m per plot. For this purpose, we apply `imageField(r, Q, ny = 3, nx = 2, dy = 11, dx = 6, plotting = TRUE)`, which provides the image shown in step 4 in Figure 6, the matrix of the coordinates of all the plots and their information matrix (including the coordinates), and the wavelength in reflectance percentage (L_1, L_2, L_3) as given in Table 1, "Qbase" object, which is the output of the of the `imageField()`.

3.4 designRaster()

This function aims to relate an image to the experimental design to allow us to fix the experimental area and identify EUs in a design image.

Syntax:

```
designRaster(R, book)
```

Where:

R: "Qbase" output object of the `imageField()`.

book: output object of an experimental designs function of the `agricolae`.

Considering that "R" and "book" are flat tables identified by rows and columns, where each EU has its identification code in the "book" and sequential (1, 2, ...) in "R," there exists an unequivocal correspondence between the information in the two tables through the identification code of the EUs.

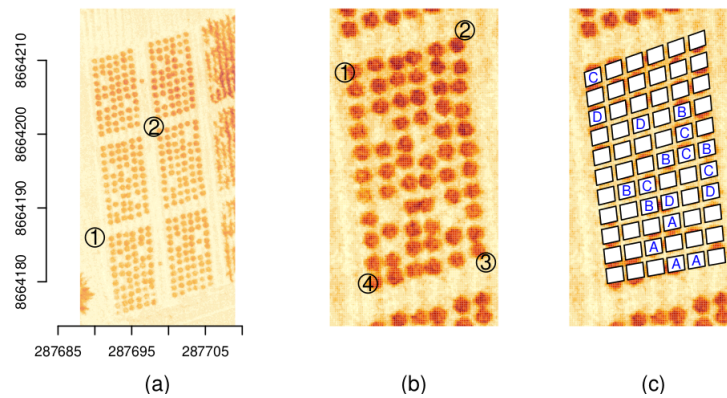


Figure 7: a) image with locator(2), b) image with locator(4) and c) EUs in the field and Completely Randomized Design generated by designRaster

Figure 7: a) image with locator(2), b) image with locator(4) and c) EUs in the field and Completely Randomized Design generated by designRaster()

This relationship is used to implement the `designRaster()`. `designRaster()` outputs two tables the first containing the EU identification of the image and design and the second the coordinates of the EUs in the field.

Example:

Consider that in the third cassava plot, we plan to test a complete randomized design with four treatments, A, B, C, and D, with replicates 4, 5, 5, and 4, respectively, where the EU is one cassava plant in the plot. For this purpose, the plot is delimited into units of 10×6 m, with new coordinates of Q. The dimensions of each EU net are 0.8×0.9 m. For design purposes, 66 (11×6) plots are available, as shown in Figure 7, and only 18 EU will be used; therefore, 48 plots will not be used, and an empty treatment is assigned to these plots.

First, the EUs in the image are generated using `imageField(r, Q, ny = 11, nx = 6, dy, dx, plotting = TRUE)`. This function outputs the Qbase table assigned to "R." Second, the design is generated with the function `design.crd(trt = c(A, B, C, D, " "), r = c(4, 5, 5, 4, 48), seed = 8)` of the `agricolae` package, where the value `seed = 8` is assigned to reproduce the design. This function generates the book table containing the design. Finally, both tables are linked by the function `designRaster(R, book)` (See 7c and Table 2).

Using the image "r" previously found and by the manual determination of points "p1 = locator(2)" (See 7a) to crop the image and "p2 = locator(4)" (See Figure 7b) to determine the vertices of the experimental area, the design in the spectral image (See Figure 7c) is generated using the following code in R:

```
op <- par(mfrow = c(1, 3), mar = c(4, 2, 0, 0))
terra::image(r, main = "", xlab = "(a)", ylab = "", cex.lab = 1.5, axes = FALSE)
axis(1)
axis(2)
p1 <- list(x = c(287690, 287698), y = c(8664186, 8664201))
points(p1, cex = 3)
text(p1, c("1", "2"), cex = 1.5)
e <- terra::ext(unlist(p1))
rc <- terra::crop(r, e)
terra::image(rc, main = "", xlab = "(b)", ylab = "", cex.lab = 1.5, axes = FALSE)
p2 <- list(x = c(287690.7, 287696.4, 287697.4, 287691.8),
          y = c(8664198, 8664200, 8664189, 8664188))
Q2 <- fourPoint(p2)
points(Q2, cex = 3)
text(Q2, c("1", "2", "3", "4"), cex = 1.5)
par(mar = c(4, 2, 0, 0))
terra::image(rc, main = "", xlab = "(c)", ylab = "", cex.lab = 1.5, axes = FALSE)
out <- imageField(r, Q2, ny = 11, nx = 6, dy = 0.8, dx = 0.9, col = "white")
trt <- c(LETTERS[1:4], " ")
k <- c(4, 5, 5, 4, 48)
plan1 <- agricolae::design.crd(trt, r = k, seed = 8)$book
plan2 <- designRaster(R = out$Qbase, book = plan1)
```


Table 2: Output designRaster()

Spatial field book													
	r	trt		x	y								
101	1	C		287691.2	8664198								
113	1	D		287691.4	8664196								
121	2	D		287693.4	8664196								
123	1	B		287695.3	8664196								
129	2	C		287695.4	8664195								
						Spectral data of the experimental area (First 6 records)							
134	2	B		287694.6	8664194	x	y	L1	L2	L3	plots	r	trt
135	3	C		287695.5	8664194	287691.5	8664198	69	20	43	101	1	C
136	3	B		287696.5	8664194	287691.4	8664198	87	27	49	101	1	C
138	4	B		287692.7	8664192	287691.5	8664198	81	25	39	101	1	C
139	4	C		287693.7	8664193	287691.5	8664198	78	18	38	101	1	C
142	5	C		287696.6	8664193	287691.3	8664198	106	21	54	101	1	C
145	5	B		287693.8	8664192	287691.3	8664198	108	12	60	101	1	C
146	3	D		287694.7	8664192								
148	4	D		287696.6	8664192								
152	1	A		287694.8	8664191								
157	2	A		287694.0	8664190								
164	3	A		287695.0	8664189								
165	4	A		287696.0	8664189								

```

xx <- plan2$design
yy <- plan2$rasterField
design <- xx[xx$trt != " ", ]
spectral <- yy[yy$trt != " ", ]
text(design[, 4], design[, 5], design[, 3], cex = 1, col = "blue")
par(op)

```

3.5 movePlot()

The objective of this function is to rectify the coordinates and the direction of the study plot to reduce errors when obtaining the coordinates of the images over time. To do this, the plot coordinates are rotated and translated.

Syntax:

movePlot(Q, q)

Where:

Q: the 4 points of the plot (initial image)

q: vector with two points, where the first one corresponds to the new location of the plot, and the second one is used to obtain the direction.

$Q = [Q[1,], Q[2,], Q[3,], Q[4,]]$, and $q = [q[1,], q[2,]]$ described in (6) allows the calculation of the rotation angle, as shown in (7). Using (8), the rotation matrix P can be determined to obtain the new rotated Q matrix

$$Q = \begin{pmatrix} Q_{1,1} & Q_{1,2} \\ Q_{2,1} & Q_{2,2} \\ Q_{3,1} & Q_{3,2} \\ Q_{4,1} & Q_{4,2} \end{pmatrix}, \quad q = \begin{pmatrix} q_{1,1} & q_{1,2} \\ q_{2,1} & q_{2,2} \end{pmatrix} \quad (6)$$

$$m_1 = \frac{Q_{2,2} - Q_{1,2}}{Q_{2,1} - Q_{1,1}}, \quad m_2 = \frac{q_{2,2} - q_{1,2}}{q_{2,1} - q_{1,1}}, \quad \theta = \arctan(m_2) - \arctan(m_1) \quad (7)$$

$$P = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}, \quad Q = Q.P \quad (8)$$

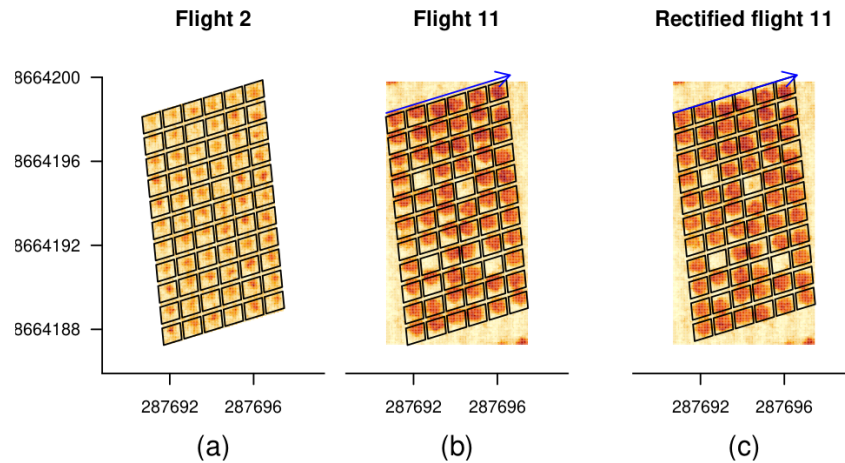


Figure 8: Cassava in plot 3, a) cropTime data with Q, b) cassava data with Q new

The translation of matrix Q to the new location is obtained using (9).

$$\delta_j = Q_{1,j} - q_{1,j}, \quad j = 1, 2 \quad Q_{i,j} = Q_{i,j} - \delta_j, \quad i = 1, 2, 3, 4 \quad j = 1, 2 \quad (9)$$

Example. Consider the image of flight 2 shown in Figure 6a and EU_3 and its point matrix Q (Q1, Q2, Q3, Q4) that limits it. In flight 11, you have the same plot, but with a small deviation as shown in Figure 6b. Q has to be corrected for use in flight 11. To rectify Q, we consider a new location and orientation given by vector $q(x, y)$, $x = (287690.7, 287696.6)$, $y = (8664198.3, 8664200.1)$. To obtain the new Q, the function `movePlot(Q, q)` is applied to obtain new distribution of subplots, without altering the dimensions. The new Q is now used in flight 11 (See Figure 8c and the values of Q in Table 3).

In R, for flight 2, data `cropTime` and `terra` package are used:

```
data(cropTime)
ax1 <- seq(287688, 287700, 4)
ay1 <- seq(8664184, 8664202, 4)
by1 <- c(8664185.9, 8664201.8)
ny <- 11
nx <- 6
dy <- 0.8
dx <- 0.9
f2 <- subset(cropTime, Flight == 2)
r2 <- terra::rast(f2[, -1], type = "xyz")
P2 <- list(x = c(287690.68, 287696.43, 287697.46), y = c(8664198.11, 8664199.87, 8664189.01))
Q2 <- fourPoint(P2)
```

For flight 11, data `cassava`, `imageField()`, and `terra` package are used:

```
ex <- list(x = range(f2$x), y = range(f2$y))
e <- terra::ext(unlist(ex))
rc <- terra::crop(r, e)
```

For rectified flight 11, `movePlot()`, `imageField()`, and `terra` package are used:

```
q <- list(x = c( 287690.7, 287696.6), y = c(8664198.3, 8664200.1))
Qnew <- movePlot(Q2, q)
```

```
op <- par(mfrow = c(1, 3), mar = c(4, 4, 2, 0), cex = 0.7)
main1 <- "Flight 2"
main2 <- "Flight 11"
main3 <- "Rectified flight 11"
terra::image(r2, axes = FALSE, main = main1, ylim = by1, xlab = "(a)", cex.lab = 1.5)
axis(1, ax1)
```

Table 3: Coordinate rectification with movePlot(Q,q)

Q coordinates image 2				rectified Q coordinates	
	x	y		x	y
Q1	287690.6800000000	8664198.109999999		287690.7000000000	8664198.300000001
Q2	287696.4300000000	8664199.869999999		287696.4516108264	8664200.054728726
Q3	287697.4600000000	8664189.010000000		287697.4716559576	8664189.193789175
Q4	287691.7100000001	8664187.250000000		287691.7200451312	8664187.439060450

```

axis(2, ay1, las = 1)
plan <- imageField(r2, Q2, ny, nx, dy, dx)
par(mar = c(4, 1, 2, 3), cex = 0.7)
terra::image(rc, main = main2, xlab = "(b)", ylab = "", cex.lab = 1.5, axes = FALSE, ylim = by1)
axis(1, ax1)
plan <- imageField(rc, Q2, ny, nx, dy, dx)
arrows(q$x[1], q$y[1], q$x[2], q$y[2], col = "blue", length = 0.1)
par(mar = c(4, 0, 2, 4), cex = 0.7)
terra::image(rc, axes = FALSE, main = main3, xlab = "(c)", ylab = "", cex.lab = 1.5, ylim = by1)
axis(1, ax1)
plan <- imageField(rc, Qnew, ny, nx, dy, dx)
arrows(q$x[1], q$y[1], q$x[2], q$y[2], col = "blue", length = 0.1)
par(op)

```

The images obtained successively from the same field on different dates are rectified; hence, the same plotting structure is applied to all images. However, some differences may arise, and it is necessary to rectify the position of the pixels.

3.6 borderPoint()

The objective of this function is to obtain the spectral information of the edge of the EUs, i.e., the information around it. This allows us to explore the edge effects on the study plot.

Syntax:

```
borderPoint(r, Rbook, distance, plotting = TRUE, ...)
```

Where:

- r: is the spectral image of the field that includes the experimental plot.
- Rbook: plot with the EUs, generated by imageField().
- distance: longitude of the border to the plot.
- plotting = TRUE: display the image of the border effect around the EUs.
- ...: plot parameter to document the image as main, axis, etc.

To obtain the spectral information of the EU edges, a new plot is constructed considering the vertices of the initial plot (Q) in Rbook, which is extended to new vertices by adding a set distance (*d*) and maintaining its direction. The imageField() is then applied to the new plot to determine the EUs in it, keeping their coordinates. Finally, the edge is obtained by removing all information of the EUs from the new plot.

Example

Consider the image shown in Figure 4. We need to obtain the spectral information of the edge of the EUs of this image at a distance of 1 m that encloses the experimental area. To this, we can apply borderPoint(r, Rbook, distance = 1) (its results are shown in Figure 9) and the matrix of the spectral information of the edge (a part of it is given in Table 4).

```

op <- par(mfrow = c(1, 2), mar = c(0, 0, 2, 0))
terra::image(r, main = "Image crop", axes = FALSE)
text(287700, 8664175.5, "(a)", cex.lab = 1.5)
Rbook <- imageField(r, Q, ny = 3, nx = 2, dy = 11, dx = 6, plotting = TRUE)
out <- borderPoint(r, Rbook, distance = 1, main = "Border effect", axes = FALSE)
text(287700, 8664175, "(b)", cex.lab = 1.5)
par(op)

```

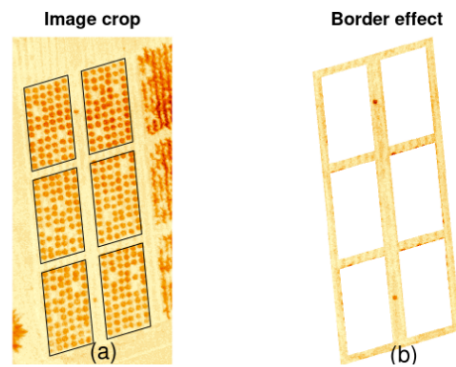


Figure 9: borderPoint() application: a) Image of the field; b) Border of EUs

Table 4: Output borderPoint()

Four points of the border area		Spectral data of the experimental edge (First 6 records)				
		x	y	L1	L2	L3
x	y	287703.4758	8664215.398	53	40	34
287688.4336	8664210.987	287703.5345	8664215.398	52	40	35
287703.6785	8664215.458	287703.5933	8664215.398	51	41	37
287707.2664	8664178.313	287703.6521	8664215.398	56	39	39
287692.0215	8664173.842	287703.2995	8664215.339	59	26	35
		287703.3582	8664215.339	62	29	36

3.7 EUsPoint()

This function determines the vertices of a specific EU from an experimental area that was already segmented with the imageField().

Syntax:

EUsPoint(Rbook, EU)

Where:

Rbook: List of objects generated by imageField().

EU: integer value that identifies the EU of interest.

The vertices matrix of the EUs is an object in Rbook, and it contains all the coordinate points (x , y), which are the boundaries of the EUs. The coordinate points are in correlative order, as shown in Figure 10a10a.

Given nx and ny , the position of each point of the EUs is determined by using the following code corresponding to the EUsPoint():

```

nx <- 2
ny <- 3
a <- c(1, 2, 2*(nx+1), 2*nx+1)
A <- NULL
for(j in 1:ny){
  for(i in seq(0, 2*(nx-1), 2)){
    A <- rbind(A, a+i)
  }
  a <- a+4*nx
}

```

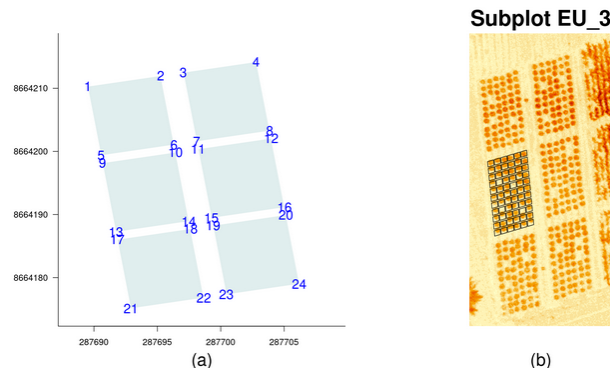
For example, with $nx = 2$ and $ny = 3$, a matrix A is generated. This matrix contains the sequence of points corresponding to the vertices of the EUs (See Table 5 and Figure 10a).

Considering EU = 3 as the EU, the coordinates of the vertices correspond to {9, 10, 14, 13} (See Figure 10a).

Example

Table 5: Numbering of the vertices of the EUs

	Q1	Q2	Q3	Q4
EU1	1	2	6	5
EU2	3	4	8	7
EU3	9	10	14	13
EU4	11	12	16	15
EU5	17	18	22	21
EU6	19	20	24	23

**Figure 10:** Effect of the EUsPoint(): a) Location of the edges, b) Generation of subplots in unit 3

We determine 66 subplots (11×6) with length $dy = 0.8m$ and width $dx = 0.9m$ on the third EU (EU_3) in Figure 6. To do this, first, `EUsPoint(Rbook, EU = 3)` is applied to get the vertices P_3 of EU_3; then, `imageField(r, P_3, ny = 11, nx = 6, dy = 0.8, dx = 0.9)` is applied. Thus, the image shown in Figure 10b is obtained. The information matrix of the subplots in EU_3 (Table 6) prints at the first few rows of the cassava:

The following instructions in R present the vertices of the plots and generate the subplots of EU_3:

```
op <- par(mfrow = c(1, 2), mar = c(4, 5, 3, 0), cex = 0.8)
XYcenter <- apply(cassava[, 1:2], 2, mean)
xArista <- range(cassava[, 1])
yArista <- range(cassava[, 2])
plot(XYcenter[1], XYcenter[2], xlim = xArista, ylim = yArista, bty = "l",
      xlab = "(a)", ylab = "", cex = 0, las = 1, cex.lab = 1.5)
Rbook <- imageField(r, Q, ny = 3, nx = 2, dy = 11, dx = 6, col = colors()[15], border = "white")
arista <- Rbook$coordinates.EU
text(arista, cex = 1, col = "blue")
terra::image(r, axes = FALSE, main = "Subplot EU_3",
             xlab = "(b)", ylab = "", cex.lab = 1.5, cex.main = 1.5)
Rbook <- imageField(r, Q, ny = 3, nx = 2, dy = 11, dx = 6, plotting = FALSE)
P_3 <- EUsPoint(Rbook, EU = 3)
Rbook3 <- imageField(r, P_3, ny = 11, nx = 6, dy = 0.8, dx = 0.9, plotting = TRUE)$Qbase
names(Rbook3)[1] <- "SubPlot"
par(op)
```

4 Application in crop development

The most important use of rPAex is to provide spectral information, such as vegetation indices, from images of PA experiments. In this example, rPAex shows the development of the cassava crop on nine evaluation dates used in the case study. In this study, the vegetation index (NDVI) and red spectral band representing the low vegetation and high soil presence were considered.

The rPAex database contains multispectral data of nine images obtained in the course of 38 flights (cropTime) for a CIP experimental field (Table 7). The cropTime database was obtained from the CIP

Table 6: First 6 records of EU-3

SubPlot	x	y	L1	L2	L3
1	287691.3109	8664198.355	97	17	43
1	287691.3697	8664198.355	97	16	46
1	287691.4285	8664198.355	90	14	45
1	287691.4872	8664198.355	78	19	42
1	287691.1346	8664198.296	102	22	61
1	287691.1934	8664198.296	109	14	63

Table 7: Description of the data 'cropTime'

Flight dates and dimension			Spectral data experimental area (First 6 records)					
	Flight	Pixels	Flight	x	y	L1	L2	L3
2014-12-18	1	18905						
2015-01-06	2	19000						
2015-01-29	6	22138	1	287696.41	8664199.86	62	39	48
2015-02-26	11	18771	1	287696.24	8664199.80	59	48	41
2015-03-30	20	18970	1	287696.30	8664199.80	53	50	44
			1	287696.36	8664199.80	53	44	52
2015-04-22	23	18406	1	287696.41	8664199.80	61	41	45
2015-06-09	30	18553						
2015-07-24	36	20480	1	287696.06	8664199.74	61	33	36
2015-08-06	38	17040						

DATAVERSE repository, the images.RAR file contains all images, one per flight. For example, flight 11 contains the file "TTC_0559_georeferenced.tif" from the folder "2015_02_26."

After downloading the file "TTC_0559_georeferenced.tif," the following code can be applied in R:

```
r <- terra::rast("TTC_0559_georeferenced.tif")
base <- imageField(r, P_3, ny = 1, nx = 1, dy = 0, dx = 0, plotting = FALSE)$Qbase
head(base)
```

The images of flights 1, 2, 6, 11, 20, 23, 30, 36, and 38 for plot 3 were extracted from each image using the definition of plot P_3 (See Figure 10b) from the example of the EUsPoint(). The code in R extracts the data from plot 3. The parameters $nx = 1$, $ny = 1$ because a single plot is considered. Since the plot is not further divided, the parameters dy and dx are zero. The accumulated data of each image are used to generate the cropTime object.

To illustrate the development of the cassava crop, during the nine flights, a color value is assigned to the NDVI to simulate the presence of canopy covers during crop growth (Figure 11a) using the trends of the NDVI and the red spectral band. The following code in R allows the following calculations:

```
ndvi <- with(cropTime, (L1 - L2)/(L1 + L2))
cropTime <- data.frame(cropTime, ndvi)
FLIGHT <- c(1, 2, 6, 11, 20, 23, 30, 36, 38)
Flight <- cropTime$Flight
f1 <- function(x) mean(x, na.rm = TRUE)
trend <- agricolae::tapply.stat(cropTime[, 2:7], Flight, f1)
ylim1 <- c(8664183.5, 8664200)
```

Each image $i = 1$ to 9, can be displayed and the trend of the spectral indices with "trend" data, where $L1 = \text{NIR}$, $L2 = \text{Red}$ and $L3 = \text{Green}$.

```
par(mfrow = c(2, 5), mar = c(0, 0, 1, 0), cex = 0.9)
for(i in 1:9){
  P <- cropTime[cropTime$Flight == FLIGHT[i], ]
  P$ndvi <- (P$L1 - P$L2)/(P$L1 + P$L2)
  I <- terra::rast(P[, c(2, 3, 7)], type = "xyz")
}
```

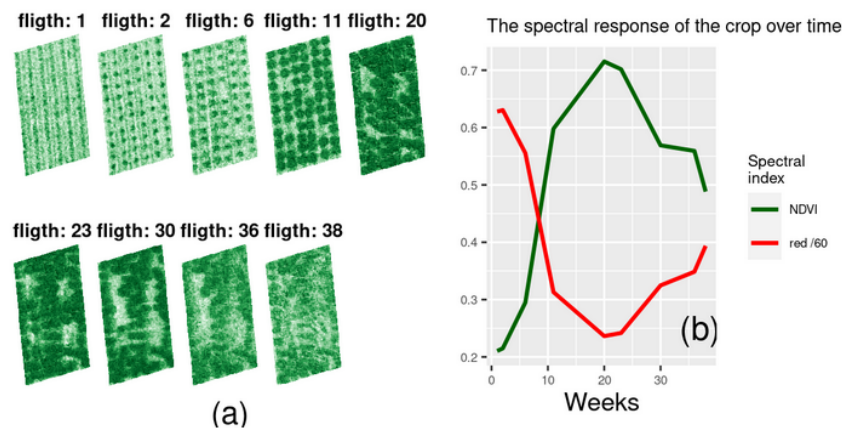


Figure 11: a) Canopy cover of the cassava crop. b) NDVI and Red spectral band during cassava growth

```
xcolor <- hcl.colors(12, "Greens 3", rev = TRUE)
xmain <- paste("flighth:", FLIGHT[i])
terra::image(I, col = xcolor, axes = FALSE, main = xmain, ylim = ylim1, cex.main = 1.5)
if(i == 8) text(287695, 8664184.8, "(a)", cex = 2)
}
par(mfrow = c(1, 1), cex = 1.2)
ggplot(data = trend) +
  theme(text = element_text(size = 15)) +
  geom_line(mapping = aes(y = ndvi, x = Flight, colour = "NDVI"), linewidth = 1) +
  geom_line(mapping = aes(y = L2/60, x = Flight, colour = "red /60"), linewidth = 1) +
  scale_linewidth(range = c(0.2, 0.8)) +
  scale_color_manual(name = "Spectral index",
    values = c("NDVI" = colors()[81], "red /60" = "red")) +
  ggtitle("The spectral response of the crop over time") +
  labs(x = "Weeks", y = "", cex.lab = 0.9) +
  theme(axis.title.x = element_text(size = 25),
    axis.title.y = element_text(size = 20)) +
  annotate("text", x = 37, y = 0.25, label = "(b)", size = 8)
```

5 Discussion

Unlike other packages, such as terra, which facilitates and supports the treatment of remote sensing images in field studies, no packages are available in R to treat images with the design of experiments. In this sense, the purpose of rPAex is to link spectral images to experimental designs by identifying the EUs, facilitating the use of spectral information, and their edges to complement the information already gathered via experimental analyses. The focus was on studying experimental designs with images to analyze vegetation indices and spectral and agronomic measurements and to compare treatments. When identifying the EUs of the experimental design using rPAex, the spectral measurements and vegetation indices become the new variables. Moreover, rPAex supports longitudinal study as the design structure remains the same throughout the experiment.

As with any process subjected to environmental factors and involving manually operated data acquisition devices, the effectiveness of rPAex depends on the accuracy and precision of the imaging instruments.

From the consecutive images obtained in an experiment, the effect of treatments on the spectral measurement may be inferred, and comparative analysis with any field measurement may be conducted. In this sense, rPAex facilitates the organization of the spectral data in the EUs. This package is available on the R platform and is free to use; hence, suggestions will allow continuous improvement of the package. Given the importance of longitudinal studies, which can be adequately treated by functional data analysis, it is necessary to include this type of analysis in rPAex.

When integrated with the **agricolae** package, rPAex allows the spatial linking of the image and spectral information with any experimental design generated before its application, assuming that the experimental plan is developed before setting up the experiment. rPAex strengthens the initial stage of experimentation with multispectral data to guide the researcher and facilitate site planning for the

experiment installation. Spectral information collected before, during, and at the end of trials will aid crop evaluation.

The integration of multispectral information with field measurements for each EU enables the study of crop phenology, statistical analysis of the experiment, and functional data analysis of EUs over time, subject to the evaluation of the study treatments. Further, regarding the limitation to fields with uniform spaces, the EUs may not necessarily have a regular shape and may have different sizes as defined by points within the area (3–4 points) and by applying the `fourPoint()`. However, this is not recommended since the EUs in the experiment have uniform shape and size.

6 Summary

The `rPAex` package was introduced in R. It contains seven functions to support the experimental design in PA with multispectral images by generating the EUs and their borders and vertices and by adjusting the plot frame by the calibration effect.

The `imageField()` integrates a spectral image with the EUs; in terms of georegistration, the pixels are labeled with the EUs. It outputs three objects, namely, the parameters of the size of the study plot, the spectral information of the EUs, and the georeference of each EU. The outputs of the function allow us to relate the experimental design created by the `agricolae` package to the field image using the `designRaster()`. The `designRaster()` creates a spectral field book and identifies the image pixels with the design characteristics for analysis. Likewise, the spectral information of the edge is obtained by the `borderPoint()`; this information allows us to study the effect in the experiment. The coordinates of a specific unit are obtained by the `EUsPoint()` to create new subplots of the unit.

7 Acknowledgments

The authors appreciate the access to DATAVERSE and repository, provided by the International Potato Center (CIP), as well as its support with the use of images in agricultural experimentation.

References

- R. A. Bailey. *Design of comparative experiments*, volume 25. Cambridge University Press, 2008. doi: 10.1017/CBO9780511611483. [p23]
- M. Basso and E. Pignatton de Freitas. A UAV guidance system using crop row detection and line follower algorithms. *Journal of Intelligent & Robotic Systems*, 97(3):605–621, 2020. doi: 10.1007/s10846-019-01006-0. [p24]
- J. Bendig, K. Yu, H. Aasen, A. Bolten, S. Bennertz, J. Broscheit, M. L. Gnyp, and G. Bareth. Combining UAV-based plant height from crop surface models, visible, and near infrared vegetation indices for biomass monitoring in barley. *International Journal of Applied Earth Observation and Geoinformation*, 39:79–87, 2015. doi: 10.1016/j.jag.2015.02.012. [p23]
- L. Griffel, D. Delparte, and J. Edwards. Using support vector machines classification to differentiate spectral signatures of potato plants infected with potato virus Y. *Computers and electronics in agriculture*, 153:318–324, 2018. doi: 10.1016/j.compag.2018.08.027. [p23]
- J. M. Guerrero, M. Guijarro, M. Montalvo, J. Romeo, L. Emmi, A. Ribeiro, and G. Pajares. Automatic expert system based on images for accuracy crop row detection in maize fields. *Expert Systems with Applications*, 40(2):656–664, 2013. doi: 10.1016/j.eswa.2012.07.073. [p24]
- R. J. Hijmans. *terra: Spatial Data Analysis*, 2023. URL <https://CRAN.R-project.org/package=terra>. R package version 1.7-55. [p23]
- J. Jung, M. Maeda, A. Chang, J. Landivar, J. Yeom, and J. McGinty. Unmanned aerial system assisted framework for the selection of high yielding cotton genotypes. *Computers and electronics in agriculture*, 152:74–81, 2018. doi: 10.1016/j.compag.2018.06.051. [p23]
- E. L. LeClerg, W. H. Leonard, and A. G. Clark. Field plot technique. *Soil Science*, 95(4):290, 1963. doi: 10.1007/978-94-009-1503-9_2. [p24]
- H. Lee. Intra-field canopy nitrogen retrieval from unmanned aerial vehicle imagery for wheat and corn crops in Ontario, Canada. 2020. doi: 10.1080/07038992.2020.1788384. [p23]

- H. Loayza, L. Silva, S. Palacios, M. Balcazar, and R. Quiroz. Dataset for: Modelling crops using high resolution multispectral images, 2018. URL <https://doi.org/10.21223/P3/UVWVLA>. [p24, 25, 26]
- F. I. Matias, M. V. Caraza-Harter, and J. B. Endelman. FIELDImageR: An R package to analyze orthomosaic images from agricultural field trials. *The Plant Phenome Journal*, 3(1):e20005, 2020. doi: 10.1002/ppj2.20005. [p24]
- M. Montalvo, G. Pajares, J. M. Guerrero, J. Romeo, M. Guijarro, A. Ribeiro, J. J. Ruz, and J. Cruz. Automatic detection of crop rows in maize fields with high weeds pressure. *Expert Systems with Applications*, 39(15):11889–11897, 2012. doi: 10.1016/j.eswa.2012.02.117. [p23, 24]
- M. Pérez-Ortiz, J. Peña, P. A. Gutiérrez, J. Torres-Sánchez, C. Hervás-Martínez, and F. López-Granados. A semi-supervised system for weed mapping in sunflower crops using unmanned aerial vehicles and a crop row detection method. *Applied Soft Computing*, 37:533–544, 2015. doi: 10.1016/j.asoc.2015.08.027. [p24]
- A. Srinivasan. *Handbook of precision agriculture: principles and applications*. CRC press, 2006. doi: 10.1201/9781482277968. [p23]
- R. Tripp. Data collection, site selection and farmer participation in on-farm experimentation. 1982. [p23]
- T. Utstumo, F. Urdal, A. Brevik, J. Dorum, J. Netland, O. Overskeid, T. W. Berge, and J. T. Gravdahl. Robotic in-row weed control in vegetables. *Computers and electronics in agriculture*, 154:36–45, 2018. doi: 10.1016/j.compag.2018.08.043. [p23]
- J. L. Willers, G. A. Milliken, J. N. Jenkins, C. G. O'Hara, P. D. Gerard, D. B. Reynolds, D. L. Boykin, P. V. Good, and K. B. Hood. Defining the experimental unit for the design and analysis of site-specific experiments in commercial cotton fields. *Agricultural Systems*, 96(1-3):237–249, 2008. doi: 10.1016/j.agsy.2007.09.003. [p24]
- M. Wójtowicz, A. Wójtowicz, and J. Piekarczyk. Application of remote sensing methods in agriculture. *Communications in Biometry and Crop Science*, 11:31–50, 2016. [p25]
- K. Wright. *desplot: Plotting Field Plans for Agricultural Experiments*, 2021. URL <https://CRAN.R-project.org/package=desplot>. R package version 1.9. [p24]
- X. Zhang, X. Li, B. Zhang, J. Zhou, G. Tian, Y. Xiong, and B. Gu. Automated robust crop-row detection in maize fields based on position clustering algorithm and shortest path method. *Computers and electronics in agriculture*, 154:165–175, 2018. doi: 10.1016/j.compag.2018.09.014. [p24]

Felipe de Mendiburu

Universidad Nacional de Ingeniería and Universidad Nacional Agraria La Molina

Av. Tupac Amaru 210, apartado 1301, Rimac

Lima, Peru

felipe.demendiburu.d@uni.pe

also

Av. La Molina s/n, La Molina

Lima, Peru

ORCID: 0000-0003-2725-5911

fmendiburu@lamolina.edu.pe

Augusto Choy

Universidad Nacional de Ingeniería and ESAN University

Av. Tupac Amaru 210, apartado 1301, Rimac

Lima, Peru

augusto.choy.p@uni.pe

also

Av. Alonso de Molina 1652, Apartado 1846, Surco

Lima, Peru

ORCID: 0000-0002-8940-6896

achoy@esan.edu.pe

Rodrigo Morales

Instituto de Investigación Agropecuaria de Panamá (IDIAP)

C. Carlos Lara 157

Panamá

ORCID: 0000-0002-7230-4578

rodrigoamoralesa@gmail.com

Roberto Quiroz

Instituto de Innovación Agropecuaria de Panamá y Sistema Nacional de Investigación, Secretaría Nacional de Ciencia y Tecnología.

David, Chiriquí 04001

Panamá

ORCID: 0000-0001-8401-2700

roberto.quiroz@catie.ac.cr

David Mauricio

Universidad Nacional Mayor de San Marcos

Av. German Amezaga s/n, Lima 15081

Lima, Peru

ORCID: 0000-0001-9262-626X

dmauricios@unmsm.edu.pe

Frame to frame interpolation for high-dimensional data visualisation using the woylier package

by Zoljargal Batsaikhan, Dianne Cook, and Ursula Laa

Abstract The woylier package implements tour interpolation paths between frames using Givens rotations. This provides an alternative to the geodesic interpolation between planes currently available in the *tourr* package. Tours are used to visualise high-dimensional data and models, to detect clustering, anomalies and non-linear relationships. Frame-to-frame interpolation can be useful for projection pursuit guided tours when the index is not rotationally invariant. It also provides a way to specifically reach a given target frame, useful for some applications. We demonstrate the method for exploring non-linear relationships between currency cross-rates.

1 Introduction

When data has up to three variables, visualization is relatively intuitive, while with more than three variables, we face the challenge of visualizing high dimensions on 2D displays. This issue was tackled by the *grand tour* (Asimov, 1985), which can be used to view data in more than three dimensions using linear projections. It is based on the idea of rotations of a lower-dimensional projection in high-dimensional space. The grand tour allows users to see dynamic low-dimensional (typically 2D) projections of higher-dimensional space. Originally, Asimov's grand tour presented the viewer with an automatic movie of projections with no user control. Since then, new work has added interactivity to the tour, giving more control to users (Buja et al., 2005). New variations include the manual (Cook and Buja, 1997) or radial tour (Laa et al., 2023), little tour, guided tour (Cook et al., 1995), local tour, and planned tour. These are different ways of selecting the sequence of projection bases for the tour; see Lee et al. (2022).

The guided tour combines projection pursuit with the grand tour, and it is implemented in the *tourr* package (Wickham et al., 2011). Projection pursuit, originally proposed in Kruskal (1969), is a procedure used to locate the projection of high-to-low dimensional space that should expose the most interesting feature of the data. It involves defining a criterion of interest, a numerical objective function that indicates the interestingness of each projection, and an optimization for selecting planes with increasing values of the function. A number of such criteria have been developed based on clustering, spread, and outliers and made available in R packages. The *tourr* contains the indexes defined in Cook et al. (1993) that can be used for clustering, outliers, and skewness, and is the only package making the guided tour available. Large data versions of these can be found in the package *PPbigdata* (Duan and Cabrera, 2024). The package *pursuit* (Ossani and Cirillo, 2026) also contains these indexes and several more. The *PPCI* (Hofmeyr and Pavlidis, 2019) does projection pursuit as part of a cluster analysis, and the *ppgmmga* (Scrucca and Serafini, 2019) provides an index for clustering based on Gaussian mixture models. The *REPPlab* (Fischer et al., 2021) provides indexes for detecting clusters and outliers.

A tour path is a sequence of projections, and we use an interpolation to produce small steps simulating a smooth movement. The current implementation of tour in the *tourr* package uses geodesic interpolation between planes. The geodesic interpolation path is the shortest path between planes with no within-plane spin (see Buja et al. (2004) for more details). As a result, the rendered target plane could be a within-plane rotation of the target plane originally specified. This is not a problem when the structure we are looking for can be identified from any rotation. However, even simple associations in 2D, such as the calculated correlation between variables, can be very different when the basis is rotated.

Most projection pursuit indexes, particularly those provided by *tourr*, are rotationally invariant. However, there are some applications where the orientation of frames does matter. One example is the splines index proposed by Grimm (2016). The splines index computes a spline model for the two variables in a projection (using the implementation in *mgcv* (Wood, 2011)), in order to measure non-linear association. It compares the variance of the response variable to the variance of residuals, such that the functional dependence is stronger when the index value is larger. It can be useful to detect non-linear relationships in high-dimensional data. However, its value will change substantially if the projection is rotated within the plane (Laa and Cook, 2020). The procedure in Grimm (2016) is less affected by the orientation because it considers only pairs of variables, and it selects the maximum value found when exchanging which variable is considered as a predictor or response variable.

Figure 1 illustrates the rotational invariance problem for a modified splines index, where we

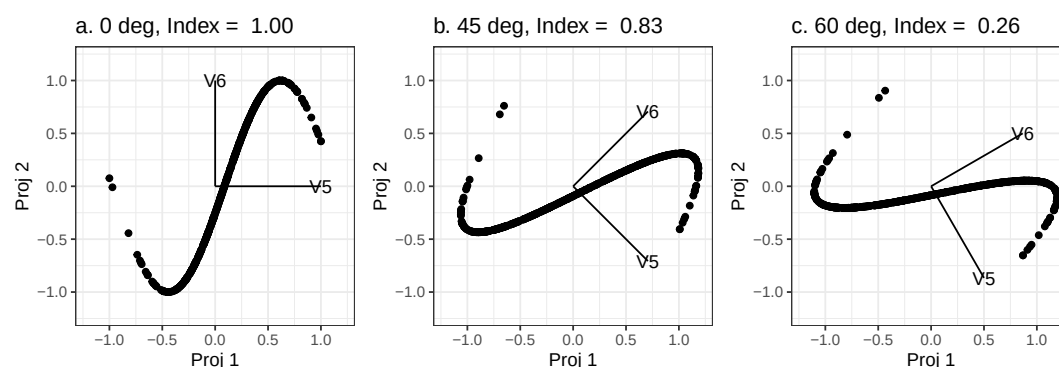


Figure 1: The impact of rotation on a spline index that is NOT rotation invariant. The index value for different within-plane rotations takes very different values: (a) the original projection has a maximum index value of 1.00, (b) axes rotated 45° drops the index value to 0.83, (c) axes rotated 60° drops the index to a very low 0.26. Geodesic interpolation between planes will have difficulty finding the maximum of an index like this because it is focused only on the projection plane, not the frame defining the plane.

always consider the horizontal direction as the predictor variable and the vertical direction as the response. Thus, our modified index computes the splines on one orientation, exaggerating the rotational variability. The example data was simulated to follow a sine curve, and the modified splines index is calculated on different within-plane rotations of the data. Although they have the same structure, the index values vary greatly.

The lack of rotation invariance of the splines index raises complications in the optimization process in the projection-pursuit guided tour as available in [tourr](#). Fixing this is one motivation of this work, and it makes the explanation of frame-to-frame interpolation easier. The goal with the frame-to-frame interpolation is that optimization would find the best within-plane rotation, and thus appropriately optimize the index.

We should note that if optimization were the only goal, it could be achieved simply by modifying the index to also optimize across all frames in the same plane by adding a within-plane rotation and index calculation. This might slow the optimization down a little, but it would achieve the objective of finding the best projection, regardless of orientation. However, there are some specific applications where we would like to reach a specific frame at the target. This is an area of potential future research, but places where we have found it to be desirable are interpolating between frames provided by two different principal component solutions, perhaps computed on two different samples, to digest the similarity and difference between them. Another recent example is to examine high-dimensional ternary diagrams (simplexes) summarizing preferential data, where we would like to rotate through very specific frames corresponding to pairs or preferences to understand how preferences might flow between groups.

A few alternatives to geodesic interpolation were proposed by [Buja et al. \(2005\)](#), including the decomposition of orthogonal matrices, Givens decomposition, and Householder decomposition. The purpose of the [woylieR](#) package is to implement the Givens paths method in R. This algorithm adapts the Givens matrix decomposition technique, which allows the interpolation to be between frames rather than planes.

This article is structured as follows. The next section provides the theoretical framework of the Givens interpolation method, followed by a section about the implementation in R. The method is applied to search for non-linear associations between currency cross-rates.

2 Background

The tour method of visualization shows a movie that is an animated high-to-low dimensional data rotation. It is a one-parameter (time) family of static projections. Algorithms for such dynamic projections are based on the idea of smoothly interpolating a discrete sequence of projections ([Buja et al., 2005](#)).

The topic of this article is the construction of the paths of projections. The interpolation of these paths can be compared to connecting line segments that interpolate points in Euclidean space. Interpolation acts as a bridge between a continuous animation and the discrete choice of sequences of projections.

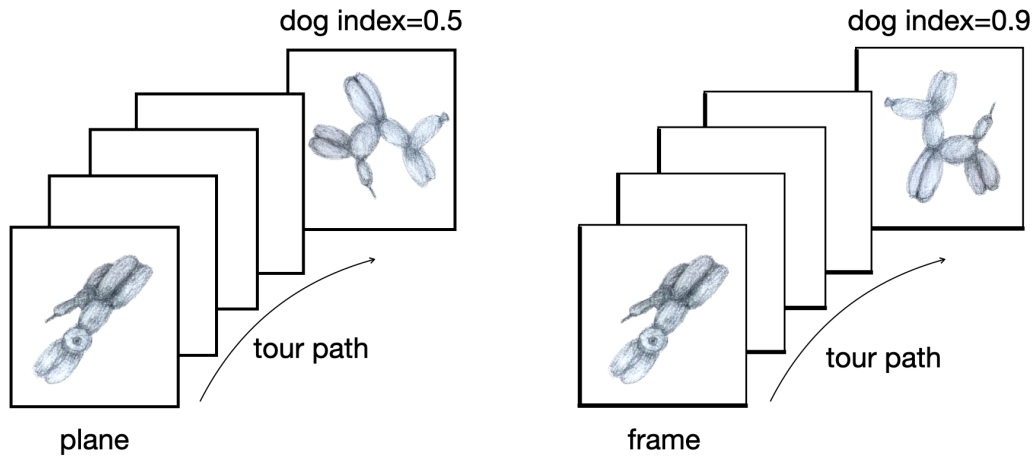


Figure 2: Plane to plane interpolation (left) and frame to frame interpolation (right). We use the dog index for illustration purposes. Orientation of the data within the plane could affect the index value. Frame-to-frame interpolation guarantees reaching a particular frame of the many defining a plane.

Interpolating paths of planes versus paths of frames

The **tour** package implements geodesic interpolation between planes, and the final interpolation step will reach the rotation of the target frame, avoiding any within-plane spin along the path. When the orientation of projections matters, interpolation between frames is required. The orientation of the frames could be important when a non-linear projection pursuit index function is used in the guided tour. This is illustrated by the different index values shown in the sketch in Figure 2, as well as the splines index for the sine curve in Figure 1.

To describe the interpolation algorithms, we will use the following notation.

- Let p be the dimension of the original data and d be the dimension onto which the data is being projected.
- A frame F is defined as a $p \times d$ matrix with pairwise orthogonal columns of unit length that satisfies

$$F^T F = I_d,$$

where I_d is the identity matrix in d dimensions.

- Paths of frames are given by continuous one-parameter families $F(t)$ where $t \in [a, z]$ represents time. We denote the starting frame (at time a) by $F_a = F(a)$ and target frame (at time z) by $F_z = F(z)$. Usually, F_z is the target frame that has been chosen according to the selected tour method. While a grand tour chooses target frames randomly, the guided tour chooses the target frame by optimizing the projection pursuit index. Interpolation methods are used to find the path that moves from F_a to F_z .

Preprojection algorithm

In order to make the interpolation algorithm simple, we carry out a preprojection step to find the subspace that the interpolation path, $F(t)$, is traversing. In other words, the preprojection step defines the joint subspace of F_a and F_z and makes sure the interpolation path is limited to that space.

The procedure starts with forming an orthonormal basis by applying Gram-Schmidt to F_z with regard to F_a : $F_\star = F_z - F_a(F_a^T F_a)^{-1} F_a^T F_z$, followed by orthonormalizing subsequent columns of $F_\star$ against all preceding columns. (The function `orthonormalise_by()` in the **tour** does this.) Then we build the preprojection basis B by combining F_a and $F_\star$ as follows:

$$B = (F_a, F_\star)$$

The dimension of the resulting orthonormal basis, B , is $p \times 2d$.

Then, we can express the original frames in terms of this basis:

$$F_a = B W_a, F_z = B W_z$$

The interpolation problem is then reduced to the construction of paths of frames $W(t)$ that interpolate between the preprojected frames W_a and W_z . By construction, W_a is a $2d \times d$ matrix of

1s and 0s. This is an important characteristic of our interpolation algorithm of choice, the Givens interpolation.

Givens interpolation path algorithm

A rotation matrix is a transformation matrix used to perform a rotation in Euclidean space. The matrix that rotates a 2D plane by an angle θ looks like this:

$$\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

If the rotation is in the plane of two selected variables, it is called a Givens rotation. Let's denote those 2 variables as i and j . The Givens rotation is used for introducing zeros, for example, when computing the QR decomposition of a matrix in linear algebra problems.

The interpolation method in the [woylieR](#) package is based on the fact that in any vector of a matrix, one can zero out the i -th coordinate with a Givens rotation in the (i, j) -plane for any $j \neq i$ (Golub and Loan, 2013). This rotation affects only coordinates i and j and leaves all other coordinates unchanged. Sequences of Givens rotations can map any orthonormal d -frame F in p -space to the standard d -frame

$$E_d = ((1, 0, 0, \dots)^T, (0, 1, 0, \dots)^T, \dots).$$

The resulting interpolation path construction algorithm from starting frame F_a to target frame F_z is illustrated below. The example is for $p = 6$ and $d = 2$.

1. Construct preprojection basis B by orthonormalizing F_z with regard to F_a with Gram-Schmidt (same procedure as described above).

In our example, F_a and F_z are $p \times d$ or 6×2 matrices that are orthonormal. The preprojection basis B is a $p \times 2d$ matrix that is 6×4 .

2. Get the preprojected frames using the preprojection basis B .

$$W_a = B^T F_a = E_d$$

and

$$W_z = B^T F_z$$

In our example, W_a looks like:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

W_z is an orthonormal $2d \times d$ matrix that looks like:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \\ a_{41} & a_{42} \end{bmatrix}$$

3. Then, we can construct a sequence of Givens rotations that maps W_z to W_a with such angles that make one element zero at a time:

$$W_a = R_m(\theta_m) \dots R_2(\theta_2) R_1(\theta_1) W_z$$

At each rotation, the angle θ_i that zeros out the next coordinate of a plane is calculated. Here $m = \sum_{k=1}^d (2d - k)$, so when $d = 2$ we need $m = 5$ rotations with 5 different angles, each making one element 0. For example, the first rotation angle θ_1 is an angle in radians between $(1, 0)$ and (a_{11}, a_{21}) . This rotation matrix would make element a_{21} zero:

$$R_1(\theta_1) = G(1, 2, \theta_1) = \begin{bmatrix} \cos \theta_1 & -\sin \theta_1 & 0 & 0 \\ \sin \theta_1 & \cos \theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Here $G(i, j, \theta_k)$ denotes a Givens rotation in components i and j by angle θ_k . In the same way, we zero out the elements a_{31} and a_{41} . Because of the orthonormality this means that now $a_{11} = 1$ and that $a_{12} = 0$. We thus need only two more rotations to zero out a_{32} and a_{42} .

Each θ_i is an angle in 2D and is computed from the polar coordinates returned by the `atan2()` function.

4. The inverse mapping is obtained by reversing the sequence of rotations with the negative of the angles; we start from the starting basis and end at the target basis.

$$R(\theta) = R_1(-\theta_1) \dots R_m(-\theta_m), W_z = R(\theta)W_a$$

Performing these rotations would go from the starting frame to the target frame in one step, but we want to do it sequentially in a number of steps so that interpolation between frames looks dynamic.

5. Next, we include the time parameter, t , so that the interpolation process can be rendered in the movie-like sequence. We break each θ_k into the number of steps, n_{step} , that we want to take from the starting frame to the target frame, which means it moves by an equal angle in each step. Here, n_{step} should vary based on the angular distance between F_a and F_z , such that when watching a sequence of interpolations, we have a fixed angular speed.
6. Finally, we reconstruct our original frames using B . This reconstruction is done at each step of interpolation so that we have the interpolated path of frames as the result.

$$F_t = BW_t$$

At each time t , we can project the data using the frame F_t .

3 Implementation

We implemented each of the steps in the Givens interpolation path algorithm in separate functions and combined them into a single function `givens_full_path()` to produce the full set of F_t . The same functions are used to integrate the Givens interpolation with the `animate()` functions of the **tourr** package. Table 1 lists the input and output of each function and its descriptions. Functions to use with `animate()` are described separately below.

When using **tourr**, we typically want to run a tour live, such that target selection and interpolation are interleaved, and the display will show the data for each frame F_t in the interpolation path. The implementation in **tourr** was described in Wickham et al. (2011), and with **woylier** we provide functions to use the Givens interpolation with the grand tour, guided tour and planned tour. To do this, we rely primarily on the function `givens_info()` which calls the functions listed in the Table above and collects all necessary information for interpolating between a given starting and target frame. The function `givens_path()` then defines the interpolation and can be used instead of `tourr::geodesic_path()`. Wrapper functions for the different tour types are available to use this interpolation, since in the `tourr::grand_tour()` and other path functions, this is fixed to use the geodesic interpolation. Calling a grand tour with Givens interpolation for direct animation will then use:

```
tourr::animate_xy(<data>, tour_path = woylier::grand_tour_givens())
```

4 Comparison of geodesic interpolation and Givens interpolation

The `givens_full_path()` function returns the intermediate interpolation step projections for a given number of steps. The code chunk below demonstrates the interpolation between 2 random bases in 5 steps.

```
givens_full_path(base1, base2, nsteps = 5)
```

To compare the path generated with the Givens interpolation to that found with geodesic interpolations, we look at the rotation of the sine data shown in Figure 1. We consider a subset in $p = 4$ dimensions where the first two dimensions contain noise and the last two contain the sine curve. Starting from a random projection, we want to interpolate towards the original sine curve. The path comparison is shown in Figure 3.

Plotting the interpolated paths

For further comparison and to check that the interpolation is moving in equally sized steps, we directly plot the interpolated paths. The space of 1D projections defines a unit sphere, while

Table 1: Primary functions in the woylier package.

name	description	input	output
<code>givens_full_path(Fa, Fz, nsteps)</code>	Construct full interpolated frames.	Starting and target frame (Fa, Fz) and number of steps	An array with nsteps matrix. Each matrix is interpolated frame in between starting and target frames.
<code>preprojection(Fa, Fz)</code>	Build a d-dimensional pre-projection space by orthonormalizing Fz with regard to Fa.	Starting and target frame (Fa, Fz)	B pre-projection $p \times 2D$ matrix
<code>construct_preframe(Fa, B)</code>	Construct preprojected frames.	A frame and the pre-projection $p \times 2D$ matrix	Pre-projected frame in pre-projection space
<code>row_rot(a, i, k, theta)</code>	Performs Givens rotation .	A frame and the pre-projection $p \times 2D$ matrix	theta angle rotated matrix a
<code>calculate_angles(Wa, Wz)</code>	Calculate angles of required rotations to map Wz to Wa.	Preprojected frames (Wa, Wz)	Names list of angles
<code>construct_moving_frame(Wt, B)</code>	Reconstruct interpolated frames using pre-projection.	Pre-projection matrix B, Each frame of Givens path	A frame of on a step of interpolation

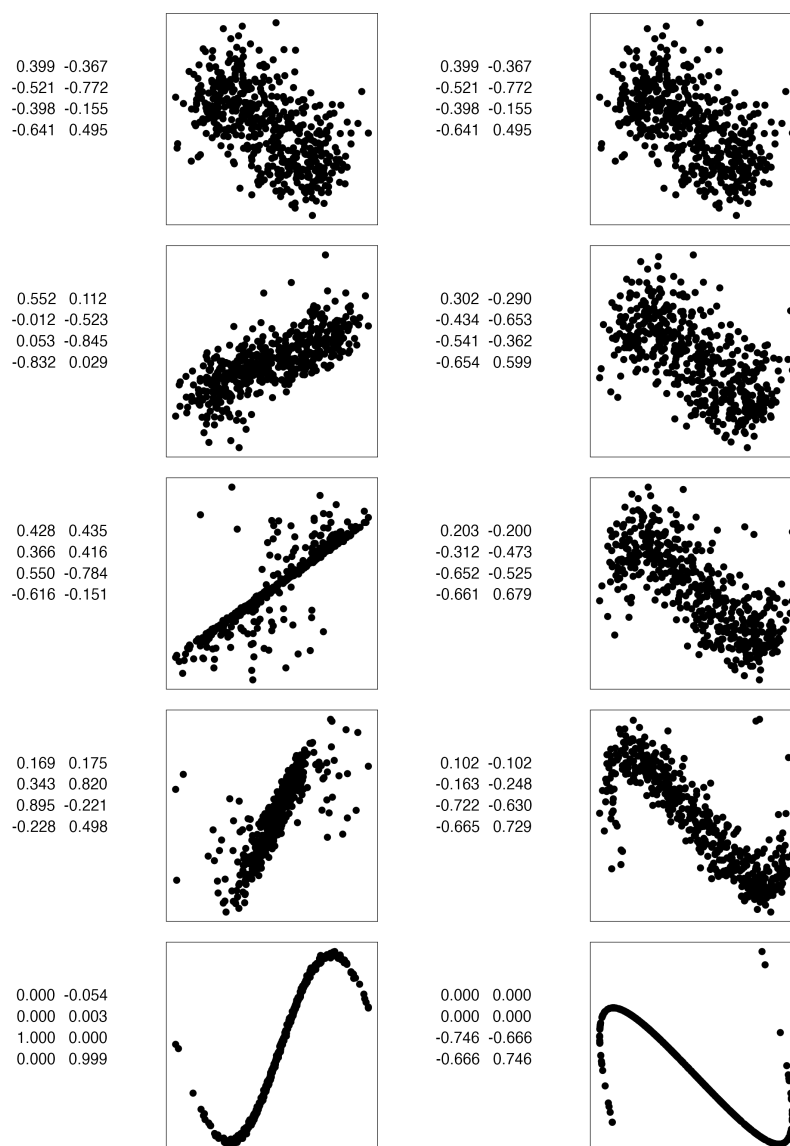


Figure 3: Comparison of Givens path and geodesic path between 2D projections. The Givens path preserves the frame ending at the provided basis (frame), while geodesic is agnostic to the particular basis. In general, the geodesic is preferred because it removes within-plane spin, but occasionally it is helpful to very specifically arrive at the prescribed basis.

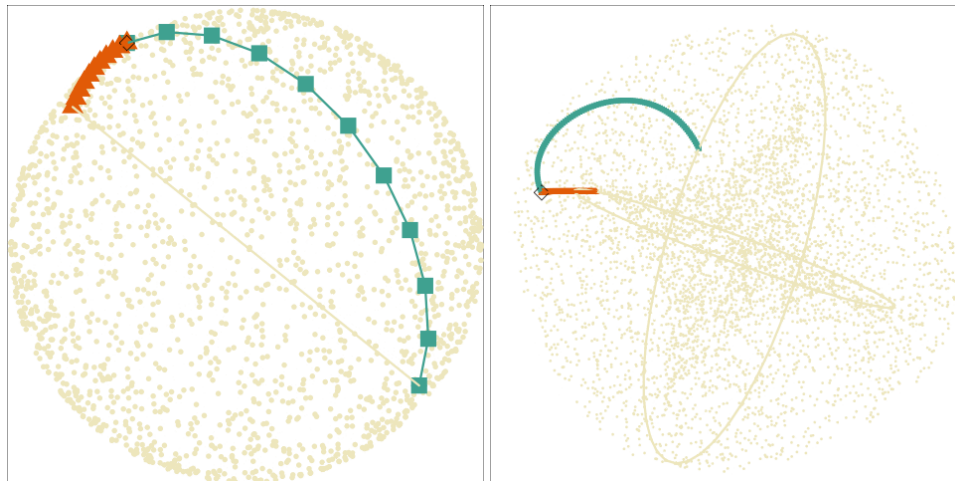


Figure 4: Interpolation steps of 1D (left) and 2D (right) projections of 3D data made with a Givens path (forest green square) and a geodesic path (dark orange triangle). The black diamond indicates the starting basis common to both interpolations. The cream points represent the space of all projections, which is a sphere for 1D projections and a torus for 2D projections. In the 1D example, geodesic arrives at the opposite side of the sphere to Givens, indicating that it has flipped the direction of the vector in order to make the shortest path to the same plane. A similar thing happens for the 2D example, geodesic flips the sign of one basis vector, but it defines the same plane, as indicated by the cream circles.

2D projections define a torus. To illustrate the space, points on the surface of the sphere and the torus shape are randomly generated by functions from the [geozoo](#) package (Schloerke, 2016). The interpolated paths are then compared within that space.

A 1D projection of data in p dimensions corresponds to a linear combination where the weights are normalized. Therefore, we can plot the point on the surface of a hypersphere. In this case, the Givens interpolation will reach the exact point, while geodesic interpolation might flip the direction and reach a point on the opposite side of the hypersphere. Figure 4 (left) shows the comparison of the interpolation steps using the same target plane, for an example with $p = 3$. Because the flipped target is close to the starting plane, the geodesic path is a lot shorter.

In the case of 2D projections, we can plot the interpolated path between 2 frames on a torus. A torus can be seen as a crossing of 2 circles that are orthonormal, as is the case with our projection onto 2D. Figure 4 (right) compares the interpolation paths for $p = 3$.

For a 2D projection, the same target plane is found when rotating the basis within the plane or when reflecting across one of the two directions (the reflected basis can then also be rotated). This means the space of target bases is constrained to two circles on the torus, and these are disconnected because the reflection corresponds to a jump. In the high-dimensional space, we can imagine the reflection as flipping over the target plane, resulting in a reflection of the normal vector on the plane. While the Givens interpolation will reach the exact basis, a geodesic interpolation towards the same target plane can land anywhere along those two circles, depending on the starting basis in the interpolation.

5 Application

This section describes the application of the Givens interpolation path with a guided tour to explore non-linear association in multivariate data. We use cross-rates for currencies relative to the US dollar. A cross-rate is an exchange rate between two currencies computed by reference to a third currency, usually the US dollar. A strong non-linear functional relationship would indicate that underlying the collection of cross-currency rates is a single latent factor explaining all the movement in that time period.

The data were extracted from Rates (2026) and contain cross-rates for ARS, AUD, EUR, JPY, KRW, MYR between 2019-11-1 and 2020-03-31. Figure 5 shows how the currencies changed relative to USD over the time period. We see some collective behavior in March of 2020 with EUR and JPY increasing in a similar manner, and smaller currencies decreasing in value. This could be understood as a consequence of flight-to-quality at these uncertain times.

We are interested in capturing this relation over time, and from the time series visualization,

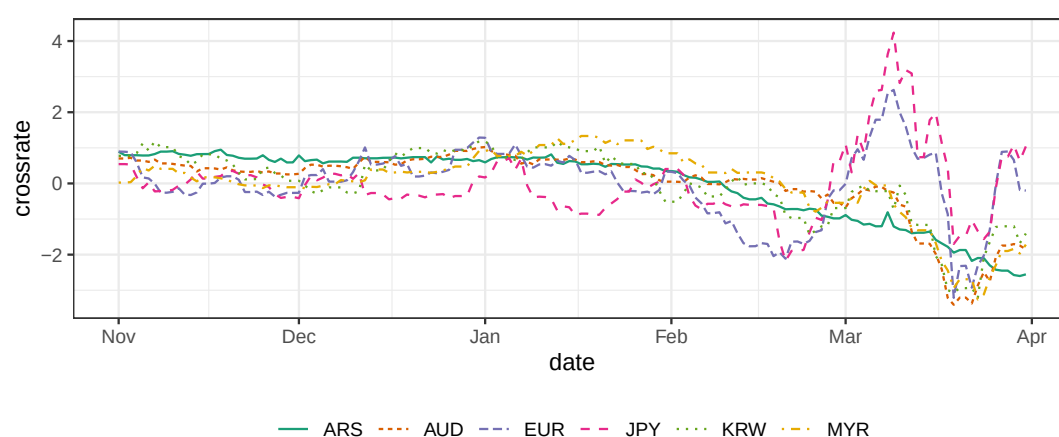


Figure 5: Examining the behaviour of six cross-currency rates (ARS, EUR, KRW, AUD, JPY, MYR) prior to and in the first month of the pandemic. All the currencies are standardised, and the sign is flipped. JPY and EUR strengthened against the USD (high values) in March, while the other currencies weakened (low values).

we expect that we can capture the main dynamics in a two-dimensional projection from the six-dimensional space of currencies. Thus, we start from $p = 6$ (the different currencies) and $n = 152$, the number of days in our sample. We expect that a projection onto $d = 2$ dimensions should capture the relation between the two groups of currencies mentioned above, and this should be identified within the noise of the random fluctuations.

We may expect that we can capture the dependence between the two groups using principal components analysis (PCA). Figure 6 shows a scatter plot matrix of the principal components (PCs) of our dataset. Indeed, we find a strong non-linear association between the first two PCs. Investigation of the rotation shows that the first PC is primarily a balanced combination between ARS, AUD, KRW, and MYR (and a smaller contribution of EUR), and the second contribution is dominated by EUR and JPY, contrasted with smaller contributions from ARS and MYR. Our next step is to use projection pursuit to further explore for non-linear functional dependence.

Guided tour optimization

We now use the splines index to further explore functional dependence beyond what is observed from PCA. Note here that because of strong linear correlations between the currencies, we start from the first four PCs, explaining over 97% of the variance in the data. To make the challenge of finding functional dependence harder, we start from a projection onto the third and fourth PCs. We hope to find a stronger functional dependence than uncovered by PCA. The `display_pca()` function in **tourr** is used to show the original variables that correspond to the 2D projection of the first four PCs, so that interpretation can be made with respect to the cross-currencies.

There are several options for optimization in the **tourr** package. Geodesic optimization (GEOO) (provided by the `search_geodesic()` function) is a derivative-free method approximating a stochastic optimization algorithm. From any starting plane, a random search in the near neighborhood delivers the most promising direction to follow. The tour using geodesic interpolation follows this direction until the index value decreases, and then a new direction search is conducted. It guarantees that the index value always increases, but it can be slow, and it can get trapped at local maxima. Typically, the best method is simulated annealing (provided by the `search_better()` function), which performs a random search of a large neighborhood and then cools down the neighborhood size as the optimization progresses. We have used this approach, inserting geodesic (SAGEO) and Givens interpolation (SAGIV) to move between targets.

The results are shown in Figure 7 and we see on the right that the guided tour with SAGIV has identified a non-linear functional dependence between the x and y axes in the final projection. The result on the left is using GEOO, and the result in the middle uses SAGEO. Since these methods do not allow for within-plane rotation, they were not able to identify the same and best functional relationship even though they are using the same starting plane and index function.

To understand the result in more detail, we use tools from **ferrn** (Zhang et al., 2021) to show how the index value is changing over the optimization in Figure 8. Points indicate selected target planes and are connected by lines showing the index value along the interpolated path between them. For ease of comparison, the horizontal line shown in all three panels indicates the maximum index value found when using Givens interpolation.

The top row shows the path found via GEOO. This search strategy only moves along geodesic

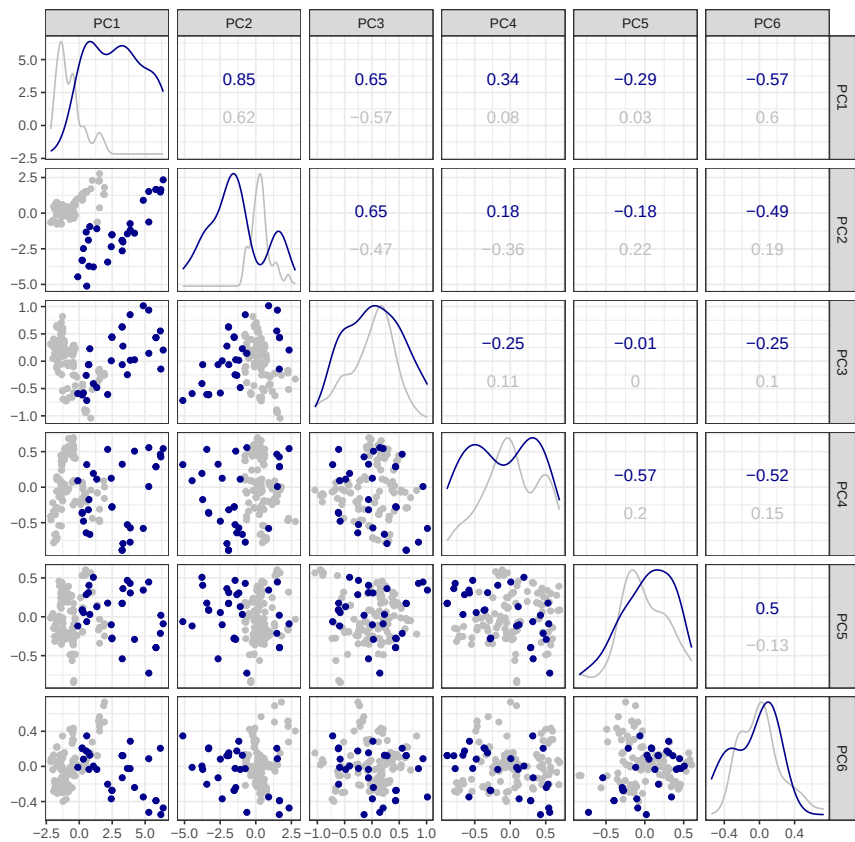


Figure 6: Scatterplot matrix of the six principal components. Observations in March 2020 are highlighted in dark blue; all other months are shown in grey. There is a strong non-linear dependence between PC1 and PC2.

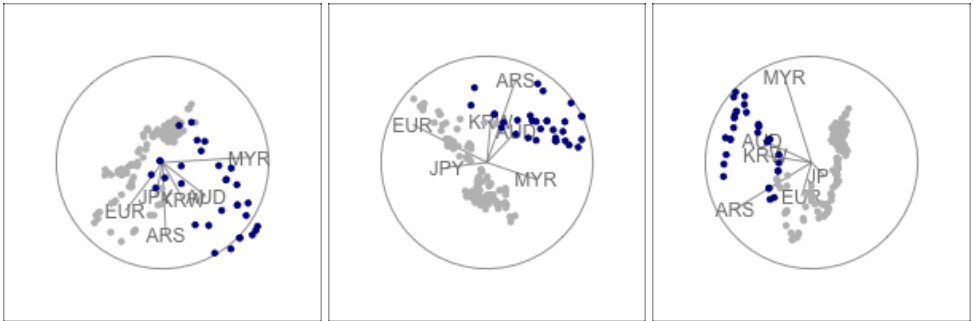


Figure 7: Final view after optimization in the guided tour using geodesic optimization (GEOO) (left), simulated annealing with geodesic interpolation (SAGEO) (middle) and simulated annealing with Givens interpolation (SAGIV) (right).

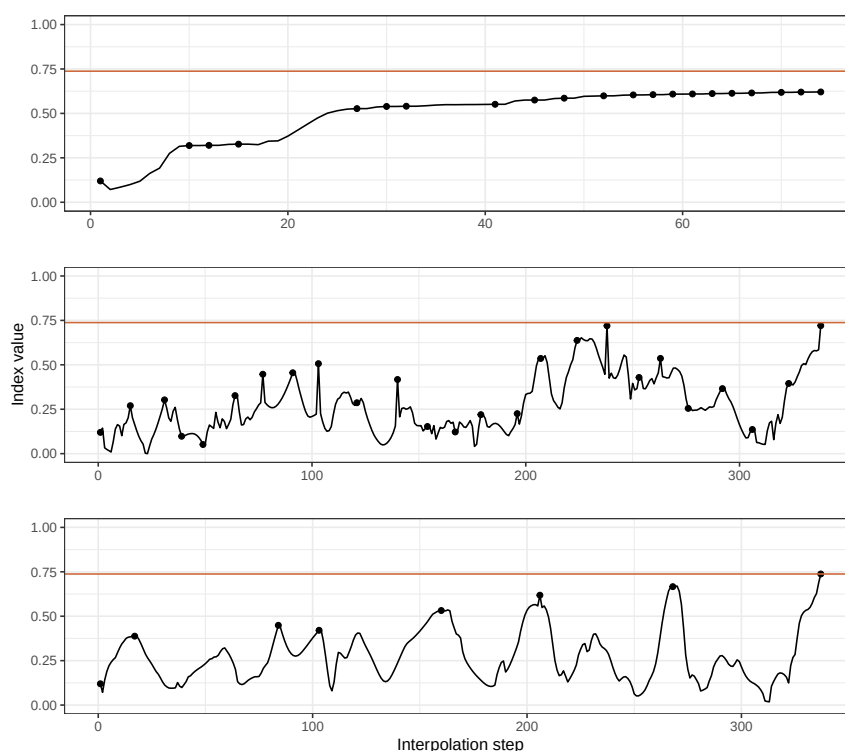


Figure 8: Splines index value along the interpolated optimization path in the guided tour using geodesic optimization (GEOO) (top), simulated annealing with geodesic interpolation (SAGEO) (middle) and with Givens interpolation (SAGIV) (bottom). Points indicate index values at target planes selected during the optimization, and the horizontal line shows the maximum across the three searches. With GEOO no within-plane rotation is possible and although the optimization always heads towards higher index values it finishes at a different frame in the same plane, with a lower than possible index value. With SAGEO, a different frame in the same plane interferes with the optimization. This does not happen with SAGIV, allowing the search to find the optimal view.

paths to a target *plane* and does not reach a specific target *frame*. By construction, this gives a smooth path. However, we can see that as a consequence, the optimization converges to a plane that does not identify the pattern of interest.

In the middle, we are showing the path found when using SAGEO. In this case, the search can suggest any target frame, also allowing for any within-plane rotation. The algorithm will compute the index value for the suggested target frame and accept it if the index value is larger than that for the current frame. However, once the target is accepted, the interpolation step will reset the orientation within the plane, potentially resulting in a frame that shows the same plane but with a lower index value. This is why we see drops in the index value even from one target plane to the next, and the optimization also does not identify the pattern of interest, even though it comes close.

Finally, in the bottom row, we see the result of the random search with Givens interpolation. While the index value can drop along an interpolation path, the value is strictly increasing for the target planes (up to small differences from numeric precision in the interpolation), as is required for the optimization. Note also that the search converged much faster (fewer target planes were selected). The length of the interpolated paths is similar because, for a fixed angular step size, the Givens interpolation will take more steps to interpolate between frames.

6 Conclusion

This paper describes the R package [woylier](#) which provides an implementation of the Givens interpolation path algorithm that can be used as an alternative interpolation method for a tour. The algorithm implemented in the [woylier](#) package is as described in [Buja et al. \(2005\)](#).

It is important to mention that [woylier](#) package should be used in conjunction with the [tourr](#) package. Currently, it is implemented for tours into 1D or 2D. A potential future development would be to generalize the interpolation for more than 2D projections of data, which could not be done here because it requires new mathematical derivations.

For most purposes, geodesic interpolation is desirable. The motivation for frame-to-frame interpolation arises from a rotational invariance problem of the current geodesic interpolation algorithm implemented in **tourr** package. Frame-to-frame interpolation provides users with the tools to move to a specified frame, not the plane it defines.

Our application shows the Givens interpolation used with a projection pursuit index, which is not rotationally invariant but ideal for detecting bivariate non-linear association in high-dimensional data. The potential applications where this might be useful are many and varied, possibly including facial recognition algorithms and morphing between images, or imputing missing time steps in high-dimensional time series.

Acknowledgements

A prior implementation using C was available in the XGobi software (Swayne et al., 1998). This new implementation was undertaken at the request of a **tourr** package user.

We thank Saurabh Jhanjee for his contributions at the start of the project. This article is created using **knitr** (Xie, 2022) and **rmarkdown** (Allaire et al., 2026; Xie et al., 2018, 2020) in R. For graphs we have used **ggplot2** (Wickham, 2016), **cowplot** (Wilke, 2020) and **GGally** (Schloerke et al., 2021), for tables we have used **kableExtra** (Zhu, 2021). The source code for reproducing this paper can be found at: <https://github.com/uschiLaa/woylie-article>.

References

- J. Allaire, Y. Xie, C. Dervieux, J. McPherson, J. Luraschi, K. Ushey, A. Atkins, H. Wickham, J. Cheng, W. Chang, and R. Iannone. *rmarkdown: Dynamic Documents for R*, 2026. URL <https://github.com/rstudio/rmarkdown>. R package version 2.31. [p52]
- D. Asimov. The grand tour: a tool for viewing multidimensional data. *SIAM J Sci Stat Comput* 6(1), pages 128–143, 1985. doi: 10.1137/0906011. [p41]
- A. Buja, D. Cook, D. Asimov, and C. B. Hurley. Theory of dynamic projections in high-dimensional data visualization. 2004. URL <http://stat.wharton.upenn.edu/~buja/PAPERS/paper-dyn-proj-math.pdf>. [p41]
- A. Buja, D. Cook, D. Asimov, and C. Hurley. Computational methods for high-dimensional rotations in data visualization. *Handbook of Statistics*, pages 391–413, 2005. doi: 10.1016/s0169-7161(04)24014-7. [p41, 42, 51]
- D. Cook and A. Buja. Manual controls for high-dimensional data projections. *Journal of Computational and Graphical Statistics*, 6(4):464–480, 1997. ISSN 1061-8600. doi: 10.2307/1390747. URL <http://www.jstor.org/stable/1390747>. [p41]
- D. Cook, A. Buja, and J. Cabrera. Projection pursuit indexes based on orthonormal function expansions. *Journal of Computational and Graphical Statistics*, 2:225–250, 1993. doi: 10.1080/10618600.1993.10474610. URL <https://doi.org/10.1080/10618600.1993.10474610>. [p41]
- D. Cook, A. Buja, J. Cabrera, and C. Hurley. Grand tour and projection pursuit. *Journal of Computational and Graphical Statistics*, 4(3):155–172, 1995. doi: 10.1080/10618600.1995.10474674. URL <https://www.tandfonline.com/doi/abs/10.1080/10618600.1995.10474674>. [p41]
- Y. Duan and J. Cabrera. *PPbigdata: Projection Pursuit for Big Data Based on Data Nuggets*, 2024. URL <https://CRAN.R-project.org/package=PPbigdata>. R package version 1.0.0. [p41]
- D. Fischer, A. Berro, K. Nordhausen, and A. Ruiz-Gazen. *REPPlab: An R package for detecting clusters and outliers using exploratory projection pursuit*, 2021. URL <https://doi.org/10.1080/03610918.2019.1626880>. [p41]
- G. H. Golub and C. F. V. Loan. *Matrix computations (4th ed)*. Johns Hopkins University Press, 2013. ISBN 0801837723. doi: 10.1353/book.23351. [p44]
- K. Grimm. *Kennzahlenbasierte Grafikauswahl*. doctoral thesis, Universität Augsburg, 2016. [p41]
- D. P. Hofmeyr and N. G. Pavlidis. PPCI: an R package for cluster identification using projection pursuit. *The R Journal*, 2019. doi: 10.32614/RJ-2019-046. URL <https://journal.r-project.org/archive/2019/RJ-2019-046/index.html>. [p41]

- J. Kruskal. Toward a practical method which helps uncover the structure of a set of observations by finding the line transformation which optimizes a new index of condensation. *Statistical computation; New York, Academic Press*, pages 427–440, 1969. doi: 10.1016/B978-0-12-498150-8.50024-0. [p41]
- U. Laa and D. Cook. Using tours to visually investigate properties of new projection pursuit indexes with application to problems in physics. *Computational Statistics*, 35:1171–1205, 2020. doi: 10.1007/s00180-020-00954-8. [p41]
- U. Laa, A. Aumann, D. Cook, and G. Valencia. New and simplified manual controls for projection and slice tours, with application to exploring classification boundaries in high dimensions. *Journal of Computational and Graphical Statistics*, 0(0):1–8, 2023. doi: 10.1080/10618600.2023.2206459. [p41]
- S. Lee, D. Cook, N. da Silva, U. Laa, N. Spyrisson, E. Wang, and H. S. Zhang. The state-of-the-art on tours for dynamic visualization of high-dimensional data. *WIREs Computational Statistics*, 14(4):e1573, 2022. doi: 10.1002/wics.1573. [p41]
- P. C. Ossani and M. A. Cirillo. *Pursuit: Projection Pursuit*, 2026. URL <https://CRAN.R-project.org/package=Pursuit>. R package version 1.1.0. [p41]
- O. E. Rates. Open exchange rates API, 2026. URL <https://openexchangerates.org>. Accessed: 2022-10-12. [p48]
- B. Schloerke. *geozoo: Zoo of Geometric Objects*, 2016. URL <https://CRAN.R-project.org/package=geozoo>. R package version 0.5.1. [p48]
- B. Schloerke, D. Cook, J. Larmarange, F. Briatte, M. Marbach, E. Thoen, A. Elberg, and J. Crowley. *GGally: Extension to 'ggplot2'*, 2021. URL <https://CRAN.R-project.org/package=GGally>. R package version 2.1.2. [p52]
- L. Scrucca and A. Serafini. Projection pursuit based on gaussian mixtures and evolutionary algorithms. *Journal of Computational and Graphical Statistics*, 28(4):847–860, 2019. doi: 10.1080/10618600.2019.1598871. [p41]
- D. F. Swayne, D. Cook, and A. Buja. XGobi: Interactive dynamic data visualization in the X window system. *Journal of Computational and Graphical Statistics*, 7(1):113–130, 1998. doi: 10.1080/10618600.1998.10474764. URL <https://github.com/ggobi/xgobi>. [p52]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p52]
- H. Wickham, D. Cook, H. Hofmann, and A. Buja. tourr: An R package for exploring multivariate data with projections. *Journal of Statistical Software*, 40(2):1–18, 2011. doi: 10.18637/jss.v040.i02. [p41, 45]
- C. O. Wilke. *cowplot: Streamlined Plot Theme and Plot Annotations for 'ggplot2'*, 2020. URL <https://CRAN.R-project.org/package=cowplot>. R package version 1.1.1. [p52]
- S. N. Wood. Fast stable restricted maximum likelihood and marginal likelihood estimation of semi-parametric generalized linear models. *Journal of the Royal Statistical Society (B)*, 73(1):3–36, 2011. doi: 10.1111/j.1467-9868.2010.00749.x. [p41]
- Y. Xie. *knitr: A General-Purpose Package for Dynamic Report Generation in R*, 2022. URL <https://yihui.org/knitr/>. R package version 1.38. [p52]
- Y. Xie, J. Allaire, and G. Golemund. *R Markdown: The Definitive Guide*. Chapman and Hall/CRC, Boca Raton, Florida, 2018. ISBN 9781138359338. doi: 10.1201/9781138359444. URL <https://yihui.org/rmarkdown/>. [p52]
- Y. Xie, C. Dervieux, and E. Riederer. *R Markdown Cookbook*. Chapman and Hall/CRC, Boca Raton, Florida, 2020. ISBN 9780367563837. doi: 10.1201/9781003097471. URL <https://yihui.org/rmarkdown-cookbook>. [p52]
- H. S. Zhang, D. Cook, U. Laa, N. Langrené, and P. Menéndez. Visual diagnostics for constrained optimisation with application to guided tours. *The R Journal*, 13(2):624–641, 2021. doi: 10.32614/RJ-2021-105. URL <https://huizezhang-sherry.github.io/ferrn/>. [p49]
- H. Zhu. *kableExtra: Construct Complex Table with 'kable' and Pipe Syntax*, 2021. URL <https://CRAN.R-project.org/package=kableExtra>. R package version 1.3.4. [p52]

Zoljargal Batsaikhan
Monash University
Department of Econometrics and Business Statistics
Clayton, VIC, Australia
<https://github.com/zolabat>
ORCID: 0009-0005-0055-1448
zoljargal11@gmail.com

Dianne Cook
Monash University
Department of Econometrics and Business Statistics
Clayton, VIC, Australia
<http://www.dicook.org>
ORCID: 0000-0002-3813-7155
dicook@monash.edu

Ursula Laa
University of Natural Resources and Life Sciences Vienna
Institute of Statistics
Vienna, Austria
<https://uschilaa.github.io>
ORCID: 0000-0002-0249-6439
ursula.laa@boku.ac.at

highlightr: Creating Highlighted Text Based on Source Frequency

by Rachel Rogers and Susan VanderPlas

Abstract Analysts sometimes need to compare a source document to documents derived from it; examples include comparing notes to an original source document, or when examining consecutive edits (such as Wikipedia article histories). By comparing multiple derivative documents to a source document, researchers can evaluate which portions of the source document are the most preserved in its derivative forms, which may indicate information regarding the importance of these phrases. The **highlightr** package provides analysis and visualization methods to map derivative documents to their source document in the form of a ‘heatmap’ of preserved information. Sequential documents are compared using collocations, or phrases of a set length. Once the frequency with which each phrase appears in derivative documents is calculated, the frequency can be mapped to a corresponding color to create a highlighting gradient. **highlightr** streamlines this process and integrates the source document and heatmap in a useful visual display.

1 Introduction

An individual’s notes can provide valuable insight into their process when learning new information. Comparing note sheets with the presented information allows researchers to determine which areas participants found worth recording. This process is particularly important when conducting studies of how jurors interpret legal arguments, using legal transcripts instead of more costly mock-trial videos. The **highlightr** package (Center for Statistics and Applications in Forensic Evidence et al., 2024) provides a set of functions developed for analyzing and visualizing participant notes to indicate which portions of the transcript were most informative.

The analysis methods in **highlightr** are useful well beyond the legal transcript study for which they were developed. The same methods can also be used to compare other documents which have a similar source, such as abstracts for conference presentations and journal papers on the same topic, or sequential edits to a Wikipedia page. In this paper, we demonstrate the use of the **highlightr** package across a number of different document comparison problems, discussing the design and function of the package as well as the adjustments which must be made as we take on problems with different source-document relationships.

By establishing similarity to the source text, we can determine which portions of the document were judged to be important and use this information to create visualizations highlighting important sections of the document based on users’ notes. This similarity is based on finding the occurrence of phrases from the source document in the derivative documents; the higher the number of occurrences, the more intense the highlighting.

2 Related work

Single document forms of text visualization include the construction of word clouds, where a cluster of frequently used words are shown, with larger words indicating more frequent occurrence. In R, word clouds can be implemented using **wordcloud** (Fellows, 2018). Similarly, frequency values can be shown directly in a barplot (Shibuya and Jensen, 2015). While word clouds and frequency barplots provide useful information about the individual frequency of words in a collection of texts, the surrounding sentence context is not taken into account. Silge visualize gendered verbs used in film scripts through digrams, or sequences of two words, where the first word consisted of a pronoun (“he” or “she”) and the second word was a screen direction. They then plotted the frequency of each word by gendered pronoun in a diverging barplot using **ggplot2** (Wickham, 2016). This text visualization has added context in the form of pronouns, when compared to word clouds. It provides a useful visualization of gender differences present in film scripts. In a more general sense, bigrams can also be visualized (and analyzed) in a graphical network, where segments indicate that two words form a bigram (Shibuya and Jensen, 2015). When evaluating a text as a whole, Shibuya and Jensen (2015) also use dispersion plots to depict the locations of n-grams, or phrases of length n, throughout a text. In a dispersion plot, one axis corresponds to the word count, and a line is drawn at the point that the n-gram appears, resulting in a barcode-like appearance that indicates the locations of the specified n-gram.

The goal of **highlightr** is not to summarize a single document, but to visualize the relationship

between a source document and its derivatives. This comparison of source and derivative documents is similar to the work of Ben Fry, who compared various editions of Charles Darwin's *On the Origin of Species* (Fry, 2018). In this work, Fry (2018) uses color to denote additions and animates deletions between six editions of Darwin's work, and notes how the language and structure change over time.

Similarly, Jänicke et al. (2014) utilize visualization tools in JavaScript to compare versions of the Bible for text reuse at a sentence level. Pairwise document comparisons are accomplished through a grid format, where a document is presented along each axis in sequential sections and similarities are shown in a heatmap form. They also provide an alternative visualization at the sentence-level in Dot Plot form, where a dot's location on the graph indicates a sentence pair. Alternatively, when viewing two texts side by side, similarities are connected in a parallel coordinate plot format, where a line is drawn between similar sentences. Jänicke et al. (2014) also perform sentence-level comparisons between multiple texts using sentence alignment flow, where a base sentence is depicted with variations connected via lines.

Comparison between multiple documents also bears some resemblance to plagiarism detection, where portions of texts are compared with references to determine if the text was copied. Parker and Hamblen (Parker and Hamblen, 1989) describe an algorithm that uses the character differences to compare similarities in computer programs, as well as other code-specific features.

Piao and McEnerly (2003) describes the TESAS algorithm, used for detecting relationships between two documents at the sentence level, with a focus on applications in journalism. This algorithm relied on shared strings, stems and synonyms for evaluating similarity between sentences in order to produce a score. The accompanying tool provided side-by-side comparisons of source and derivative sentences. Similarly, Olsen et al. (2011) describe the application of sequence alignment algorithms, traditionally used in bioinformatics, to compare passages of text. To visualize the correspondences between texts, Olsen et al. (2011) place texts side-by-side with color-coded similarities.

Visualizations for source and derivative text comparisons usually consist of texts placed next to each other, with color to indicate similarities and differences (Olsen et al., 2011; Fry, 2018; Piao and McEnerly, 2003). This type of data display uses the method of juxtaposition, which requires individuals to remember the content of the first document when comparing to the second document (Gleicher et al., 2011). These displays are limited in the number of texts that can be compared, as well as the length of texts, given the mental resources necessary for more complex comparisons. In contrast, the **highlightr** method focuses on a single "source" text, where similarities to derivative documents are directly mapped. This method relies on explicit encoding (Gleicher et al., 2011), since the relationship of interest is directly mapped in the visualization. While this reduces the amount of information displayed, in that all derivative documents are not presented alongside the source document, the method provides a simpler display of similarity.

This display method borrows from the methods seen in single-document text analysis. Similar to wordclouds and barplots, the output of interest is frequency. Instead of considering the frequency of words, the method considers the frequency of n-grams and the location of these n-grams throughout the document, much like dispersion plots (Shibuya and Jensen, 2015). However, this visualization differs in that its function is to visualize an entire document, and the frequency of this source document's n-grams in derivative documents, in order to provide a cohesive picture of the most borrowed phrases.

The collocation approach used in **highlightr** is similar to the shingling approach, where overlapping sequences of words (or 'shingles') are compared across documents (Olsen et al., 2011; Smith et al., 2013). Unlike longest common substring approaches (Amir et al., 2018; Crochemore et al., 2017), where the longest common string of characters between two texts is identified, the shingle (and collocation) approach focuses on finding matches in a source text of phrases of a set length. This allows for source text visualizations to be created based on the frequency for all phrases (or collocations) in a document, providing a view of the document as a whole.

While the word "collocation" traditionally refers to words that occur more frequently together - such as "white house" (Merriam-Webster, 2023), tools for collocation analysis helpfully provide the frequency of occurrence for strings of a specified length. This method allows us to calculate the frequency of word sequences (or n-grams) in derivative documents, and map these word frequencies back to the corresponding word sequence in the source document. When the word sequences of the derivative documents do not directly correspond to the source document, fuzzy matching can be used to find the closest matching source document sequence. This approximate string matching allows for the inclusion of inexact strings (Green, 2025).

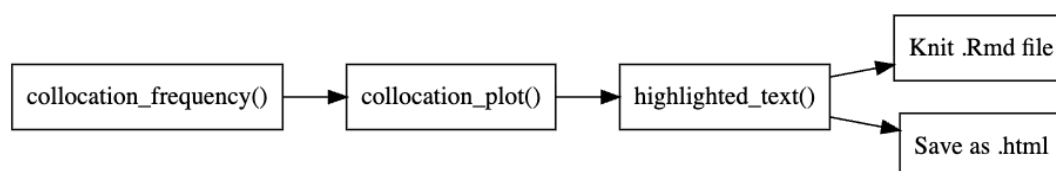


Figure 1: A flowchart of highlightr functions used to generate highlighted html output. Arrows indicate inputs.

3 Highlightr

3.1 Collocations

Collocation analysis calculates the frequency of sequentially-appearing word strings (n-grams) to identify words that tend to ‘stick together’. This process has been used to assist in identifying common multi-word phrases with a set meaning, such as “ice cream”. In this analysis, the collocation function of [quanteda](#) ([Benoit et al., 2018](#)) is used to consider longer strings ([Schweinberger, 2023](#)). Each set of words, or collocation, is found for the source document, then located in the derivative documents to calculate a frequency for each phrase.

Once sequences of words from derivative documents have been mapped to the source document, the average frequency is taken per word. When a collocation appears multiple times in the same source document, the derivative document frequency is divided by the number of appearances in the source document. To create a continuous highlighting effect (without breaks between words), each word is weighted by the collocations containing that word. Initially, each word is assigned a single color using **ggplot2** color scale mappings from frequency to hue. Then, HTML gradient boxes are created for each word, where the left color corresponds to the frequency of the previous word, while the right color corresponds to the frequency of the current word, producing a smooth visual transition between each word.

The process for creating highlighted text is depicted in Figure 1. The `collocation_frequency()` function performs the following operations: tokenizes the derivative and source documents; summarizes collocation frequencies in reference to the tokenized source document; maps collocation frequencies to the fully formatted source document and averages the collocation count for each word. Collocation frequencies are calculated by either using only direct matches or including indirect matches with `fuzzy=TRUE`. This function provides the averaged frequency values for each word, which lends itself to further analysis and visualization methods. For example, frequency values can be graphed in the order that they appear in the text to provide an overview of the document, as seen in Figures 4 and 5, which compare the effect of tuning parameter values. When used interactively, this visualization method can make it easy to detect phrases with extreme frequency. However, the entirety of the text is not easily included in the visualization. For this purpose, the frequencies must be transformed into a gradient and mapped onto the source document.

The `collocation_frequency()` object is then used by `collocation_plot()` to create a plot object with corresponding highlight colors. Other objects that include a column for the desired text, word order, and value of interest can also be inputted into `collocation_plot()` to generate color values for each word. The final step is to input this object into `highlighted_text()` to appropriately format the html tags to create the highlighting effect. This output can then be viewed through .Rmd by including the object outside of a code chunk, or through outputting the object as an .html file.

The visualization generated from this output follows common visualization design principles, such as strategic use of color, simplicity in design, and careful consideration of *what* the visualization is meant to convey ([Midway, 2020](#)). In contrast to using a table to represent frequencies, this visualization allows for the comparison of values ([Gelman et al., 2002](#)) within a source document, which provides the audience with a whole-of-document perspective of the data. This focus on the source document’s entirety with reference to several derivative documents simultaneously directly informs the visualization method implemented. Additionally, the use of semantically-resonant colors has been shown to increase performance in graphical evaluations ([Lin et al., 2013](#)). Therefore, the color yellow is used to represent higher frequency counts, as it is associated with traditional highlighters.

3.2 Fuzzy matching

The [zoomerjoin](#) package ([Green, 2025](#)) facilitates fuzzy matching on the indirect collocations - participant edits such as spelling errors, abbreviated words, or otherwise minor changes within a sentence. Leveraging fuzzy matching allows these minor changes to be matched back to their source using

Table 1: Similarities and differences between ‘rabbit’ and ‘rabid’ using n-grams of length 3

Relationship	rab	abb	bbi	bit	abi	bid
Intersection	1	0	0	0	0	0
Union	1	1	1	1	1	1

Jaccard similarity as approximated through the MinHash algorithm (Green, 2025). Two parameters, `n_bands` and `band_width`, are used in **zoomerjoin** to control the number of bands used and the width of the bands in the MinHash algorithm when computing fuzzy matches. These parameters can be passed to **zoomerjoin** through specifications in `collocation_frequency()` in order to adjust the comparison coverage of the algorithm.

Jaccard similarity is the intersection of two sets divided by the union of the sets (Wu et al., 2022). In this application, each collocation is considered a set. The units of interest are specified as n-grams in **zoomerjoin**. Green (2025) recommend to set a size of 2 or 3 for names, and a size of 5 or 6 for sentences. Because collocation length is usually longer than a name but shorter than a sentence, the default for fuzzy matching in **highlightr** has been set at 4 characters. This means that a subset of 4 characters in the two strings being compared will be counted in the intersection if they are identical.

For example, take the two following strings: “rabbit” and “rabid”. With a three-character n-gram width, the similarity between the two words can be visualized as Table 1. Here, there is a single common 3-gram (‘rab’), and a total of six 3-grams between the two words. This would result in a Jaccard distance of $\frac{1}{6} = 0.167$, indicating that the similarity between the two words is low.

Because the intent of the derivative document is clearer with simple typos (such as a single letter change) than with larger differences (such as changing multiple words), the count for these fuzzily matched observations are weighted by the Jaccard similarity value. A Jaccard similarity value of 1 would indicate identical strings, while a Jaccard similarity value of 0 would indicate no similarity between strings. The default threshold to be considered a match in **highlightr** is 0.7, corresponding to the default threshold used in **zoomerjoin** (Green, 2025).

Another factor in the calculation is the number of closest matches per ‘misspelled’ collocation. Some derivative collocations may have the same Jaccard similarity value for multiple source document collocations. In these cases, the frequency of the ‘misspelled’ derivative collocations are split evenly between the closest source document collocations, resulting in a total fuzzy frequency per source document collocation of:

$$f_i(x) = \sum_{i=1}^n \frac{x_i * d_i}{n_i} \quad (1)$$

where x_i is the count for derivative document collocation i , d_i is the Jaccard similarity value between the source document collocation and the ‘misspelled’ derivative document collocation i , and n_i is the number of closest source document collocation matches for derivative document collocation i . This quantity is then added to the total of direct collocation matches, before dividing by the number of times the collocation appears in the source document.

The Jaccard method is implemented due to simplicity and interpretability. This method is meant to capture misspellings, and provides a built-in method of weighting indirect matches in a way that is easy to explain and add to direct match frequency values. Other methods for detecting semantic similarity, such as the **word2vec** package (Wijffels and Watanabe, 2025) or Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2019), may be considered in conjunction with this method. However, the interpretability of the output would be less clear-cut than the 0-1 similarity value output of the Jaccard method.

3.3 Optimizing collocation length

The default collocation length in **highlightr** is 5. This length was chosen as a default to balance two considerations: the frequency with which a phrase may appear in the source document, and the number of sequential words that may appear in derivative documents. If the phrase is too short, a phrase may appear multiple times throughout the source document, which would complicate mapping the phrase back to a specific portion of source document. For example, setting the collocation length to 2 would pick up phrases such as “in the”, which may occur multiple times in a single document. On the other hand, it is unlikely that a large amount of derivative documents would have an exact 10-word phrase, especially if the documents represent notes or abbreviations. In a newspaper article matching scenario, Smith et al. (2013) utilize a collocation length between 5 and 7 for their shingling approach, and discuss the tradeoff between allowing flexibility in textual variation through a shorter

length and detection of fixed phrases.

The appropriate collocation length may differ based on the length of the source document, the number of derivative documents, and the type of derivation; notes, subsequent versions, and other types of derivatives may have different expected collocation lengths.

We use the following equation in order to measure the smoothness of transitions between words:

$$S(f) = \frac{\sum_{i=1}^{m-1} (w_i - w_{i+1})^2}{(m-1) * k} \quad (2)$$

Here, w_i is the average collocation frequency for word i of the source document collocation (where frequency is computed either using the fuzzy matching shown in Equation (1), or using non-fuzzy matches), m is the number of words in the source document, and k is the number of derivative documents. By computing the sum of squared distance between the frequencies of consecutive words, we can evaluate the smoothness of transitions between words in the document on the whole. Dividing by $m-1$ and k provides a form of standardization, as we would expect a larger sum of squares when there are more words and when the frequency values are higher (e.g. there are more derivative documents).

If the collocation length is too long, the smoothness parameter will approach 0, meaning that the entire document will be the same color, which limits the utility of visualizing collocations. However, when collocation length is short, there are large differences in the collocation counts of sequential words, indicating that whole phrases are not being recognized.

Equation (2) produces an L-shaped curve resembling exponential decay, where the shortest collocation graphed (length 2) has a high value that quickly decreases. The goal is to find a collocation of minimum length that results in ‘smooth transitions’; functionally, as with many parameter optimization problems, this means identifying an ‘elbow’ in the curve where both collocation length and variation between collocation counts are acceptably low. This ‘elbow’ or ‘knee’ approach is often used in k-means clustering (Schubert, 2023) and operations research (Thomas and Sheldon, 1999). It can be defined as the location where the slope of the curve changes from large to small (Thomas and Sheldon, 1999). While the best method for identifying the elbow of a curve is contested (Schubert, 2023), visualization provides a “useful clue” (Thomas and Sheldon, 1999) for collocation length values that can be more thoroughly explored.

4 Practical applications of highlightr

4.1 Abstract comparison

In the first example, we compare the abstract from the author’s dissertation (Rogers, 2024) (source document) to conference presentations given on portions of the same material (derivative documents). Since the presentation abstracts cover portions of the same content as the dissertation, we would expect to see some overlap between the subjects and wording. This comparison provides a concise example of how **highlightr** functions, since abstracts are generally limited to less than 200 words. In this case, since we do not expect an exact match between the wording in different summaries, we use fuzzy matching to include indirect matches.

The `collocation_frequency()` function compares the source and derivative documents. It is assumed that both the source and derivative texts are located in a single data frame. The row corresponding to the source document, as well as the column containing the text to be compared, must be specified in the function. The function first tokenizes the source and derivative documents, a process of splitting the text in the document into words, allowing for a comparison across documents. Next, the tokenized source and derivative documents are compared using collocations.

```
# Read in data
abstract <- readr::read_csv("data/abstracts.csv")

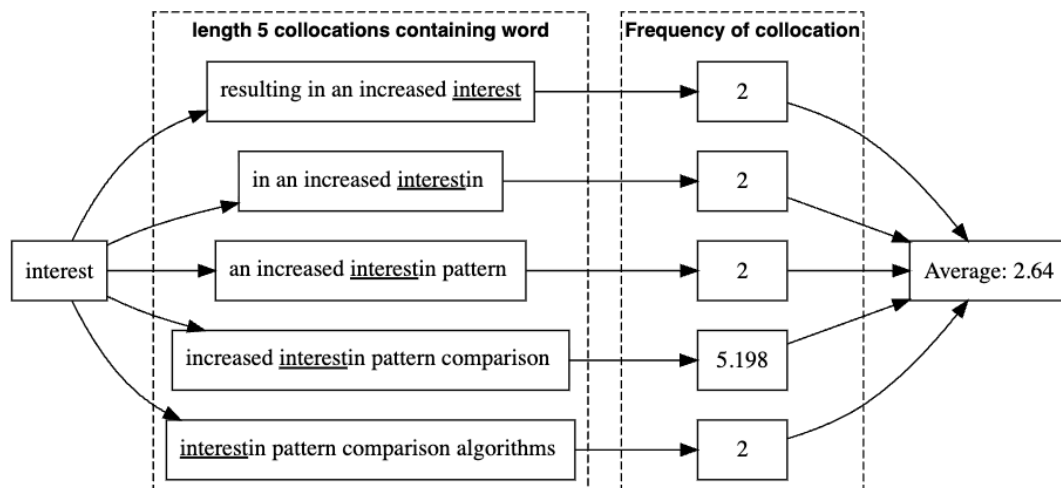
# Calculate collocation frequency using fuzzy matching and map to source document

collocation_abstract <-
  collocation_frequency(abstract,
    source_row=which(abstract$Conference=="Dissertation"),
    text_column = "page_notes", fuzzy=TRUE)
```

Table 2 shows the fuzzy collocation results for a collocation of length 5 (the default collocation length). These results are organized by the first word of the collocation. The “Word Location in

Table 2: Dissertation Collocations of Length 5

Word	Word Location in Collocation					First Collocation	Frequency
	1st	2nd	3rd	4th	5th		
Subjective	2	NA	NA	NA	NA	subjective pattern comparison has been	2
pattern	2	2	NA	NA	NA	pattern comparison has been subject	2
comparison	2	2	2	NA	NA	comparison has been subject to	2
has	2	2	2	2	NA	has been subject to increased	2
been	2	2	2	2	2	been subject to increased scrutiny	2
subject	2	2	2	2	2	subject to increased scrutiny by	2

**Figure 2:** A flowchart of the collocation process for the word ‘interest’ with collocation length 5.

Collocation” identifies the word’s place in the collocation frequency calculated. The “1st” column provides the collocation frequency for the collocation that begins with the word in the “Word” column (shown as the “First Collocation”). Note that the first word of the source document (“Subjective”) only appears in a single collocation, resulting in empty values for the remaining “Word Location in Collocation” columns. Similarly, the second word of the source document (“pattern”) appears in two collocations, resulting in two collocation values. The first frequency value corresponds to the collocation where “pattern” is the first word: “pattern comparison has been subject”, while the second frequency value corresponds to the collocation where “pattern” is the second word: “Subjective pattern comparison has been”. This pattern continues for the rest of words in the source document. The “Frequency” column is the average of the collocations that a word appears in - this is the value used when assigning color to the text.

This collocation process is outlined in the flowchart in Figure 2, corresponding to the word ‘interest’ in the first sentence of the source document abstract: “Subjective pattern comparison has been subject to increased scrutiny by the courts and by the general public, resulting in an increased **interest** in pattern comparison algorithms that provide quantitative assessments of similarity for use by forensic scientists.” Because the collocation is of length 5 and the word is not at the beginning of the document, there are five collocations that contain this instance of the word ‘interest’. While most of these collocations have a frequency of 2, meaning that they appeared in approximately two derivative documents (while accounting for indirect matches), we can see that the phrase “increased interest in pattern comparison” has a higher frequency count, when fuzzy matches are included, for a value of approximately 5.198. This higher frequency count would affect all words contained in the collocation, creating a more yellow highlight around the phrase. The values for collocations containing the word ‘interest’ are averaged together to provide the number used in the highlighting effect: 2.64.

This data is then assigned a gradient corresponding to the frequency through **ggplot2** using `collocation_plot()`, and the gradient is converted into html tags using `highlighted_text()`:

```
freq_plot <- collocation_plot(collocation_abstract)
page_highlight <- highlighted_text(freq_plot)
```

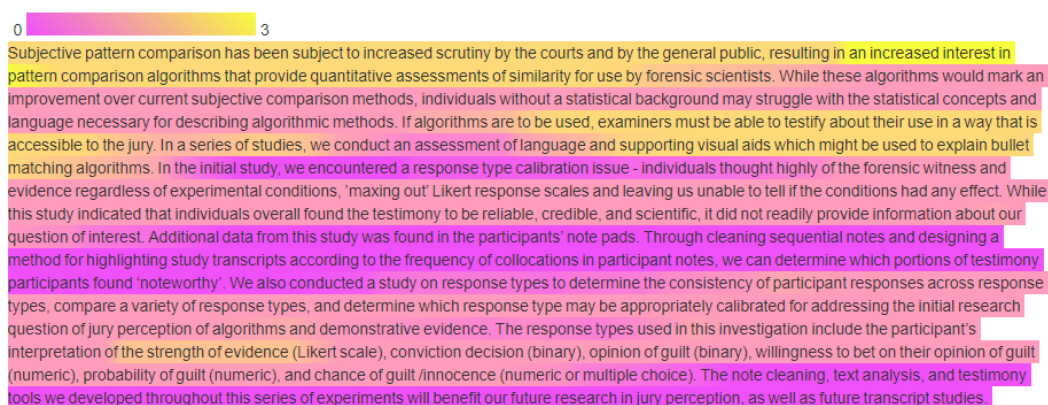


Figure 3: Source Document with Collocation Length 5 using Fuzzy Matching and Default n-gram Width of 4

This results in the output shown in Figure 3. Here, the highest weighted frequency is around 3, with particular emphasis on the phrase “an increased interest in”. Two tuning adjustments that can be made in the case of fuzzy matching are a change in n-gram width (specified through `n_gram_width`) and a change in the threshold of what is considered a match (specified through `threshold`). The default parameter for n-gram width is 4, meaning that the fuzzy matching process considers sequences of 4 characters when computing Jaccard similarity. The default parameter for threshold is 0.7, meaning that a Jaccard value of 0.7 or higher is necessary to be considered a ‘match’.

A comparison of frequency values for thresholds ranging from 0.5 to 0.95 and n-gram widths ranging from 2 to 10 are shown in Figures 4 and 5, respectively. Frequency values for the default settings are shown in red. In both cases, frequency values are similar across tuning parameters, with most settings following similar trends. There is some divergence in both cases around the phrase “algorithms and demonstrative evidence” (as shown in Figures 4 and 5), where lower set values demonstrate slightly higher frequencies. There is also a high peak around the phrase “bullet matching algorithms” for smaller set values, indicating that relaxing ‘match’ qualifications results in higher frequency values for this phrase. While the overall trend is similar, different values chosen for these tuning parameters can result in different emphasis for specific phrases. These options of n-gram width and threshold are only available in the case of fuzzy matching, as it determines the computation of the Jaccard similarity.

Alternatively, non-fuzzy matching can also be used by excluding the `fuzzy=TRUE` argument from `collocation_frequency()`, as follows:

```
collocation_abstract <-
  collocation_frequency(abstract,
                        source_row=which(abstract$Conference=="Dissertation"),
                        text_column = "page_notes")
```

This provides a document that only includes direct matches between collocations, shown in Figure 6. When indirect matches are excluded, the maximum value on the scale decreases from four to two, and there is an equal highlighting throughout most of the initial sentence. While the removal of fuzzy matches results in the exclusion of close phrases, the nonfuzzy document has a clearer interpretation. The count directly represents the average number of times the collocations associated with a word appeared in derivative documents, whereas fuzzy match counts are the average of a weighted collocation frequency.

We compare the collocation smoothness curve to the text itself in order to find a collocation length that is acceptably smooth. Figure 7 shows the relationship between collocation length and smoothness, where 5 is marked as the default collocation length. As this graph demonstrates, smoothness values quickly decrease as the collocation length increases, with values approaching 0. In this case, a collocation of length 5 does not appear to be ill-suited to the graph. One limitation of this example is the small sample size - we only have 7 derivative documents. This results in the small smoothness values on the y-axis, as shown in Figure 7.

Figure 8 shows the dissertation abstract when the collocation length is 3. In this case, it is difficult to pick out main passages from the document - “increased interest”, “matching algorithms”, and “demonstrative evidence” appear to be commonly repeated, but this provides a limited view of the document as a whole. While the maximum count here is 4, this information seems like it could have been better represented as a table of collocations and frequencies.

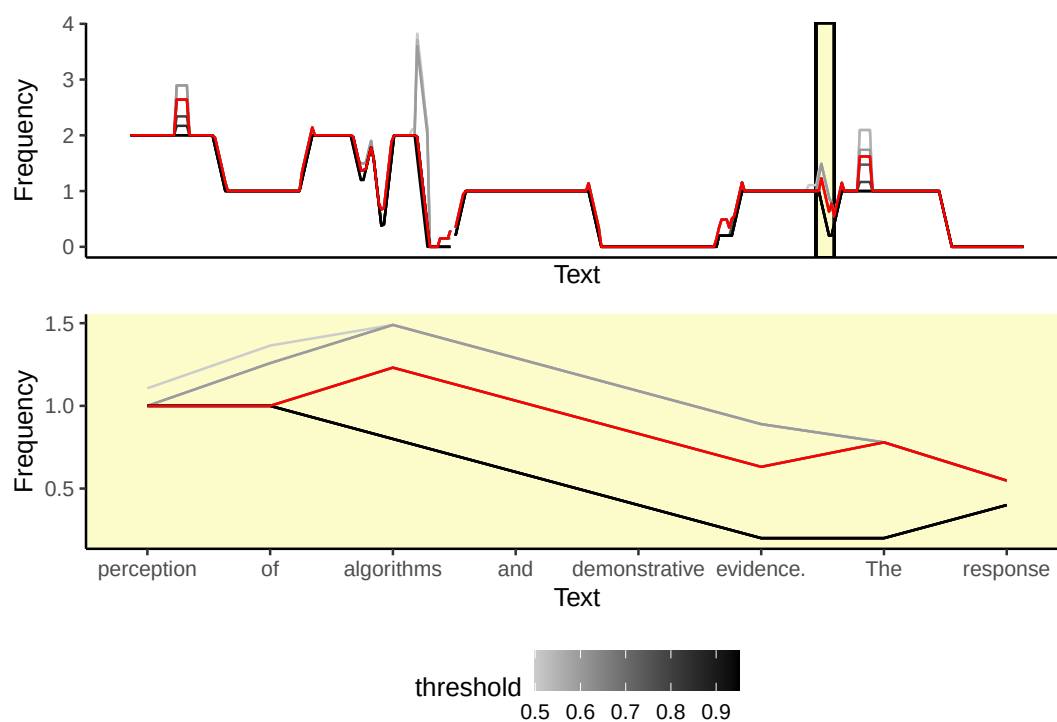


Figure 4: Graph of Frequency Values by Word for Thresholds from 0.5 to 0.95 with Collocation Length 5 using Fuzzy Matching.

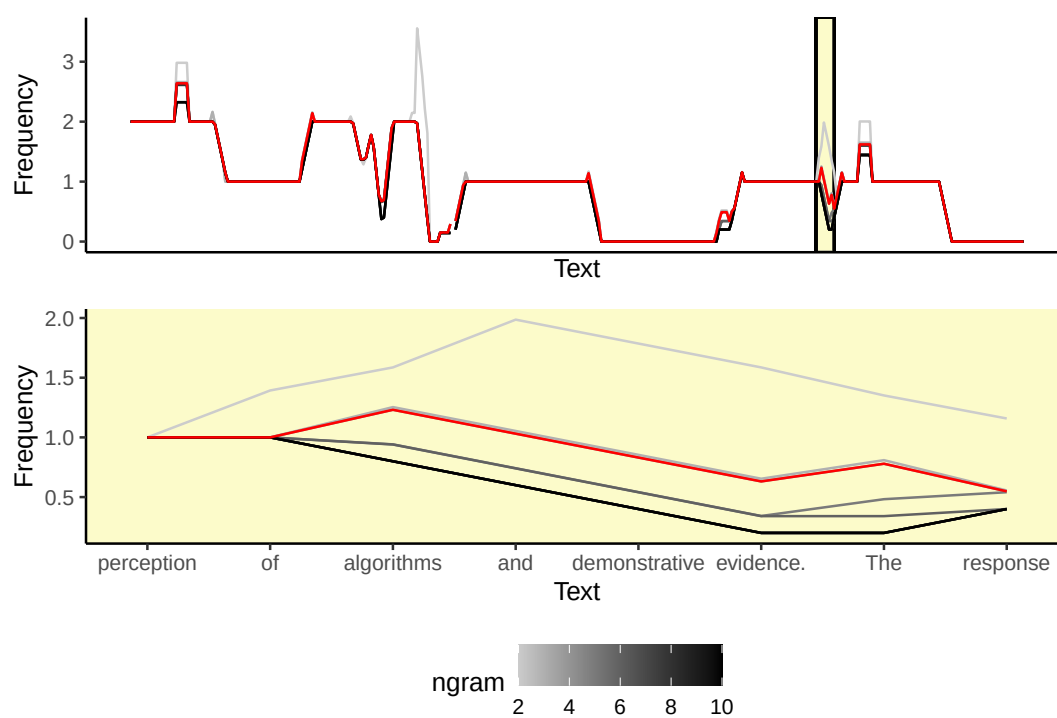


Figure 5: Graph of Frequency Values by Word for N-Gram Widths from 2 to 10 with Collocation Length 5 using Fuzzy Matching.

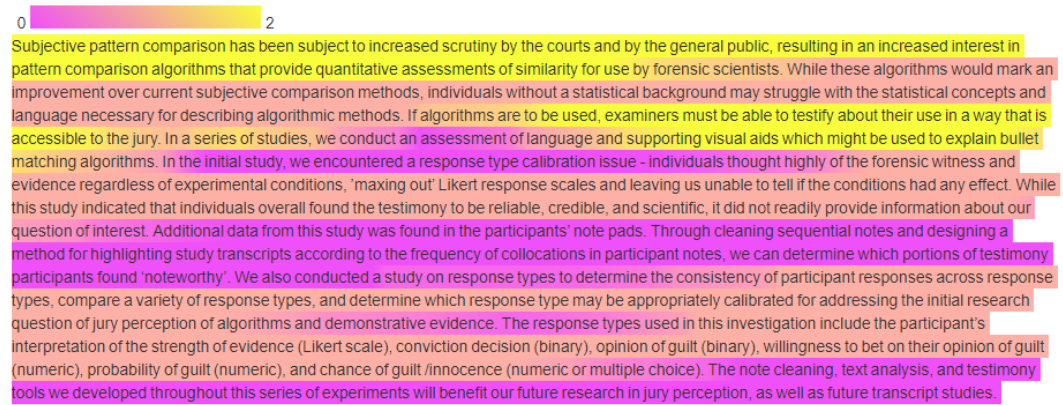


Figure 6: Source Document with Collocation Length 5 without Fuzzy Matching

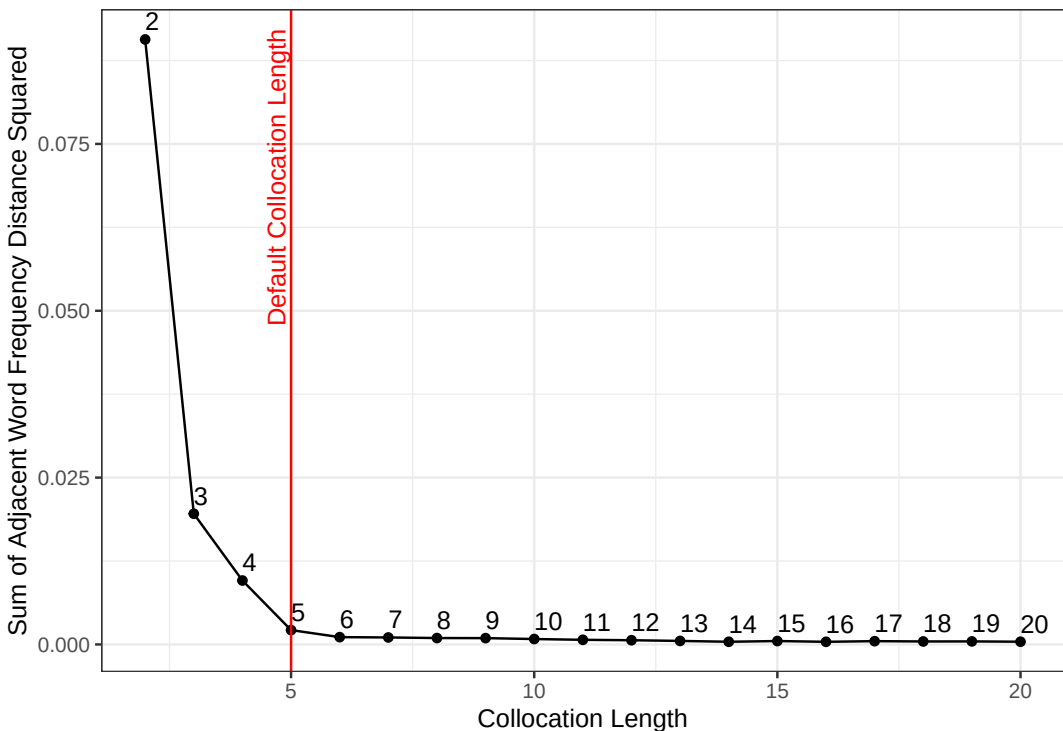


Figure 7: Smoothness of Summary Highlighting in Abstract Example

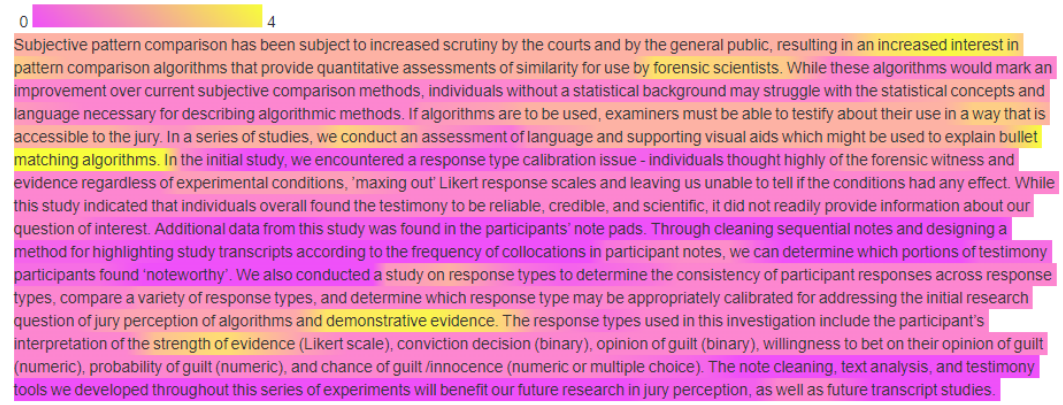


Figure 8: Source Document with Collocation Length 3 using Fuzzy Matching

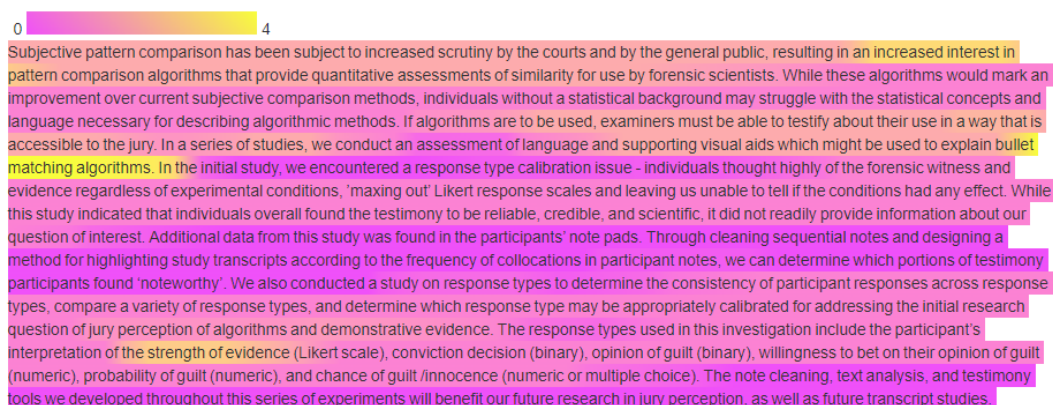


Figure 9: Source Document with Collocation Length 4 using Fuzzy Matching

Increasing the collocation length to 4 results in Figure 9. This graphic resembles Figure 8, but features less emphasis on short phrases such as “demonstrative evidence” and “forensic scientists”. The emphasis on “bullet matching algorithms”, however, is still strong.

By further increasing the collocation length to 5, shown in Figure 3, there are smoother transitions and more focus on phrases than was seen in Figure 9. Smoother transitions are particularly evident in the first sentence of the document, which is similar across several documents. We can begin to see overall themes of the work which were highlighted in different conference presentations - pattern comparison, jury accessibility, and strength of evidence. Phrases that were not replicated in derivative documents are also clearer: ‘response type calibration’, ‘cleaning sequential notes’, and ‘future research in jury perception’.

Based on these images, a collocation of length 5 balances the smoothness needed for fluid interpretation of this data and the short length necessary to capture as many observations as possible. This length is also located in the elbow of the distribution. While a collocation of length 5 is shown as a good fit for this application of the function, other applications may benefit from the selection of a different collocation length. The optimization method described above can be applied to explore collocation values in other applications, and tuning plot comparisons such as Figures 4 and 5 can be created to visualize differences in parameter values.

4.2 Wikipedia edits: Tim Walz

highlightr can also be applied to analyze Wikipedia edit history. Often, political candidates’ pages will be cleaned up as they announce their entry into a race; this edit frequency can sometimes be informative when US presidential candidates announce their selection of a vice president. Here, we demonstrate the use of **highlightr** to examine former Vice Presidential Candidate Tim Walz’s Wikipedia page (Tim, 2024), comparing Wikipedia edits from approximately August 15, 2024 to August 29, 2024 to edits from before he was nominated as a vice presidential candidate. We scraped the 150 most recent versions of Tim Walz’s Wikipedia article on August 29, 2024 (resulting in articles dating back to August 18, 2024), as well as 150 articles from before he was nominated (with a start date of July 31, 2024, and an end date of April 22, 2023). Due to the large amount of edits around Walz’s nomination, these edits were not included in this analysis.

In this analysis, we aim to determine how his article was “cleaned up” for the presidential election, and whether information was added or removed to improve his public profile. Figure 10 shows the frequency of edits to Walz’s article by date. There are many more edits in the later part of 2024 than in earlier time periods. This is especially noticeable after Walz was nominated.

The most recent article and the oldest article in record are both used as source documents for this comparison. The highlighted versions of these documents are shown in Figure 11. As this figure demonstrates, the length of Walz’s Wikipedia article has doubled. Through the use of **highlightr**, we can see that this increase in length is due not only to recent news in the 18-month period between the newest and the oldest article, but also to the inclusion of older events as well as elaboration on previously mentioned topics.

In the most recent article, we can see that Walz’s endorsement of President Obama’s candidacy in 2008, as well as President Trump appointing Walz to “the bipartisan Council of Governors”, are relatively recent additions. These additions are shown in Figure 12. Neither former president was mentioned in the oldest article scraped. Another section has also been added regarding Walz’s actions

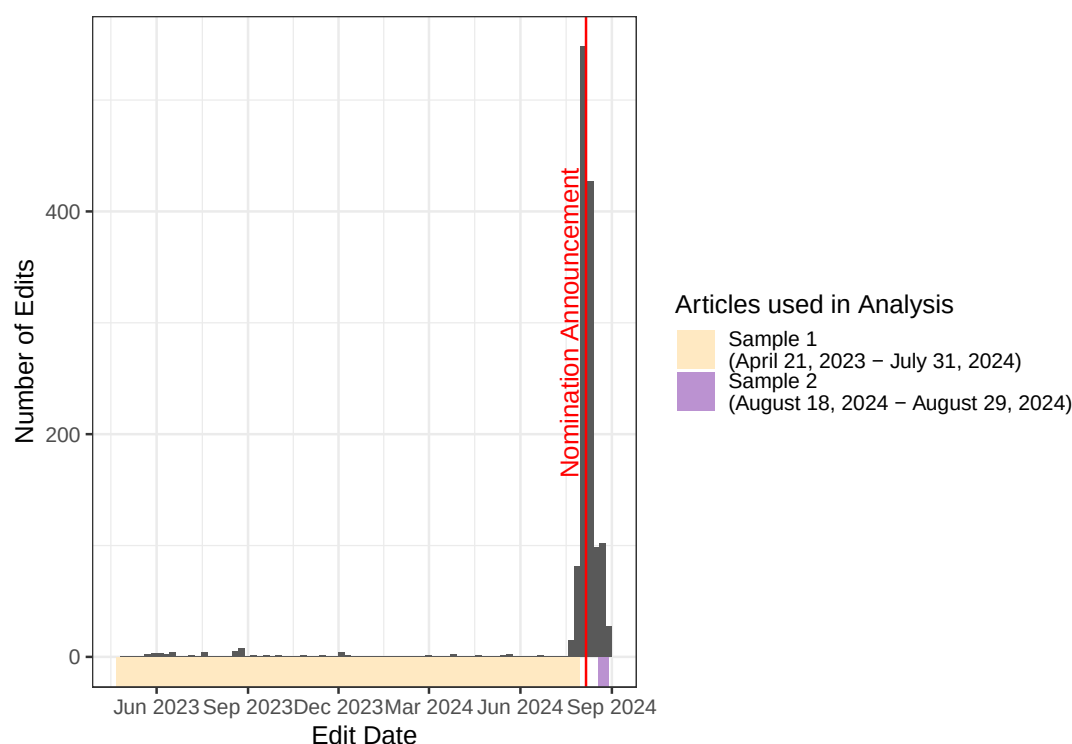


Figure 10: Number of Edits between April 22, 2023 and August 29, 2024. Sample 1, corresponding to articles before nomination, ranges from April 21, 2023 to July 31, 2024. Sample 2, corresponding to articles after nomination, ranges from August 18, 2024 to August 29, 2024. The red line indicates the date Walz’s nomination was announced (August 6, 2024).

related to tribal nations. These items were not referenced in the oldest article scraped (from April 22, 2023), despite most of these actions predating the article. This paragraph is shown in Figure 13.

The most notable phrase that is not commonly repeated from the oldest article in the database is that Walz “has been endorsed by the NRA multiple times.” In comparing with the August 29, 2024 article, we can see that more detail has been added, stating that Walz later denounced the NRA and believes in gun regulation. This comparison can be seen in Figure 14.

5 Conclusion

The package **highlightr** has diverse applications that can help researchers synthesize text data to evaluate changes, assess similarities, and highlight common information across documents. This package works by evaluating the frequency with which collocations from a source document appear in derivative documents, mapping this frequency to a color scale, and creating a gradient output of these frequencies to form a highlighting effect in HTML outputs. These applications have been demonstrated on the author’s submitted abstracts, as well as the Wikipedia article for Tim Walz.

While originally developed to compare participant notes to an online transcript, this package can generally be used to compare texts. In an extension of the abstract comparison, this method can be used to compare revised articles to earlier iterations, demonstrating which portions are consistent across edits as well as which portions have been updated. This visualization method has applications in visualizing borrowed phrases in literature and newspapers, as described by [Smith et al. \(2013\)](#) and [Olsen et al. \(2011\)](#). Similarly, this method can be applied to differing versions of the same text, such as [Fry \(2018\)](#)’s analysis of *On the Origin of Species*, to visualize the passages from the first version that were retained, or [Jänicke et al. \(2014\)](#)’s analysis of Bible passages. This form of literature analysis can also extend to versions of fairy tales in order to visualize changes over time. With some modifications to the string division method, this visualization method would also have applications in bioinformatics. This method provides a new way to visualize an entire document with regards to several derivative documents by mapping average phrase frequency directly onto the source document through a color gradient. Popular and unpopular phrases are instantly identifiable through color indicators, while the overall document view provides context.



Figure 11: The highlighted versions of the most recent (left) and oldest (right) articles about Tim Walz used in this analysis.

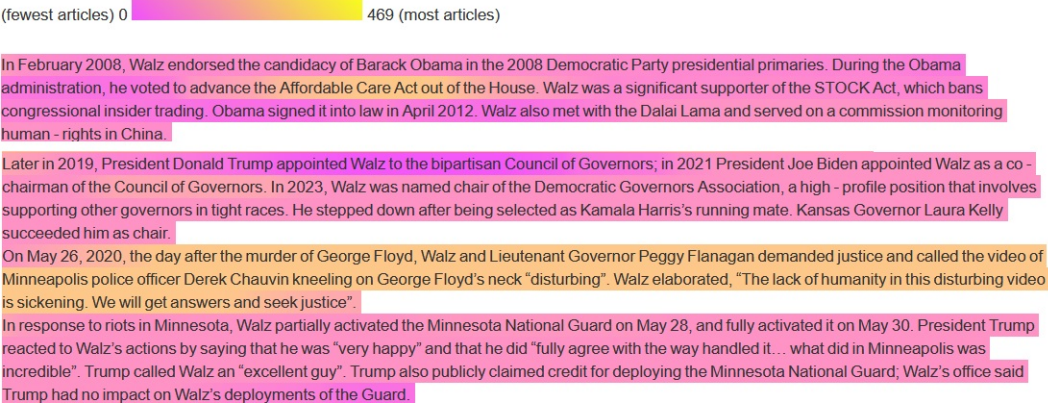


Figure 12: Newer additions to Walz’s Wikipedia page relating to former presidents. The purple highlighting indicates less frequent mentions

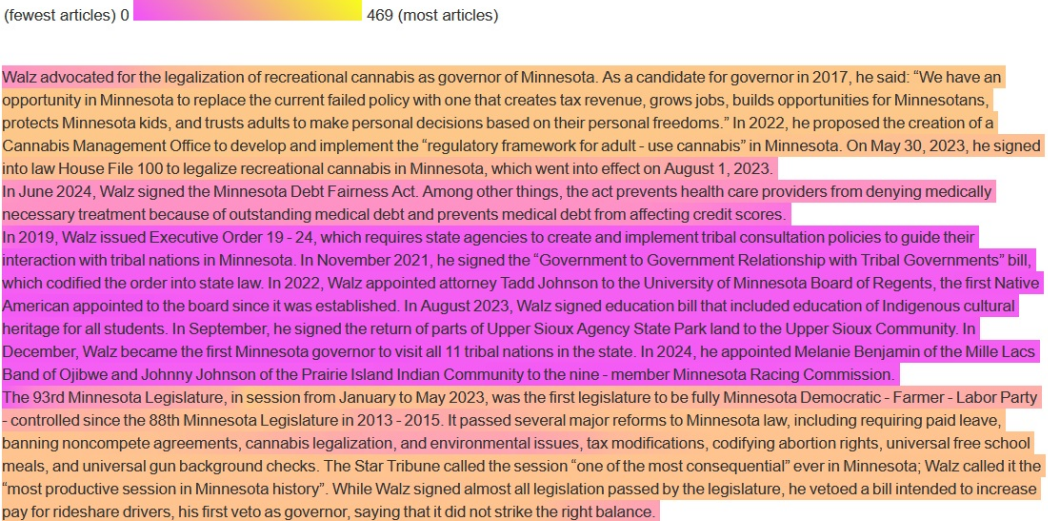


Figure 13: Highlighted comparison with the August 19, 2024 Wikipedia page as the source document. The purple indicates that this paragraph appeared in fewer previous articles than the surrounding paragraphs.

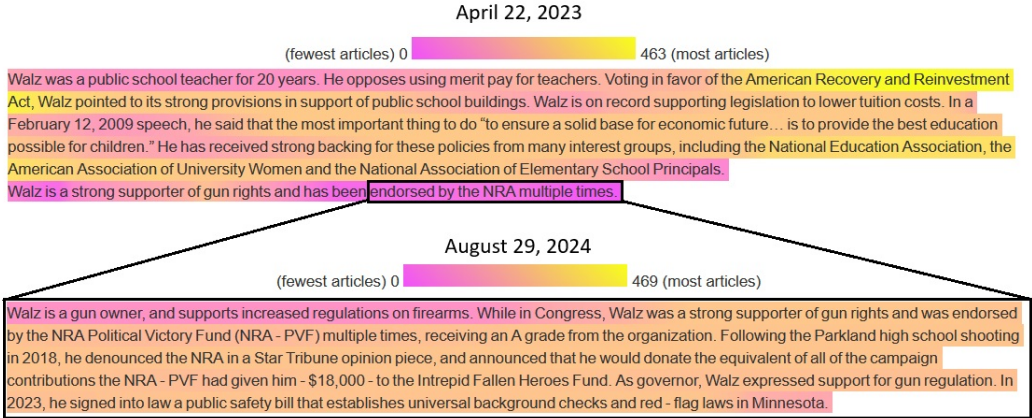


Figure 14: A Comparison between how Walz’s NRA support was described on Wikipedia in April 2023 versus August 2024. Purple indicates a lower frequency, while yellow indicates a higher frequency.

6 Acknowledgements

This work was funded (or partially funded) by the Center for Statistics and Applications in Forensic Evidence (CSAFE) through Cooperative Agreements 70NANB15H176 and 70NANB20H019 between NIST and Iowa State University, which includes activities carried out at Carnegie Mellon University, Duke University, University of California Irvine, University of Virginia, West Virginia University, University of Pennsylvania, Swarthmore College and University of Nebraska, Lincoln.

This paper and corresponding examples used R (R Core Team, 2025) and the following packages: **DiagrammeR** (Iannone and Roy, 2024), **gt** (Iannone et al., 2026), **highlightr** (Center for Statistics and Applications in Forensic Evidence et al., 2024), **imager** (Barthelme, 2025), **knitr** (Xie, 2025), **pander** (Daróczy and Tsegelskyi, 2025), **patchwork** (Pedersen, 2025), **plotly** (Sievert, 2020), **purrr** (Wickham and Henry, 2025), **rjtools** (O'Hara-Wild et al., 2025), **rmarkdown** (Allaire et al., 2025), **rvest** (Wickham, 2025a), **stringr** (Wickham, 2025b), **tidyverse** (Wickham et al., 2019), and **webshot2** (Chang, 2025).

References

- Tim Walz, Aug. 2024. URL https://en.wikipedia.org/wiki/Tim_Walz. [p64]
- J. Allaire, Y. Xie, C. Dervieux, J. McPherson, J. Luraschi, K. Ushey, A. Atkins, H. Wickham, J. Cheng, W. Chang, and R. Iannone. *rmarkdown: Dynamic Documents for R*, 2025. URL <https://github.com/rstudio/rmarkdown>. R package version 2.30. [p68]
- A. Amir, P. Charalampopoulos, S. P. Pissis, and J. Radoszewski. Longest Common Substring Made Fully Dynamic, July 2018. URL <http://arxiv.org/abs/1804.08731>. arXiv:1804.08731 [cs]. [p56]
- S. Barthelme. *imager: Image Processing Library Based on 'CImg'*, 2025. URL <https://CRAN.R-project.org/package=imager>. R package version 1.0.8. [p68]
- K. Benoit, K. Watanabe, H. Wang, P. Nulty, A. Obeng, S. Müller, and A. Matsuo. quanteda: An R package for the quantitative analysis of textual data. *Journal of Open Source Software*, 3(30):774, 2018. doi: 10.21105/joss.00774. URL <https://quanteda.io>. [p57]
- Center for Statistics and Applications in Forensic Evidence, R. Rogers, and S. VanderPlas. *highlightr: Highlight Conserved Edits Across Versions of a Document*, 2024. URL <https://CRAN.R-project.org/package=highlightr>. R package version 1.0.2. [p55, 68]
- W. Chang. *webshot2: Take Screenshots of Web Pages*, 2025. URL <https://CRAN.R-project.org/package=webshot2>. R package version 0.1.2. [p68]
- M. Crochemore, C. S. Iliopoulos, A. Langiu, and F. Mignosi. The longest common substring problem. *Mathematical Structures in Computer Science*, 27(2):277–295, Feb. 2017. ISSN 0960-1295, 1469-8072. doi: 10.1017/S0960129515000110. URL <http://www.cambridge.org/core/journals/mathematical-structures-in-computer-science/article/longest-common-substring-problem/1DA95504B11A551E2BD130AA834CF41>. Publisher: Cambridge University Press. [p56]
- G. Daróczy and R. Tsegelskyi. *pander: An R 'Pandoc' Writer*, 2025. URL <https://CRAN.R-project.org/package=pander>. R package version 0.6.6. [p68]
- J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL <https://aclanthology.org/N19-1423/>. [p58]
- I. Fellows. *wordcloud: Word Clouds*, 2018. URL <https://CRAN.R-project.org/package=wordcloud>. R package version 2.6. [p55]
- B. Fry. Tracing the Origin, May 2018. URL https://medium.com/@ben_fry/tracing-the-origin-65011dc20877. [p56, 65]
- A. Gelman, C. Pasarica, and R. Dodhia. Let's Practice What We Preach: Turning Tables into Graphs. *The American Statistician*, 56(2):121–130, 2002. ISSN 0003-1305. URL <https://www.jstor.org/stable/3087382>. Publisher: [American Statistical Association, Taylor & Francis, Ltd.]. [p57]

- M. Gleicher, D. Albers, R. Walker, I. Jusufi, C. D. Hansen, and J. C. Roberts. Visual comparison for information visualization. *Information Visualization*, 10(4):289–309, Oct. 2011. ISSN 1473-8716, 1473-8724. doi: 10.1177/1473871611416549. URL <https://journals.sagepub.com/doi/10.1177/1473871611416549>. [p56]
- B. Green. *zoomerjoin: Superlatively Fast Fuzzy Joins*, 2025. URL <https://CRAN.R-project.org/package=zoomerjoin>. R package version 0.2.1. [p56, 57, 58]
- R. Iannone and O. Roy. *DiagrammeR: Graph/Network Visualization*, 2024. URL <https://CRAN.R-project.org/package=DiagrammeR>. R package version 1.0.11. [p68]
- R. Iannone, J. Cheng, B. Schloerke, S. Haughton, E. Hughes, A. Lauer, R. François, J. Seo, K. Brevoort, and O. Roy. *gt: Easily Create Presentation-Ready Display Tables*, 2026. URL <https://CRAN.R-project.org/package=gt>. R package version 1.3.0. [p68]
- S. Jänicke, A. Geßner, M. Büchler, and G. Scheuermann. Visualizations for Text Re-use. In *2014 International Conference on Information Visualization Theory and Applications (IVAPP)*, pages 59–70, Jan. 2014. URL <https://ieeexplore.ieee.org/document/7294398/>. [p56, 65]
- S. Lin, J. Fortuna, C. Kulkarni, M. Stone, and J. Heer. Selecting Semantically-Resonant Colors for Data Visualization. *Computer Graphics Forum*, 32(3pt4):401–410, 2013. ISSN 1467-8659. doi: 10.1111/cgf.12127. URL <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12127>. _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12127>. [p57]
- Merriam-Webster. Definition of COLLOCATION, Apr. 2023. URL <https://www.merriam-webster.com/dictionary/collocation>. [p56]
- S. R. Midway. Principles of Effective Data Visualization. *Patterns*, 1(9):100141, Dec. 2020. ISSN 2666-3899. doi: 10.1016/j.patter.2020.100141. URL <https://www.sciencedirect.com/science/article/pii/S2666389920301896>. [p57]
- 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'*, 2025. URL <https://CRAN.R-project.org/package=rjtools>. R package version 1.0.18.1. [p68]
- M. Olsen, R. Horton, and G. Roe. Something Borrowed: Sequence Alignment and the Identification of Similar Passages in Large Text Collections. *Digital Studies / Le champ numérique*, 2(1), May 2011. ISSN 1918-3666. doi: 10.16995/dscn.258. URL <https://www.digitalstudies.org/article/id/7224/>. [p56, 65]
- A. Parker and J. Hamblen. Computer algorithms for plagiarism detection. *IEEE Transactions on Education*, 32(2):94–99, May 1989. ISSN 1557-9638. doi: 10.1109/13.28038. Conference Name: IEEE Transactions on Education. [p56]
- T. L. Pedersen. *patchwork: The Composer of Plots*, 2025. URL <https://CRAN.R-project.org/package=patchwork>. R package version 1.3.2. [p68]
- S. S. L. Piao and T. McEnery. A Tool for Text Comparison. *Proceedings of the Corpus Linguistics*, 2003. [p56]
- 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/>. [p68]
- R. Rogers. Measuring Jury Perception of Explainable Machine Learning and Demonstrative Evidence. *Dissertations and Doctoral Documents from University of Nebraska-Lincoln*, 2023–, Apr. 2024. URL <https://digitalcommons.unl.edu/dissunl/69>. [p59]
- E. Schubert. Stop using the elbow criterion for k-means and how to choose the number of clusters instead. *SIGKDD Expl. Newsl.*, 25(1):36–42, July 2023. ISSN 1931-0145. doi: 10.1145/3606274.3606278. URL <https://dl.acm.org/doi/10.1145/3606274.3606278>. [p59]
- M. Schweinberger. *Analyzing Co-Occurrences and Collocations in R*. The University of Queensland, Australia. School of Languages and Cultures, Brisbane, 2023.05.31 edition, 2023. <https://ladal.edu.au/coll.html>. [p57]
- Y. Shibuya and K. E. Jensen. Mining for constructions in texts using N-gram and network analysis. *Globe: A Journal of Language, Culture and Communication*, 2, Oct. 2015. ISSN 2246-8838. doi: 10.5278/ojs.globe.v2i0.1113. URL <https://journals.aau.dk/index.php/globe/article/view/1113>. [p55, 56]

- C. Sievert. *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman and Hall/CRC, 2020. ISBN 9781138331457. URL <https://plotly-r.com>. [p68]
- J. Silge. She Giggles, He Gallops. URL <https://pudding.cool/2017/08/screen-direction>. [p55]
- D. A. Smith, R. Cordell, and E. M. Dillon. Infectious texts: Modeling text reuse in nineteenth-century newspapers. In *2013 IEEE International Conference on Big Data*, pages 86–94, Oct. 2013. doi: 10.1109/BigData.2013.6691675. URL <https://ieeexplore.ieee.org/document/6691675/>. [p56, 58, 65]
- C. Thomas and B. Sheldon. The "Knee of a Curve"—Useful Clue but Incomplete Support. *Military Operations Research*, 4(2):17–24, 1999. ISSN 1082-5983. URL <https://www.jstor.org/stable/43940795>. Publisher: Military Operations Research Society. [p59]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p55]
- H. Wickham. *rvest: Easily Harvest (Scrape) Web Pages*, 2025a. URL <https://CRAN.R-project.org/package=rvest>. R package version 1.0.5. [p68]
- H. Wickham. *stringr: Simple, Consistent Wrappers for Common String Operations*, 2025b. URL <https://CRAN.R-project.org/package=stringr>. R package version 1.6.0. [p68]
- H. Wickham and L. Henry. *purrr: Functional Programming Tools*, 2025. URL <https://CRAN.R-project.org/package=purrr>. R package version 1.0.4. [p68]
- H. Wickham, M. Averick, J. Bryan, W. Chang, L. D. McGowan, R. François, G. Golemund, 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. [p68]
- J. Wijnffels and K. Watanabe. *word2vec: Distributed Representations of Words*, 2025. URL <https://CRAN.R-project.org/package=word2vec>. R package version 0.4.1. [p58]
- W. Wu, B. Li, L. Chen, J. Gao, and C. Zhang. A Review for Weighted MinHash Algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 34(6):2553–2573, June 2022. ISSN 1558-2191. doi: 10.1109/TKDE.2020.3021067. URL <https://ieeexplore.ieee.org/abstract/document/9184977>. [p58]
- Y. Xie. *knitr: A General-Purpose Package for Dynamic Report Generation in R*, 2025. URL <https://yihui.org/knitr/>. R package version 1.51. [p68]

Rachel Rogers
 University of Technology-Sydney
 School of Mathematical and Physical Sciences
 Sydney, New South Wales, Australia
 ORCID: 0000-0002-4145-9630
rachel.rogers@uts.edu.au

Susan VanderPlas
 University of Nebraska-Lincoln
 Department of Statistics
 Lincoln, Nebraska, United States of America
 ORCID: 0000-0002-3803-0972
susan.vanderplas@unl.edu

nimblewomble: An R package for Bayesian Wombling with nimble

by Aritra Halder and Sudipto Banerjee

Abstract This exposition presents **nimblewomble**, a software package to perform wombling, or boundary analysis, using the **nimble** Bayesian hierarchical modeling language in the **R** statistical computing environment. Wombling is used widely to track regions of rapid change within the spatial reference domain. Specific functions in the package implement Gaussian process models for point-referenced spatial data followed by predictive inference on rates of change over curves using line integrals. We demonstrate model-based Bayesian inference using posterior distributions featuring simple analytic forms while offering uncertainty quantification over curves.

1 Introduction

Detecting regions of rapid change is an important exercise in spatial data science as they harbor effects not easily explained by predictors incorporated into a spatial regression model for point-referenced spatial data. For example, environmental health scientists are often keen on identifying regions where exposure levels display rapid change or sharp gradients. Formal statistical detection of such regions can lead to data-driven discoveries of latent risk factors and other predictors that drive the rapid change in exposure surfaces. This exercise is referred to as wombling (Womble, 1951; Gleyze et al., 2001; Banerjee, 2010). Wombling results in curves that track regions of interest. Identifying such regions serves as crucial guides for interventions. For example, to determine whether natural features such as rivers or mountain ranges represent significant zones of rapid change in weather patterns; to identify significant boundaries (over geographic areas) in disease rates, such as cancer, or to track variations in access to health care across geographical regions. Measurement scales of the spatial data usually dictate the methods required for wombling. For areal data, boundaries delineate neighboring regions (see, e.g. Gao et al., 2023; Wu and Banerjee, 2025). Predictive inference is sought for smooth curves. We evaluate spatial gradients along a curve while assessing its candidacy for a boundary (see, e.g., Banerjee and Gelfand, 2006; Qu et al., 2021; Halder et al., 2024), which requires specifying the smoothness of the spatial process (see, e.g. Kent, 1989; Banerjee et al., 2003).

In this software package, we are concerned with point-referenced wombling. Developing an easily accessible software that implements Bayesian wombling for use by the wider scientific community is faced with several challenges, perhaps the most severe being the need for two-dimensional quadrature to enable posterior inference. Our contributions here lie in the use of analytic closed forms for posteriors that require at most one-dimensional quadrature, greatly easing the computational burden and enabling efficient Bayesian inference for hierarchical spatial models (see, e.g. Banerjee et al., 2014) via **nimble** (de Valpine et al., 2017) within the **R** (R Core Team, 2021) statistical environment.

Several **R** packages exist for point-referenced spatial modeling, with **spBayes** (Finley et al., 2007) and **R-INLA** (Lindgren and Rue, 2015) being more widely used. However, they do not address boundary analysis, or wombling, in any capacity. In recent years, **nimble** has found increased use in Bayesian modeling applications (see, e.g. Turek et al., 2016; Ponisio et al., 2020; Goldstein and de Valpine, 2022). Notable **R**-packages that use **nimble** include **BayesNSGP** (Risser and Turek, 2020) and **nimbleEcology** (Goldstein et al., 2024). The Bayesian hierarchical framework in **nimblewomble** is similar to **spBayes**. We take advantage of the *one-line call and execute* feature of **nimble** to develop Markov Chain Monte Carlo (MCMC) algorithms for fitting Gaussian processes (GPs). This makes the underlying code for **nimblewomble** easily accessible and customizable for wider use.

We demonstrate our developments using the Matérn class of covariance kernels (see, e.g., Abramowitz et al., 1988). They are a popular choice in the literature for GPs (see, e.g., Rasmussen and Williams, 2005). They feature a fractal parameter that provides explicit control over process smoothness. Our package offers three choices for the fractal parameter, allowing for flexible process specification. Gradient estimation is done on a grid. We show an example of generating an equally spaced grid. Users can also specify a grid of their choice. Wombling requires a curve; we use contours for that purpose. We demonstrate the procedure for obtaining contours using the **raster** package. Alternatively, a curve of choice can also be used. Our vignettes show an example that uses the `locator()` function. The user annotates points on the interpolated surface and a smooth Bézier curve is generated for use. Finally, posterior samples from models in **spBayes** can also be used for wombling with **nimblewomble**. The same kernel needs to be used in both packages to ensure valid inference.

The ensuing discussion describes the necessary methodological details in brief. We begin with a general overview of the functions within **nimblewomble**. 2.6 contains worked-out examples to

Type	Function	Purpose	Description
Main	materncov1 materncov2 gaussian	covariance kernel	Matérn covariance with $\nu = \frac{3}{2}, \frac{5}{2}$ and ∞ (squared exponential kernel)
	gp_fit	model fitting	Fits a Gaussian Process with non-informative priors. Produces posterior samples for θ
	zbeta_samples	model fitting	Posterior samples for $Z(s)$ and β
	sprates spwombling	inference	Posterior samples for $\partial Z(s)$ and $\partial^2 Z(s)$ Posterior samples for $\Gamma(C)$
	sp_ggplot	plotting	Interpolated spatial surface plots
	significance	–	Determines significance for posterior estimates
Internal	pnorm_nimble	–	Computes the Cumulative Distribution Function (CDF) for the standard Gaussian probability distribution
	gamma_int	–	Incomplete Gamma Function
	gamma1.mcov1 gamma1n2.gauss gamma1n2.mcov2	–	Cross-covariance terms for the posterior distribution of wombling measures
	gradients_matern1 curvatures_matern2 curvatures_gaussian	–	Posterior samples of rates of change (gradients and curvatures) for various kernels
	wombling_matern1 wombling_matern2 wombling_gaussian	–	Posterior samples for wombling measures for various kernels
	zbeta_matern1 zbeta_matern2 zbeta_gaussian	–	Posterior samples of spatial effects and intercept for various kernels
	zXbeta	–	Posterior samples of spatial effects and intercept in the presence of covariates

Table 1: Summary of functions that are required for wombling using `nimblewomble`.

demonstrate the workflow. 2.8 houses an application of the package to perform boundary analysis on a spatial transcriptomics dataset.

2 General Package Overview

The `nimblewomble` package is available for download on the Comprehensive R Archive Network (CRAN). It contains functions that are required to perform wombling. These functions are described in 1. The main functions can broadly be classified into four categories: covariance kernels, model fitting, inference on rates of change and line integrals, and graphical displays. All functions, with the exception of plotting, are scripted as `nimbleFunctions` with wrapper functions that are callable through **R**. This enables fast execution using their compiled C++ counterparts. We generate spatial graphics using `ggplot2` (Wickham, 2011) and `MBA` (Finley, 2024). Other internal helper functions within the package serve specific computational purposes, for example, the incomplete Gamma integral is computed by `gamma_int`. They are primarily for internal use and are described under the internal type (row subsections) within 1.

3 Spatial Processes for Rates of Change

We consider $\{Y(s) : s \in \mathcal{S} \subseteq \mathcal{R}^2\}$ to be a univariate weakly stationary random field with zero mean and a positive definite covariance $K(s, s') = \text{Cov}(Y(s), Y(s'))$ for locations $s, s' \in \mathcal{R}^2$. Mean square smoothness (see, e.g., Stein, 1999) at an arbitrary location s_0 requires $Y(s_0 + hu) = Y(s_0) + hu^T \partial Y(s_0) + h^2 u^{\otimes 2T} \partial^{\otimes 2} Y(s_0) + o(h^3 \|u\|^3)$, where, $u = (u_1, u_2)^T \in \mathcal{R}^2$ is an arbitrary vector of directions, ∂ is the gradient operator, $\partial Y(s_0) = \left(\frac{\partial}{\partial s_x} Y(s_0), \frac{\partial}{\partial s_y} Y(s_0) \right)^T = (\partial_x Y(s_0), \partial_y Y(s_0))^T$, $\otimes$ is the Kronecker vector product. Hence, $u^{\otimes 2} = (u_1^2, u_1 u_2, u_2 u_1, u_2^2)^T$ and $\partial^{\otimes 2} = \partial \otimes \partial$. Note that $\partial^{\otimes 2} Y(s_0)$ is the vectorized Hessian. The processes $\partial Y(s_0)$ and $\partial^{\otimes 2} Y(s_0)$ govern rates of change in $Y(s_0)$. The *gradient*, or first-order rate of change, is captured by $\partial Y(s_0)$ while *curvature* is captured by $\partial^{\otimes 2} Y(s_0)$. *Mean square differentiability* (see Banerjee and Gelfand, 2003) of the first and second order for $Y(s_0)$ guarantees

that $\mathbf{u}^\top \partial Y(s_0)$ and $(\mathbf{u} \otimes \mathbf{v})^\top \partial^{\otimes 2} Y(s_0)$ are well-defined respectively, for any set of direction vectors $\mathbf{u}, \mathbf{v} \in \mathfrak{R}^2$. We note that the entries of $\partial^{\otimes 2} Y(s_0)$ contain duplicates, both $\partial_{xy}^2 Y(s_0) = \frac{\partial^2}{\partial s_x \partial s_y} Y(s_0)$ and $\partial_{yx}^2 Y(s_0)$ are included. To avoid singularities that arise from duplication, we work with $\tilde{\partial}^{\otimes 2} Y(s_0) = \left(\partial_{xx}^2 Y(s_0), \partial_{xy}^2 Y(s_0), \partial_{yy}^2 Y(s_0) \right)^\top$ comprised of only unique derivatives.

Statistical inference is devised for the joint process, $\mathcal{L}^* Y(s) = \left(Y(s), \partial Y(s)^\top, \tilde{\partial}^{\otimes 2} Y(s)^\top \right)^\top$. Note that in $\mathfrak{R}^2$, $\partial Y(s)$ and $\tilde{\partial}^{\otimes 2} Y(s)$ are 2×1 and 3×1 vectors respectively. Validity of the inference is considered at length in [Banerjee et al. \(2003\)](#); [Halder et al. \(2024\)](#). The process $\mathcal{L}^* Y(s)$ is also weakly stationary with a cross-covariance matrix of order 6×6 given by,

$$\mathbf{V}_{\mathcal{L}^*}(\Delta) = \begin{pmatrix} K(\Delta) & \partial K(\Delta)^\top & \tilde{\partial}^2 K(\Delta)^\top \\ -\partial K(\Delta) & -\partial^2 K(\Delta) & -\tilde{\partial}^3 K(\Delta)^\top \\ \tilde{\partial}^2 K(\Delta) & \tilde{\partial}^3 K(\Delta) & \tilde{\partial}^4 K(\Delta) \end{pmatrix}, \quad (1)$$

where $\Delta = s - s'$, $\partial K(\Delta)$ is a 2×1 vector of gradients, $\tilde{\partial}^2 K(\Delta)$ is a 3×1 vector of unique curvatures, $\tilde{\partial}^3 K(\Delta)$ is a 3×2 matrix of third derivatives, $\partial^2 K(\Delta)$ is the 2×2 Hessian and $\tilde{\partial}^4 K(\Delta)$ is a 3×3 matrix of fourth-order derivatives. Evidently, for $\mathbf{V}_{\mathcal{L}^*}(\Delta)$ in 1 to be valid all entries need to be well-defined.

Let $Y(s) \sim GP(0, K(\cdot; \theta))$ denote a Gaussian process (GP) where $K(\Delta; \theta) = \text{Cov}(Y(s), Y(s'))$ with process parameters $\theta = \{\sigma^2, \phi\}$. We will denote $K(\Delta; \theta) = K(\Delta)$ to ease notation. The covariance function satisfies $\sum_{i=1}^N \sum_{j=1}^N a_i a_j K(\Delta_{ij}) > 0$ for any collection of coordinates $\{s_i : i = 1, \dots, N\}$. Under isotropy we have $K(\Delta) = \tilde{K}(\|\Delta\|)$. Let $\mathbf{y} = (y(s_1), \dots, y(s_N))^\top$ be the observed realization over $\mathcal{S}$, $\Sigma_{\mathbf{y}}$ be the $N \times N$ covariance matrix with entries $K(s_i, s_j)$, $i, j = 1, \dots, N$ and s_0 an arbitrary location of interest. The joint distribution is as follows:

$$\mathbf{y}, \partial Y(s_0), \tilde{\partial}^2 Y(s_0) \mid \theta \sim \mathcal{N}_{N+5} \left(\mathbf{0}_{N+5}, \begin{pmatrix} \Sigma_{\mathbf{y}} & \mathbf{K}_1 & \mathbf{K}_2 \\ -\mathbf{K}_1^\top & -\partial^2 K(0) & -\tilde{\partial}^3 K(0)^\top \\ \mathbf{K}_2^\top & \tilde{\partial}^3 K(0) & \tilde{\partial}^4 K(0) \end{pmatrix} \right), \quad (2)$$

where, $\mathcal{N}_d$ denotes the d -variate Gaussian distribution, $\mathbf{K}_1 = (\partial K(\delta_{10})^\top, \dots, \partial K(\delta_{N0})^\top)^\top$, $\mathbf{K}_2 = (\tilde{\partial}^2 K(\delta_{10})^\top, \dots, \tilde{\partial}^2 K(\delta_{N0})^\top)^\top$ and $\delta_{i0} = s_i - s_0$, $i = 1, \dots, N$. The resulting posterior predictive distribution for rates of change at s_0 is $P(\partial Y(s_0), \tilde{\partial}^2 Y(s_0) \mid \mathbf{y}) = \int P(\partial Y(s_0), \tilde{\partial}^2 Y(s_0) \mid \mathbf{y}, \theta) P(\theta \mid \mathbf{y}) d\theta$. Posterior sampling proceeds in a one-for-one fashion corresponding to posterior samples of θ . From 2 the resulting full conditional distribution is obtained as follows:

$$\begin{pmatrix} \partial Y(s_0) \\ \tilde{\partial}^2 Y(s_0) \end{pmatrix} \mid \mathbf{y} \sim \mathcal{N}_5 \left(- \begin{pmatrix} \mathbf{K}_1 \\ \mathbf{K}_2 \end{pmatrix}^\top \Sigma_{\mathbf{y}}^{-1} \mathbf{y}, \begin{pmatrix} -\partial^2 K(0) & -\tilde{\partial}^3 K(0)^\top \\ \tilde{\partial}^3 K(0) & \tilde{\partial}^4 K(0) \end{pmatrix} - \begin{pmatrix} \mathbf{K}_1 \\ \mathbf{K}_2 \end{pmatrix}^\top \Sigma_{\mathbf{y}}^{-1} \begin{pmatrix} \mathbf{K}_1 \\ \mathbf{K}_2 \end{pmatrix} \right). \quad (3)$$

We use the Matérn class of kernels, $K(\|\Delta\|, \theta) = \sigma^2 \Gamma(\nu)^{-1} 2^{1-\nu} \left(\sqrt{2\nu\phi} \|\Delta\| \right)^\nu K_\nu \left(\sqrt{2\nu\phi} \|\Delta\| \right)$, where $K_\nu(\cdot)$ is the modified Bessel function of the second kind ([Abramowitz et al., 1988](#)) featuring a fractal parameter ν that controls process smoothness, a spatial range parameter ϕ and an overall variance parameter σ^2 .

4 Spatial Wombling

The wombling exercise seeks posterior predictive inference on line integrals

$$\mathbf{\Gamma}(C) = \left(\int_C \mathbf{u}^\top \partial Y(s) ds, \int_C \mathbf{u}^{\otimes 2^\top} \partial^{\otimes 2} Y(s) ds \right)^\top, \quad (4)$$

where C is a curve of interest to the investigator. Average wombling measures are defined as $\bar{\mathbf{\Gamma}}(C) = \mathbf{\Gamma}(C) / \ell(C)$, where ℓ is the arc-length measure. For closed curves we replace $\int$ with $\oint$ in 4. The choice of direction is crucial when measuring rates of change. The curve C typically tracks a region of rapid change in the reference domain and hence, the direction normal to C is naturally of interest. We denote the normal to C at s by $\mathbf{n}(s)$ and set $\mathbf{u} = \mathbf{n}(s)$ in the line integrals of 4. The curve C is deemed to be a *wombling boundary* if any entry of $\mathbf{\Gamma}(C)$ is large. Focusing on the choices for C , not all curves ensure the existence of $\mathbf{n}(s)$ at every s . Parametric smooth curves offer some respite in that regard. We work with $C = \{s(t) = (s_1(t), s_2(t)) : t \in \mathcal{I} \subset \mathfrak{R}\}$. As t varies over $\mathcal{I}$, $s(t)$ traces out C . We assume $\|s'(t)\| \neq 0$ which ensures $\mathbf{n}(s) = \|s'(t)\|^{-1} (s'_2(t), -s'_1(t))^\top$ is well-defined.

The arc-length, $\ell(C) = \int_{\mathcal{I}} \|s'(t)\| dt$. For parametric curves $\Gamma(C)$ can be expressed as, $\Gamma(C) = \left(\int_{\mathcal{I}} \mathbf{n}(s(t))^T \partial Y(s(t)) \|s'(t)\| dt, \int_{\mathcal{I}} \mathbf{n}(s(t))^{\otimes 2T} \partial^{\otimes 2} Y(s(t)) \|s'(t)\| dt \right)^T$. Let $\mathcal{I} = [0, t^*]$, and the curve traced out over $\mathcal{I}$ be denoted as C_{t^*} . Statistical inference for $\Gamma(C_{t^*})$ follows from $\partial Y(s)$ and $\tilde{\partial}^2 Y(s)$ being GPs, as seen in 3, $\Gamma(C_{t^*}) \sim \mathcal{N}_2(\mathbf{0}_2, \mathbf{K}_{\Gamma}(t^*, t^*))$, where $\mathbf{K}_{\Gamma}(t^*, t^*)$ is a 2×2 matrix with entries

$$k_{ij}(t^*, t^*) = (-1)^i \int_0^{t^*} \int_0^{t^*} \mathbf{a}_i(t_1)^T \partial^{i+j} K(\Delta(t_1, t_2)) \mathbf{a}_j(t_2) \|s'(t_1)\| \|s'(t_2)\| dt_1 dt_2, \quad (5)$$

where $\mathbf{a}_1(t) = \mathbf{n}(s(t))$, $\mathbf{a}_2(t) = \mathcal{E}_2 \mathbf{n}(s(t))^{\otimes 2}$, with $\mathcal{E}_2 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ being an elimination matrix and $\Delta(t_1, t_2) = s_2(t) - s_1(t)$ for $i, j = 1, 2$. Predictive inference on $\Gamma(C)$ requires

$$\mathcal{Y}, \Gamma(C_{t^*}) | \boldsymbol{\theta} \sim \mathcal{N}_{N+2} \left(\mathbf{0}_{N+2}, \begin{pmatrix} \Sigma_{\mathcal{Y}} & \mathcal{G}_{\Gamma}(t^*)^T \\ \mathcal{G}_{\Gamma}(t^*) & \mathbf{K}_{\Gamma}(t^*, t^*) \end{pmatrix} \right), \quad (6)$$

where $\mathcal{G}_{\Gamma}(t^*)^T = \begin{pmatrix} \gamma_1(t^*)^T \\ \vdots \\ \gamma_N(t^*)^T \end{pmatrix}$ is an $N \times 2$ matrix with entries

$$\gamma_i(t^*) = \left(\int_0^{t^*} \mathbf{n}(s(t))^T \partial K(\Delta_j(t)) \|s'(t)\| dt, \int_0^{t^*} \mathbf{n}(s(t))^{\otimes 2T} \partial^{\otimes 2} K(\Delta_j(t)) \|s'(t)\| dt \right)^T, \quad (7)$$

where $\Delta_j(t) = s(t) - s_j$. Posterior predictive inference proceeds one-for-one (similar to 3) using $\Gamma(C_{t^*}) | \mathcal{Y} \sim \mathcal{N}_2 \left(-\mathcal{G}_{\Gamma}(t^*) \Sigma_{\mathcal{Y}}^{-1} \mathcal{Y}, \mathbf{K}_{\Gamma}(t^*, t^*) - \mathcal{G}_{\Gamma}(t^*) \Sigma_{\mathcal{Y}}^{-1} \mathcal{G}_{\Gamma}(t^*)^T \right)$.

In practice, modern computing environments store curves as a set of points. As a result, it suffices to demonstrate wombling for rectilinear approximations to smooth curves where predictive inference is performed iteratively on segments. We show the inference for one generic segment. Let $C = \{s(t_0), s(t_1), \dots, s(t_{n_p})\}$, then the i -th segment, $C_{t_i} = \{s(t) = s(t_{i-1}) + t\mathbf{u}_i : t \in [0, t_i]\}$, $t_i = \|s(t_i) - s(t_{i-1})\|$ and $\mathbf{u}_i = t_i^{-1}(s(t_i) - s(t_{i-1}))$. Clearly, $\|\mathbf{u}_i\| = 1$, $\|s'(t)\| = 1$ and the normal to C_{t_i} is $\mathbf{u}_i^{\perp} = (u_{i2}, -u_{i1})^T$. For predictive inference on $\Gamma(C_{t_i})$, note that we need $\Delta(t_1, t_2) = (t_2 - t_1)\mathbf{u}_i$ in 5 and $\Delta_j(t) = \Delta_{i-1,j} + t\mathbf{u}_i = (s_{i-1} - s_j) + t\mathbf{u}_{i-1}$ in 7.

4.1 Wombling with Closed Forms

We acknowledge that 5 requires 2-dimensional quadrature, which is computationally expensive to evaluate. We work with the Matérn kernel for which closed-form analytic expressions exist for the entries of $\mathbf{K}_{\Gamma}(t^*, t^*)$ improving on Banerjee and Gelfand (2006); Halder et al. (2024) (see Theorems 1 & 2 in the Supplement). Apart from reduced computation time resulting from the reduction of a surface integral to closed-form expressions, the Matérn class is uniquely suited for wombling, featuring a dedicated fractal parameter that controls process smoothness and thereby guaranteeing validity of posterior inference on derivatives and line integrals (see, e.g., Halder et al., 2024). Our R-package, *nimblewomble* features Matérn kernels with $\nu = \frac{3}{2}, \frac{5}{2}$ and ∞ (squared exponential).

5 Bayesian Hierarchical Models

A Bayesian hierarchical model is specified as follows:

$$Y(s) = \mu(s, \boldsymbol{\beta}) + Z(s) + \epsilon(s), \quad (8)$$

where $\mu(s, \boldsymbol{\beta}) = x(s)^T \boldsymbol{\beta}$, $Z(s) \sim GP(0, K(\cdot; \sigma^2, \phi))$ and $\epsilon(s)$ is a white noise process (i.e., $\epsilon(s_i) \stackrel{iid}{\sim} N(0, \tau^2)$ over any finite collection of locations). The process parameters are $\boldsymbol{\theta} = \{\sigma^2, \phi, \tau^2\}$. Predictive inference for $\mathcal{L}^* Z(s)$ evaluates $P(\partial Z(s)^T, \tilde{\partial}^2 Z(s)^T | \mathcal{Y}) = \int P(\partial Z(s)^T, \tilde{\partial}^2 Z(s)^T | \mathcal{Z}, \boldsymbol{\theta}) P(\mathcal{Z} | \mathcal{Y}, \boldsymbol{\theta}) P(\boldsymbol{\theta} | \mathcal{Y}) d\boldsymbol{\theta} d\mathcal{Z}$. Similarly for the wombling measures, $P(\Gamma_{\mathcal{Z}}(C_{t^*}) | \mathcal{Y}) = \int P(\Gamma_{\mathcal{Z}}(C_{t^*}) | \mathcal{Z}, \boldsymbol{\theta}) P(\mathcal{Z} | \mathcal{Y}, \boldsymbol{\theta}) P(\boldsymbol{\theta} | \mathcal{Y}) d\boldsymbol{\theta} d\mathcal{Z}$. A customary *collapsed* posterior, that is generated by marginalizing $\mathcal{Z}$ from the likelihood (see, e.g. Finley et al., 2019), for $\boldsymbol{\theta}$ is specified as follows:

$$P(\boldsymbol{\theta} | \mathcal{Y}) \propto U(\phi | a_{\phi}, b_{\phi}) \times IG(\sigma^2 | a_{\sigma}, b_{\sigma}) \times IG(\tau^2 | a_{\tau}, b_{\tau}) \times \mathcal{N}_N(\mathcal{Y} | \mathbf{X}\boldsymbol{\beta}, \Sigma + \tau^2 \mathbf{I}_N), \quad (9)$$

where $\Sigma = \sigma^2 \mathbf{R}_{\mathcal{Z}}(\phi)$, with $\mathbf{R}_{\mathcal{Z}}(\phi)$ being the correlation matrix corresponding to $K(\cdot; \sigma^2, \phi)$, $U(\cdot | \cdot)$ is the uniform distribution and $IG(\cdot | \cdot)$ is the inverse-gamma distribution. Hyper-parameters are chosen such that a weakly informative prior is specified on $\boldsymbol{\theta}$. Posterior samples for $\mathcal{Z}$ and $\boldsymbol{\beta}$ are generated one-for-one corresponding to posterior samples of $\boldsymbol{\theta}$ using a Gibbs sampling scheme.

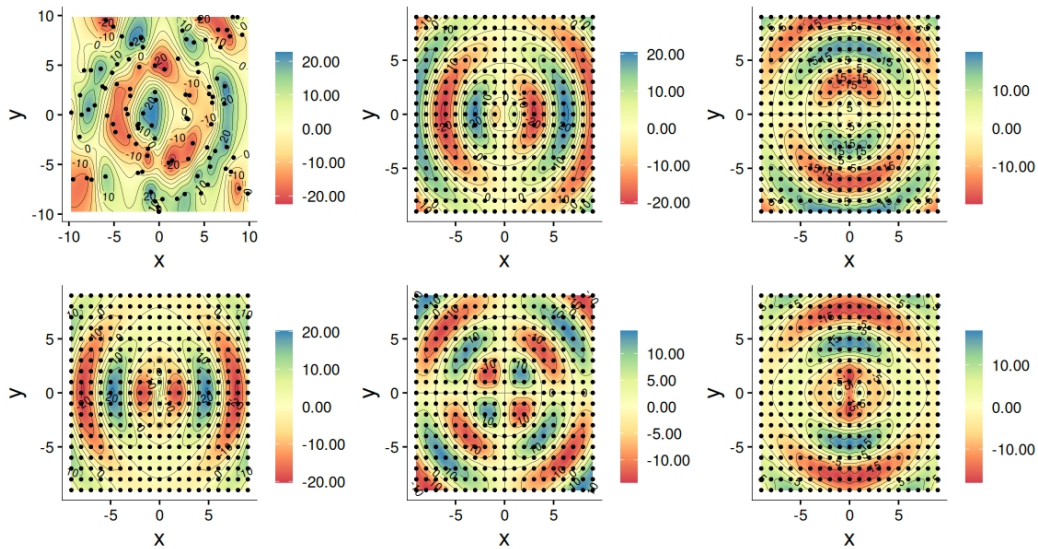


Figure 1: Patterned data used for experiments. Top row: (left) simulated process (middle) ∂_x (right) ∂_y . Bottom row: (left) ∂_{xx}^2 (middle) ∂_{xy}^2 (right) ∂_{yy}^2 . The grid used is overlaid on the plots.

Posterior sampling in 9 is straightforward in nimbleCode as seen in the code for `gp_model` below, which forms the core of our `gp_fit` function (see 1). We use hyper-parameter settings that result in weakly informative inverse gamma priors for σ^2 and τ^2 . We take advantage of the *one-line call and execute* feature of nimble using the `buildMCMC` and `runMCMC` functions to obtain posterior samples from 9 thereby, fitting 8.

```
#####
# Collapsed Metropolis-Hastings #
# for covariance parameters      #
#####

gp_model <- nimbleCode({
  # Priors #
  phi ~ dunif(0, 10)
  sigma2 ~ dinvgamma(shape = 1, rate = 1)
  tau2 ~ dinvgamma(shape = 2, rate = 1)

  # Initialization #
  mu[1:N] <- zeros[1:N] # vector of 0s
  cov[1:N, 1:N] <- kernel(dists[1:N, 1:N], phi, sigma2, tau2)

  # Likelihood #
  y[1:N] ~ dmnorm(mu[1:N], cov = cov[1:N, 1:N])
})
```

Note that for different choices kernel is replaced with the corresponding kernel choice in 1.

6 Workflow of nimblewomble

We detail the workflow for nimblewomble using simulated data. We produce data using patterns that yield closed-form expressions for rates of change. This helps calibrate the predictive performance of nimblewomble. We generate $N = 100$ observations over $[-10, 10] \times [-10, 10] \subseteq \mathbb{R}^2$ arising from, $y(s) \sim \mathcal{N}_1(\mu_0(s) = 20 \sin ||s||, \tau^2)$. We set $\tau^2 = 1$. Here, the true values of gradients are available in closed form. For example, $\partial_x \mu_0(s_G) = 20 \cos ||s_G|| s_{x,G} ||s_G||^{-1}$, where $s_G = (s_{x,G}, s_{y,G})$ lies on a grid overlaid on the domain of reference (in this case: $[-10, 10] \times [-10, 10]$). Other gradient and curvature processes are computed similarly by differentiating $\mu_0(s)$. Running the following code generates the simulated data and produces plots in Figure 1.

```
set.seed(1)
# Generating Simulated Data
```

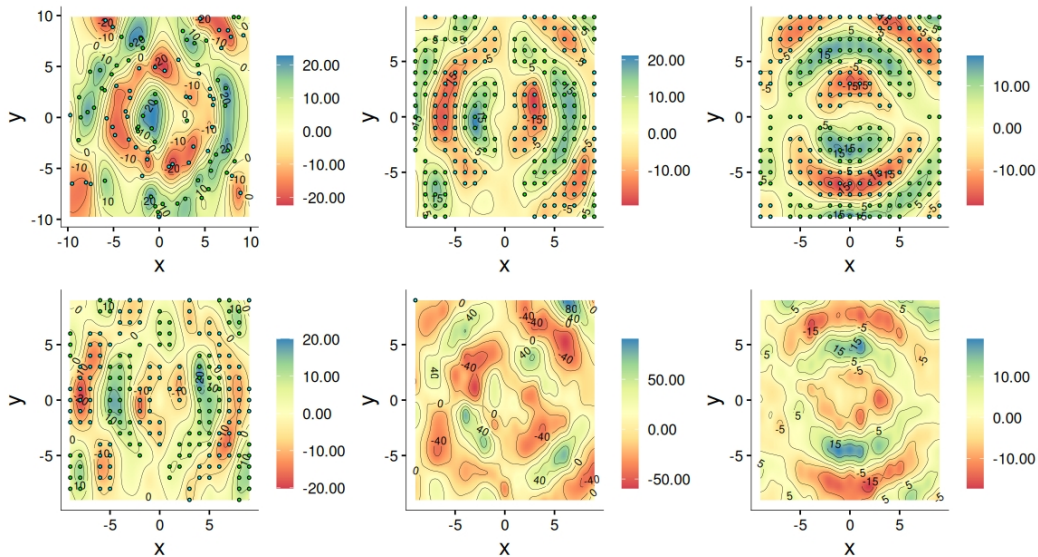


Figure 2: Estimated patterns with highlighted significant locations: positive (green) negative (cyan).

```

N = 1e2
tau = 1
coords = matrix(runif(2 * N, -10, 10), ncol = 2); colnames(coords) = c("x", "y")
y = rnorm(N, mean = 20 * sin(sqrt(coords[, 1]^2 + coords[, 2]^2)), sd = tau)

# Create equally spaced grid of points
xsplitted = seq(-10, 10, by = 1)[-c(1, 21)]
ysplitted = seq(-10, 10, by = 1)[-c(1, 21)]
grid = as.matrix(expand.grid(xsplitted, ysplitted), ncol = 2)
colnames(grid) = c("x", "y")

#####
# Process for True Rates of Change #
#####
# Gradient along x
true_sx = round(20 * cos(sqrt(grid[,1]^2 + grid[,2]^2)) *
                grid[,1]/sqrt(grid[,1]^2 + grid[,2]^2), 3)
# Gradient along y
true_sy = round(20 * cos(sqrt(grid[,1]^2 + grid[,2]^2)) *
                grid[,2]/sqrt(grid[,1]^2 + grid[,2]^2), 3)
# Plotting
sp_ggplot(data_frame = data.frame(coords, z = y))
sp_ggplot(data_frame = data.frame(grid[-which(is.nan(true_sx)),],
                                  z = true_sx[-which(is.nan(true_sx))]))

```

We fit the model in 8 using `gp_fit` and a Matérn kernel with $\nu = \frac{5}{2}$ to the simulated data. This allows for inference on gradients and curvatures. Running the code below first generates posterior samples of θ from 9 followed by posterior samples for $Z(s)$ and β one-for-one with θ . The `mc_sp` object is a list comprised of (a) MCMC samples for θ stored in `mc_sp$mcmc` and (b) the estimates: median and 95% confidence intervals (CIs) stored in `mc_sp$estimates`. Posterior samples for $Z(s)$ and β are obtained using `zbeta_samples` as seen below. The model object contains samples for θ , Z and β .

```

require(nimble)
require(nimblewomble)

#####
# Fit a Gaussian Process #
#####
# Posterior samples for theta
mc_sp = gp_fit(coords = coords, y = y, kernel = "matern2")
# Posterior samples for Z(s) and beta
model = zbeta_samples(y = y, coords = coords,

```

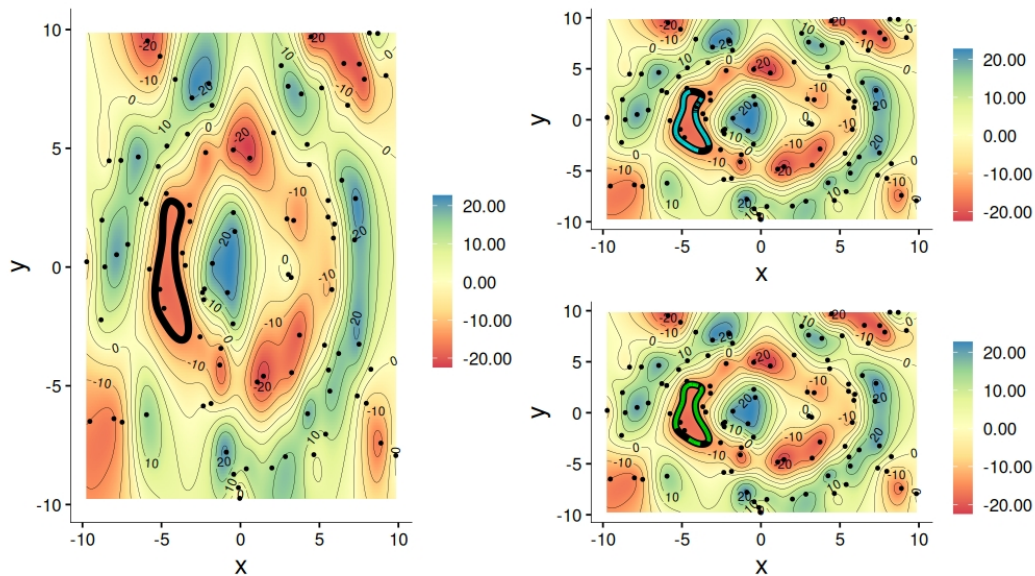



Figure 3: (Left) Curve chosen for wombling, (Right-top) gradient wombling measure for line segments, (Right-bottom) curvature wombling measure for line segments. Significant segments are highlighted: positive (green) negative (cyan).

```
model = mc_sp$mcmc,
kernel = "matern2")
```

Next, we estimate gradients and curvatures using the posterior samples of ϕ , σ^2 and $\mathcal{Z}$ using the `sprates` function. The output stored in `gradients` contains posterior samples and estimates: median and 95% CIs for gradients and curvatures required to produce the plots in Figure 2. Posterior sampling is done one-for-one for samples of ϕ , σ^2 and $\mathcal{Z}$.

```
#####
# Rates of Change #
#####
gradients = sprates(grid = grid,
                    coords = coords,
                    model = model,
                    kernel = "matern2")
# Plot estimated gradients along x
sp_ggplot(data_frame = data.frame(grid,
                                   z = gradients$estimate.sx[, "50%"],
                                   sig = gradients$estimate.sx$sig))
```

The wombling exercise requires a curve. The easiest choice of curves is contours. In R, a rasterized surface using the `raster` package can be used to lift contours from the interpolated surface (as seen in the plot Figure 1 top row left). The code below shows an example. The curve is shown in Figure 3.

```
require(MBA)
require(raster)
# Rasterized Surface
surf <- raster(mba.surf(data.frame(cbind(coords, z = y)),
                      no.X = 300,
                      no.Y = 300,
                      h = 5,
                      m = 2,
                      extend = TRUE, sp = FALSE)$xyz.est)
# convert raster surface to contours
x = rasterToContour(surf, nlevel = 10)

x.levels <- as.numeric(as.character(x$level))
# Curve from a region of relatively low values
curves.pm.subset = subset(x, level == -15)
```

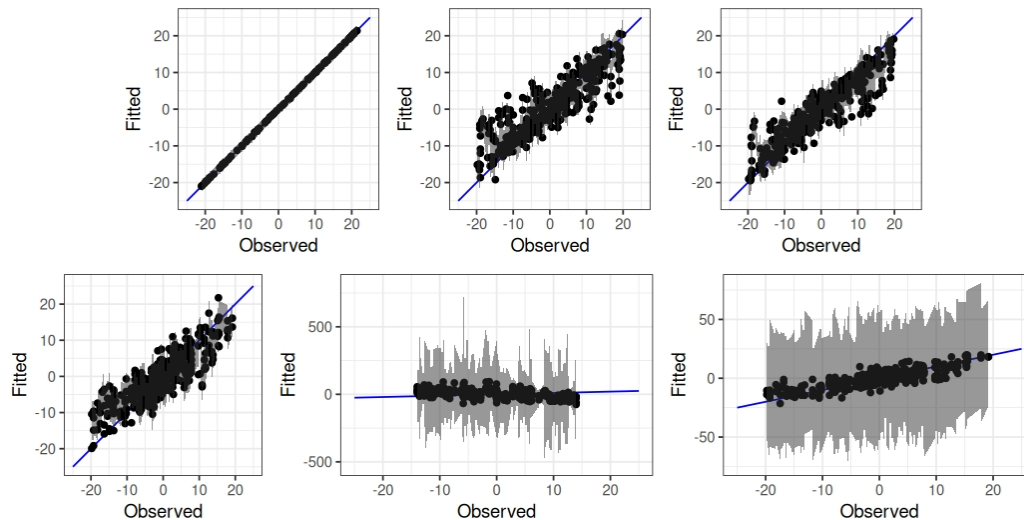



Figure 4: Diagnostics assessing quality of fit using observed vs. fitted values for (Top) (left) response (center) ∂_x (right) ∂_y (Bottom) (left) ∂_{xx}^2 (center) ∂_{xy}^2 (right) ∂_{yy}^2 with 95% credible bands.

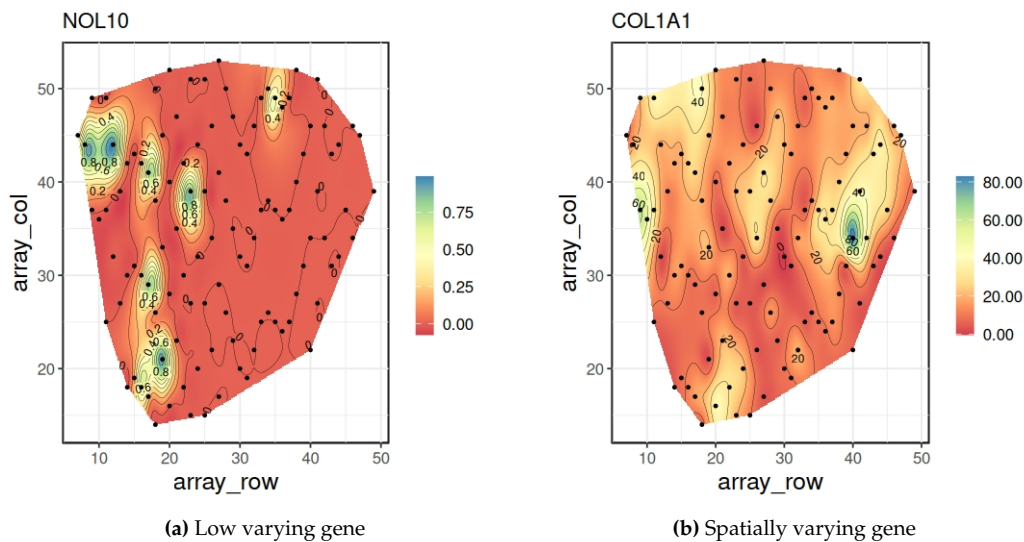


Figure 5: Surfaces for gene expression of a low varying gene and a spatially varying gene.

Wombling is performed on this curve using the `spwombling` function. Posterior samples of $\Gamma(C)$, where C is the chosen curve, are generated one-for-one with σ^2 , ϕ and $\mathcal{Z}$. The code below provides an example. The output is comprised of posterior samples (`wm$wm.mcmc`) of $\Gamma(C)$ and estimates: median and 95% CI (`wm$estimate.wm`). It also produces the plots in Figure 3.

```
require(patchwork)
#####
# Wombling #
#####
wm = spwombling(coords = coords,
                curve = curve,
                model = model,
                kernel = "matern2")
# Total wombling measure for gradient
colSums(wm$estimate.wm.1[, -4])
# Total wombling measure for curvature
colSums(wm$estimate.wm.2[, -4])

# Color code line segments based on significance
# of gradient based wombling measure
```

```

col.pts.1 = sapply(wm$estimate.wm.1$sig, function(x){
  if(x == 1) return("green")
  else if(x == -1) return("cyan")
  else return(NA)
})
# Color code line segments based on significance
# of curvature based wombling measure
col.pts.2 = sapply(wm$estimate.wm.2$sig, function(x){
  if(x == 1) return("green")
  else if(x == -1) return("cyan")
  else return(NA)
})
#####
# Plots for Wombling #
#####
p1 = sp_ggplot(data_frame = data.frame(coords, y))
# Plot in Figure 3 (left)
p2 = p1 + geom_path(curve, mapping = aes(x, y), linewidth = 2)
# Plot in Figure 3 (top-right): gradient
p3 = p1 + geom_path(curve, mapping = aes(x, y), linewidth = 2) +
  geom_path(curve, mapping = aes(x, y),
    colour = c(col.pts.1, NA), linewidth = 1, na.rm = TRUE)
# Plot in Figure 3 (bottom-right): curvature
p4 = p1 + geom_path(curve, mapping = aes(x, y), linewidth = 2) +
  geom_path(curve, mapping = aes(x, y),
    colour = c(col.pts.2, NA), linewidth = 1, na.rm = TRUE)

p2 + (p3/p4) # generates Fig. 3

```

We conclude the workflow with some brief comments on assessing the quality of fit. The default setting of `gp_fit` generates 10,000 posterior samples, with a 5,000 burn-in. The model fit was satisfactory: $\hat{\tau}^2 = 0.384$ (0.145, 1.433) containing the true value of 1, $\hat{\sigma}^2 = 344.680$ (194.292, 687.247) and $\hat{\phi} = 0.380$ (0.302, 0.489) which can be obtained from `mc_sp$estimates`. We achieved $\approx 96\%$ coverage for the estimated rates of change and wombling measures at the line segment level. For the wombling measure, $\widehat{\Gamma(C)} = (-108.765, 154.565)^T$, with 95% CIs being $(-182.098, -36.290)$ and $(69.053, 241.664)$ respectively, containing the true values $\Gamma(C) = (-131.149, 144.010)^T$. They are obtained from `wm$estimate.wm.1` and `wm$estimate.wm.2`. The curve C forms a *wombling boundary*. Figure 4 shows further diagnostics for model fit.

7 Further Computational Details

We provide further details about sensitivity to prior choices, convergence and scalability that affect computational aspects that may arise in practical applications for our package.

Alternative prior choices and sensitivity Although there are numerous precedents for our prior choices (see, for e.g., Banerjee and Gelfand, 2006; Loro et al., 2023; Halder et al., 2024), alternative priors are available for use to the interested investigator (see, for e.g., Gelman, 2006). In reference to the code snippet and discussion in 2.5 there are several alternative choices available for σ , for example, the half Cauchy, the uniform and the folded-noncentral- t distributions. Within NIMBLE, the uniform and half Cauchy distributions are easily implemented (`sigma ~ dcauchy(0, 1) T(0, )`, where the `T(0, )` truncates the Cauchy) while, the folded-noncentral- t requires some effort (using `dt_nonstandard()`). The resulting posterior is robust to these alternative choices. The algorithm for posterior sampling under these complex priors is *automatically determined* by NIMBLE once the model is specified. Under such complex non-conjugate priors a Metropolis Hastings algorithm is usually chosen by NIMBLE, which is shown in the console when fitting the model. The inference for gradients remains unaffected since it is done one-for-one using the posterior samples for σ^2 , τ^2 and ϕ .

Convergence diagnostics The posterior MCMC samples are stored as an `mcmc` object that communicates well with the R-package `coda`. Running `traceplot(mc_sp)` produces trace plots to assess convergence. The `gp_fit()` function also contains a `nchains` argument which defaults to 1; increasing this to ≥ 2 allows for computing $\hat{R}$ using the `gelman.diag()` function for further assessment. We checked posterior convergence using trace plots, which showed satisfactory evidence for convergence.

Scalability Addressing scalability for the package, we repeated the model fit under increasing sample sizes. The model fit takes: 18.37 secs. for $N = 100$, 2.52 mins. for $N = 500$ and 16.39 mins. for $N = 10^3$.

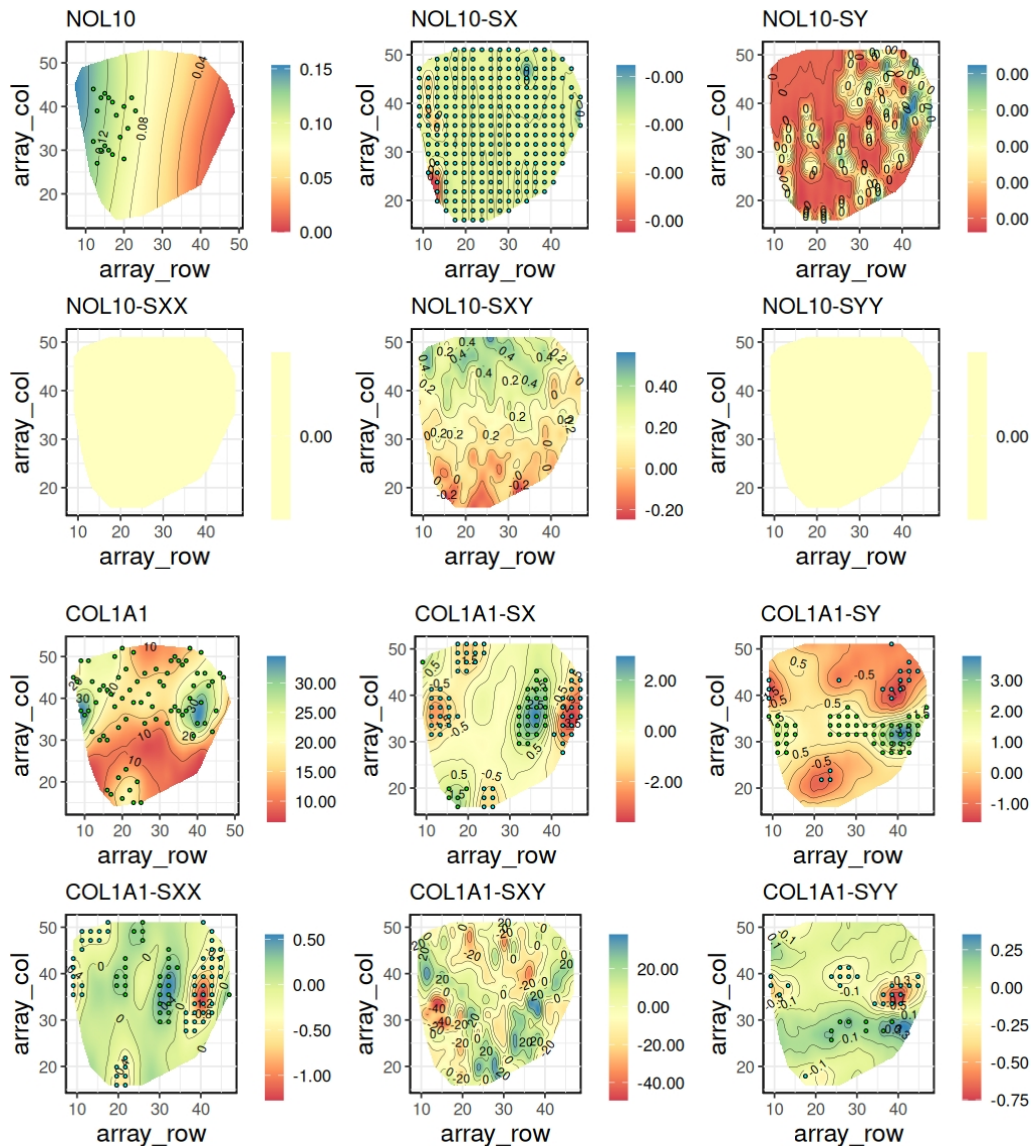


Figure 6: Plots comparing gradients for the two genes. First two rows are for the low varying gene. Bottom two rows are for the high varying gene. Significant grid locations are highlighted.

Setting $N = 100$, for a grid of size 400, generating posterior samples for gradients takes 10.02 mins. and for a curve containing 257 points, posterior samples for wobbling measures take 32.12 mins. Using a coarser representation of grids and curves (fewer points) improves the runtime when working with larger N . These computations were performed on a 12th Gen. Intel®Core™i9-12.9K processor with 24 cores running Ubuntu OS and 96GB of RAM.

8 nimblewomble in Action: Spatial Omics

We demonstrate the workflow of nimblewomble on a spatial omics dataset which is also supplied with the package. The data is an abridged version of what can be found in the Gene Expression Omnibus (accession number [GSE144239](#)) (see, e.g., [Ji et al., 2020](#)). It consists of gene expressions with tumor sampling locations for human squamous cell carcinoma, commonly known as *skin cancer*. Detecting variation in gene expression is key to identifying genetic pathways specific to the cancer type. This has led to a large body of research that focuses on identifying spatially varying genes (SVGs) (see, e.g., [Svensson et al., 2018](#); [Sun et al., 2020](#); [Weber et al., 2023](#); [Chen et al., 2024](#)). We use rates of change to investigate differences between a SVG (COL1A1) and a low variance gene (NOL10).

```
#####
# Load the Data #
```

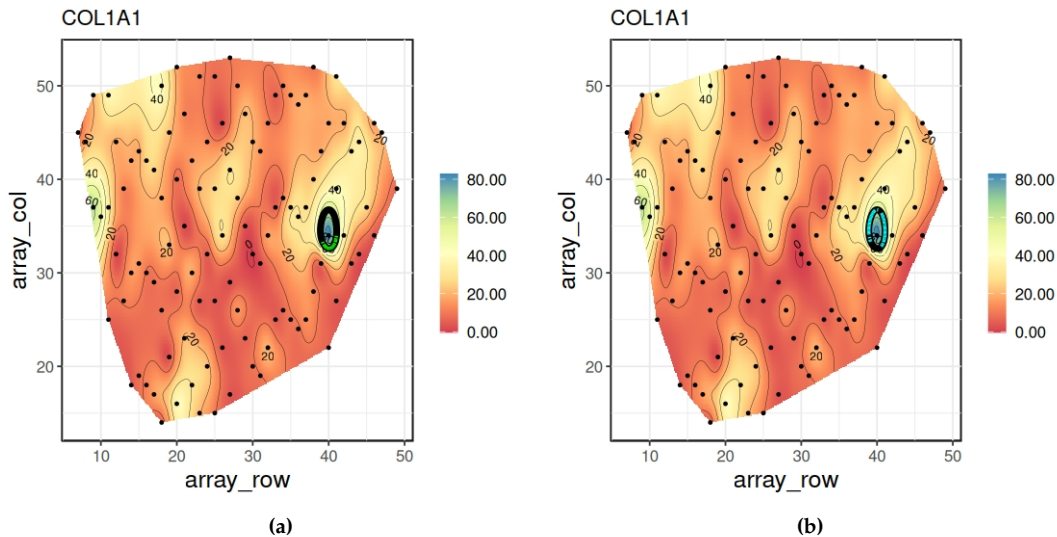


Figure 7: Line segment level inference for (a) gradient and (b) curvature wombling measures.

```
#####
load("genes.RData")
coords = genes[, 1:2]
y = genes[, 4]; gene = "COL1A1"
N = length(y)
# Make a spatial plot of the genetic expression
sp_ggplot(data_frame = data.frame(coords, z = y),
           extend = FALSE, title = gene)
```

The data can be loaded into the **R** console by running the above code. Running `sp_ggplot` produces an interpolated spatial plot of the raw gene expression counts as seen in Figure 5b. Using `genes[, 3]` and running the same code produces Figure 5a. Comparing the ranges for the two plots, the differences in expression are immediate.

We begin by fitting the GP model to individual gene expressions using `gp_fit`. For COL1A1: $\hat{\tau}^2 = 120.667$ (68.168, 198.105), $\hat{\sigma}^2 = 225.652$ (78.750, 529.670); $\hat{\phi} = 0.118$ (0.015, 0.217), while for NOL10: $\hat{\tau}^2 = 0.081$ (0.063, 0.017), $\hat{\sigma}^2 = 0.676$ (0.191, 6.211); $\hat{\phi} = 0.004$ (0.001, 0.016). These can be accessed by running `$estimates` on the object that stores `gp_fit`. We use a Matérn kernel with $\nu = \frac{5}{2}$.

Following up with gradient and curvature estimation using `sprates`, the resulting plots are shown in Figure 6. The top two rows are for NOL10, while the bottom two rows are for COL1A1, which is an SVG. In each set, the first plot shows the fitted process followed by the gradients: $-SX$, $-SY$ and the curvatures: $-SXX$, $-SXY$ and $-SYY$. Comparing the magnitude of corresponding gradients and curvatures, we see more significant grid locations show up for rates of change for the SVG as compared to NOL10. This provides a more detailed picture of the manifestation of variation in expression in the SVG rather than comparing estimates of overall variance.

Finally, we pick a curve within the expression surface of COL1A1 that tracks a region of high expression (see Figure 5b) and performed wombling using `spwombling`. The results for the gradient-based wombling measure and the curvature-based wombling measure are shown in Figure 7a and Figure 7b respectively; $\hat{\Gamma}(C) = (18.176, -8.976)^T$ with corresponding 95% CI being $(-5.353, 55.965)$ for the gradient measure and $(-27.904, -0.073)$ for the curvature measure, indicating that C forms a *curvature boundary*. Curves like C provide a deeper look into the tumor micro-environment.

9 Summary

We have developed an easy-to-use software for boundary analysis, or wombling, under a Bayesian framework. We hope that it will find use in many applications; for example, see Banerjee and Gelfand (2006); Halder et al. (2024). The `sp_ggplot` function also features an option to supply shape-files for more mainstream geostatistical applications. Further examples are available in the GitHub repository: [arh926/nimblewomble/](https://github.com/arh926/nimblewomble/). Boundary analysis requires the investigator to pre-select curves. Identifying such curves in context of the application often proves crucial for detecting differential behavior in the response variable. Nimble facilitates an accessible MCMC framework that is immensely helpful in

developing the statistical inference for rates of change and boundary analysis.

Future developments can proceed along many directions. We hope to expand the software to include spatiotemporal wombling (see, e.g., Halder et al., 2024). Similar frameworks can be developed for generalized linear models, particularly focusing on zero-inflated models (see, e.g. Finley et al., 2011; Halder et al., 2021) which provide a more realistic setting for analyzing raw gene-expression counts. We also plan to include code for inference on directional data considering Bayesian inference for the direction of maximum gradient and curvature (see, e.g., Wang and Gelfand, 2014; Wang et al., 2018). Finally, developments in wombling have relied on the assumption of stationarity and isotropy. In hindsight, the assumptions yielded simplified mathematical expressions. Smoothness for nonstationary processes can be developed (see, e.g. Paciorek and Schervish, 2006) depending on the smoothness of location specific covariances. Investigations for process smoothness can also be considered for (geometrically) anisotropic processes (see, e.g., Allard et al., 2016)—we now define $K(||\Delta||) = \tilde{K}(r)$, where $r = \sqrt{\Delta^T A \Delta}$, with a positive definite matrix, A corresponding to rotations and translations. The derivatives are then taken with respect to the radial profile, r .

References

- M. Abramowitz, I. A. Stegun, and R. H. Romer. Handbook Of Mathematical Functions With Formulas, Graphs, and Mathematical Tables, 1988. [p1, 3]
- D. Allard, R. Senoussi, and E. Porcu. Anisotropy models for spatial data. *Mathematical Geosciences*, 48(3):305–328, 2016. [p12]
- S. Banerjee. Spatial Gradients and Wombling. In A. E. Gelfand, P. Diggle, P. Guttorp, and M. Fuentes, editors, *Handbook of Spatial Statistics*, pages 559–575. Taylor & Francis, 08 2010. ISBN 978-1-4200-7287-7. doi: 10.1201/9781420072884-c31. [p1]
- S. Banerjee and A. E. Gelfand. On smoothness properties of spatial processes. *Journal of Multivariate Analysis*, 84(1):85–100, Jan. 2003. ISSN 0047-259X. [p2]
- S. Banerjee and A. E. Gelfand. Bayesian Wombling: Curvilinear Gradient Assessment Under Spatial Process Models. *Journal of the American Statistical Association*, 101(476):1487–1501, 2006. [p1, 4, 9, 11]
- S. Banerjee, A. E. Gelfand, and C. F. Sirmans. Directional Rates of Change Under Spatial Process Models. *Journal of the American Statistical Association*, 98(464):946–954, Dec. 2003. ISSN 0162-1459. [p1, 3]
- S. Banerjee, B. P. Carlin, and A. E. Gelfand. *Hierarchical Modeling and Analysis for Spatial Data*. Chapman and Hall/CRC, New York, 2 edition, Sept. 2014. ISBN 978-0-429-13717-4. [p1]
- J. Chen, C. Xiong, Q. Sun, G. W. Wang, G. P. Gupta, A. Halder, Y. Li, and D. Li. Investigating spatial dynamics in spatial omics data with StarTrail, May 2024. [p10]
- P. de Valpine, D. Turek, C. J. Paciorek, C. Anderson-Bergman, D. T. Lang, and R. Bodik. Programming With Models: Writing Statistical Algorithms for General Model Structures With NIMBLE. *Journal of Computational and Graphical Statistics*, 26(2):403–413, Apr. 2017. ISSN 1061-8600. [p1]
- A. O. Finley. *MBA: Multilevel B-Spline Approximation*, 2024. R package version 0.1-2. [p2]
- A. O. Finley, S. Banerjee, and B. P. Carlin. spBayes: An R Package for Univariate and Multivariate Hierarchical Point-referenced Spatial Models. *Journal of Statistical Software*, 19:1–24, Apr. 2007. ISSN 1548-7660. [p1]
- A. O. Finley, S. Banerjee, and D. W. MacFarlane. A Hierarchical Model for Quantifying Forest Variables Over Large Heterogeneous Landscapes With Uncertain Forest Areas. *Journal of the American Statistical Association*, 106(493):31–48, Mar. 2011. ISSN 0162-1459. [p12]
- A. O. Finley, A. Datta, B. D. Cook, D. C. Morton, H. E. Andersen, and S. Banerjee. Efficient Algorithms for Bayesian Nearest Neighbor Gaussian processes. *Journal of Computational and Graphical Statistics*, 28(2):401–414, 2019. [p4]
- L. Gao, S. Banerjee, and B. Ritz. Spatial Difference Boundary Detection for Multiple Outcomes Using Bayesian Disease Mapping. *Biostatistics*, 24(4):922–944, Oct. 2023. ISSN 1465-4644. [p1]
- A. Gelman. Prior distributions for variance parameters in hierarchical models (comment on article by Browne and Draper). *Bayesian Analysis*, 1(3):515–534, 2006. [p9]

- J. F. Gleyze, J. N. Bacro, and D. Allard. Detecting Regions of Abrupt Change: Wombling Procedure and Statistical Significance. In P. Monestiez, D. Allard, and R. Froidevaux, editors, *geoENV III — Geostatistics For Environmental Applications*, pages 311–322, Dordrecht, 2001. Springer Netherlands. ISBN 978-94-010-0810-5. [p1]
- B. R. Goldstein and P. de Valpine. Comparing N-mixture Models and GLMMs for Relative Abundance Estimation in a Citizen Science Dataset. *Scientific Reports*, 12:12276, 2022. [p1]
- B. R. Goldstein, D. Turek, L. Ponisio, and P. de Valpine. nimbleEcology: Distributions for Ecological Models in nimble, 2024. URL <https://cran.r-project.org/package=nimbleEcology>. R package version 0.5.0. [p1]
- A. Halder, S. Mohammed, K. Chen, and D. K. Dey. Spatial Tweedie Exponential Dispersion Models: An Application to Insurance Rate-Making. *Scandinavian Actuarial Journal*, 2021(10):1017–1036, 2021. [p12]
- A. Halder, S. Banerjee, and D. K. Dey. Bayesian Modeling with Spatial Curvature Processes. *Journal of the American Statistical Association*, 119(546):1155–1167, 2024. doi: 10.1080/01621459.2023.2177166. [p1, 3, 4, 9, 11, 12]
- A. L. Ji, A. J. Rubin, K. Thrane, S. Jiang, D. L. Reynolds, R. M. Meyers, M. G. Guo, B. M. George, A. Mollbrink, J. Bergenstr hle, L. Larsson, Y. Bai, B. Zhu, A. Bhaduri, J. M. Meyers, X. Rovira-Clav , S. T. Hollmig, S. Z. Aasi, G. P. Nolan, J. Lundeborg, and P. A. Khavari. Multimodal Analysis of Composition and Spatial Architecture in Human Squamous Cell Carcinoma. *Cell*, 182(2):497–514.e22, July 2020. ISSN 1097-4172. [p10]
- J. T. Kent. Continuity Properties for Random Fields. *The Annals of Probability*, 17(4):1432–1440, 1989. ISSN 0091-1798. Publisher: Institute of Mathematical Statistics. [p1]
- F. Lindgren and H. Rue. Bayesian Spatial Modelling with R-INLA. *Journal of Statistical Software*, 63: 1–25, Feb. 2015. ISSN 1548-7660. [p1]
- P. A. D. Loro, M. Mingione, J. Lipsitt, C. M. Batteate, M. Jerrett, and S. Banerjee. Bayesian Hierarchical Modeling and Analysis for Actigraph Data from Wearable Devices. *The Annals of Applied Statistics*, 17(4):2865–2886, Dec. 2023. ISSN 1932-6157, 1941-7330. doi: 10.1214/23-AOAS1742. URL <https://projecteuclid.org/journals/annals-of-applied-statistics/volume-17/issue-4/Bayesian-hierarchical-modeling-and-analysis-for-actigraph-data-from-wearable/10.1214/23-AOAS1742.full>. [p9]
- C. J. Paciorek and M. J. Schervish. Spatial modelling using a new class of nonstationary covariance functions. *Environmetrics*, 17(5):483–506, 2006. doi: 10.1002/env.785. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/env.785>. [p12]
- L. Ponisio, P. de Valpine, N. Michaud, and D. Turek. One size does not fit all: Customizing MCMC methods for hierarchical models using NIMBLE. *Ecology and Evolution*, 10:2385–2416, 2020. [p1]
- K. Qu, J. R. Bradley, and X. Niu. Boundary Detection Using a Bayesian Hierarchical Model for Multiscale Spatial Data. *Technometrics*, 63(1):64–76, Jan. 2021. ISSN 0040-1706. [p1]
- R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2021. [p1]
- C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, Nov. 2005. ISBN 978-0-262-25683-4. [p1]
- M. D. Risser and D. Turek. Bayesian inference for high-dimensional nonstationary Gaussian processes. *Journal of Statistical Computation and Simulation*, 90(16):2902–2928, Nov. 2020. ISSN 0094-9655. [p1]
- M. L. Stein. *Interpolation of Spatial Data*. Springer Series in Statistics. Springer, New York, NY, 1999. [p2]
- S. Sun, J. Zhu, and X. Zhou. Statistical analysis of spatial expression patterns for spatially resolved transcriptomic studies. *Nature Methods*, 17(2):193–200, Feb. 2020. ISSN 1548-7105. Publisher: Nature Publishing Group. [p10]
- V. Svensson, S. A. Teichmann, and O. Stegle. SpatialDE: identification of spatially variable genes. *Nature Methods*, 15(5):343–346, May 2018. ISSN 1548-7105. Publisher: Nature Publishing Group. [p10]
- D. Turek, P. de Valpine, and C. J. Paciorek. Efficient Markov chain Monte Carlo sampling for hierarchical hidden Markov models. *Environmental and Ecological Statistics*, 23:549–564, 2016. [p1]

- F. Wang and A. E. Gelfand. Modeling Space and Space-Time Directional Data Using Projected Gaussian Processes. *Journal of the American Statistical Association*, 109(508):1565–1580, 2014. [p12]
- F. Wang, A. Bhattacharya, and A. E. Gelfand. Process modeling for slope and aspect with application to elevation data maps. *TEST*, 27(4):749–772, Dec. 2018. ISSN 1863-8260. [p12]
- L. M. Weber, A. Saha, A. Datta, K. D. Hansen, and S. C. Hicks. nnSVG for the scalable identification of spatially variable genes using nearest-neighbor Gaussian processes. *Nature Communications*, 14(1): 4059, July 2023. ISSN 2041-1723. Publisher: Nature Publishing Group. [p10]
- H. Wickham. ggplot2. *Wiley Interdisciplinary Reviews: Computational Statistics*, 3(2):180–185, 2011. [p2]
- W. H. Womble. Differential Systematics. *Science*, 114(2961):315–322, Sept. 1951. Publisher: American Association for the Advancement of Science. [p1]
- K. L. Wu and S. Banerjee. Assessing Spatial Disparities: A Bayesian Linear Regression Approach, Mar. 2025. [p1]

Aritra Halder

Department of Biostatistics & Epidemiology
3215 Market Street, Philadelphia, PA 19104
<https://orcid.org/0000-0002-5139-3620>
aritra.halder@drexel.edu

Sudipto Banerjee

Department of Biostatistics
University of California, Los Angeles
650 Charles E. Young Drive South, Los Angeles, CA 90095
<https://orcid.org/0000-0002-2239-208X>
sudipto@ucla.edu

exametrika: Test Data Engineering with R

by Koji Kosugi and Kojiro Shojima

Abstract The exametrika package implements statistical models for test data engineering proposed by Shojima (2022), providing a comprehensive framework for analyzing educational test data. The package offers eleven models organized into three categories: classical test theory methods (CTT, IRT), latent structure models (latent class analysis, latent rank analysis, biclustering, and Rankclustering), and advanced network models (Bayesian network model, locally dependent LRA, locally dependent biclustering, bicluster network model, and deep bicluster network model). This paper focuses on latent structure models, including latent class analysis, latent rank analysis, biclustering, and Rankclustering, which combines rank-based ordering with clustering. For advanced network models, we demonstrate the application of Bayesian network models to test data. These models are designed to capture complex structures inherent in test data and deepen understanding of examinee abilities and item characteristics. The package employs EM algorithms for efficient and accurate parameter estimation. Through detailed explanations of theoretical backgrounds, implementation methods, and practical applications to educational test data, we demonstrate the utility of the diverse analytical approaches provided by the exametrika package.

1 Introduction

Educational testing plays a crucial role in contemporary society, with significant social implications. In Japan, for instance, the National Center Test for University Admissions is taken by nearly 500,000 examinees annually, serving as a critical gateway to higher education. Standardized tests are employed across various domains: in schools, they measure educational effectiveness and student progress; in corporate settings, they assess candidate qualifications for employment; in professional certification programs, they validate specialized knowledge and skills; and in educational research, they provide data for evaluating teaching methodologies. The impact of these test results extends beyond scores, influencing educational policies, career opportunities, and professional development. This underscores the importance of developing robust and meaningful testing methodologies.

Test theory has evolved from classical test theory to modern approaches as a scientific discipline. Item response theory (IRT) has enabled detailed estimation of item characteristics and examinee abilities. While IRT provides valuable precision for many psychometric applications, this level of detail raises questions about practical significance in certain educational contexts. For example, although IRT can estimate ability parameters with high precision, the practical relevance of minute score differences (e.g., 0.01 point changes) may be limited for providing actionable feedback to learners. University grading systems typically use a 0-4 scale, which proves sufficient for most practical purposes. Moreover, such precise measurements often fail to provide actionable feedback to examinees regarding specific learning needs or improvement strategies.

Our exametrika package provides a new test data analysis method. In addition to classical test data analysis methods, the package includes latent structure models such as latent class analysis, latent rank analysis, biclustering, and Rankclustering, which combines rank-based ordering with clustering. Furthermore, it provides network models to illustrate learning roadmaps, including the Bayesian network model, locally independent latent rank model, locally independent biclustering, and bicluster network model. We developed the exametrika package based on Shojima (2022)'s theoretical framework in "Test Data Engineering", originally designed for binary data analysis, with subsequent extensions to support ordinal and nominal response formats as the models have evolved.

For latent structure models, research related to LRA includes Croon (1990)'s ordered latent class analysis (LCA), modeling of performance level setting by Torres Iribarra et al. (2015), and research on construct-level representation in psychometric models by Diakow et al. (2014). Moreover, diverse theoretical approaches exist in the field of biclustering, such as the sparse singular value decomposition approach by Lee et al. (2010), the probabilistic latent block model by Govaert and Nadif (2010), the bipartite spectral graph partitioning method by Dhillon (2001), and biclustering models for ordinal data by Matechou et al. (2016).

For network models, Bayesian networks have been widely applied across diverse fields, including engineering, marketing, behavioral sciences, and cognitive psychology (Pearl, 1988; Jensen and Nielsen, 2007; Darwiche, 2009; Koski and Noble, 2009; Scutari and Denis, 2014). In educational measurement, Almond et al. (2015) applied the model to assessment design, and Culbertson (2016) and Reichenberg (2018) provided comprehensive reviews of its applications. Network models are particularly well-suited for representing learning pathways that examinees should follow, and discovering these pathways from test data holds significant educational value.

The `exametrika` package integrates these methodological traditions into a unified system for educational measurement. While all models in the package offer valuable analytical capabilities, the Rankclustering model serves as the conceptual core, introducing an ordering constraint on examinee clusters in the probabilistic biclustering framework. The package’s ordered clustering approach and various visualization capabilities enable the provision of clear learning roadmaps for both test administrators and examinees.

Beyond mere ability estimation and item analysis, the primary aim of this package is to provide specific, actionable guidance on what each examinee should learn next. While traditional approaches focus on measurement precision, `exametrika` prioritizes practicality in educational settings, offering test administrators the ability to develop effective instructional strategies and examinees the opportunity to understand their learning progress clearly. This approach transforms test assessment from a mere evaluation tool into a powerful educational resource that promotes learning.

The remainder of this paper is organized as follows. Section 2 provides an overview of the package’s capabilities and workflow. Section 3 presents the latent structure models in detail with R code and examples. Section 4 describes the network models with practical demonstrations. Finally, Section 5 offers a discussion of the package’s contributions and future directions.

2 Package overview

The `exametrika` package implements a comprehensive suite of statistical models for test data engineering, supporting both exploratory and confirmatory approaches to educational test data analysis. Additionally, several models accommodate not only binary test data but also polytomous response formats. This section provides an overview of the package’s functions, organized into three major categories: classical measurement methods, latent structure analysis models, and advanced network approaches.

2.1 Available models

The package integrates classical psychometric approaches with modern latent structure analysis and network models, as shown in Table 1.

Table 1: Main Analysis Models in `exametrika`

Category	Model	Main Function	Data Types	Primary Outputs
Classical Methods	Classical Test Theory	CTT()	Binary	Item statistics, reliability
	Item Response Theory	IRT()	Binary	Item parameters, ability estimates
	Graded Response Model	GRM()	Ordinal	Item parameters, ability estimates
Latent Structure Analysis	Latent Class Analysis	LCA()	Binary, Ordinal, Nominal	Class membership, conditional probabilities
	Latent Rank Analysis	LRA()	Binary, Ordinal, Rated	Rank membership, item response profiles
Advanced Network Models	Biclustering / Rankclustering	Biclustering()	Binary, Ordinal, Nominal	Item & examinee clusters, response patterns
	Bayesian Network Model	BNM()	Binary	Network structure, conditional dependencies
	Latent Dependence LRA	LD_LRA()	Binary	LRA with local dependencies
	Local Dependence Biclustering	LDB()	Binary	Biclustering with local dependencies
	Biclusternet Network Model	BINET()	Binary	Integrated biclustering + network

Classical methods include basic test and item statistics, along with CTT and IRT. Test statistics refer to the distribution of total test scores, including mean, standard error, variance, standard deviation, skewness, kurtosis, minimum, maximum, range, quartile deviation, IQR, and stanine scores. Item statistics output pass rates, item odds, item thresholds, entropy, and item-total correlations. For CTT, Cronbach’s alpha coefficient and McDonald’s omega coefficient are computed. The `Dimensionality()` function outputs eigenvalues of the correlation matrix and scree plots to verify unidimensionality. IRT implements logistic models, including 2-parameter, 3-parameter, and 4-parameter models. Estimated item parameters, item-level fit indices, and overall test fit indices are output, and examinee ability parameters are estimated simultaneously. Furthermore, the `plot()` function can draw item characteristic curves (ICC), test characteristic curves (TCC), and information functions (IIF, TIF). The GRM model works similarly for ordinal data.

Latent structure analysis includes four models: latent class analysis (LCA), latent rank analysis (LRA), biclustering, and Rankclustering. Unlike IRT, LCA does not estimate continuous examinee parameters but classifies examinees into latent classes based solely on observed response patterns. LRA extends LCA by assuming ordinality among the latent classes. Both models output expected correct response rates for each class/rank as their primary output, allowing users to understand the characteristics of each class/rank. Item-level model fit indices, overall test fit indices, and examinees’ class/rank membership probabilities are also available. Visualization of these results is possible with the `plot()` function. Biclustering simultaneously clusters both examinees and items. Item clusters are called fields, and examinee clusters are called classes. A model that assumes ordinality for classes

is specifically called Rankclustering. The primary output is expected correct response rates for each class/rank by field, allowing users to understand which class of examinees tends to score higher in which field (item domain). It also reveals which fields an examinee at a given rank should strengthen to advance to the next rank.

Network models represent the correlation structure among items as a network. Specifically, connections between items are represented as a DAG (Directed Acyclic Graph), revealing influence relationships among items. Based on the given network structure, conditional correct response rates are estimated, and item-level fit indices and overall test fit indices are available. By analyzing these conditional correct response rates by LRA rank or by Rankclustering field-rank combinations rather than by individual items, learning pathways for each rank can be revealed. The BINET model, which combines biclustering and network models, provides a learning roadmap showing which fields examinees at each rank should strengthen to advance to higher ranks.

For latent structure analysis and advanced network models, the number of latent classes/ranks, fields, or the DAG structure must be specified externally. However, auxiliary functions for exploring these structures also exist, as shown in Table 2.

Table 2: Structure Exploration Tools

Category	Function	Description
Latent Structure Analysis	GridSearch()	Explore optimal class/rank and field numbers for LCA, LRA, Biclustering
	Biclustering_IRM()	Explore Rankclustering structure via Infinite Relational Model
Advanced Network Models	BNM_GA()	Learn BNM network structure via Genetic Algorithm
	BNM_PBIL()	Learn BNM network structure via PBIL
	LDLRA_PBIL()	Learn LD-LRA network structure via PBIL

All models output item-level and overall model fit indices through unified functions. From the model likelihood, null model likelihood, and saturated model likelihood, model chi-square and null model chi-square are calculated, which are then transformed to output NFI, RFI, IFI, TLI, CFI, RMSEA, AIC, CAIC, and BIC. For latent structure models, the GridSearch() function exhaustively searches for optimal numbers of ranks/classes and fields based on these fit indices. Rankclustering can also search for the optimal number of ranks using the Infinite Relational Model via Biclustering_IRM(). For advanced network models, functions are provided to search for DAG structures based on genetic algorithms. Using these auxiliary functions, users can find optimal model structures and perform more accurate analyses. However, exploratory structure approaches often yield results that are difficult to interpret in practice, and in the domain of educational test data, practitioners typically have some prior knowledge about inter-item and inter-field structures. Therefore, it is recommended to construct DAGs utilizing expert domain knowledge rather than relying solely on data-driven exploration.

2.2 Data preparation and workflow

The dataFormat() function serves as the primary data gateway for all analyses in exametrika. It takes raw data matrices as input, validates the data, and produces a standardized exametrika class object. The function signature is:

```
dataFormat(data, na = NULL, id = 1, Z = NULL, w = NULL,
           response.type = NULL, CA = NULL)
```

The na argument specifies values to treat as missing, id indicates the column containing examinee IDs (default is 1), Z is an optional missing indicator matrix, and w is an item weight vector. The response.type argument specifies the data type: “binary” for dichotomous responses, “ordinal” for ordered polytomous data, “nominal” for unordered categories, or “rated” for polytomous data with correct answers (requiring the CA argument). If response.type is NULL (default), the type is automatically detected: data containing only 0/1 values is classified as binary, data with three or more ordered categories as ordinal, and data with a CA specification as rated.

The function also handles labeling automatically. For examinee IDs, if the first column contains character strings or factor values, it is treated as the ID column; otherwise, row names are used if available, or sequential IDs (“Student1”, “Student2”, ...) are generated. Item labels are taken from column names, or generated as “Item1”, “Item2”, ... if not provided. For polytomous data, category labels are preserved from factor levels if the input uses factors; otherwise, they are generated automatically based on the observed values. Most models assume binary data, while latent structure models (LCA, LRA, and Biclustering) accommodate polytomous responses.

Figure 1 presents the typical data analysis workflow. The workflow begins with raw data preparation using dataFormat(), which creates exametrika class objects. Users then select an appropriate

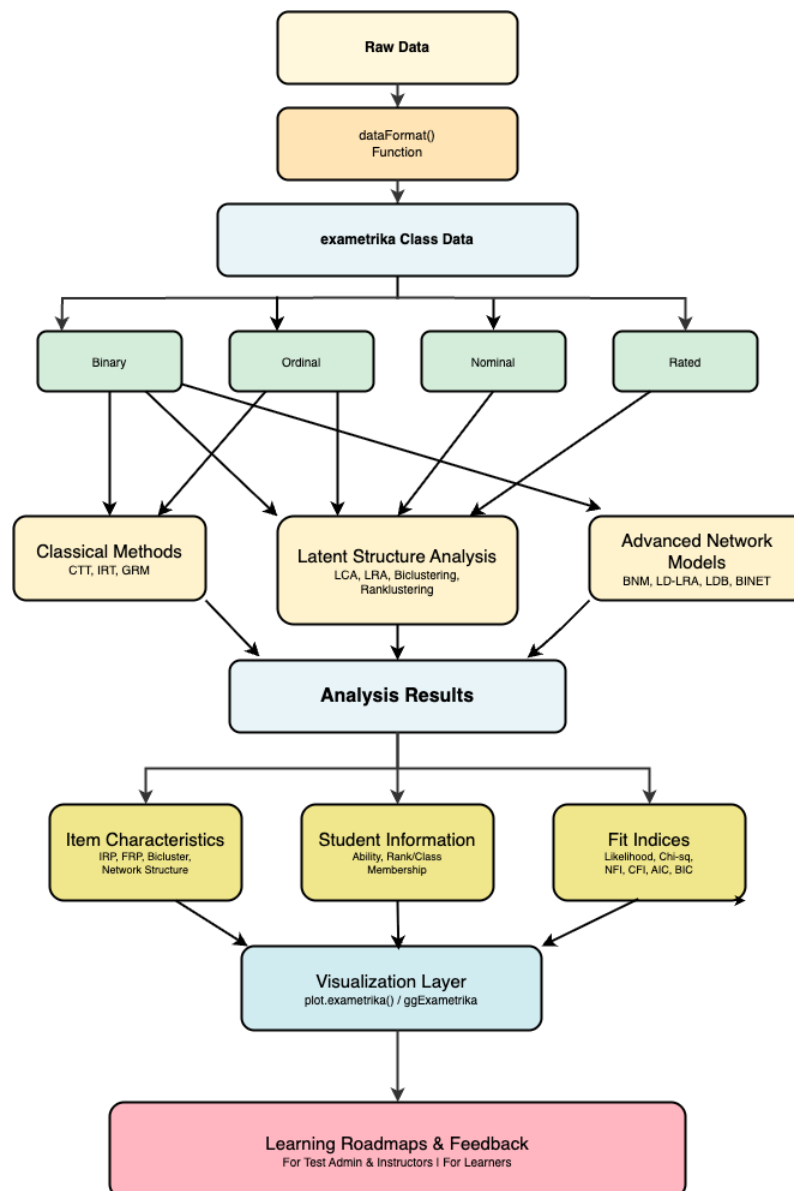


Figure 1: Data analysis workflow in exametrika.

main analysis model based on their research questions. Optional structure exploration tools can automatically discover optimal model parameters. The resulting analysis objects contain membership probabilities, response profiles, and fit indices, which are visualized and interpreted to generate learning roadmaps.

For visualization, the package inherits the base R `plot()` function, providing standard diagnostic plots for every model. Standard graphical parameters can be supplied through the `...` argument of `plot()` and are forwarded consistently to all plot types, including to the manually drawn axes. Users can therefore customize point symbols (`pch`), axis-label orientation (`las`), text sizes (`cex`), colors, and line types, and can override the default axis labels and titles. For example, `plot(result.LRA, type = "IRP", items = 1:4, nc = 2, nr = 2, las = 2, pch = 16)` rotates the axis labels and changes the plotting symbol. In addition, the package returns all the information needed for visualization within its result objects, so users who prefer a `ggplot2`-based grammar can build fully custom plots; the companion `ggExametrika` package, available on CRAN, provides `ggplot2`-based renderings of these visualizations. This design keeps the computational core lightweight while still offering the customization flexibility expected of base R graphics.

3 Latent structure models

This section presents the latent structure models implemented in the *exametrika* package. These models classify examinees into discrete ability groups rather than estimating continuous ability parameters, providing more interpretable results for educational practice.

3.1 Latent class analysis and latent rank analysis

Model

Latent class analysis (LCA) and latent rank analysis (LRA) both handle examinee ability as discrete classes. However, they differ significantly in their approach to ability scales: LCA handles ability as a nominal scale, whereas LRA handles it as an ordinal scale. In LCA, classes have no inherent order and represent different ability patterns. In contrast, LRA establishes a clear ordinal relationship between classes (ranks), where higher ranks indicate higher ability levels.

We position LRA as a nonparametric IRT model because it does not define the relationship between ability and response probability using specific mathematical functions (such as logistic or normal distribution functions). Traditional IRT models estimate these functions' parameters (e.g., discrimination and difficulty), whereas LRA directly estimates the probability of correct responses for items at each rank. This characteristic allows LRA to flexibly adapt to data characteristics without requiring assumptions about specific functional forms.

This nonparametric nature has important practical implications. The extremely fine-grained measurement provided by traditional IRT models' continuous ability scales may not always be practical. For instance, in a 100-point test, it is uncommon to have precisely one examinee at each possible score point. Similarly, with binary scoring, the observed response patterns typically do not cover all possible combinations (the total number of possible patterns being 2 to the power of the number of items). Fine-grained measurement on a continuous ability scale may lack substantive meaning in such situations.

Such fine resolution is often unnecessary, particularly in diagnostic tests. Instead, it is important to clearly indicate which items are achievable at each ability level and utilize this information as a learning roadmap. Continuous scores can be inconvenient from the perspectives of feedback to examinees and practical application in educational settings. LRA addresses these practical needs by providing ordered ability levels, enabling more practical educational assessment.

Furthermore, LRA's item reference profile (IRP) reveals the probability of correct responses for each rank, providing concrete information about which items examinees at each ability level can answer correctly. This approach offers a more intuitive understanding of learning achievement, rather than only numerical scores. Test administrators can utilize these ordered ability levels to develop teaching plans and provide specific learning advice to examinees.

Let us define the basic notations. Let J be the number of items, S be the number of examinees, and R be the number of ranks. The observed data matrix $\mathbf{U}$ can be defined as follows:

$$\mathbf{U} = \{u_{sj}\}, \quad u_{sj} \in \{0, 1\},$$

where u_{sj} represents the binary response of examinee s to item j (1 for correct, 0 for incorrect). We also define a missing value indicator matrix $\mathbf{Z}$ of the same size as $\mathbf{U}$:

$$\mathbf{Z} = \{z_{sj}\}, \quad z_{sj} \in \{0, 1\},$$

where $z_{sj} = 1$ indicates that the response of examinee s to item j is observed, and $z_{sj} = 0$ indicates a missing value.

The rank reference matrix $\mathbf{\Pi}_R$ can be defined as follows:

$$\mathbf{\Pi}_R = \begin{bmatrix} \pi_{11} & \cdots & \pi_{1R} \\ \vdots & \ddots & \vdots \\ \pi_{J1} & \cdots & \pi_{JR} \end{bmatrix} = \{\pi_{jr}\},$$

where each element π_{jr} ($0 \leq \pi_{jr} \leq 1$) represents the probability of correct response to item j at rank r .

The rank membership matrix $\mathbf{M}_R$ is defined as:

$$\mathbf{M}_R = \begin{bmatrix} m_{11} & \cdots & m_{1R} \\ \vdots & \ddots & \vdots \\ m_{S1} & \cdots & m_{SR} \end{bmatrix},$$

where each row $\mathbf{m}_s$ represents the rank membership profile of examinee s , satisfying $\mathbf{1}'_R \mathbf{m}_s = 1$.

Given these definitions, the likelihood of the observed data is:

$$l(\mathbf{U} | \boldsymbol{\Pi}) = \prod_{s=1}^S \prod_{j=1}^J \prod_{r=1}^R \left\{ \left(\pi_{jr} \right)^{u_{sj}} \left(1 - \pi_{jr} \right)^{1-u_{sj}} \right\}^{z_{sj}}$$

The EM algorithm maximizes this likelihood to estimate the model parameters.

Algorithm

The algorithm involves three key matrices:

- $\mathbf{M}_R$: The rank membership matrix representing the probability of each examinee belonging to each rank.
- $\mathbf{F}$: The filter matrix that imposes ordinal structure through smoothing.
- $\mathbf{S}$: The smoothed rank membership matrix, obtained by applying $\mathbf{F}$ to $\mathbf{M}_R$.

The relationship between these matrices is defined as $\mathbf{S} = \mathbf{M}_R \mathbf{F}$.

The EM algorithm iteratively updates these matrices through the following cycle:

1. **Initialization**: Set initial values for $\boldsymbol{\Pi}_R^{(0)}$.
2. **Repeat until convergence**:
 - 2-1. **E-step**: Calculate the rank membership matrix $\mathbf{M}_R^{(t)}$ using the current parameter estimates.
 - 2-2. **Smoothing**: Obtain the smoothed rank membership matrix $\mathbf{S}^{(t)}$ by applying the filter matrix $\mathbf{F}$ to $\mathbf{M}_R^{(t)}$.
 - 2-3. **M-step**: Update the rank reference matrix $\boldsymbol{\Pi}_R^{(t)}$ using $\mathbf{S}^{(t)}$.
 - 2-4. **Ordering**: For each item j , sort $\pi_j^{(t)}$ to maintain the ordinal constraint.

The smoothing process is crucial for maintaining the ordinal structure of ranks. The filter matrix $\mathbf{F}$ assigns weights primarily to the current rank and slightly to the L ranks before and after, ensuring a stepwise progression. Our method constructs the filter matrix using a kernel of size $2L + 1$ that gradually decreases from the center f_0 to L ranks on both sides:

$$\mathbf{F}^* = \begin{bmatrix} f_0 & \cdots & f_L & & & \\ \vdots & \ddots & \vdots & \ddots & & \\ f_L & \ddots & f_0 & \ddots & f_L & \\ & \ddots & \vdots & \ddots & \vdots & \ddots \\ & & f_L & \ddots & f_0 & \ddots & f_L \\ & & & \ddots & \vdots & \ddots & \vdots \\ & & & & f_L & \cdots & f_0 \end{bmatrix},$$

$$\mathbf{F} = \mathbf{F}^* \oslash \{ \mathbf{1}_R (\mathbf{1}'_R \mathbf{F}^*) \}$$

where $\oslash$ represents element-wise division.

In this package, we used a kernel $\mathbf{f}$ of size $L = 3$, with its magnitude adjusted according to the number of ranks as follows:

$$f_0 = \begin{cases} 1.05 - 0.05R & (1 \leq R \leq 5) \\ 1.00 - 0.04R & (5 < R \leq 10) \\ 0.80 - 0.02R & (10 < R \leq 20) \end{cases}$$

These coefficient values are defined in Shojima (2022) based on practical experience and empirical analysis rather than theoretical derivation. The smoothing strength is adjusted according to the number of ranks: fewer ranks require stronger smoothing to maintain clear ordinal distinctions, while more ranks benefit from weaker smoothing to allow finer differentiation between adjacent ranks. Without this smoothing process, the model becomes latent class analysis due to the absence of ordinal constraints.

In the M-step, we update $\Pi_R^{(t)}$. The rank reference matrix is updated as follows:

$$\pi_{jr}^{(t)} = \frac{S_{1jr}^{(t)} + \beta_1 - 1}{S_{1jr}^{(t)} + S_{0jr}^{(t)} + \beta_1 + \beta_0 - 2}$$

Here, we assume a beta distribution $B(\beta_0, \beta_1)$ as the prior distribution for $\pi_{jr}^{(t)}$. Furthermore, $S_{1jr}^{(t)}$ and $S_{0jr}^{(t)}$ are elements of the $J \times R$ matrices $\mathbf{S}_1$ and $\mathbf{S}_0$:

$$\mathbf{S}_1^{(t)} = (\mathbf{Z} \odot \mathbf{U})' \mathbf{S}^{(t)}, \quad \mathbf{S}_0^{(t)} = \left\{ \mathbf{Z} \odot (\mathbf{1}_S \mathbf{1}_J' - \mathbf{U}) \right\}' \mathbf{S}^{(t)}$$

By default, the package uses $B(\beta_0, \beta_1) = (1, 1)$, which corresponds to a uniform (noninformative) prior distribution. This design choice reflects a deliberate philosophy: noninformative priors allow data characteristics to speak for themselves without imposing external assumptions, supporting exploratory analysis of test data patterns. With this default, the MAP estimate and MLE become identical, yielding $\pi_{jr}^{(t)} = S_{1jr}^{(t)} / (S_{0jr}^{(t)} + S_{1jr}^{(t)})$. For users who wish to incorporate prior knowledge or regularization, alternative priors can be specified via the `beta1` and `beta2` arguments in the `LRA()` function.

The convergence criterion checks whether the change in expected log-likelihood is less than a threshold value $c = 10^{-4}$.

Example

The `exametrika` package enables users to perform this analysis by specifying a test dataset and the number of ranks. Here, we demonstrate an example using the binary sample data `J15S500`. This dataset has 15 items and 500 samples in binary format. The `LRA` function runs when users specify the number of ranks.

The returned object contains the IRP, representing the relationship between items and ranks as the transpose of Π_R .

```
result.LRA <- LRA(J15S500, nrank = 6)
result.LRA$IRP
```

```
#>      IRP1      IRP2      IRP3      IRP4      IRP5      IRP6
#> Item01 0.58505704 0.63186539 0.7080067 0.7866549 0.8530561 0.8977529
#> Item02 0.52472724 0.62904525 0.7554820 0.8454942 0.8829038 0.8750385
#> Item03 0.61342178 0.60950806 0.7082092 0.7726317 0.8009246 0.8386955
#> Item04 0.44061750 0.60725617 0.7937016 0.8821678 0.9394150 0.9763276
#> Item05 0.64652323 0.74523579 0.8214571 0.8366280 0.8623113 0.9052983
#> Item06 0.64708578 0.77479139 0.9109422 0.9674623 0.9633072 0.9154799
#> Item07 0.40904237 0.51767342 0.7204344 0.8402237 0.8898201 0.9002571
#> Item08 0.33751077 0.42924464 0.6023919 0.7134688 0.7346823 0.6979226
#> Item09 0.35228771 0.31985180 0.2977586 0.2822659 0.3773630 0.5419243
#> Item10 0.49961957 0.57931903 0.6861960 0.7290801 0.7166365 0.7531038
#> Item11 0.09580910 0.07930581 0.1357061 0.2856802 0.4723506 0.6172879
#> Item12 0.06478922 0.09822972 0.1556541 0.2394945 0.4213656 0.6359273
#> Item13 0.29076602 0.48420870 0.7152043 0.7733618 0.7499489 0.7784993
#> Item14 0.48350628 0.59491451 0.7285399 0.8490429 0.9327099 0.9766344
#> Item15 0.39812723 0.57448159 0.7561590 0.8270643 0.8354638 0.8342372
```

The IRP (Item Reference Profile) is a $\text{rank} \times \text{item}$ matrix showing the expected probability of correct response for each item at each rank. Each row represents a rank (from lowest to highest ability), and each column represents an item. Values should increase monotonically from lower to higher ranks, reflecting that higher-ability examinees are more likely to answer correctly. From the IRP, test administrators can identify two key item characteristics: (1) the rank where an item's correct response

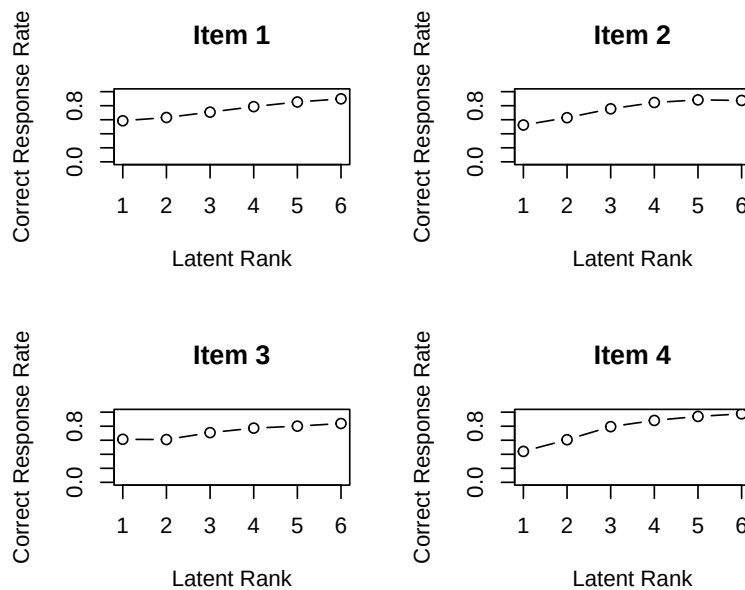


Figure 2: IRP plots

probability is closest to 0.5 indicates the ability level at which the item is most informative (analogous to the location parameter in IRT), and (2) the maximum change in correct response probability between adjacent ranks indicates how well an item discriminates between ability levels (analogous to the discrimination parameter in IRT).

Additionally, the package calculates the model's fit indices using a saturated model where the number of ranks equals the number of unique response patterns and a null model where all examinees are assigned to a single rank.

```
result.LRA$TestFitIndices
```

```
#>  model_log_like bench_log_like null_log_like model_Chi_sq null_Chi_sq model_df
#> 1      -3857.982      -3560.005      -4350.217      595.9544      1580.424      138.4909
#>  null_df      NFI      RFI      IFI      TLI      CFI      RMSEA      AIC
#> 1      195 0.6229148 0.469051 0.6827429 0.5350704 0.6698025 0.0813612 318.9726
#>      CAIC      BIC
#> 1 -403.2032 -264.7123
```

We determined the appropriate number of ranks for the administered test by referring to these fit indices. For each examinee, a rank membership profile (**S**) is also output, and the system identifies the rank with the highest probability for each individual as an estimated value. The package presents the ease of rank movement as rank-up and rank-down odds.

The plots provided by this package offer rich information for test administrators. The IRP plot shows the expected correct response probability for each item across ranks as line graphs, which should be monotonically increasing.

```
plot(result.LRA, type = "IRP", items = 1:4, nc = 2, nr = 2)
```

The RMP (Rank Membership Profile) represents each examinee's probability of belonging to each rank. For each examinee, the profile shows a probability distribution across all ranks, with probabilities summing to 1. Examinees with probability concentrated in a single rank have clear ability classification, while those with probability spread across adjacent ranks (e.g., ranks 3 and 4) are in transitional stages between ability levels. The rank with the highest probability becomes the estimated rank for that examinee.

```
plot(result.LRA, type = "RMP", students = 1:4, nc = 2, nr = 2)
```

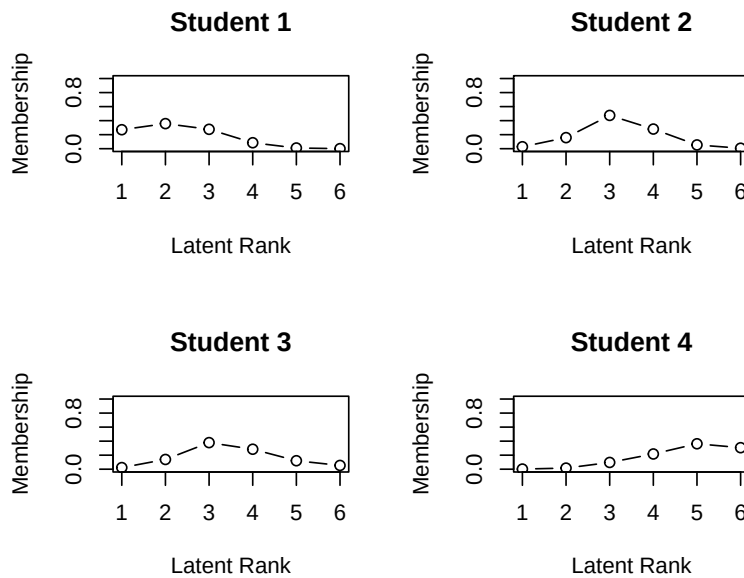


Figure 3: RMP plots

3.2 Biclustering and rankclustering

Model

LRA focuses on classifying examinees; however, certain cases exist where we prefer to classify items from the test administrator's perspective, such as by content area or difficulty level.

Biclustering is a method that simultaneously clusters the data matrix (examinees $\times$ items) in both horizontal (examinees) and vertical (items) directions. Examinees are classified into latent classes (nominal clusters), whereas items are classified into latent fields (nominal clusters). LCA performs one-way clustering of examinees, latent field analysis performs one-way clustering of items, whereas biclustering performs two-way simultaneous clustering (two-mode clustering).

In biclustering, examinees are classified into latent classes (nominal clusters). However, applying a filter matrix similar to LRA to this class membership profile can transform them into latent ranks with an ordinal structure. This method is called "Rankclustering," a term that reflects its unique approach to rank-based clustering. Rankclustering can be achieved by simply adding one step of smoothing the class membership profile to the biclustering estimation procedure.

To formalize the biclustering model, let C be the number of latent classes/ranks and F be the number of latent fields. The bicluster reference matrix Π_B is defined as follows:

$$\Pi_B = \begin{bmatrix} \pi_{11} & \cdots & \pi_{1F} \\ \vdots & \ddots & \vdots \\ \pi_{C1} & \cdots & \pi_{CF} \end{bmatrix} = \{\pi_{fc}\},$$

where each element π_{fc} represents the probability of correct responses from examinees in class/rank c to items in field f .

The class membership matrix $\mathbf{M}_C$ and field membership matrix $\mathbf{M}_F$ are defined as follows:

$$\mathbf{M}_C = \begin{bmatrix} m_{11} & \cdots & m_{1C} \\ \vdots & \ddots & \vdots \\ m_{S1} & \cdots & m_{SC} \end{bmatrix}, \quad \mathbf{M}_F = \begin{bmatrix} m_{11} & \cdots & m_{1F} \\ \vdots & \ddots & \vdots \\ m_{J1} & \cdots & m_{JF} \end{bmatrix}.$$

For rankclustering, the rank membership matrix $\mathbf{M}_R$ is obtained by filtering the class membership matrix: $\mathbf{M}_R = \mathbf{M}_C \mathbf{F}$.

The likelihood function is:

$$l(\mathbf{U} \mid \Pi_B) = \prod_{s=1}^S \prod_{j=1}^J \prod_{f=1}^F \prod_{c=1}^C \left(\pi_{fc}^{u_{sj}} (1 - \pi_{fc})^{1-u_{sj}} \right)^{z_{sj} m_{sc} m_{jf}}$$

Algorithm

The biclustering algorithm uses the EM algorithm to estimate three key matrices. The estimation procedure proceeds as follows:

1. **Initialization:** Set initial values for $\mathbf{M}_F^{(0)}$ and $\Pi_B^{(0)}$
2. **Iteration:**
 - **E-step:** Estimate $\mathbf{M}_C^{(t)}$ from $\mathbf{U}$, $\Pi_B^{(t-1)}$, and $\mathbf{M}_F^{(t-1)}$
 - **Smoothing:** For rankclustering, transform $\mathbf{M}_C^{(t)}$ into $\mathbf{M}_R^{(t)} = \mathbf{M}_C^{(t)} \mathbf{F}$
 - **E-step:** Estimate $\mathbf{M}_F^{(t)}$ from $\mathbf{U}$, $\Pi_B^{(t-1)}$, and $\mathbf{M}_R^{(t)}$
 - **M-step:** Estimate $\Pi_B^{(t)}$ from $\mathbf{U}$, $\mathbf{M}_R^{(t)}$, and $\mathbf{M}_F^{(t)}$
 - Check convergence criterion

The E-step calculates class membership probability for examinee s :

$$m_{sc}^{(t)} = \frac{l(\mathbf{u}_s | \pi_c^{(t-1)}) \pi_c}{\sum_{c'=1}^C l(\mathbf{u}_s | \pi_{c'}^{(t-1)}) \pi_{c'}}$$

where π_c represents the prior distribution (uniform in the package). Similarly, field membership probability for item j :

$$m_{jf}^{(t)} = \frac{l(\mathbf{u}_j | \pi_f^{(t-1)}) \pi_f}{\sum_{f'=1}^F l(\mathbf{u}_j | \pi_{f'}^{(t-1)}) \pi_{f'}}$$

In the M-step, we update π_{fc} using a beta distribution as the conjugate prior:

$$\pi_{fc}^{(t)} = \frac{U_{1fc}^{(t)} + \beta_1 - 1}{U_{0fc}^{(t)} + U_{1fc}^{(t)} + \beta_0 + \beta_1 - 2}$$

Because we set $B(\beta_0, \beta_1) = (1, 1)$ as the prior distribution, MAP and MLE become identical. Users can specify alternative priors via the `beta1` and `beta2` arguments. The convergence criterion is that the change in expected log-likelihood be less than $c = 10^{-4}$.

For further details on the mathematical formulation and implementation, see [Shojima \(2022\)](#).

Example

The `exametrika` package enables users to perform biclustering with minimal specification. We only need to specify the data, the number of fields, and the number of classes. By setting the `method` argument to `R`, rankclustering is performed; by setting it to `B`, standard biclustering is performed.

```
result.Rankclustering <- Biclustering(J35S515, nflid = 5, ncls = 6, method = "R")
```

The array plot is an important tool for visually evaluating the results. In this plot, correct responses (1) are displayed in black and incorrect responses (0) in white. The left panel shows the original response data, while the right panel shows the data reorganized by rankclustering.

```
plot(result.Rankclustering, type = "Array")
```

In the right panel, examinees are sorted by estimated rank (lower ranks at the top, higher ranks at the bottom) and items are arranged by estimated field. The ordinal structure of ranks is visually confirmed by the decreasing black areas from bottom to top, and the field arrangement reveals difficulty differences across fields.

```
result.Rankclustering$TestFitIndices
```

```
#>  model_log_like bench_log_like null_log_like model_Chi_sq null_Chi_sq model_df
#> 1    -7273.063    -5891.314    -9862.114    2763.498    7941.601 1166.164
#>  null_df      NFI      RFI      IFI      TLI      CFI      RMSEA      AIC
#> 1    1155 0.6520226 0.6553538 0.7642464 0.7668873 0.7646342 0.05162221 431.1703
#>      CAIC      BIC
#> 1 -5684.386 -4518.223
```



Figure 4: Array plot comparing original response patterns (left) and reorganized patterns after rankclustering (right). Black cells indicate correct responses and white cells indicate incorrect responses.

Π_B represents the biclustering result and is output as the field-rank profile (FRP), which shows the correspondence between fields and ranks. The FRP is a rank $\times$ field matrix where each cell value π_{fc} represents the expected probability that an examinee belonging to rank c will correctly answer items belonging to field f . When examined row-wise (by rank), a pattern of increasing correct response rates as ranks rise confirms that the ordinal structure is functioning properly. When examined column-wise (by field), differences in difficulty across fields become apparent—for example, if a particular field shows low correct response rates across all ranks, this indicates that the field consists of generally difficult items. Test administrators can use the FRP to make interpretations such as “examinees at rank 3 have nearly mastered fields 1 and 2, but fields 4 and 5 remain challenging,” forming the foundation for creating learning roadmaps.

```
result.Rankclustering$FRP
```

```
#>           Rank1      Rank2      Rank3      Rank4      Rank5      Rank6
#> Field1 0.64951141 0.79197342 0.88093148 0.9140316 0.9381373 0.9701273
#> Field2 0.09362918 0.27752126 0.61746787 0.9036746 0.9835653 0.9979939
#> Field3 0.22472428 0.32079306 0.44027355 0.6163989 0.7507110 0.8988009
#> Field4 0.10394873 0.18598451 0.28100007 0.3614656 0.6030432 0.8411943
#> Field5 0.03293073 0.05469207 0.09344104 0.1298351 0.2618470 0.5980969
```

The field membership matrix $\mathbf{M}_F$ shows the relationship between items and fields. This is an item $\times$ field probability matrix indicating how likely each item belongs to each field. For each row (item), the field with the highest probability becomes that item’s estimated field assignment. Items with probability concentrated in a single field have clear membership, while items with probability distributed across multiple fields have boundary characteristics and may relate to multiple content areas. Test administrators can reference the FieldMembership to confirm item groupings and assign meaningful names to fields based on domain knowledge (e.g., “basic calculations,” “applied problems,” “word problems”), enabling specific feedback to examinees.

```
result.Rankclustering$FieldMembership
```

```
#>           Field1      Field2      Field3      Field4      Field5
```



```
#> Item01 1.000000e+00 2.28832e-86 5.706565e-83 2.091043e-165 0.000000e+00
#> Item02 7.749584e-123 7.678734e-34 4.502569e-03 9.954974e-01 5.965152e-51
#> Item03 1.124141e-90 6.575315e-41 1.000000e+00 4.767645e-11 3.471920e-80
#> Item04 2.959054e-177 2.415474e-81 3.602966e-19 1.000000e+00 1.255540e-23
#> Item05 4.544645e-208 5.303588e-111 1.309392e-28 9.999999e-01 9.276257e-08
#> Item06 2.888383e-232 2.320900e-132 3.903128e-38 2.703468e-04 9.997297e-01
#> Item07 6.756857e-40 2.189049e-40 1.000000e+00 5.920375e-33 2.217949e-135
#> Item08 1.392498e-142 1.206256e-108 5.383242e-09 1.000000e+00 2.034326e-36
#> Item09 1.936847e-121 4.171067e-67 2.413280e-02 9.758672e-01 4.765915e-49
#> Item10 7.597617e-142 1.803765e-85 6.852640e-09 1.000000e+00 4.816190e-38
#> Item11 8.692776e-96 1.000000e+00 2.833556e-10 1.163245e-23 1.575095e-99
#> Item12 6.561729e-156 1.891926e-59 4.591764e-12 1.000000e+00 8.419090e-35
#> Item13 2.746779e-217 1.535184e-121 6.388788e-31 9.974116e-01 2.588399e-03
#> Item14 1.846552e-312 2.006191e-210 5.068314e-81 6.012586e-32 1.000000e+00
#> Item15 0.000000e+00 5.660785e-252 4.334981e-99 8.988673e-44 1.000000e+00
#> Item16 1.683654e-234 8.377009e-131 1.770654e-38 1.176466e-04 9.998824e-01
#> Item17 6.259979e-193 6.735073e-109 1.800244e-23 1.000000e+00 1.036304e-14
#> Item18 0.000000e+00 2.808356e-303 2.383790e-116 1.265126e-55 1.000000e+00
#> Item19 0.000000e+00 6.688295e-300 1.702523e-114 2.206307e-54 1.000000e+00
#> Item20 0.000000e+00 1.055875e-296 1.366911e-113 9.864649e-54 1.000000e+00
#> Item21 1.254745e-52 1.000000e+00 1.394255e-31 2.281997e-70 3.228828e-188
#> Item22 1.147398e-63 1.000000e+00 2.856247e-29 6.567723e-63 4.205833e-172
#> Item23 1.043217e-49 1.000000e+00 1.176340e-19 1.392666e-56 5.231867e-169
#> Item24 3.387781e-69 1.000000e+00 3.323447e-24 3.656098e-55 4.950092e-158
#> Item25 2.295422e-101 1.000000e+00 1.008429e-21 4.659872e-39 1.375879e-119
#> Item26 2.532972e-115 1.000000e+00 3.360541e-17 7.719020e-28 1.144821e-96
#> Item27 5.079939e-115 1.457272e-08 9.994785e-01 5.214970e-04 1.167602e-60
#> Item28 1.395343e-229 5.851348e-109 1.235318e-34 1.707549e-02 9.829245e-01
#> Item29 1.923332e-222 2.552749e-122 2.118045e-32 2.728950e-01 7.271050e-01
#> Item30 0.000000e+00 6.880478e-264 7.518294e-99 2.398740e-44 1.000000e+00
#> Item31 1.000000e+00 7.607062e-77 6.876091e-66 1.708284e-141 5.869369e-315
#> Item32 1.000000e+00 3.763630e-90 7.260497e-65 1.088366e-139 2.603302e-311
#> Item33 4.213879e-98 3.789204e-72 1.000000e+00 2.827306e-08 3.418544e-70
#> Item34 1.121011e-213 1.034039e-171 3.812205e-31 9.368479e-02 9.063152e-01
#> Item35 9.250487e-280 3.735999e-217 1.772960e-65 2.911130e-23 1.000000e+00
```

By combining the FRP and FieldMembership information, test administrators can create a can-do chart showing “which areas and items need to be mastered to advance to the next rank,” providing learners with a concrete roadmap for learning. When we output the rank membership matrix **S** for examinees, the rank-up odds and rank-down odds are also provided, indicating how close each examinee is to advancing or declining in rank.

```
head(result.Rankclustering$Students)
```

```
#>      Membership 1 Membership 2 Membership 3 Membership 4 Membership 5
#> Student001 7.306275e-01 2.676161e-01 0.001756384 2.530204e-08 8.959856e-16
#> Student002 1.013677e-11 1.822511e-06 0.007688111 3.203504e-01 6.716934e-01
#> Student003 6.132982e-01 3.822317e-01 0.004470062 9.297209e-08 7.848836e-15
#> Student004 4.105535e-05 2.965646e-02 0.756516510 2.134293e-01 3.566324e-04
#> Student005 7.582381e-01 2.400191e-01 0.001742819 3.503843e-08 4.617900e-16
#> Student006 2.567149e-01 6.901288e-01 0.053138690 1.764034e-05 5.055875e-12
#>      Membership 6 Estimate Rank-Up Odds Rank-Down Odds
#> Student001 1.370827e-28      1 0.3662826041      NA
#> Student002 2.662291e-04      5 0.0003963551      0.47692958
#> Student003 5.044221e-27      1 0.6232395801      NA
#> Student004 2.677414e-11      3 0.2821211892      0.03920134
#> Student005 3.675098e-29      1 0.3165484318      NA
#> Student006 9.433808e-24      2 0.0769982206      0.37198109
```

4 Advanced network models

4.1 Model

The probability that an examinee who correctly answers item A will also correctly answer item B can be expressed as a conditional probability. By arranging items in order of their pass rates, the learning pathway becomes visible. Bayesian network models, which represent these conditional probabilities as directed acyclic graphs (DAGs), can be applied to test data analysis.

The *exametrika* package implements several network-based models with different structural assumptions. The basic Bayesian Network Model (BNM) represents conditional probabilities between items in a network format; given an externally specified DAG structure, the model estimates conditional correct response rates based on the graph.

Building on this foundation, Latent Dependence LRA (LD-LRA) and Local Dependence Biclustering (LDB) combine latent structure analysis with Bayesian network modeling. In these models, nodes represent items, and a separate network structure is specified for each latent rank—reflecting the idea that item dependencies may differ across developmental stages. For example, in early learning stages, mastering one concept may have strong dependencies on other foundational concepts, while in advanced stages, items may become more independent as learners have integrated the material.

The Bicluster Network Model (BINET) takes a fundamentally different approach: instead of items as nodes, BINET uses latent classes (ranks) as nodes. The network structure is specified for each field (locus), representing how examinees transition between latent classes within content areas. BINET requires two inputs: (1) a field configuration vector (*conf*) specifying which field each item belongs to, and (2) an adjacency file or list defining parent-child relationships among classes at each field. This configuration is typically provided via a CSV file containing three columns: parent class, child class, and the field where the relationship holds.

These network models involve complex structures, and at present they support only binary data (extension to polytomous data remains a future development goal). The package documentation ([Kosugi, 2023](#)) provides detailed examples for LD-LRA, LDB, and BINET, including sample adjacency file formats. This paper presents only a basic example of BNM execution; readers interested in the more advanced models are referred to [Shojima \(2022\)](#) and the package documentation.

4.2 Example

The following example demonstrates BNM execution using the sample data J5S10, which contains 5 items and 10 examinees.

First, we define the DAG structure as an edge list specifying parent-child relationships between items. In this example, Item01 is the root node, Item02 has Item01 as its parent, Items 03 and 04 each have Item02 as their parent, and Item05 has both Item03 and Item04 as parents.

From these node relationships, we construct the Parent Item Response Pattern (PIRP). The PIRP represents all possible combinations of parent node responses. For an item with k parent nodes, there are 2^k possible patterns. In this example, Item01 is the root with no parents (1 pattern), Items 02, 03, and 04 each have one parent (2 patterns each), and Item05 has two parents (4 patterns).

```
DAG <- matrix(c("Item01", "Item02",
               "Item02", "Item03",
               "Item02", "Item04",
               "Item03", "Item05",
               "Item04", "Item05"), ncol = 2, byrow = TRUE)
g <- igraph::graph_from_data_frame(DAG)
adj_mat <- as.matrix(igraph::get.adjacency(g))
```

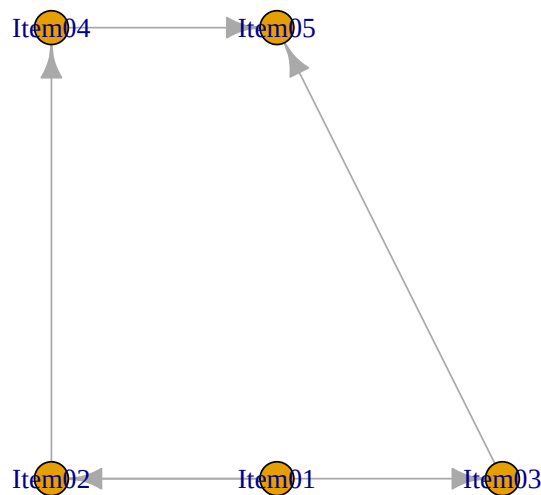
The code above constructs the DAG structure using the *igraph* package. The edge list is defined as a matrix specifying parent-child pairs, then converted to a graph object and finally to an adjacency matrix. Alternatively, the DAG structure can be read from a CSV file containing the edge list, which is convenient for larger networks.

With the adjacency matrix defined, we can now fit the Bayesian Network Model:

```
result.BNM <- BNM(J5S10, adj_matrix = adj_mat)
result.BNM

#> Adjacency Matrix
```

```
#>      Item01 Item02 Item03 Item04 Item05
#> Item01      0      1      0      0      0
#> Item02      0      0      1      1      0
#> Item03      0      0      0      0      1
#> Item04      0      0      0      0      1
#> Item05      0      0      0      0      0
#> [1] "Your graph is an acyclic graph."
#> [1] "Your graph is connected DAG."
```



```
#>
#> Parameter Learning
#>      PIRP 1 PIRP 2 PIRP 3 PIRP 4
#> Item01  0.600
#> Item02  0.250    0.5
#> Item03  0.833    1.0
#> Item04  0.167    0.5
#> Item05  0.000   NaN  0.333  0.667
#>
#> Conditional Correct Response Rate
```

#>	Child Item	N of Parents	Parent Items	PIRP	Conditional CRR
#> 1	Item01	0	No Parents	No Pattern	0.600
#> 2	Item02	1	Item01	0	0.250
#> 3	Item02	1	Item01	1	0.500
#> 4	Item03	1	Item02	0	0.833
#> 5	Item03	1	Item02	1	1.000
#> 6	Item04	1	Item02	0	0.167
#> 7	Item04	1	Item02	1	0.500
#> 8	Item05	2	Item03, Item04	00	0.000
#> 9	Item05	2	Item03, Item04	01	NA
#> 10	Item05	2	Item03, Item04	10	0.333
#> 11	Item05	2	Item03, Item04	11	0.667

```
#>
#> Model Fit Indices
#>      value
#> model_log_like -27.046
#> bench_log_like -8.935
#> null_log_like -28.882
#> model_chi_sq 36.222
#> null_chi_sq 39.894
#> model_df 20.000
#> null_df 25.000
#> NFI 0.092
#> RFI 0.000
#> IFI 0.185
#> TLI 0.000
```

```
#> CFI          0.000
#> RMSEA       0.300
#> AIC         -3.778
#> CAIC        -29.829
#> BIC         -9.829
```

The output shows conditional correct response rates calculated from the data. The section labeled “Conditional Correct Response Rate” displays these probabilities for each item given the response patterns of its parent nodes. Item01, as the root node with no parents, simply shows its marginal pass rate. Item02 has Item01 as its parent, showing conditional pass rates of 0.25 when Item01 is answered correctly and 0.50 when Item01 is answered incorrectly. Similarly, Item05 has two parent nodes (Item03 and Item04), and its conditional pass rates vary by the four possible parent response patterns: 0.00 when both parents are incorrect (00), NA when the pattern 01 does not exist in the data, 0.33 when the pattern is 10, and 0.67 when both parents are correct (11). The “Parameter Learning” and “Conditional Correct Response Rate” sections contain the same information, with the latter providing more detailed output.

Below the conditional probabilities, model fit indices are displayed, allowing evaluation of how well the specified network structure fits the observed data.

4.3 Exploratory learning of network structures

These models require the network structure between items to be specified externally, but it is also possible to discover structures exploratively from the data. However, searching all possible combinations of adjacency matrices causes exponential combinatorial explosion as the number of items increases. In test theory, item pass rates flow from higher to lower, so items can be arranged by pass rate order to construct a Directed Acyclic Graph (DAG). This allows only the upper triangular portion of the item-by-item adjacency matrix to be the estimation target, significantly reducing the search space.

The `BNM_GA()` function shown in Table 2 uses a genetic algorithm to learn network structures. Additionally, `BNM_PBIL()` uses the Population-Based Incremental Learning (PBIL) method (Baluja, 1994; Fukuda et al., 2014). In PBIL, multiple realizations of adjacency matrices (with elements of 0 or 1) are first generated randomly from a generational Gene (an adjacency matrix with values between 0 and 1). The model is executed for each realization, and they are ranked based on fitness indices. Similar to genetic algorithms, selection is performed and the generational Gene is updated. By repeating this process, an adjacency matrix Gene that survives across generations is obtained. The final adjacency matrix is determined by rounding the Gene values or similar methods.

However, as with biclustering and rankclustering, it is often more practical and meaningful for test administrators to explicitly specify models based on domain knowledge rather than relying solely on exploratory analysis using fitness indices.

5 Discussion

This article presents the *exametrika* package, which is specifically designed for educational test data analysis, providing valuable information for both test administrators and examinees. A key design philosophy of the package is the recognition that continuous and highly precise numerical measurements are not always necessary or even desirable in educational settings. While IRT provides ability estimates with many decimal places, such precision rarely translates into actionable educational guidance. Teachers and learners benefit more from knowing which learning stage an examinee has reached and what steps are needed to advance to the next stage. The latent structure models in *exametrika* are designed to meet this need. In practice, when analyzing test data, even large-scale tests typically require only a dozen or so classes/ranks for adequate model fit. The resolution of examinees achievable through testing—at least from the perspective of statistical fit—is of this order, and practically speaking, there are few situations requiring finer granularity.

Item classification is also an important concern in educational tests and psychological scales. Many tests and scales are designed to measure multiple subdomains, and understanding how items group together is crucial for evaluating test structure and validity. In *exametrika*, biclustering and rankclustering models address this need by simultaneously classifying both items and examinees, providing insights into the content structure of the test and examinees’ ability profiles. The Field Reference Profile provided by these models—representing expected correct response rates for each class/rank within each field—offers concrete information about which content areas examinees have mastered and which require further study, enabling the construction of practical learning roadmaps. These models also provide field membership probabilities and class/rank membership probabilities,

indicating the degree to which each item belongs to each field and each examinee belongs to each class/rank. Visualizing these probabilities reveals whether examinees are clearly classified into specific classes/ranks or span multiple categories. Of course, this can also be expressed numerically through rank-up and rank-down odds, providing feedback that is understandable and valuable for examinees. For test administrators, examining how items decompose into fields provides information for evaluating the content structure of the test and revising it as needed. While the default approach performs exploratory item clustering, confirmatory biclustering and rankclustering are also available when item groupings are known in advance.

Furthermore, *exametrika* provides a flexible and interpretable analytical approach to test data through Bayesian network models (BNM, BINET). These models can capture complex dependencies among examinee response patterns, offering insights into latent structures inherent in test data. Unlike traditional latent variable models, BNM and BINET focus on relationships among observed variables, providing a new perspective for test data analysis. These models help examinees understand their response patterns and provide learning roadmaps, while giving test administrators information useful for designing educational steps. LD-LRA and LDB, which construct rank-specific network models, can capture different network structures based on examinees' ability ranks, providing powerful tools for understanding relationships among items at different ability levels; these structures can also be learned exploratively using genetic algorithms. BINET provides an overall learning course for the test set, showing which fields students at a given rank should study to advance to higher ranks. In this way, the Bayesian network models in *exametrika* provide a flexible and interpretable approach to test data analysis, offering valuable insights to both examinees and test administrators.

The extension of *exametrika* to polytomous data types opens significant possibilities beyond traditional educational testing. For ordinal data such as Likert scales commonly used in psychological measurement, the package offers an alternative analytical framework that does not rely on factor-analytic assumptions. Recent discussions in psychological measurement, led by [Borsboom \(2005\)](#), have raised fundamental questions about whether latent factors represent substantive psychological entities or are merely statistical summaries. Traditional psychometric models assume that observed responses are caused by underlying latent traits, but this causal interpretation remains contentious for many psychological constructs. The clustering approach in *exametrika* sidesteps this debate entirely—it classifies response patterns and item groupings without making claims about underlying psychological processes. This descriptive, pattern-based approach may prove valuable for scale development and validation, particularly when the existence of continuous latent traits is questionable. Furthermore, the network-based models in *exametrika* (BNM, BINET) align with the growing interest in psychological network analysis ([Borsboom, 2005](#); [Hood, 2009](#); [Marsman et al., 2018](#)), which focuses on relationships among observed variables rather than latent constructs.

While *exametrika* offers flexible, data-driven analytical tools, users should be mindful of certain operational considerations. The nonparametric nature of many models in the package means they are highly adaptive to the specific dataset at hand, which can be both a strength and a limitation. Models optimized for one dataset may not generalize well to new samples, particularly when sample sizes are small. Domain knowledge plays a crucial role in obtaining meaningful results—rather than relying solely on exploratory structure learning algorithms (such as GA or PBIL for network discovery), practitioners should consider specifying models based on theoretical understanding of the content domain. When designing tests or scales intended for analysis with *exametrika*, it is beneficial to plan the item structure in advance, considering how items group into fields and how response patterns should reflect developmental stages. The package's emphasis on providing interpretable feedback to examinees represents a distinctive advantage over purely statistical approaches; the outputs are designed not just for researchers but for practical educational application. Readers interested in technical details are encouraged to consult [Shojima \(2022\)](#) and the package website ([Kosugi, 2023](#)), which includes comprehensive documentation and example analyses.

6 Conclusion

The *exametrika* package represents a significant advancement in educational test data analysis by providing a comprehensive framework that surpasses traditional approaches. By focusing on meaningful learning roadmaps rather than precise numerical measurements, the package offers practical insights for test administrators and examinees. Integrating LRA, biclustering, and the Bayesian network model provides a flexible and interpretable approach to analyzing test data. In contrast, extending nominal and ordinal data types opens new possibilities for psychological measurement.

The package's emphasis on ordered clustering and its rich visualization capabilities make it particularly valuable for educational practice. Providing clear learning pathways and diagnostic information helps bridge the gap between test results and actionable educational interventions. As educational measurement continues to evolve, the *exametrika* package offers a promising direction

for rendering test data analysis more meaningful and valuable in real educational settings.

7 Acknowledgments

This research was supported by the Domestic Research Fellowship Program of Senshu University in 2023 (Reiwa 5).

References

- R. G. Almond, R. J. Mislevy, L. S. Steinberg, D. Yan, and D. M. Williamson. *Bayesian Networks in Educational Assessment*. Statistics for Social and Behavioral Sciences. Springer, New York, 2015. doi: 10.1007/978-1-4939-2125-6. [p85]
- S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, 1994. [p99]
- D. Borsboom. *Measuring the Mind: Conceptual Issues in Contemporary Psychometrics*. Cambridge University Press, Cambridge, 2005. ISBN 9780521541011. [p100]
- M. A. Croon. Latent class analysis with ordered latent classes. *British Journal of Mathematical and Statistical Psychology*, 43(2):171–192, 1990. [p85]
- M. J. Culbertson. Bayesian networks in educational assessment: The state of the field. *Applied Psychological Measurement*, 40(1):3–21, 2016. doi: 10.1177/0146621615590401. [p85]
- A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, Cambridge, 2009. [p85]
- I. S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Advances in Neural Information Processing Systems*, pages 577–583, 2001. [p85]
- R. Diakow, D. Torres Irribarra, and M. Wilson. Some comments on representing construct levels in psychometric models. *Measurement: Interdisciplinary Research and Perspectives*, 12(3):139–170, 2014. [p85]
- S. Fukuda, Y. Yamanaka, and T. Yoshihiro. A probability-based evolutionary algorithm with mutations to learn Bayesian networks. *International Journal of Interactive Multimedia and Artificial Intelligence*, 3(1):7–13, 2014. doi: 10.9781/ijimai.2014.311. URL <https://ijimai.researchcommons.org/ijimai/vol3/iss1/8>. [p99]
- G. Govaert and M. Nadif. Latent block model for contingency table. *Communications in Statistics—Theory and Methods*, 39(1):1–23, 2010. [p85]
- S. B. Hood. Validity in psychological testing and scientific realism. *Theory & Psychology*, 19(4):425–444, 2009. doi: 10.1177/0959354309336310. [p100]
- F. V. Jensen and T. D. Nielsen. *Bayesian Networks and Decision Graphs*. Springer, New York, 2nd edition, 2007. [p85]
- T. Koski and J. Noble. *Bayesian Networks: An Introduction*. Wiley Series in Probability and Statistics. John Wiley & Sons, Chichester, 2009. [p85]
- K. Kosugi. exametrika: An R package for latent rank analysis and biclustering, 2023. URL <https://kosugitti.github.io/exametrika/>. Accessed: 2025-05-15. [p97, 100]
- S.-J. Lee, J. Huang, J. Hu, and E. P. Xing. Biclustering via sparse singular value decomposition. *Biometrics*, 66(4):1087–1095, 2010. [p85]
- M. Marsman, L. J. Waldorp, C. J. Albers, M. E. Timmerman, R. van Bork, A. Cramer, D. Borsboom, and E.-J. Wagenmakers. An introduction to network psychometrics: Relating Ising network models to item response theory models. *Multivariate Behavioral Research*, 53(1):15–35, 2018. doi: 10.1080/00273171.2017.1379379. [p100]
- E. Matechou, J. Evans, and E. Wit. Biclustering models for two-mode ordinal data. *Psychometrika*, 81(2):413–435, 2016. [p85]

- J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988. [p85]
- R. Reichenberg. Dynamic Bayesian networks in educational measurement: Reviewing and advancing the state of the field. *Applied Measurement in Education*, 31(4):335–350, 2018. doi: 10.1080/08957347.2018.1495217. [p85]
- M. Scutari and J.-B. Denis. *Bayesian Networks: With Examples in R*. Chapman & Hall/CRC Texts in Statistical Science. CRC Press, Boca Raton, 2014. [p85]
- K. Shojima. *Test Data Engineering*. Springer, 2022. URL <https://link.springer.com/book/10.1007/978-981-16-9986-3>. [p85, 91, 94, 97, 100]
- D. Torres Irribarra, R. Diakow, R. Freund, and M. Wilson. Modeling for directly setting theory-based performance levels. *Measurement: Interdisciplinary Research and Perspectives*, 13(1):1–47, 2015. [p85]

Koji Kosugi
Senshu University
School of Human Sciences
2-1-1 Higashi-mita, Tama-ku, Kawasaki, Kanagawa 214-8580, Japan
<https://kosugitti.github.io/kosugitti10/>
ORCID: 0000-0001-5816-0099
kosugi@psy.senshu-u.ac.jp

Kojiro Shojima
The National Center for University Entrance Examinations
Research Division
2-19-23, Komaba, Meguro-ku, Tokyo, Japan
<http://shojima.stars.ne.jp/tde/index.htm>
shojima@rd.dnc.ac.jp

DeepLearningCausal: R Package for Estimating Treatment Effects Using Deep Neural Networks and Ensemble Learning

by Nguyen Khoi Huynh, Yang Yang, and Bumba Mukherjee

Abstract Conventional machine learning (ML) methods for causal inference are widely used for estimating causal effects and uncovering heterogeneous treatment effects. Yet, deep learning, which leverages deep neural networks to address selection bias and unobserved confounders, is underutilized for estimating sample and population average treatment effects. Further, conventional ML methods are employed to estimate treatment effects under the assumption that all units in the sample comply with the treatment, which is often implausible. To address these methodological challenges, this article introduces our DeepLearningCausal R-package that enables users to employ deep neural networks to estimate conditional average treatment effects (CATEs) from samples via meta-learners and population average treatment effects on the treated (PATT) in settings with treatment noncompliance. DeepLearningCausal also includes functions to compute and illustrate conformal prediction (CP) intervals for the estimated meta-learner individual treatment effects (ITEs), visualize heterogeneous treatment effects, and estimate CATEs and PATT by using weighted ensemble learning.

1 Introduction

Causal inference constitutes a vital component of scientific research in biology, economics, epidemiology, medicine, political science, and sociology (Pearl, 2009; Imbens and Rubin, 2015; Athey and Imbens, 2016; Li et al., 2024). Many puzzles in the natural and social sciences are causal questions and answers to these questions are statistically evaluated by estimating conditional average treatment effects (CATE) obtained from randomized control trials or (structural) causal models estimated on experimental and observational samples (Hill, 2011; Yao et al., 2018; Hu et al., 2021; Tikka, 2023). Yet the following methodological challenges often impede the accuracy of treatment effects estimated from samples: unobserved confounders, high-dimensional data, low external validity to the target population, and complex non-linear relationships between the treatment and outcome measures.

To address these challenges, scholars use conventional machine learning (ML) algorithms such as causal trees (Athey and Imbens, 2016), kernel methods (Alaa and Van Der Schaar, 2018), or causal boosting (Powers et al., 2018) that capture complex non-linear relationships when estimating treatment effects. Others have used targeted learning which leverages the Super Learner approach for estimating targeted causal estimands (van der Laan and Rose, 2011; Schuler and Rose, 2019). Researchers have also accounted for unobserved confounders by developing *meta-learner* models that tackle the said issue by decomposing estimation of CATEs into multiple sub-problems solved by conventional ML methods (Künzel et al., 2019; Nie and Wager, 2021). Some scholars have addressed external validity by extending the generalizability of sample treatment effects to the target population by using standard ML methods to estimate population average treatment effects on the treated from experimental and observational data (Hartman et al., 2015; Balzer et al., 2016; Zhang et al., 2019; Ottoboni and Populos, 2020). These statistical learning approaches that employ machine learning for causal inference have been widely disseminated via software packages that are discussed in the next section.

But unlike the widespread use of standard ML algorithms for causal inference, deep learning methods have been underutilized for the estimation of not just sample treatment effects such as CATEs but also population average treatment effects. This is surprising as deep learning methods based on deep neural networks capture heterogeneous treatment effects with low bias, account for unobserved confounders and selection bias, and assess causality in complex settings (Farrell et al., 2021; Koch et al., 2024). Moreover, the aforementioned statistical learning approaches are applied to settings which assume that all individual units in the experimental and observational sample have fully complied with the assigned treatment, even though some units (in reality) neither comply with nor accept the treatment. Not addressing treatment noncompliance of this sort leads to biased results when generalizing estimated treatment effects from samples to the broader target population.

We thus present our R package **DeepLearningCausal** (available at <https://cran.r-project.org/package=DeepLearningCausal>) in this paper which overcomes these limitations and enables researchers to employ deep learning methods to estimate CATEs from samples and population average treatment effects on the treated (PATT) from data that exhibit treatment noncompliance. **DeepLearningCausal** allows users to employ deep learning based on deep neural networks to estimate CATEs from four meta-learner models: T-learner, S-learner, X-learner, and R-learner. Our package also pro-

vides functions to use deep neural networks to estimate the PATT from a newly developed causal model that addresses treatment noncompliance by some units in the sample. Further, **DeepLearningCausal** provides an interface to Python that enables users to leverage Python's extensive deep learning ecosystem (within their R script) for estimating the CATE and PATT via deep neural networks. The package provides users with substantial flexibility to substantially customize their deep neural network architecture for deep learning estimation, and enables users to conduct conformal inference of individual treatment effects obtained from the meta-learner models. The analysis presented below primarily focuses on our package's functionality for *deep learning estimation* of the meta-learner CATEs and the PATT (given treatment noncompliance) using **reticulate** (Ushey et al., 2025), **tensorflow** (Allaire and Tang, 2024; Abadi et al., 2015), and **keras** (Kalinowski et al., 2025; Chollet et al., 2015) which provide access to Python's deep learning libraries. The functions and procedure for estimating the meta-learner CATEs and PATT using weighted ensemble learning and R neural nets are provided in our package's GitHub repository: [hknd23/DeepLearningCausal](https://github.com/hknd23/DeepLearningCausal).

2 Background and other R packages

Although parametric models are often used for estimating average treatment effects, they rely on the assumption that the functional form of the relationship between variables is correctly specified. But if the assumed model is misspecified, the estimates can be biased, and identification of causal effects can be challenging in the presence of unmeasured confounders (Jonzon et al., 2023; Tikka, 2023). Scholars have thus used conventional machine learning methods (e.g., gradient boosting, random forests) to address the aforementioned challenge when estimating CATEs in samples or the PATT in the target population, and one recent study uses deep learning for estimating CATEs from two meta-learner models (Hartman et al., 2015; Johansson et al., 2016; Chernozhukov et al., 2018; Yao et al., 2018; Künzel et al., 2019; Zhang et al., 2020; Koch et al., 2024). Deep learning techniques for causal inference that address challenges such as unstructured variables, unobserved confounders, or unknown intervention include either the use of *general* deep neural networks that involve using multilayer perceptrons or *specific* deep neural network architectures including convolutional neural networks and variational autoencoders (Goodfellow et al., 2016; Koch et al., 2024). Other studies have implemented "targeted learning" that employs super learner-based weighted ensemble learning to estimate specific causal parameters (estimands) by *targeting* the data-fitting process to the target parameter (van der Laan and Rose, 2011; Schuler and Rose, 2019).

Conventional ML, targeted learning, and deep learning methods for estimating causal effects address the limitations of parametric models as these methods learn complex relationships without requiring explicit functional forms for the association between variables (Zhang et al., 2020; Nie and Wager, 2021; Okasa, 2022; Hu and Ji, 2022). Given these advantages, it is not surprising that conventional ML or super learner methods are employed for estimating various causal models, including Augmented Inverse Probability Weighting, Double/Debiased Machine Learning, causal estimands that employ targeted learning, and meta-learner models (Glynn and Quinn, 2010; van der Laan and Rose, 2011; Chernozhukov et al., 2018; Knaus, 2022; Bach et al., 2024; Okasa, 2022). While the review by Koch et al. (2024) introduces deep learning for causal effect estimation in the social sciences, researchers also use standard ML methods to estimate population average treatment effects on the treated from experimental data with and without treatment noncompliance (Hartman et al., 2015; Balzer et al., 2016; Ottoboni and Populos, 2020).

In line with these trends in the causal machine learning literature, numerous R packages have been developed that permit users to use machine learning methods for causal inference. The package **DoubleML** Bach et al. (2024) focuses on estimation of the nuisance parts in causal models by machine learning methods and computation of the Neyman orthogonal score functions, while **lmt** by Díaz et al. (2021) employs ensemble learning for estimating causal effects based on longitudinal treatment indicators. The **CIMTx** package by Hu and Ji (2022) uses Bayesian additive regression trees to estimate multiple treatments with a focus on binary outcomes, and the package **causaloptim** Jonzon et al. (2023) focuses on causal graphs and bounds for causal inference that have implications for machine learning model development. The **SVMMatch** package by Ratkovic (2015) uses support vector machines for causal effect estimation, the **htetree** package by Xu et al. (2023) provides functions to estimate heterogeneous treatment effects with tree-based machine learning algorithms, and **twangContinuous** by Coffman and Griffin (2021) uses gradient boosting machines for estimating continuous treatments. Python packages such as **CausalML** by Zhao and Liu (2023) and **EconML** by Battocchi et al. (2019) also provide ML methods for estimating treatment effects. There also exists open-source R code that uses ML methods for estimating meta-learner models (Künzel et al., 2019; Nie and Wager, 2021).

To our knowledge, our **DeepLearningCausal** package is the first that allows users to employ deep learning based on deep neural networks to estimate CATEs from several meta-learner models and the PATT from a new estimator that accounts for treatment noncompliance by some units (Ottoboni

and Populos, 2020). **DeepLearningCausal** also permits users to automatically install the **reticulate** R package, **TensorFlow**, and **Keras3** (R and Python packages) to integrate Python code and libraries used for deep neural networks within an R environment. This feature enables estimation of treatment effects via a highly customized deep neural network architecture using cutting-edge optimizers such as Adam and RMSprop. Further, our package permits users to implement the recently developed conformal prediction (CP) framework to conduct inference of target parameters such as individual treatment effects from meta-learner models. Additional functions in our package permit estimation of the CATE and PATT via the Super Learner-based weighted ensemble learning, illustration of heterogeneous treatment effects, and assessment of correlation of CATEs from meta-learner models.

3 Models

We use the Neyman-Rubin potential outcomes framework to conceptualize estimation of the CATEs and the PATT (Imbens and Rubin, 2015). Assume the dataset $\mathcal{D}(Y_i, \mathbf{X}_i, W_i)_{i=1}^n$ with $(Y_i, \mathbf{X}_i, W_i) \stackrel{i.i.d.}{\sim} \mathcal{P}$ where i denotes an individual subject, $Y \in \mathcal{Y}$ is a binary or continuous outcome of interest, $\mathbf{X}_i \in \mathcal{X} \subset \mathbb{R}^d$ is the vector of covariates, and $W_i \in \{0, 1\}$ is the binary treatment assigned according to the propensity score $\pi(\mathbf{x}) = P(W_i = 1 | \mathbf{X}_i = \mathbf{x})$. The two potential outcomes (POs) represent the outcome under the treatment $Y_i^{(1)}$ and control $Y_i^{(0)}$. Since only one of the POs is observed, the observed outcome is: $Y = W_i Y_i^{(1)} + (1 - W_i) Y_i^{(0)}$. The ATE is the difference between the two POs: $\tau = E[Y_i^{(1)} - Y_i^{(0)}]$. The ATE is *identifiable* when the following assumptions hold: consistency, SUTVA (Stable Unit Treatment Value Assumption), ignorability (POs are independent of the treatment assignment given the covariates), and positivity (each i has a non-zero probability of belonging to the control and treated groups). These assumptions also permit identification of the CATEs defined below.

3.1 CATEs and non-parametric regression framework

Given *identifiability*, the conditional average treatment effect (CATE) is the expected difference between the two POs conditional on the covariates $\mathbf{X}_i = \mathbf{x}$:

$$\tau(\mathbf{x}) = \mathbb{E}[Y_i^{(1)} - Y_i^{(0)} | \mathbf{X}_i = \mathbf{x}] = \mu_1(\mathbf{x}) - \mu_0(\mathbf{x}) \quad (1)$$

Equation (1) reveals that the CATE is defined as the difference between the response surface under treatment, $\mu_1(\mathbf{x})$, and under control, $\mu_0(\mathbf{x})$:

$$\mu_w(\mathbf{x}) := \mathbb{E}[Y^{(w)} | \mathbf{X}_i = \mathbf{x}] \quad (2)$$

While the CATE can be estimated via numerous methods, we focus below on *meta-learners* which are non-parametric regression approaches that model the outcome surface Y as a function of the treatment assignment W_i , the covariates $\mathbf{X}_i$, and the error term ε_i : $Y = f(\mathbf{X}_i, W_i) + \varepsilon_i$ where $\varepsilon_i \sim N(0, \sigma^2)$ and $f(\mathbf{X}_i, W_i) = \mathbb{E}[Y | \mathbf{X}_i, W_i]$. We turn to describe the four meta-learner models that estimate the CATE via the non-parametric regression framework: T-learner, S-learner, X-learner, and R-learner.

3.2 Meta-learner models: T-learner and S-learner

Meta-learner models decompose estimation of the CATE $\hat{\tau}(\mathbf{x})$ into multiple prediction problems (Künzel et al., 2019). The machine or deep learning method employed to solve the prediction problem is called a *base-learner*. Most of the prediction problems in meta-learner models amount to estimating the conditional means of the outcome and the treatment. The latter are referred to as *nuisance* functions, because they are not of primary interest themselves, but are needed to derive $\hat{\tau}(\mathbf{x})$. Meta-learner models are typically categorized into *conditional mean regression* and *pseudo-outcome* methods (Okasa, 2022). We thus formally describe two main conditional mean regression methods (T and S-learner models) and then turn to present two key pseudo-outcome methods (X and R-learner models).

To begin with, the T-learner model estimates $\tau(\mathbf{x})$ by estimating the conditional mean function in each treatment arm, $\mu_0(\mathbf{x}) = \mathbb{E}[Y | \mathbf{X}_i = \mathbf{x}, W_i = 0]$, and the conditional mean function under treatment, $\mu_1(\mathbf{x}) = \mathbb{E}[Y | \mathbf{X}_i = \mathbf{x}, W_i = 1]$. Accordingly, the T-learner estimates the CATE by fitting two separate response surfaces for the treated and control groups respectively, and then computing their difference:

$$\hat{\tau}(\mathbf{x}) = \hat{\mu}_1(\mathbf{x}) - \hat{\mu}_0(\mathbf{x}) \quad (3)$$

This task is implemented by fitting two separate prediction models for the base learner: the first to

the control group's data and the second to the treatment group's data. The base learner can be fit by weighted ensemble learning or deep neural networks which we focus on below. Specifically, deep neural network (DNN)-estimated CATEs from the T-learner model are obtained by using feed-forward networks. This entails fitting a separate network for each regression task, namely the PO surface for the treated, μ_1 , and the control, μ_0 , groups. The two networks are trained using,

$$\mathcal{L}_F + \lambda \sum_1^{\mu_w} \mathfrak{R}(\Theta_{\mu_w}) \quad (4)$$

$\mathcal{L}_F$ is the loss function of the general penalized objective function, $\mathfrak{R}(\cdot)$ is the regularization term, Θ_{μ_w} is the network's weights where $W_i \in (0, 1)$, and λ is the hyperparameter that controls the trade-off between the standard loss and regularization term. Once the two POs surfaces are estimated, CATEs from the DNN-estimated T-learner are obtained by the difference: $\hat{\tau}(\mathbf{x}) = \hat{\mu}_1(\mathbf{x}) - \hat{\mu}_0(\mathbf{x})$. This means that DNN-estimation of the T-learner fits the response variable Y by assuming that the response surfaces μ_w are group-specific, and thus dependent on different conditional means $f_w(\cdot)$ and error terms ε_w . The pseudocode for the DNN-estimated T-learner is:

Algorithm: DNN-estimated T-learner

Input: $\mathbf{X}, Y, W$; **Output:** $\hat{\tau}$

- 1: $\hat{\mu}_0 = NN_1(Y^0 \sim \mathbf{X}^0) \triangleright$ Estimate POs surfaces
 - 2: $\hat{\mu}_1 = NN_2(Y^1 \sim \mathbf{X}^1)$
 - 3: $\hat{\tau}(\mathbf{x}) = \hat{\mu}_1(\mathbf{x}) - \hat{\mu}_0(\mathbf{x}) \triangleright$ Estimate CATE
-

Algorithm: DNN-estimated T-learner

Input: $\mathbf{X}, Y, W$; **Output:** $\hat{\tau}$

- 1: $\hat{\mu}_0 = NN_1(Y^0 \sim \mathbf{X}^0) \triangleright$ Estimate POs surfaces
 - 2: $\hat{\mu}_1 = NN_2(Y^1 \sim \mathbf{X}^1)$
 - 3: $\hat{\tau}(\mathbf{x}) = \hat{\mu}_1(\mathbf{x}) - \hat{\mu}_0(\mathbf{x}) \triangleright$ Estimate CATE
-

where NN_1 denotes deep neural networks. The S-learner uses the whole sample to fit a single model in which the observed outcome values are modeled as a function of the covariates and the treatment to obtain $\hat{\mu}(\mathbf{x}, w) = \hat{\mathbb{E}}[Y | \mathbf{X}_i = \mathbf{x}; W_i = w]$. Hence, the S-learner's CATEs are estimated as,

$$\hat{\tau}(\mathbf{x}) = \hat{\mu}(\mathbf{x}, 1) - \hat{\mu}(\mathbf{x}, 0) \quad (5)$$

This expression implies that the S-Learner fits a single response surface $\mu(\cdot)$ by using regressors $(\mathbf{X}_i, W_i)$ through a base learner and estimates CATEs by taking the difference between the two conditional average POs represented by the fitted $\hat{\mu}(\cdot)$ with $W_i = 1$ and $W_i = 0$. The group-specific conditional average POs in the S-learner stem from the same model, with conditional mean function $\mu(\cdot)$ and error term ε_i . Formally, the deep neural network-based (DNN-based) S-learner uses feed-forward networks to fit a separate network for the regression task, namely the single response surface. This network is trained by using the general loss function defined above. Once the response surface is estimated, the DNN-based S-learner obtains the CATE by the difference stated in equation (5). The pseudocode for estimating the DNN-based S-learner is:

Algorithm: DNN-estimated S-learner

Input: $\mathbf{X}, Y, W$; **Output:** $\hat{\tau}$

- 1: $\hat{\mu} = NN_1(Y \sim (\mathbf{X}, W)) \triangleright$ Estimate response surface
 - 2: $\hat{\tau}(\mathbf{x}) = \hat{\mu}(\mathbf{x}, 1) - \hat{\mu}(\mathbf{x}, 0) \triangleright$ Estimate CATE
-

Algorithm: DNN-estimated S-learner

Input: $\mathbf{X}, Y, W$; **Output:** $\hat{\tau}$

- 1: $\hat{\mu} = NN_1(Y \sim (\mathbf{X}, W)) \triangleright$ Estimate response surface
 - 2: $\hat{\tau}(\mathbf{x}) = \hat{\mu}(\mathbf{x}, 1) - \hat{\mu}(\mathbf{x}, 0) \triangleright$ Estimate CATE
-

3.3 Individual Treatment Effects (ITEs) and conformal prediction

Existing studies have primarily focused on machine learning algorithms or (less frequently) deep learning to produce point estimates of CATEs from meta-learner models, which quantify the expected difference in treatment outcomes for individuals with specific characteristics (Wager and Athey, 2019; Künzel et al., 2019). While CATEs account for different covariates by averaging effects across similar individuals, it tends to overlook individual-level heterogeneity. Recent studies have addressed this issue by focusing on individual treatment effects (ITEs) that offer more accurate predictions tailored to each individual (Alaa and Van Der Schaar, 2018).

Building on this, our package permits users to apply the recently developed work on weighted split conformal quantile regression (CQR) for producing conformal prediction (CP) intervals for ITEs obtained from the said meta-learner models (Lei and Candès, 2021; Alaa et al., 2023). The CQR framework that integrates the concept of conformal prediction with quantile regression enables direct inference of the target parameters (ITEs) obtained in the two conditional mean regression methods: the S-learner and T-learner model.

Weighted conformal prediction provides a model-agnostic and distribution-free framework that produces interval predictions with the desired coverage probability. The main idea of (split) conformal prediction is to compute a conformity score by fitting a predictive model on the training set and then evaluating the conformity score on the calibration set to quantify the uncertainty of future predictions. Doing so allows researchers to estimate and construct intervals of ITEs by computing the empirical quantile of conformity scores evaluated in a held-out calibration set. We turn to briefly present below how we employ the weighted split CQR framework to estimate and construct CP intervals for the S-learner model's ITEs. The CQR framework for estimating and constructing CP intervals for the T-learner's ITEs is summarized in the GitHub repository of our package.

To this end, recall that the meta-learner CATEs are defined as $\tau(\mathbf{x}) = \mathbb{E}[Y_i^{(1)} - Y_i^{(0)} | \mathbf{X}_i = \mathbf{x}]$. Our focus is to employ the weighted split CQR procedure to estimate and construct valid prediction intervals $\hat{C}_{1-\alpha}(x_i)$ for the S-learner's ITEs: $\hat{\tau}(x_i) = \hat{Y}_i(1) - \hat{Y}_i(0)$ where $\hat{Y}_i(1)$ and $\hat{Y}_i(0)$ are predicted potential outcomes under treatment and control, respectively. The algorithm for doing so is summarized as:

Algorithm: Weighted split CQR procedure for S-learner's ITEs

Step 1. Randomly split training data into two parts:

- ▷ A model training subset to fit the predictive model $\hat{f}(x, w)$.
- ▷ A calibration subset to compute nonconformity scores and determine the empirical error distribution. Denote the calibration subset as $\mathcal{C}$.

Step 2. Compute Nonconformity Scores:

- ▷ For each $i \in \mathcal{C}$, compute the absolute residual (nonconformity score) as $r_i = |y_i - \hat{Y}_i(w_i)|$. $\hat{Y}_i(w_i)$ is the model's predicted outcome given treatment w_i .

Step 3. Assign Propensity-Based Weights:

- ▷ Since some calibration units are more informative than others, assign *overlap weights* $g_i = \hat{p}_i(1 - \hat{p}_i)$
- ▷ These overlap weights (which emphasize units in the regions of covariate overlap) are based on the estimated propensity score $\hat{p}_i = P(W_i = 1 | X_i)$
- ▷ Normalize the weights: $\tilde{g}_i = \frac{g_i}{\sum_{j \in \mathcal{C}} g_j}$

Step 4. Compute the weighted quantile of calibration residuals:

- ▷ Compute the weighted $(1 - \alpha)$ quantile of the calibration residuals:
 $q_{1-\alpha} = \min\{q : \sum_{i: r_i \leq q} \tilde{w}_i \geq 1 - \alpha\}$
- ▷ $q_{1-\alpha}$ represents the smallest residual threshold

Step 5. Construct conformal prediction intervals:

- ▷ Construct a two-sided conformal interval around the estimated treatment effect for each x_i :
 $\hat{C}_{1-\alpha}(x_i) = [\hat{\tau}(x_i) - q_{1-\alpha}, \hat{\tau}(x_i) + q_{1-\alpha}]$
-

3.4 X-learner and R-learner (meta-learner) models

Pseudo-outcome methods such as the X-learner and R-learner models presented here first estimate nuisance functions (e.g., the conditional means of the outcome and the propensity score) and then combine these estimates into a pseudo-outcome $\hat{\psi}$ (Künzel et al., 2019; Nie and Wager, 2021). The X-learner specifically computes two group-specific estimators, $\hat{\tau}_1(\mathbf{x})$ and $\hat{\tau}_0(\mathbf{x})$, and combines them with a weighting function to estimate the CATEs, $\hat{\tau}(\mathbf{x})$. Künzel et al. (2019) proposed three steps to estimate $\hat{\tau}(\mathbf{x})$ from the X-learner that we adopt. First, we estimate the two PO response surfaces,

Algorithm: Weighted split CQR procedure for S-learner's ITEs**Step 1.** Randomly split training data into two parts:

- ▷ A model training subset to fit the predictive model $\hat{f}(x, w)$
- ▷ A calibration subset to compute nonconformity scores and determine the empirical error distribution. Denote the calibration subset as $\mathcal{C}$.

Step 2. Compute Nonconformity Scores:

- ▷ For each $i \in \mathcal{C}$, compute the absolute residual (nonconformity score) as $r_i = |y_i - \hat{Y}_i(w_i)|$. $\hat{Y}_i(w_i)$ is the model's predicted outcome given treatment w_i

Step 3. Assign Propensity-Based Weights:

- ▷ Since some calibration units are more informative than others, assign *overlap weights* $g_i = \hat{p}_i (1 - \hat{p}_i)$
- ▷ These overlap weights (which emphasize units in the regions of covariate overlap) are based on the estimated propensity score $\hat{p}_i = P(W_i = 1 | X_i)$
- ▷ Normalize the weights: $\tilde{g}_i = \frac{g_i}{\sum_{j \in \mathcal{C}} g_j}$

Step 4. Compute the weighted quantile of calibration residuals:

- ▷ Compute the weighted $(1 - \alpha)$ quantile of the calibration residuals: $q_{1-\alpha} = \min \{q : \sum_{i: r_i \leq q} \tilde{w}_i \geq 1 - \alpha\}$
- ▷ $q_{1-\alpha}$ represents the smallest residual threshold

Step 5. Construct conformal prediction intervals:

- ▷ Construct a two-sided conformal interval around the estimated treatment effect for each x_i : $\hat{C}_{1-\alpha}(x_i) = [\hat{\tau}(x_i) - q_{1-\alpha}, \hat{\tau}(x_i) + q_{1-\alpha}]$

$\mu_1(\mathbf{x})$ and $\mu_0(\mathbf{x})$, by using deep neural networks (or weighted ensemble learning) as the base-learner. The estimated functions are $\hat{\mu}_1(\mathbf{x})$ and $\hat{\mu}_0(\mathbf{x})$. Second, we obtain the imputed treatment effects $\tilde{D}$ by computing the difference between the observed outcomes Y and the counterfactual outcomes estimated for the treatment not assigned using the corresponding regression surface $\hat{\mu}_w$,

$$\tilde{D}^1 = Y^1 - \hat{\mu}_0(\mathbf{X}_i) \text{ if } W_i = 1 \quad (6)$$

$$\tilde{D}^0 = \hat{\mu}_0(\mathbf{X}_i) - Y^0 \text{ if } W_i = 0 \quad (7)$$

for the treated and control groups. If $\hat{\mu}_0 = \mu_0$ and $\hat{\mu}_1 = \mu_1$ then $\tau(\mathbf{x}) = \mathbb{E}[\tilde{D}_1 | \mathbf{X}_i = \mathbf{x}] = \mathbb{E}[\tilde{D}_0 | \mathbf{X}_i = \mathbf{x}]$. $\tilde{D}$ is an unbiased estimator of τ when μ_0 and μ_1 are known. We next estimate the group-specific CATEs, $\hat{\tau}_0$ and $\hat{\tau}_1$, in two separate non-parametric pseudo-outcome regressions, using the imputed treatment effects as the response variable and the covariates $\mathbf{X}$ as regressors:

$$\tilde{D}_1 = \tau_1(\mathbf{X}_i) + \eta_1 \text{ if } W_i = 1 \quad (8)$$

$$\tilde{D}_0 = \tau_0(\mathbf{X}_i) + \eta_0 \text{ if } W_i = 0 \quad (9)$$

η_w is a general error term. $\hat{\tau}_0$ and $\hat{\tau}_1$ can be estimated from these expressions by using ensemble learning or deep neural networks as the base-learner. Third, we estimate CATEs by the weighted average:

$$\hat{\tau}(\mathbf{x}) = g(\mathbf{x}) \hat{\tau}_0(\mathbf{x}) + (1 - g(\mathbf{x})) \hat{\tau}_1(\mathbf{x}) \quad (10)$$

We use the propensity score $g(\mathbf{x}) = \pi(\mathbf{x})$ as the weighting function. This requires estimating $\hat{\pi}(\mathbf{x})$ with a separate neural network (or via ensemble learning). Hence, the pseudocode for obtaining CATEs from the DNN-estimated X-learner is:

Algorithm: DNN-estimated X-learner**Input:** $\mathbf{X}, Y, W, *g$; **Output:** $\hat{\tau}$

- 1: $\hat{\mu}_0 = NN_1(Y^0 \sim \mathbf{X}^0)$ ▷ {Estimate response surfaces}
- 2: $\hat{\mu}_1 = NN_2(Y^1 \sim \mathbf{X}^1)$
- 3: $\tilde{D}^1 = Y^1 - \hat{\mu}_0(\mathbf{X}^1)$ ▷ Compute imputed treatment effects
- 4: $\tilde{D}^0 = \hat{\mu}_1(\mathbf{X}^0) - Y^0$
- 5: $\hat{\tau}_1 = NN_3(\tilde{D}^1 \sim \mathbf{X}^1)$ ▷ {Estimate group-specific CATEs}
- 6: $\hat{\tau}_0 = NN_4(\tilde{D}^0 \sim \mathbf{X}^0)$
- 7: $\hat{\tau}(\mathbf{x}) = g(\mathbf{x}) \hat{\tau}_0(\mathbf{x}) + (1 - g(\mathbf{x})) \hat{\tau}_1(\mathbf{x})$ ▷ Average estimates

Algorithm: DNN-estimated X-learner**Input:** $\mathbf{X}, Y, W, g$; **Output:** $\hat{\tau}$

- 1: $\hat{\mu}_0 = NN_1(Y^0 \sim \mathbf{X}^0)$ ▷ Estimate response surfaces
- 2: $\hat{\mu}_1 = NN_2(Y^1 \sim \mathbf{X}^1)$
- 3: $\tilde{D}^1 = Y^1 - \hat{\mu}_0(\mathbf{X}^1)$ ▷ Compute imputed treatment effects
- 4: $\tilde{D}^0 = \hat{\mu}_1(\mathbf{X}^0) - Y^0$
- 5: $\hat{\tau}_1 = NN_3(\tilde{D}^1 \sim \mathbf{X}^1)$ ▷ Estimate group-specific CATEs
- 6: $\hat{\tau}_0 = NN_4(\tilde{D}^0 \sim \mathbf{X}^0)$

of the excluded data by minimizing the prediction errors via cross-validation. Second, plug in the estimates from the previous step to estimate $\hat{\tau}(x_i)$ by minimizing (11) via parameter tuning on the k -folds. The weighted pseudo-outcome regression can be fit by deep neural networks (or weighted ensemble learning) that allow modification of the loss function by passing the weights $(W_i - \hat{\pi}(X_i))^2$. The pseudocode for obtaining CATEs from the deep neural network-estimated R-learner is:

Algorithm: DNN-estimated R-learner

Input: X, Y, W ; **Output:** $\hat{\tau}$

- 1: $\hat{m} = NN_1(Y \sim X) \triangleright$ Estimate *nuisance* parameters
 - 2: $\hat{\pi} = NN_2(W \sim X)$
 - 3: $\hat{\psi}_R(X_i) = \frac{Y_i - \hat{m}(X_i)}{(W_i - \hat{\pi}(X_i))} \triangleright$ Compute pseudo-outcome
 - 4: $\hat{\tau} = NN_3(\hat{\psi}_R \sim X_i) \triangleright$ Estimate CATE
-

Algorithm: DNN-based R-learner

Input: X, Y, W ; **Output:** $\hat{\tau}$

- 1: $\hat{m} = NN_1(Y \sim X) \triangleright$ Estimate *nuisance* parameters
 - 2: $\hat{\pi} = NN_2(W \sim X)$
 - 3: $\hat{\psi}_R(X_i) = \frac{Y_i - \hat{m}(X_i)}{(W_i - \hat{\pi}(X_i))} \triangleright$ Compute pseudo-outcome
 - 4: $\hat{\tau} = NN_3(\hat{\psi}_R \sim X_i) \triangleright$ Estimate CATE
-

3.5 Estimating PATT from experiments with noncompliance

Scholars such as [Ottonboni and Populos \(2020\)](#) have developed the “PATT-C” model to obtain *population* average treatment effects on the treated (PATT) from experimental (e.g., RCTs) and observational samples in which noncompliance with treatment is prevalent. We focus here on presenting the PATT-C model. To this end, let Y_{iGR} be the potential outcome for individual i in group $G_i \in \{0, 1\}$. $G = 0$ in Y_{iGR} when i is in the target population and $G = 1$ when i is in the experimental sample (e.g. RCT).

Define $W_i \in \{0, 1\}$ as the binary treatment assignment. Let $R_i \in \{0, 1\}$ denote whether or not i received the treatment. Because the treatment is randomly assigned in the experimental sample, W_i and R_i are observed in this sample when $G_i = 1$ for i . Let $X_i \in \{0, 1\}$ be the pretreatment covariates that influence selection into the experimental sample, treatment assignment in the population, and treatment noncompliance. $W_i = 0$ for individuals in the population who do not receive the treatment. $W_i = 1$ for those who receive the treatment but can decide whether or not to accept the treatment.

Let $C_i \in \{0, 1\}$. $C_i = 1$ when i complies with the received treatment, and $C_i = 0$ when i does not comply with the treatment. C_i is only observed for individuals in the treated group in the experimental sample. Hence, for individuals who comply with the treatment, $W_i = R_i$. For individuals in the population, we only observe R_i but not W_i . Five assumptions permit identification of PATT from experimental samples with treatment noncompliance ([Ottonboni and Populos, 2020](#), 110). The first assumption is *consistency* which implies that each i has the same response to the received treatment whether or not i is in the experimental sample: $Y_{i0R} = Y_{i1R}; R = \{0, 1\} \forall i$. The second is *conditional independence* of compliance and sample and treatment assignment: $C_i \perp G_i, W_i | X_i, 0 < \mathbb{P}(C_i = 1 | X_i) < 1$. The third assumption is *strong ignorability* of sample assignment for the *treated*: $(Y_{i01}, Y_{i11}) \perp G_i | (X_i, W_i = 1, C_i = 1)$. The fourth assumption is *strong ignorability* of sample assignment for the *control*: $(Y_{i00}, Y_{i10}) \perp G_i | (X_i, W_i = 1, C_i = 1; 0 < \mathbb{P}(G_i = 1 | X_i, W_i = 1, C_i = 1) < 1$.

The third and fourth assumptions imply strong ignorability of sample assignment for treated and control non-compliers since compliance—as per the second assumption—is also independent of sample and treatment assignment conditional on the covariates. These two assumptions also presuppose that the response surface is the same for compliers in the experimental sample and the target population. The fifth assumption is one-sided noncompliance, $\mathbb{P}(R_i | W_i = 0) = 0 \forall i$, which implies that individuals assigned to control are not allowed to receive the treatment.

The estimated PATT $\hat{\tau}_p$ from the sample with treatment assignment is the average causal effect of taking up treatment assigned on individuals who received treatment in the population. This follows from the first two assumptions, which ensure that the potential outcomes do not differ based on sample assignment or receipt of treatment: $\hat{\tau}_p = \mathbb{E}(Y_{i01} - Y_{i100} | G_i = 0, R_i = 1)$. Building on this and the five aforementioned assumptions permits identification of the PATT $\hat{\tau}_p$ from samples with treatment noncompliance:

$$\hat{\tau}_p = \mathbb{E}_{01} [\mathbb{E}(Y_{i11} | G_i = 1, R_i = 1, X_i)] - \mathbb{E}_{01} [\mathbb{E}(Y_{i10} | G_i = 1, R_i = 0, C_i = 1, X_i)]$$

$E_{01} [E(\cdot | \dots, X_i)]$ denotes the expectation with respect to the distribution of X_i for those in the target population that received the treatment (see (Ottoboni and Populos, 2020, 111)).

Two datasets are required to estimate the PATT $\hat{\tau}_p$ from samples with treatment noncompliance. The first is the experimental or RCT sample which is the sample of individual participants drawn from the target population, but in which some participants have not complied with the treatment. The second is the commensurate observational data from the target population in which some individuals have received the treatment but may or may not have accepted the treatment. Estimating the PATT $\hat{\tau}_p$ from these datasets with treatment noncompliance is implemented in four steps.

First, train a model via deep neural networks (or weighted ensemble learning) to predict the probability of compliance as a function of the covariates X_i by using the group assigned to treatment in the experimental sample. Second, use the model trained in the first step to predict which observations in the experimental sample assigned to the control group would have complied with the treatment had they been assigned to the treated group. Third, for both observed compliers in the treated group and predicted compliers in the control group in the experimental sample, train a model via deep neural networks (or weighted ensemble learning) using the covariates X_i and information about whether i received (R_i) the treatment to predict the outcome response in the sample. This third step leads to $E(Y_{i1R} | G_i = 1; R_i, X_i)$ for $R_i \in \{0, 1\}$. Fourth, estimate the potential outcome Y_{i1R} for all i that received the treatment in the target population by using the model from the third step. This produces the potential outcome for i in group G that receives the treatment (R). The difference between the mean counterfactual Y_{i11} and the mean counterfactual Y_{i10} leads to the estimate of $\hat{\tau}_p$. The pseudo-code for estimating the PATT in settings with noncompliance via deep neural networks is summarized as:

Algorithm: DNN-estimated PATT

Input: X, Y, W ; **Output:** $\hat{\tau}_p$

1: $\hat{C}_i = NN_1(C \sim (X, W)) \triangleright$ Train Models

2: $\hat{Y}_{iGR} = NN_2(Y \sim (X, R))$

3: $E(Y_{i1R} | G_i = 1; R_i, X_i)$ for $R_i \in \{0, 1\} \triangleright$ Predict outcome response using X_i and R_i

4: $\hat{\tau}_p = E(Y_{i11} | G_i = 1; R_i = 1, X_i) - E(Y_{i10} | G_i = 1; R_i = 0, C_i = 1, X_i) \triangleright$ Estimate PATT

Algorithm: DNN-estimated PATT	
Input: X, Y, W, R ; Output: $\hat{\tau}_p$	
1: $\hat{C}_i = NN_1(C \sim (X, W))$	$\triangleright$ Train Models
2: $\hat{Y}_{iGR} = NN_2(Y \sim (X, R))$	
3: $E(Y_{i1R} G_i = 1; R_i, X_i)$ for $R_i \in \{0, 1\}$	$\triangleright$ Predict outcome response using X_i and R_i
4: $\hat{\tau}_p = E(Y_{i11} G_i = 1; R_i = 1, X_i) -$	$\triangleright$ Estimate PATT
$E(Y_{i10} G_i = 1; R_i = 0, C_i = 1, X_i)$	

4 Deep learning estimation of treatment effects

This section provides an introduction for building deep neural network models for estimating the meta-learner CATEs and the PATT in settings with treatment noncompliance. Our package also provides functions to estimate the CATEs and the PATT via weighted ensemble learning (see Table 1).

Deep Neural Networks

We use deep learning based on deep neural networks to estimate CATEs from the meta-learner models and the PATT from datasets with treatment noncompliance. The deep neural networks method employed here for estimating CATEs and the PATT is broadly based on the architecture illustrated in Figure 1. This figure shows that deep neural networks are structured as stacks of input and output layers on top of each other with multiple hidden layers added between the input and output layers. More specifically, in the case of deep neural networks, the predictor variable and covariates from a given dataset are captured by the number of neurons that constitute the input layer of the neural network. The covariates that include the treatment variable thus comprise the input layer for the network. The deep neural network learning process begins with the forward propagation phase that takes in the raw data (neurons) from the input layer (see Figure 1) that can be trained to learn the potential outcome(s).

The neurons from the input layer then go through the hidden layers, where the neurons perform deep learning on the data. With respect to such learning, deep neural network estimation focuses on

feed-forward networks tasked with minimizing the mean squared error in the prediction of observed outcomes. The information from this exercise is then passed to the next layer. This process is repeated until convergence is achieved or the iteration reaches the specified maximum number. After training, inputting the same unit into the networks of (for example) the meta-learner model of interest produces predictions to estimate the CATE for each unit, as shown in Figure 1.

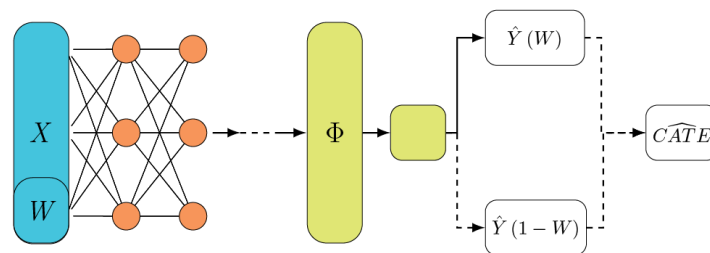


Figure 1: Estimating CATE Using Deep Neural Networks

4.1 Neural network architecture, model training, and treatment effects estimation

Deep neural networks (that is, deep learning) can be employed to estimate the CATEs and PATT from datasets by using Python libraries such as **Keras3** (Chollet et al., 2015) and **TensorFlow** (Abadi et al., 2015). The deep learning estimation procedure for causal inference consists of four steps. First, after loading and splitting their dataset into training and test sets, researchers need to build their neural network model for conducting the necessary estimation exercise. This entails defining their deep neural network architecture which involves specifying layers, activation functions, and the overall structure. Second, once the model is defined, researchers must compile it by specifying an optimizer for updating the model's weights to minimize the loss, define a loss function, and choose metrics.

While numerous optimization algorithms such as Adam, Stochastic Gradient Descent, AdaGrad, or RMSprop can be employed to update the model's weights for loss minimization, the applications presented below focus on using Adam (Adaptive Moment Estimation) for deep learning estimation of the CATE and PATT. Adam is an adaptive learning rate algorithm that leverages past gradient information to accelerate convergence, handle sparse gradients, dynamically adjust the learning rate for each individual parameter within a model, and minimize hyperparameter tuning. Apart from choosing an optimization algorithm, a loss function appropriate for the task must be defined, and metrics must be chosen to monitor during training and evaluation.

Third, after compiling the model, researchers need to train the model. This is implemented by fitting the model to the training data which involves setting the number of times the model iterates over the entire training dataset (denoted as epochs), choosing the number of samples processed before updating model weights (that is, batch_size), and setting the proportion of the training data used for validation during training via validation_split. Fourth, after training the model, researchers can proceed to estimate causal effects such as CATEs in the held-out test data. Since the loss function only minimizes the factual error to estimate $\hat{Y}$ (this is because we only observe one potential outcome for each unit), one has to artificially toggle the treatment to obtain $\hat{Y}(1)$ and $\hat{Y}(0)$ for each unit. After doing so, one can plug in the predictions to calculate the predicted CATEs for the meta-learner model of interest. To save space, we discuss the procedure and code for estimating the PATT in settings with treatment noncompliance in Section 7 below.

5 Loading the DeepLearningCausal package and datasets

5.1 Functions, Reticulate, TensorFlow and Keras

Estimating the CATEs and PATT via deep neural networks requires the use of libraries in Python while also simultaneously using R code and modules. Our **DeepLearningCausal** package allows users to leverage Python libraries within the R programming environment and seamlessly use Python's robust deep learning ecosystem in their R session. To see how, users need to first install the **DeepLearningCausal** package from CRAN via `install.packages("DeepLearningCausal")`, and from its GitHub repository ([hknd23/DeepLearningCausal](https://github.com/hknd23/DeepLearningCausal)). The **DeepLearningCausal** package automatically installs the **reticulate** R package that facilitates interoperability between Python and R, and it also automatically installs the **TensorFlow** and **Keras3** R packages as dependencies.

TensorFlow and **Keras3** allow for functions from the deep learning Python libraries **TensorFlow** and **Keras**(3.0) to work in tandem with R. However, given that the R packages only provide the interface, users must have the Python version of **TensorFlow** and **Keras** installed for the packages to work properly through reticulate. Hence, to streamline the installation process, users must call the function `python_ready()` to create a virtual environment for Python, and check for as well as install the necessary Python versions of the packages (TensorFlow, Keras3, NumPy). `python_ready()` thus ensures that both the R and Python versions of TensorFlow and Keras3 are installed. After installation of our package from CRAN, users can import the package by calling `library(DeepLearningCausal)` to import the functions in the package. Table 1 lists the functions in **DeepLearningCausal**:

Table 1: Brief description of functions in **DeepLearningCausal**

Function	Description
<code>metalearner_deeplearning()</code>	Deep neural network estimation of CATEs for S, T, X, and R-learner using reticulate, tensorflow and keras3
<code>pattc_deeplearning()</code>	Deep neural network estimation of PATT using reticulate, tensorflow and keras3
<code>conformal_plot()</code>	Conformal prediction for inference of ITEs from S and T-learner
<code>hte_plot()</code>	Heterogeneous Treatment Effects plot from meta-learner and PATT-C models
<code>metalearner_ensemble()</code>	Weighted ensemble learning estimation of CATEs for S, T, X, and R-learner using Super Learner (GitHub repo)
<code>metalearner_neural()</code>	Deep neural network estimation of CATEs for S, T, X, and R-learner using R neural net (GitHub repo)
<code>pattc_ensemble()</code>	Weighted ensemble learning estimation of PATT using Super Learner (GitHub repo)
<code>pattc_neural()</code>	Deep neural network estimation of PATT using R neural net (GitHub repo)

Our package first includes functions (see Table 1) which enable users to employ deep neural networks using **reticulate**, **TensorFlow** and **Keras3**, which gives them access to Python's deep learning libraries in their R session. Furthermore, our package includes the `hte_plot()` function that allows users to plot heterogeneous treatment effects and apply the conformal prediction (CP) procedure for generating predictive intervals for ITEs from two meta-learner models. All visualizations produced by the package are generated using **ggplot2** (Wickham, 2016).

The **DeepLearningCausal** package provides users with two additional options for estimating CATEs from the meta-learner models and the PATT in settings with treatment noncompliance. First, if users prefer to employ deep learning estimation without enabling Python, our package provides two functions that allow them to implement deep neural network estimation of the CATE and PATT by using the R neural net package: `metalearner_neural()` and `pattc_neural()`. Second, to employ the functions in our package to estimate the meta-learner CATEs and the PATT from the PATT-C model by using weighted ensemble learning, users need to install the **SuperLearner** package from CRAN:

5.2 Datasets

Two datasets are included as examples to illustrate the applicability of the functions in our package: the survey experiment sample (`exp_data_full`) and the commensurate representative population-level survey response dataset (`pop_data_full`). nts are randomly assigned to the control group or the binary . These two datasets are described in the package's GitHub repository ([hknd23/DeepLearningCausal](https://github.com/hknd23/DeepLearningCausal)). The survey experiment sample is generated from responses to survey questions fielded online to 728 respondents in India in 2022. After reading a vignette about a hypothetical crisis between their country and a foreign adversary, respondents are randomly assigned to the control group or the binary *strong leader* "treatment" variable coded as 1 for those exposed to a hawkish policy prescription for the said international crisis by a strong populist leader as opposed to a non-populist leader in their country.

```
data("exp_data_full")
str(exp_data_full)

#> 'data.frame': 514 obs. of 16 variables:
#> $ strong_leader : int 1 0 1 1 1 1 1 1 1 0 ...
#> $ support_war : int 0 1 0 1 1 0 1 1 1 1 ...
#> $ compliance : int 1 0 1 1 0 1 0 0 1 0 ...
```

```

#> $ female      : num  1 0 1 1 1 0 1 0 1 1 ...
#> $ age          : int   38 28 25 32 21 60 38 44 31 20 ...
#> $ income       : int    7 2 5 7 9 7 4 3 4 2 ...
#> $ religion     : int    1 6 1 9 1 6 5 6 6 9 ...
#> $ hindu        : int    0 1 0 0 0 1 0 1 1 0 ...
#> $ practicing_religion: int  4 1 1 1 4 3 2 1 1 1 ...
#> $ education    : int    4 3 4 4 8 7 4 3 4 4 ...
#> $ political_ideology : int  4 5 6 10 5 3 3 10 9 5 ...
#> $ employment  : int    1 6 2 2 5 1 1 2 3 6 ...
#> $ employed     : int    1 0 0 0 0 1 1 0 0 0 ...
#> $ marital_status : int    6 6 6 6 6 1 1 1 1 6 ...
#> $ married      : int    0 0 0 0 0 1 1 1 1 0 ...
#> $ job_loss     : int    1 1 4 1 1 1 4 4 2 1 ...

data("pop_data_full")
str(pop_data_full)

#> 'data.frame':  11813 obs. of  17 variables:
#> $ strong_leader : num  0 0 NA 0 0 NA 1 0 0 1 ...
#> $ support_war   : int   NA NA NA NA NA NA NA NA NA NA ...
#> $ compliance    : int    0 0 0 0 0 0 1 0 0 1 ...
#> $ female       : num  0 1 0 1 1 0 1 0 0 0 ...
#> $ age          : int   48 38 42 26 39 24 28 20 42 68 ...
#> $ income       : int    3 6 3 2 4 3 4 4 4 3 ...
#> $ religion     : num  31 31 31 31 31 31 31 31 31 31 ...
#> $ hindu        : int    1 1 1 1 1 1 1 1 1 1 ...
#> $ practicing_religion: int  2 2 4 2 2 2 3 4 4 2 ...
#> $ education    : int    2 2 6 1 1 1 8 8 7 7 ...
#> $ political_ideology : int  5 5 5 5 5 -1 5 5 5 5 ...
#> $ employment  : int    3 6 1 2 6 3 6 7 1 5 ...
#> $ employed     : int    1 0 1 1 0 1 0 0 1 0 ...
#> $ marital_status : int    1 1 1 1 1 6 1 6 1 1 ...
#> $ married      : int    1 1 1 1 1 0 1 0 1 1 ...
#> $ job_loss     : int   NA NA NA NA NA NA NA NA NA NA ...
#> $ year         : int   2001 2001 2001 2001 2001 2001 2001 2001 2001 2001 ...

```

After random assignment to treatment or control, respondents are asked whether or not they support fighting a war against a foreign adversary. This generates the binary *support war* outcome measure. We record the vignette screen time latency and conduct factual manipulation checks to generate the binary *compliance* variable coded as 1 for respondents who understood and followed the instructions associated with the *strong leader* treatment and thus complied with this treatment; it is coded 0 for noncompliers. The survey experiment sample also includes confounders that operationalize the respondents' demographic features, dispositional traits, and attitudinal characteristics.

The second dataset in the package (`pop_data_full`) is the commensurate representative population-level World Values Survey response data from India for 1995, 2001, 2006, 2012, and 2022. This survey response data includes the binary *support war* outcome variable that mirrors the binary "support war" outcome measure in the India survey experiment sample. It also includes responses to a question about the respondents' preference for a *strong leader* that is compatible with the "strong leader" treatment variable in the survey experiment sample. The binary *compliance* indicator in this data is 1 for those who responded to the strong leader preference question as this indicates compliance with the treatment proxy; it is 0 for respondents who did *not* answer this question as non-response indicates treatment noncompliance. The population-level survey response data include the same demographic, dispositional, and attitudinal covariates of respondents included in the survey experiment sample.

The following formula that specifies each model's (S-Learner, T-Learner, X-Learner, R-Learner, PATT-C) outcome measure *support war* and list of covariates drawn from the package's two example datasets helps us illustrate the applicability of the functions in **DeepLearningCausal**:

```

response_formula <- support_war ~ age + female + education + income + employed +
  job_loss + hindu + political_ideology

```


6 Deep learning estimation of meta-learner models

6.1 Deep neural networks for meta-learners

After initializing Python and setting up the virtual environment with `python_ready()`, the `metalearner_deeplearning()` function in **DeepLearningCausal** enables users to use TensorFlow and Keras3 to estimate CATEs from the S, T, X, and R-learner models. To save space, we focus on demonstrating below the applicability of the `metalearner_deeplearning()` function for the S-learner model by using the survey experiment dataset in our package. We also briefly present below some results from the deep learning-estimated T and X-learner models. More details about the procedure and results from the deep learning-estimated T, X and R-learner models by calling the `metalearner_deeplearning()` function are provided in the package's GitHub repository.

To begin with, the `metalearner_deeplearning()` function displayed below indicates that users can specify the arguments `train.data` and `test.data` to separately train the meta-learners on their training data and estimate CATEs with their test data. If a single dataset is specified, then the model will use cross-validation to train the meta-learners and estimate CATEs. Hence, for a single dataset, users can specify `nfolds`, which defines the number of folds to split data for cross-validation. Next, the `metalearner_deeplearning()` function permits users to specify the covariates in the S-learner model via `cov.formula = response.formula`. This includes the *support war* outcome measure and other confounders from the India survey experiment data, which is specified with `data = exp_data_full`. The treatment variable from this data is specified with `treat.var = "strong_leader"`. Users must specify `meta.learner.type = "S.Learner"` to estimate the S-learner model or specify `meta.learner.type =` to the meta-learner they want to estimate.

The arguments indicate that the `metalearner_deeplearning()` function permits users to substantially customize their deep neural network architecture for estimating CATEs from the meta-learner models. For instance, the `hidden.layer` argument permits users to specify the number of hidden layers and the number of neurons in each layer, while `hidden.activation` is the activation function for the hidden layers. Users can either specify a single value to use one activation function for all hidden layers (`hidden.activation = "relu"`) or a vector to set different activation functions for each hidden layer (`hidden.activation = c("relu", "tanh")`). Users can also specify arguments in `metalearner_deeplearning()` based on the outcome variable. For a binary outcome which involves a classification model, the output layer's activation function is `output.activation` while `output.units` is the number of units for the output layer. For regression models, the output layer's activation function is `output.activation = "linear"`. The argument `loss = "binary_crossentropy"` allows users to define the loss function for classification models, while `loss = "mean_squared_error"` defines the loss for regression models.

The arguments to reconfigure and train the deep neural networks for estimation include the optimization algorithm (options include "adam", "adagrad", "rmsprop", "sgd"), the number of epochs or iterations for a full pass of the data through the network to update weights, the size of each batch (`batch_size`) to split the training data, and `verbose` for monitoring the training progress. The arguments for metrics evaluation include `metrics = "accuracy"` for classification models and `metrics = "mean_squared_error"` for regression models. Arguments to specify the tuning of hyperparameters to monitor and mitigate overfitting include: `validation_split` that checks for overfitting, `patience` that stops the model early if overfitting is detected, and `dropout_rate` that mitigates overfitting to specify the proportion of neurons in each layer that can be dropped during training.

Next, the `plot(slearner_deep$m1_model_history[[1]])` in our **DeepCausalLearning** package allows users to illustrate the trace plot of the loss function and accuracy plots from the S-learner model. Moreover, as illustrated below, if the validation split is specified, then the trace and accuracy plots obtained from `plot(slearner_deep$m1_model_history[[1]])` will help users visualize how well the deep neural network model is learning and identifying potential problems such as overfitting for both the training and validation datasets:

```
plot(slearner_deep$m1_model_history[[1]])
```

The bottom half of Figure 2 illustrates the trace plot of the loss function for the S-learner. This plot shows model performance through loss after each epoch. The y-axis of the plot shows the loss value (e.g., mean squared error, cross-entropy) while the x-axis indicates the training progress (e.g., epochs, iterations). The said trace plot figure reveals that the loss decreases steadily and smoothly on *both* the training and validation sets before flattening out. This means that the neural network model that we trained is learning effectively and generalizing well to new data.

The accuracy plot at the top half of Figure 2 shows how the model's accuracy changes over time (epochs or training steps) on the training dataset and a separate validation dataset. The x-axis of the

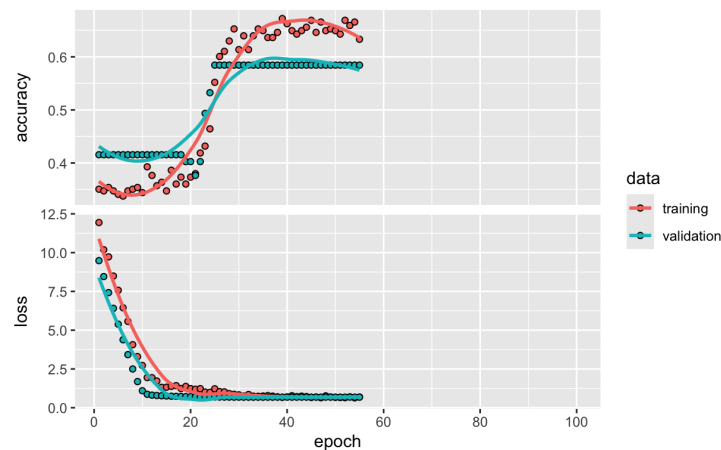


Figure 2: Accuracy Plot and Trace Plot of the Loss Function for the S-Learner

accuracy plot represents the training epochs or iterations while the y-axis represents the accuracy score. Note that the gap between training and validation accuracy in the accuracy plot decreases, which indicates stable learning and an increase in accuracy. Overall, the steady decrease in loss but increase in accuracy illustrated in Figure 2 indicates stable learning. Furthermore, the trace and accuracy plots were cut off at the 54th epoch, meaning that the model stopped early to avoid overfitting. Finally, in addition to the S-learner, the function `metalearner_deeplearning()` also allows users to build and customize the deep neural network architecture for estimating the CATEs from the T, X, and R-learner models, which is described in the GitHub repository of our package.

6.2 CATEs, Conformal Inference, Heterogeneous Treatment Effects

Users can view the results of their deep neural network-estimated S and T-learner with `print(slearner_deep)` and `print(tlearner_deep)` respectively:

```
#> Method:
#> Deep Learning S.Learner
#> Formula:
#> support_war ~ age + female + education + income + employed + job_loss + hindu + political_ideology
#> Treatment Variable: strong_leader
#> CATEs percentiles:
#>      10%      25%      50%      75%      90%
#> -0.0017127156 -0.0017046928 -0.0015885234 -0.0009122491 -0.0009065032

#> Method:
#> Deep Learning T.Learner
#> Formula:
#> support_war ~ age + female + education + income + employed + job_loss + hindu + political_ideology
#> Treatment Variable: strong_leader
#> CATEs percentiles:
#>      10%      25%      50%      75%      90%
#> -0.24167932 -0.16082895 -0.04291850  0.02078736  0.05503685
```

The reported results provide information about the estimated CATE percentiles from the meta-learner model of interest. Next, the `conformal_plot()` function in **DeepLearningCausal** implements the conformal prediction framework to conduct direct inference of individual treatment effects (ITEs) obtained from the deep learning-estimated S and T-learner models. To see how, first note that individual treatment effects (ITEs) from the S-learner and T-learner are calculated as the difference between a subject's predicted outcome with and without a specific treatment.

The conformal intervals for the deep learning S and T-learner are generated directly within the `metalearner_deeplearning()` function when the argument `conformal = TRUE` is specified. This is displayed below for just the S-learner to save space:

Setting `conformal = TRUE` constructs uncertainty intervals around each estimated individual treatment effect (ITE). The argument `alpha = 0.1` specifies the miscoverage rate, corresponding to

approximately 90% nominal coverage. `calib_frac = 0.5` divides the test sample into two parts: one is used for model prediction, and the other serves as a calibration set to compute residuals that quantify the typical deviation between model predictions and observed outcomes in finite samples.

The conformal adjustment is then determined by the *weighted quantile* of these residuals, where the weights are derived from the estimated propensity scores to correct for treatment–control imbalance. Setting `prob_bound = TRUE` ensures that the predicted probabilities and resulting conformal intervals are restricted to the $[-1, 1]$ feasible range for binary outcomes. After estimation, the function stores the conformal intervals in conformal (S-learner) and conformal (T-learner) which can be visualized using:

```
conformal_plot(slearner_deep, binary.outcome = TRUE, prop = .2, seed = 1234)
conformal_plot(tlearner_deep, binary.outcome = TRUE, prop = .2, seed = 1234)
```

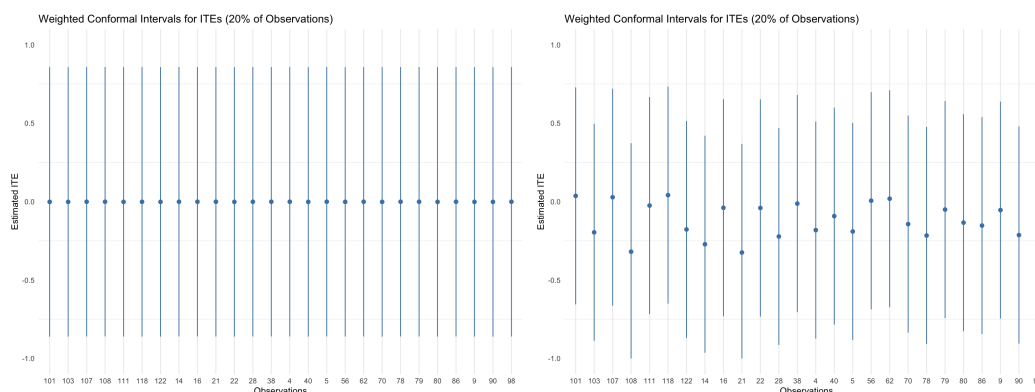


Figure 3: Conformal Predictions from Deep Neural Network-Based Meta Learners

The argument specifies that the outcome variable is binary, which constrains the conformal intervals to the logical probability bounds $[-1, 1]$. indicates that 20% of the test observations are randomly selected for display in the plot to maintain readability while preserving representativeness of the overall distribution. Figure 3 illustrates these intervals for the S-learner and T-learner. Each vertical line in these figures represents an observation’s uncertainty range, and the length of the line reflects the model’s local calibration residuals—wider intervals indicating greater uncertainty in the predicted ITEs.

Unlike the S-learner, which estimates both potential outcomes within a single network, the T-learner fits separate models for the treated and control groups. As a result, its conformal intervals often appear narrower and more heterogeneous across observations, reflecting that uncertainty is estimated conditional on treatment status rather than pooled across all units. By contrast, the S-learner tends to yield more conservative coverage due to shared outcome modeling, while the T-learner captures finer treatment-specific variation in individual effects.

Apart from the S- and T-learner models, our package permits users to customize their deep neural network architecture for deep learning estimation of the X- and R-learner models using **reticulate**, **TensorFlow**, and **Keras3**. The procedure and results for the R-learner are presented in the package’s GitHub repository. After the X-learner model is estimated via their customized deep neural network architecture, users can view the model’s results with `print(xlearner_deep)`:

```
print(xlearner_deep)

#> Method:
#> Deep Learning  X.Learner
#> Formula:
#> support_war ~ age + female + education + income + employed +   job_loss + hindu + political_ideology
#> Treatment Variable:  strong_leader
#> CATEs percentiles:
#>      10%      25%      50%      75%      90%
#> 0.2017969 0.2017969 0.2017969 0.2017969 0.2017969
```

These results report the deep neural network-estimated CATE percentiles from the X-learner model. The conformal prediction framework described earlier can also be employed to compute and illustrate conformal prediction (CP) intervals for individual treatment effects (ITEs) estimated from the two pseudo-outcome meta-learner models: the X and R-learner. The procedure as well as code

for extracting CP intervals for the estimated ITEs from pseudo-outcome meta-learners have been developed and described by [Alaa et al. \(2023\)](#) and are thus not summarized here to save space.

As such, the estimated CATEs from all the meta-learner models in our package are stored in the element `CATEs`. The DNN-estimated CATEs from the (i) X-learner stored in `CATEs` can be called with `xlearner_deep$CATEs`, (ii) S-learner can be called with `slearner_deep$CATEs`, and (iii) T-learner can be called with `tlearner_deep$CATEs`. Users can thus employ the stored CATEs from all these three DNN-estimated meta-learner models (and the R-learner model) to illustrate and analyze treatment effect heterogeneity across distinct subgroups in their sample by using the `hte_plot()` function in our package. As displayed below, the `hte_plot()` function gives users the flexibility to enter any number of covariates for which she or he would like to assess heterogeneous treatment effects.

By default, the cutoff point in `hte_plot()` is set as the median of the interval and continuous covariates, and classifies binary variables (e.g., gender) into two categories (female and male). Using `hte_plot()`, we illustrate below the heterogeneous treatment effects (HTEs) from the estimated S, T, and X-learner model for the following subgroups of respondents in our experimental sample: male and female, employed and unemployed, and left-leaning versus right-leaning political ideology.

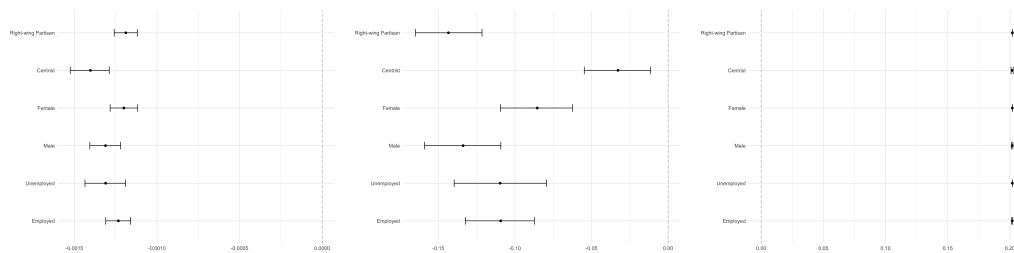


Figure 4: HTEs from Deep Neural Network-Based Meta Learners

The HTE estimates show some evidence of treatment effect heterogeneity obtained from the deep neural network-based S and X-learner but not the T-learner model. We also extract and illustrate the HTE plots obtained from the deep neural network estimated R-learner model in the GitHub repository of our package.

6.3 Comparing Meta-Learner Treatment Effects and Supplementary Analysis

The **DeepLearningCausal** package also enables users to plot the histograms and pairwise correlations of the estimated individual treatment effects from the deep neural network-estimated meta-learner models by using the **psych** package's `pairs.panels()`:

```
allCATEs_deep <- data.frame("S_learner" = slearner_deep$CATEs,
                           "T_learner" = tlearner_deep$CATEs,
                           "X_learner" = xlearner_deep$CATEs,
                           "R_learner" = rlearner_deep$CATEs)
psych::pairs.panels(allCATEs_deep, breaks = 30)
```

These figures suggest that the correlation of the estimated individual treatment effects across the meta-learner models estimated via deep neural networks using *reticulate*, *TensorFlow*, and *Keras3* is weak and inconsistent. The individual treatment effects obtained from the deep neural network-based T-learner are also negatively correlated with those extracted from the S-learner and X-learner models.

To save space, we focus on reporting the meta-learner CATEs obtained from deep neural network estimation using *reticulate*, *TensorFlow*, and *Keras3*. However, the `metalearner_ensemble()` function (see Table 1) also enables users to estimate CATEs from the four meta-learner models via deep neural networks using R neural net. The function in our package permits users to employ weighted ensemble learning via the Super Learner approach for estimating CATEs from the S-, T-, X-, and R-learner. The procedure for estimating the meta-learner models using the two aforementioned functions is described in the package's GitHub repository ([hknd23/DeepLearningCausal](https://github.com/hknd23/DeepLearningCausal)).

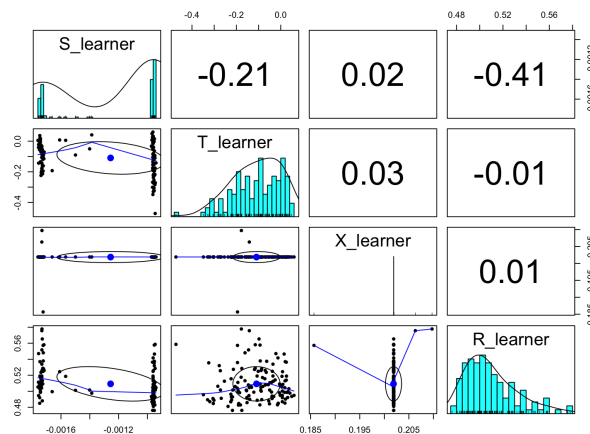


Figure 5: Treatment Effects Distribution and Correlations: Deep Neural Network-based Meta Learners

7 Deep learning for estimating PATT with treatment noncompliance

7.1 Deep neural networks for estimating the PATT

The `pattc_deeplearning()` function in the **DeepLearningCausal** package enables users to estimate the PATT by employing the survey experiment data and the population-level survey data in our package. To start with, the argument `exp.data = exp_data_full` in the aforementioned function specifies the data input for the experimental sample which serves as the training data. `pop.data = pop_data_full` is the data input for the representative population-level data (this is the World Values Survey in the package) which serves as the test data. `exp.data = exp_data_full` and `pop.data = pop_data_full` are both employed to estimate the PATT in settings with treatment noncompliance.

The argument `response.formula = response_formula` permits users to specify the outcome variable (e.g., *support war*) and the confounders in the PATT-C model. `treat.var` and `compl.var` allow users to specify the treatment variable and the compliance variable, respectively.

The arguments displayed above reveal that the `pattc_deeplearning()` function provides users with substantial options to customize and tune the hyperparameters of both the compliance and response models separately in the PATT-C estimator to obtain the PATT in settings with treatment noncompliance. These arguments include for example `exp.data`, `pop.data` and `treat.var` that users can employ to specify their data inputs. `compl.hidden.layer` permits users to specify the number of layers and neurons in each layer for the compliance model that users can specify, while `response.hidden.layer` is employed to specify the number of layers and neurons in each layer for the response model. The argument `compl.hidden.activation` is the activation function for the hidden layers (e.g., “relu”) in the compliance model for each hidden layer, and `response.hidden.activation` denotes the activation function for the hidden layers in the response model.

Users can also specify the arguments for the response model based on the outcome variable. For a binary outcome variable, the activation function of the response model’s output layer is `response.output_activation` which is set to “sigmoid”. The argument `response.output_units` is the number of units for the response model’s output layer. Next, users can define the response model’s loss function as `response.loss = "binary_crossentropy"` for classification and `response.loss = "mean_squared_error"` for regression. The argument `compl.algorithm` and `response.algorithm` denote the optimization algorithms (e.g., “adam”, “adagrad”, “rmsprop”) for the compliance and response models, respectively.

`compl.epoch` and `response.epoch` denote iterations of the full pass of the data through the network to update weights for the compliance and response models, respectively. The argument `batch_size` permits users to split the training data, while `verbose` allows users to monitor the training progress. The arguments for the evaluation metrics are `response.metrics = "accuracy"` for when the response model is a classification model and `response.metrics = "mean_squared_error"` for a regression model. Arguments that specify the tuning of hyperparameters to monitor and mitigate overfitting for deep neural networks when estimating the PATT are `compl.validation_split` for the compliance model, and `response.validation_split` for the response model. Users can specify `compl.patience` and `response.patience` to stop the model early if overfitting is detected for the compliance and response models. `compl.dropout_rate` permits users to set the proportion of neurons in each layer that

can be dropped during training of the compliance model to mitigate overfitting. The argument `nboot = 1000` sets the number of bootstrap samples to 1,000. Finally, `plot(deeppattc$complier_history)` and `plot(deeppattc$response_history)` can be employed to illustrate the trace plots of the loss and metric functions of the complier and response models used for estimating the PATT. This enables users to assess whether their deep neural network architecture is learning and identifying overfitting when estimating PATT in data with treatment noncompliance.

Once the deep neural networks are trained on the experimental data, users can call `deeppattc$pop_counterfactual` to extract the predicted probabilities of the outcome measure for population treatment compliers *if* they are all in the treated group (the counterfactual treated group) and *if* they are all in the control group (the counterfactual control group). They can then use `plot(deeppattc)` to illustrate the distribution of these predicted probabilities of the outcome measure for population compliers in the counterfactual treated group (blue histogram) and counterfactual control group (red histogram):

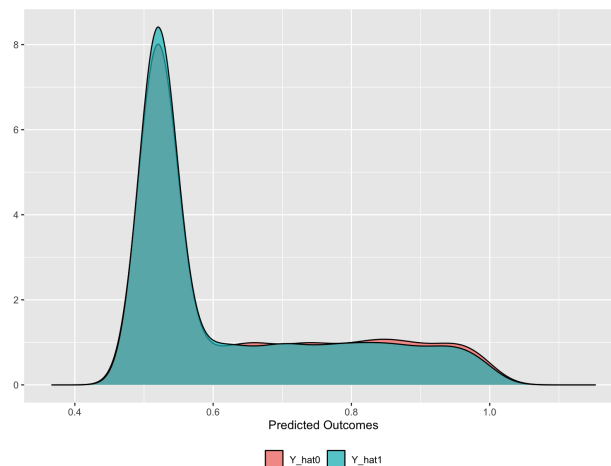


Figure 6: Counterfactual Predictions Deep Learning PATT-C

This figure indicates that the share of population compliers in the counterfactual treated group who support war is not different than their peers in the counterfactual control group. The predicted probabilities of the outcomes obtained from this exercise can also be employed for additional analyses, such as identifying members in the target population who are most receptive to the treatment.

7.2 Heterogeneous Treatment Effects and supplementary analysis

Our **DeepLearningCausal** package also enables users to extract and illustrate heterogeneous treatment effects for specified subgroups in the target population once the PATT has been estimated from datasets with treatment noncompliance using their deep neural network architecture. Heterogeneous treatment effects are estimated by taking differences across response surfaces for a given covariate, and response surfaces are estimated with neural network-based mean predictions when deep neural networks are employed for estimating the PATT. Users can call the `hte_plot()` function to extract and illustrate the heterogeneous treatment effects for the deep neural network-estimated PATT-C model. The HTE plot is shown below for the *support war* outcome measure for the subgroups listed earlier:

```
hte_plot(deeppattc, selected_vars = c( "employed", "female", "political_ideology"),
         cut_points = c(.5,.5,5), custom_labels= c( "Employed", "Unemployed",
                                                    "Male", "Female", "Centrist",
                                                    "Right-wing Partisan"))
```

In addition to deep learning estimation of the PATT in settings with treatment noncompliance, our **DeepLearningCausal** package also allows users to employ the R **neuralnet** package for estimating the PATT (from the PATT-C model) via deep neural networks with `pattc_neural()`. Additionally, the `pattc_ensemble()` function in **DeepLearningCausal** permits users to estimate the PATT from datasets that exhibit treatment noncompliance by using weighted ensemble learning from `library(SuperLearner)` that focuses on four candidate machine learning algorithms mentioned earlier. The procedure for estimating the PATT in settings with treatment noncompliance via deep neural networks (from R neural net) and weighted ensemble learning using the super learner is described in our package's GitHub repository ([hknd23/DeepLearningCausal](https://github.com/hknd23/DeepLearningCausal)).

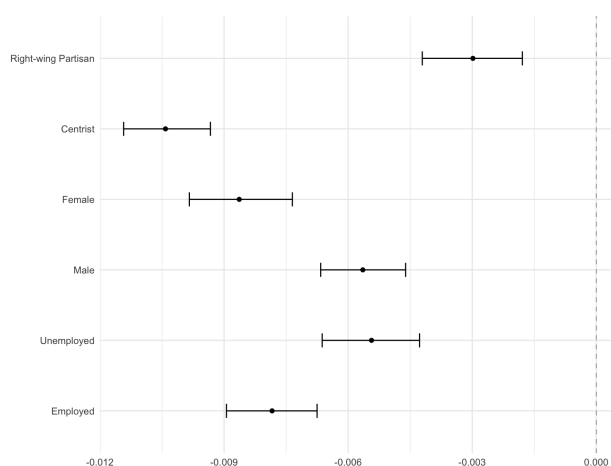


Figure 7: HTEs for Selected Covariates of Interest

8 Conclusion

The rapid surge in research on causal machine learning has led to the development of statistical software packages that enable users to employ conventional machine learning, weighted ensemble learning or targeted learning to estimate a wide variety of causal models (e.g., van der Laan and Rose, 2011; Nie and Wager, 2021; Knaus, 2022; Hu and Ji, 2022; Zhao and Liu, 2023). More recent research on causal inference has, however, sought to demonstrate how deep causal learning, which leverages deep neural networks, can address challenges stemming from high-dimensional data, unobserved confounders, and selection bias (Goodfellow et al., 2016; Li et al., 2024; Koch et al., 2024). Yet barring some open-source Python code that introduces deep learning for causal inference (Koch et al., 2024), there is, to our knowledge, no R package that allows users to employ deep learning methods for estimating treatment effects in settings with noncompliance.

Our R package addresses this lacuna by offering functions that enable the use of deep learning—deep neural networks—to estimate the meta-learner CATEs in samples and the PATT from a newly developed causal estimand that addresses treatment noncompliance (Künzel et al., 2019; Ottoboni and Populos, 2020). Another feature of our package is that it enables users to import Python’s deep learning modules directly into their R session to construct and customize their deep neural network architecture for estimating treatment effects. It also permits users to apply the conformal prediction (CP) procedure to generate predictive intervals for ITEs from a set of conditional mean regression meta-learner models.

Future work can build on the general deep neural network architecture for causal inference in numerous ways. For instance, R packages can be developed that allow users to employ specific deep neural network models such as Convolutional Neural Networks, Recurrent Neural Networks, Variational Autoencoders, and Conditional Generative Adversarial Networks. These specific models offer several benefits for causal inference such as learning non-linear relationships between features and counterfactual outcomes, imputation of missing counterfactual outcomes, learning latent representations of data, predicting the conditional outcome and propensity score, and extending causal inference to text data, networks, and images. The deep learning-estimated causal models in our R package can be applied to high-dimensional data, large-scale RCTs conducted for target populations, and observational data with unknown interventions and long-range temporal dependencies. Doing so will expand the application of the causal models in the package across several fields that employ randomized trials, panel data, or high-dimensional observational data.

References

- M. Abadi et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. URL <https://www.tensorflow.org/>. accessed 2025-11-05. [p104, 111]
- A. Alaa and M. Van Der Schaar. Limits of estimating heterogeneous treatment effects: Guidelines for practical algorithm design. *Proceedings of the 35th International Conference on Machine Learning*, 80: 129–138, 2018. URL <https://proceedings.mlr.press/v80/aaa18a.html>. [p103, 107]
- A. Alaa, Z. Ahmed, and M. van der Laan. Conformal meta-learners for predictive inference of

- individual treatment effects. *Proceedings of the 37th Conference on Neural Information Processing Systems*, 48:1–22, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/94ab02a30b0e4a692a42ccd0b4c55399-Paper-Conference.pdf. [p107, 117]
- J. J. Allaire and Y. Tang. *tensorflow: R Interface to 'TensorFlow'*, 2024. URL <https://CRAN.R-project.org/package=tensorflow>. R package version 2.16.0. [p104]
- S. Athey and G. W. Imbens. Recursive partitioning for heterogeneous causal effects. *Proceedings of the National Academy of Sciences*, 113(27):7353–7360, 2016. URL <https://doi.org/10.1073/pnas.1510489113>. [p103]
- P. Bach, M. S. Kurz, V. Chernozhukov, M. Spindler, and S. Klaassen. DoubleML: An object-oriented implementation of double machine learning in R. *Journal of Statistical Software*, 108(3):1–56, 2024. URL <https://doi.org/10.18637/jss.v108.i03>. [p104]
- L. B. Balzer, M. L. Petersen, and M. J. van der Laan. Targeted estimation and inference for the sample average treatment effect in trials with and without pair-matching. *Statistics in Medicine*, 35(21):3717–3732, 2016. URL <https://doi.org/10.1002/sim.6965>. [p103, 104]
- K. Battocchi, E. Dillon, M. Hei, G. Lewis, P. Oka, M. Oprescu, and V. Syrgkanis. EconML: A Python Package for ML-Based Heterogeneous Treatment Effects Estimation, 2019. URL <https://github.com/py-why/EconML>. Version 0.x. [p104]
- V. Chernozhukov, D. Chetverikov, M. Demirer, E. Duflo, C. Hansen, W. Newey, and J. Robins. Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1):C1–C68, 2018. URL <https://doi.org/10.1111/ectj.12097>. [p104]
- F. Chollet et al. Keras, 2015. URL <https://keras.io>. accessed 2025-11-05. [p104, 111]
- D. Coffman and B. Griffin. *twangContinuous: Toolkit for Weighting and Analysis of Nonequivalent Groups - Continuous Exposures*, 2021. URL <https://CRAN.R-project.org/package=twangContinuous>. R package version 1.0.0. [p104]
- I. Díaz, N. Williams, K. Hoffman, and E. Schneck. Non-parametric causal effects based on longitudinal modified treatment policies. *Journal of the American Statistical Association*, 2021. URL <https://doi.org/10.1080/01621459.2021.1955691>. [p104]
- M. Farrell, T. Liang, and S. Misra. Deep neural networks for estimation and inference. *Econometrica*, 89(1):181–213, 2021. URL <https://doi.org/10.3982/ECTA16901>. [p103]
- A. N. Glynn and K. M. Quinn. An introduction to the augmented inverse propensity weighted estimator. *Political analysis*, 18(1):36–56, 2010. URL <https://doi.org/10.1093/pan/mpp036>. [p104]
- I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Cambridge, MA: MIT Press, 2016. URL <https://mitpress.mit.edu/9780262035613/deep-learning/>. [p104, 120]
- E. Hartman, R. Grieve, R. Ramsahai, and J. S. Sekhon. From sate to patt: Combining experimental with observational studies to estimate population treatment effects. *Journal of Royal Statistical Society Series A. Stat. Soc.*, 178(3):757–778, 2015. URL <https://doi.org/10.1111/rssa.12094>. [p103, 104]
- J. Hill. Bayesian nonparametric modeling for causal inference. *Computational and Graphical Statistics*, 20(1):217–240, 2011. URL <https://doi.org/10.1198/jcgs.2010.08162>. [p103]
- L. Hu and J. Ji. CimtX: An R package for causal inference with multiple treatments using observational data. *R Journal*, 14(3):213–230, 2022. URL <https://journal.r-project.org/articles/RJ-2022-058/>. [p104, 120]
- L. Hu, J. Ji, and F. Li. Estimating heterogeneous survival treatment effect in observational data using machine learning. *Statistics in Medicine*, 40(21):4691–4713, 2021. URL <https://doi.org/10.1002/sim.9090>. [p103]
- G. W. Imbens and D. Rubin. *Causal inference for statistics, social, and biomedical sciences: An introduction*. Cambridge, NY: Cambridge University Press, 2015. URL <https://doi.org/10.1017/CBO9781139025751>. [p103, 105]
- F. Johansson, U. Shalit, and D. Sontag. Learning representations for counterfactual inference. in. *Proceedings of the 35th International Conference on Machine Learning*, 48:3020–3029, 2016. URL <https://dl.acm.org/doi/10.5555/3045390.3045708>. [p104]

- G. Jonzon, M. Sachs, and E. Gabriel. Accessible computation of tight symbolic bounds on causal effects using an intuitive graphical interface. *R Journal*, 15(4):53–68, 2023. URL <https://journal.r-project.org/articles/RJ-2023-083/>. [p104]
- T. Kalinowski, J. J. Allaire, and F. Chollet. *keras3: R Interface to 'Keras'*, 2025. URL <https://CRAN.R-project.org/package=keras3>. R package version 1.4.0. [p104]
- M. Knaus. Double machine learning-based program evaluation under unconfoundedness. *The Econometrics Journal*, 25(3):602–627, 2022. URL <https://doi.org/10.1093/ectj/utac015>. [p104, 120]
- B. J. Koch, T. Sainburg, P. E. Geraldo, S. Jiang, Y. Sun, and J. G. Foster. A primer on deep learning for causal inference. *Sociological Methods and Research*, 54(2):1–51, 2024. URL <https://doi.org/10.1177/00491241241234866>. [p103, 104, 120]
- S. Künzel, J. Sekhon, P. Bickel, and B. Yu. Metalearners for estimating heterogeneous treatment effects using machine learning. *Proceedings of the National Academy of Sciences*, 116(1):4156–4165, 2019. URL <https://doi.org/10.1073/pnas.1804597116>. [p103, 104, 105, 107, 120]
- L. Lei and E. Candès. Conformal inference of counterfactuals and individual treatment effects. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 83(5):911–938, 2021. URL <https://doi.org/10.1111/rssb.12445>. [p107]
- Z. Li, X. Guo, and S. Qiang. A survey of deep causal models and their industrial applications. *Artificial Intelligence Review*, 57(1):298, 2024. URL <https://doi.org/10.1007/s10462-024-10886-0>. [p103, 120]
- X. Nie and S. Wager. Quasi-oracle estimation of heterogeneous treatment effects. *Biometrika*, 108(2): 299–319, 2021. URL <https://doi.org/10.1093/biomet/asaa076>. [p103, 104, 107, 108, 120]
- G. Okasa. Meta-learners for estimation of causal effects: Finite sample cross-fit performance. *arXiv preprint arXiv:2201.12692*, 2022. URL <https://doi.org/10.48550/arXiv.2201.12692>. [p104, 105]
- K. N. Ottoboni and J. V. Populos. Estimating population average treatment effects from experiments with noncompliance. *Journal of Causal Inference*, 8:108–130, 2020. URL <https://doi.org/10.1515/jci-2018-0035>. [p103, 104, 109, 110, 120]
- J. Pearl. *Causality*. Cambridge, NY: Cambridge University Press, 2009. URL <https://doi.org/10.1017/CBO9780511803161>. [p103]
- S. Powers, J. Qian, K. Jung, A. Schuler, N. Shah, T. Hastie, and R. Tibshirani. Some methods for heterogeneous treatment effect estimation in high dimensions. *Statistics in Medicine*, 37(11):671–679, 2018. URL <https://doi.org/10.1002/sim.7623>. [p103]
- M. Ratkovic. *SVMMatch: Causal Effect Estimation and Diagnostics with Support Vector Machines*, 2015. URL <https://CRAN.R-project.org/package=SVMMatch>. R package version 1.1. [p104]
- M. S. Schuler and S. Rose. Targeted maximum likelihood estimation for causal inference in observational studies. *American Journal of Epidemiology*, 185(1):65–73, 2019. URL <https://doi.org/10.1093/aje/kww165>. [p103, 104]
- S. Tikka. Identifying counterfactual queries with the R package cfid. *R Journal*, 15(2):330–343, 2023. URL <https://journal.r-project.org/articles/RJ-2023-053/>. [p103, 104]
- K. Ushey, J. J. Allaire, and Y. Tang. *reticulate: Interface to 'Python'*, 2025. URL <https://CRAN.R-project.org/package=reticulate>. R package version 1.42.0. [p104]
- M. J. van der Laan and S. Rose. *Targeted Learning: Causal Inference for Observational and Experimental Data*. Springer, NY: Springer Nature, 2011. URL <https://link.springer.com/book/10.1007/978-1-4419-9782-1>. [p103, 104, 120]
- A. Wager and S. Athey. Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*, 113(523):1228–1242, 2019. URL <https://doi.org/10.1080/01621459.2017.1319839>. [p107]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. URL <https://ggplot2.tidyverse.org>. [p112]
- J. Xu, T. Shinkre, and J. Brand. *htetree: Causal Inference with Tree-Based Machine Learning Algorithms*, 2023. URL <https://CRAN.R-project.org/package=htetree>. R package version 0.1.18. [p104]

- L. Yao, S. Li, Y. Li, M. Huai, J. Gao, and A. Zhang. Representation learning for treatment effect estimation from observational data. *Advances In Neural Information Processing Systems*, 31(1):2633–2643, 2018. URL <https://doi.org/10.1002/sim.7623>. [p103, 104]
- Y. Zhang, A. Bellot, and M. Schaar. Learning overlapping representations for the estimation of individualized treatment effects. *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pages 1005–1014, 2020. URL <https://doi.org/10.48550/arXiv.2001.04754>. [p104]
- Z. Zhang, Z. Hu, and C. Liu. Estimating the population average treatment effect in observational studies with choice-based sampling. *The International Journal of Biostatistics*, 15(1):965–972, 2019. URL <https://doi.org/10.1515/ijb-2018-0093>. [p103]
- Y. Zhao and Q. Liu. Causalm: Python package for causal inference machine learning. *SoftwareX*, 21: 101294, 2023. URL <https://doi.org/10.1016/j.softx.2022.101294>. [p104, 120]

Nguyen Khoi Huynh
Hitotsubashi Institute for Advanced StudyHitotsubashi University
Kunitachi, Tokyo, Japan
ORCID: 0000-0002-6234-7232
khoinguyen.huynh@r.hit-u.ac.jp

Yang Yang
Department of Political ScienceThe Pennsylvania State University
University Park, PA, USA
ORCID: 0009-0004-6135-4555
yky5272@psu.edu

Bumba Mukherjee
Department of Political ScienceThe Pennsylvania State University
University Park, PA, USA
ORCID: 0000-0002-3453-601X
bumba.mukherjee@psu.edu

SFDesign: An R Package for Space-Filling Designs

Shangkun Wang, Weijun Xie and V. Roshan Joseph

Abstract Space-filling designs play a crucial role in surrogate modeling, uncertainty quantification, and optimization of computer models. The SFDesign package offers a comprehensive suite of functions to construct various types of space-filling designs, including maximin designs, clustering-based designs, maximin Latin hypercube designs, maximum projection designs, and uniform designs. This paper provides a review of commonly used space-filling designs and highlights the advantages of SFDesign over the existing packages. We demonstrate how SFDesign produces high-quality designs with improved space-filling properties, making it a valuable tool for practitioners to design computer experiments.

1 Introduction

Space-filling designs are a class of experimental designs for deterministic computer experiments, which are widely used in science and engineering applications (Joseph, 2026). They ensure that the design points are spread evenly throughout the experimental region, minimizing gaps such that no region is left unexplored.

The motivation behind space-filling designs arises from the need to build accurate surrogate models, such as Gaussian process (or kriging) models for computer simulations (Santner et al., 2018; Gramacy, 2020). A good experimental design should place points strategically in the experimental region to capture variations of the response surface while avoiding unnecessary clustering. There are two general strategies for constructing designs: model-based designs and space-filling designs. Model-based designs, such as the maximum entropy design (Currin et al., 1991) and the integrated mean squared prediction design (Sacks et al., 1989), are often limited in practice due to their dependence on unknown correlation parameters in the Gaussian process model. In contrast, space-filling designs offer a model-free approach, providing a more robust framework for model fitting.

There are several space-filling criteria for assessing the quality of a design (Joseph, 2016). Among them, the minimax and maximin criteria are widely adopted geometric measures, both of which have asymptotic optimality justification for Gaussian process modeling (Johnson et al., 1990). However, in high-dimensional spaces, designs optimized using these criteria often exhibit poor projections in lower-dimensional subspaces. Latin hypercube design (LHD) addresses this issue by constraining that each factor of the design is evenly spaced throughout its range (McKay et al., 1979). It is often combined with the aforementioned minimax or maximin criteria to achieve both good coverage and projection property (Morris and Mitchell, 1995; van Dam, 2008). Maximum projection (MaxPro) designs further improve over maximinLHD, which ensures better projection properties across all subsets of factors. Uniformity is another desirable property for designs, particularly when the objective is integration or uncertainty quantification (Fang et al., 2005).

Several open-source packages are available to construct space-filling designs, especially Latin hypercube designs. The **lhs** package provides functionalities to generate random LHD and options to optimize the design based on various criteria such as maximin, S-optimality, etc. (Carnell, 2024). The **DiceDesign** package provides a comprehensive suite of design choices, including model-based designs via exchange algorithms or stochastic processes, as well as optimized LHDs using genetic algorithms or simulated annealing (Dupuy et al., 2015). **SLHD** package (Ba, 2015) uses simulated annealing to generate maximin LHD, which can also accommodate categorical factors using sliced LHD. The **maximin** package deals with maximin design (Sun and Gramacy, 2024). It improves an existing design by swapping out or optimizing the row with the minimum distance. For minimax designs, the **minimaxdesign** package employs clustering algorithms combined with particle swarm optimization to achieve globally optimal designs (Mak, 2021). For MaxPro design, the **MaxPro** package provides functionality to generate LHDs optimized for the MaxPro criterion, with further refinement made possible through continuous optimization (Ba and Joseph, 2018). In this article, we introduce the **SFDesign** package (Wang and Joseph, 2025) which offers considerable improvements over existing packages for generating maximin designs, MaxPro designs, uniform designs, and minimax designs. We also include a function to generate clustering-based designs, which is not available in other packages. A list of the major functions is presented in Table 1.

Name	Description
maximin.crit	maximin criteria
maxpro.crit	maximum projection criteria
uniform.crit	wrap-around discrepancy
cluster.error	clustering error
randomLHD	random LHD
maximinLHD	LHD optimized for maximin criteria
maxproLHD	LHD optimized for MaxPro criteria
uniformLHD	LHD optimized for wrap-around discrepancy
customLHD	LHD optimized for a user-defined criteria
clustering.design	designs generated by clustering algorithm
maximin.optim	a continuous optimizer for maximin-distance design
maxpro.optim	a continuous optimizer for MaxPro design
uniform.optim	a continuous optimizer for uniform design
continuous.optim	a continuous optimizer for user-defined criteria

Table 1: Major functions of the **SFDesign** package

2 Review of space-filling design

In this section, we introduce the notation and formally define various space-filling criteria. Let $\mathcal{X} \subset \mathbb{R}^p$ denote the experimental region of interest, which is typically assumed to be the unit hypercube $\mathcal{X} = [0, 1]^p$. Denote a design of size n by the design matrix $D_n = [x_1, \dots, x_n]^T$, where x_i^T represents the i th design point.

A maximin design aims to maximize the minimum pairwise distance between design points, ensuring that points are spread as far apart as possible. It can be obtained by maximizing

$$\phi_{\text{Mm}}(D_n) = \min_{i \neq j} \|x_i - x_j\|_2, \quad (1)$$

where $\|\cdot\|_2$ denotes the Euclidean norm. On the other hand, minimax designs focus on reducing the largest gap within the design space $\mathcal{X}$ by minimizing:

$$\phi_{\text{mM}}(D_n) = \max_{x \in \mathcal{X}} \min_i \|x - x_i\|_2.$$

This criterion ensures that the maximum distance from any point in the experimental region $\mathcal{X}$ to the nearest design point is minimized, effectively reducing the largest uncovered region in the space.

Factor sparsity is commonly observed in real physical systems; that is, only a few factors out of the numerous factors in the system may affect the response. Therefore, in deterministic simulations, we do not want replications when we project points to a lower subspace. A Latin hypercube design (LHD) ensures good one-dimensional projections (McKay et al., 1979). In an LHD, each column of D_n is a permutation of $[0.5/n, 1.5/n, \dots, (n-0.5)/n]$, which guarantees that the projections of the points in each dimension are evenly distributed. For an n -point design with p factors, there are n^p possible LHD configurations; however, not all of them are space-filling. To enhance the space-filling properties of LHDs, we can optimize them with respect to the maximin criterion (Morris and Mitchell, 1995). The optimization is usually done by minimizing a reciprocal distance criterion

$$\phi_{\text{rec}}(D_n) = \left\{ \frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{1}{\|x_i - x_j\|_2^r} \right\}^{1/r}, \quad (2)$$

which is proportional to $1/\phi_{\text{Mm}}(D_n)$ for large enough r .

Although maximin LHDs ensure good one-dimensional and full-dimensional space-filling properties, they may not be good when projected onto other subspaces. To improve the projection properties of a design for all possible subspaces, maximum projection (MaxPro) design was proposed (Joseph et al., 2015). It minimizes :

$$\phi_{\text{maxpro}}(D_n) = \left\{ \frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{1}{\prod_{l=1}^p (x_{il} - x_{jl})^2} \right\}^{1/p}. \quad (3)$$

Compared to (2), the only difference is that the Euclidean distance in the denominator is replaced by a

“product distance”, but this small change ensures good projections to all subspaces.

Uniformity of the points in the experimental region is another important property of space-filling designs, particularly in the context of mean estimation of a model output. Let $F_n(x) = 1/n \sum_{i=1}^n I(x_i \leq x)$ be the empirical distribution of the design points, where I is the indicator function in which the comparison is done component-wise. Let $f: \mathcal{X} \rightarrow \mathbb{R}$ denote the computer code. In order for the mean estimation error $|\mathbb{E}(f) - \sum_{i=1}^n f(x_i)/n|$ to be small for *any* f , it is ideal for $F_n(x)$ to closely approximate the uniform distribution. Several discrepancies have been proposed, such as star L_p -discrepancy and modified L_p -discrepancy, to measure the uniformity of designs (Fang et al., 2000). In this package, we utilize the wrap-around discrepancy because of its computational simplicity and nice projection properties (Hickernell, 1998):

$$\phi_{\text{wa}}(D_n) = -\left(\frac{4}{3}\right)^p + \frac{1}{n^2} \sum_{i,j=1}^n \prod_{k=1}^p \left[\frac{3}{2} - |x_{ik} - x_{jk}|(1 - |x_{ik} - x_{jk}|) \right]. \quad (4)$$

Finally, we introduce clustering-based designs, which is justified as a way to minimize an upper bound of the integrated prediction error of a Gaussian process. The clustering criterion is:

$$\phi_{\text{cluster}}(D_n) = \sum_{i=1}^n \int_{V_i} \|x - x_i\|^\alpha dx. \quad (5)$$

where $V_i, i = 1, \dots, n$ are the Voronoi regions. For $\alpha = 1$, the criterion is the same as K -median problem. For α large enough, the criterion can serve as a surrogate criterion for the minimax design (Mak and Joseph, 2018).

3 Optimization algorithms

In this section, we present the optimization algorithms employed in this package and illustrate their effectiveness through comparisons with existing packages.

3.1 Designs based on pairwise distance

Many of the criteria reviewed in Section 2.2, such as maximin criterion, MaxPro, and wrap-around discrepancy, are functions of the pairwise distances among design points. Motivated by this observation, we propose a general optimization framework for improving such criteria, with and without constraints to Latin hypercube designs.

For Latin hypercube designs, the optimization framework consists of two stages: an initial optimization phase using simulated annealing (Algorithm 1, Morris and Mitchell (1995)), followed by a deterministic local search refinement (Algorithm 2). Both stages iteratively enhance the design by swapping coordinates between selected pairs of design points.

In the first stage, design point pairs are randomly selected, and swaps are accepted with a certain probability, even if they do not immediately improve the criterion. This probabilistic acceptance helps prevent the algorithm from getting trapped in local optima, enabling more effective exploration of the design space.

In the second stage, a deterministic local search is performed by systematically scanning the design matrix. Every coordinate of each pair of design points is swapped, but the swap is only accepted if it leads to an improvement in the criterion. This greedy approach efficiently refines the design, converging towards a local optimum.

For designs that are not restricted to Latin hypercubes, the optimization process begins with the optimized LHD obtained through the above process or some initial design provided by the user and then transitions to continuous optimization using the low-storage BFGS algorithm from the NLOpt package (Liu and Nocedal, 1989; Johnson, 2008). If a surrogate objective function is used, such as in maximin designs, an additional simulated annealing step can be applied at the end to directly optimize the original criterion.

We present three specific designs implemented in the packages as follows.

Maximin design

Instead of directly optimizing the maximin criterion, we start by minimizing the average reciprocal distance in (2). The advantage of this criterion is that it helps differentiate between designs that have the same maximin measure. The reciprocal distance criterion is equivalent to the maximin criterion for

Algorithm 1 SA: simulated annealing for LHD

Require: Design matrix $D \in \mathbb{R}^{(n \times p)}$, criterion ϕ , Initial temperature T , temperature decay rate λ , Maximum number of iterations I_1 , Maximum number of iterations without improving the global best criterion I_2 .

- 1: Initialize the local and global best criterion value by the current criterion value $\phi_l = \phi_g = \phi(D)$.
- 2: Initialize the global best design $D_g = D$.
- 3: Initialize a flag variable for the loop $\text{flag} = \text{True}$.
- 4: Initialize a counter for total number of iteration $i = 0$.
- 5: **while** flag **do**
- 6: $\text{flag} = \text{False}$.
- 7: **while** $i_{\text{nochange}} < I_2$ **do**
- 8: **if** $i > I_1$ **then** break.
- 9: **end if**
- 10: Randomly select two rows l, m from $[1, \dots, n]$ and one column k from $[1, \dots, p]$.
- 11: Update counter $i = i + 1$.
- 12: Set $D_{\text{try}} = D$ and swap x_{lk} and x_{mk} in D_{try} .
- 13: Compute the criterion $\phi_{\text{try}} = \phi(D_{\text{try}})$.
- 14: **if** $\phi_{\text{try}} < \phi_g$ **then**
- 15: $\text{flag} = \text{False}$.
- 16: $\phi_g = \phi_{\text{try}}, D_g = D_{\text{try}}$.
- 17: $\phi_l = \phi_{\text{try}}, D_l = D_{\text{try}}$.
- 18: $i_{\text{nochange}} = 1$.
- 19: **else**
- 20: $i_{\text{nochange}} = i_{\text{nochange}} + 1$.
- 21: **if** $\phi_{\text{try}} < \phi_l$ **then**
- 22: $\text{flag} = \text{False}$.
- 23: $\phi_l = \phi_{\text{try}}, D_l = D_{\text{try}}$.
- 24: **else**
- 25: With probability $\exp(-(\phi_{\text{try}} - \phi_l)/T)$, $\phi_l = \phi_{\text{try}}, D_l = D_{\text{try}}$.
- 26: **end if**
- 27: **end if**
- 28: **end while**
- 29: $T = \lambda T$.
- 30: **end while**
- 31: **Return:** D_g .

large enough r . Motivated by the choice in Joseph et al. (2015), we use $r = 2p$ as the default setting in our package.

The gradient of ϕ_{rec}^r with each element in D_n is:

$$\frac{\partial \phi_{\text{rec}}^r}{\partial x_{ik}} = \frac{-2r}{n(n-1)} \sum_{j \neq i} \frac{x_{ik} - x_{jk}}{\|x_i - x_j\|_2^{r+2}}, \quad (6)$$

which can be used for gradient-based continuous optimization of maximin design.

Algorithm 2 Deterministic Swap: deterministic swap for LHD

Require: Design matrix $D \in \mathbb{R}^{(n \times p)}$, criterion ϕ , number of passes of the whole design matrix P , Maximum number of iterations I .

```

1: Initialize best criterion value by the current criterion value  $\phi_{\text{best}} = \phi(D)$ .
2: Initialize a counter for total number of iteration  $i = 0$ .
3: for  $j = 1, \dots, P$  do
4:    $\text{flag} = \text{False}$ .
5:   for  $j = 1, \dots, n - 1$  do
6:     for  $m = j, \dots, n$  do
7:       for  $k = 1, \dots, p$  do
8:         if  $i > I$  then break.
9:       end if
10:      Update counter  $i = i + 1$ .
11:      Set  $D_{\text{try}} = D$  and swap  $x_{jk}$  and  $x_{mk}$  in  $D_{\text{try}}$ .
12:      Compute the criterion  $\phi_{\text{try}} = \phi(D_{\text{try}})$ .
13:      if  $\phi_{\text{try}} < \phi_{\text{best}}$  then
14:         $\text{flag} = \text{True}$ .
15:         $\phi_{\text{best}} = \phi_{\text{try}}$ ,  $D = D_{\text{try}}$ .
16:      end if
17:    end for
18:  end for
19: end for
20: if not flag then
21:   Break out of all the loops.
22: end if
23: end for
24: Return:  $D$ .

```

The following codes generate a two-dimensional seven-point maximin LHD:

```

R> n <- 7
R> p <- 2
R> D <- maximinLHD(n, p)$design

```

By default, it will go through the two-stage optimization. To perform continuous optimization with a simulated annealing (SA) step at the end, the following code can be used:

```

R> D <- maximin.optim(D, sa = TRUE)

```

Figure 1a presents the generated seven-point maximin design by the above code (denoted as SFD) alongside with the optimal solution. The design points with the smallest pairwise distance nearly coincide with the optimal points, with the exception of the top-right point, which deviates slightly. This deviation occurs because its placement has minimal impact on the maximin criterion. Maximin designs are closely related to the sphere packing problem (Conway and Sloane, 2013). A comparison of the maximin criterion of the design generated by this package with the best sphere packing solution recorded in the literature (<http://hydra.nat.uni-magdeburg.de/packing/>, denoted as record) is shown in Figure 1b. It can be seen that the designs generated by **SFDesign** are almost optimal. Note that the best sphere packing solutions are available for only two- or three-dimensional problems, whereas our package can be used for any number of dimensions making it useful for computer experiments.

In the above example, we use maximin LHD as the initial design for the `maximin.optim` function. However, if our final goal is to generate a maximin design without LHD constraints, optimizing over

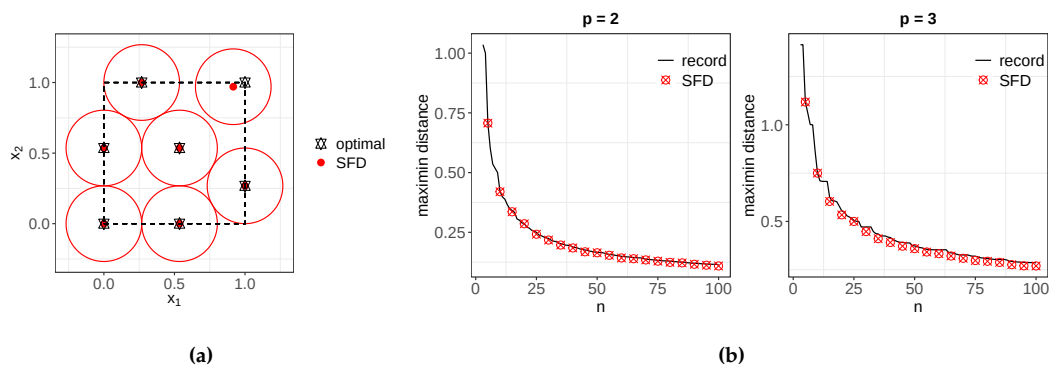


Figure 1: (a) Seven-point maximin design in two dimension generated by **SFDesign**. Red circles are centered at the design points and the radius is half the maximin distance. The stars are the optimal design; (b) Comparison of the maximin design for two and three dimension generated by **SFDesign** (denoted as SFD) with the recorded optimal sphere packing solution (denoted as record). The larger, the better.

an initial maximin LHD may be inefficient, especially for high dimensions where maximin design points should usually be located at corners. To address this issue, the `maximin.optim` has an option to set `find.best.ini = TRUE`. In this case, in addition to the user-supplied initial design, new initial designs are generated internally for optimization. Given the size of the design n , the function will first construct the closest full factorial design. If the number of points in this design is less than n , additional points are added sequentially to augment the design; otherwise, points are removed sequentially to reach the desired size. Initial designs constructed this way typically yield a smaller reciprocal distance criterion after continuous optimization.

We present a comparison with the `maximinSLHD` function from the **SLHD** package (Ba, 2015) and the `maximin` function from the **maximin** package (Sun and Gramacy, 2024) in Figure 4. In the figure, SLHD denotes the maximin LHD generated by **SLHD**; SFD.LHD refers to the LHD obtained after deterministic swap (Algorithm 2); SFD is the result of applying continuous optimization to SFD.LHD, with the setting `find.best.ini = TRUE`, `sa = FALSE`; `maximin` denotes the designs produced by the **maximin** package and it directly minimizes the maximin criterion (Equation 1) instead of the reciprocal distance.

For Latin hypercube designs, a considerable improvement is observed following the deterministic swap given by our package. When the LHD constraint is removed, further enhancement is achieved through continuous optimization, and the result is always better than **maximin**. In higher dimensions (e.g., $p = 10$), it should be noted that almost all SFD designs came from the internal generation of full factorial-based initial designs rather than the maximin LHD-based initial designs.

We highlight here the efficiency of the deterministic swap algorithm for improving maximin Latin hypercube designs, as illustrated in the left panel of Figure 2. For a fair comparison, the total number of swaps is fixed at 15,000. In our implementation, 7,500 swaps are allocated to the simulated annealing stage and 7,500 to the deterministic swapping stage. Before the onset of deterministic swapping, our method exhibits performance similar to **SLHD**, since both methods employ the same algorithmic framework during the SA stage. Once the algorithm transitions to deterministic swapping, the criterion value decreases faster than **SLHD**.

Besides generating Latin hypercube design and optimizing designs with the continuous optimizer `maximin.optim`, **SFDesign** provides the function `maximin.remove` for users to generate maximin designs from a large pool of candidate points. In Figure 3 we show an example of generating a maximin of size 20 from 1000 candidates.

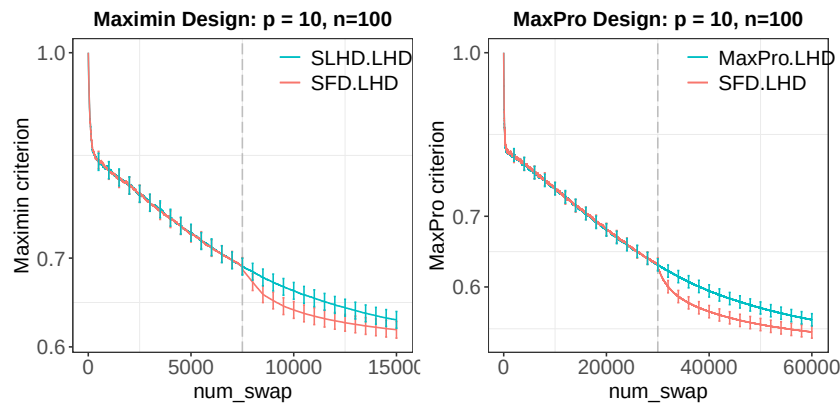


Figure 2: Trajectory of the criterion value with respect to the number of swaps performed by the algorithm. The gray dashed lines indicate when the deterministic swap stage starts. The solid lines represent the mean $\bar{m}$ over 100 experiments $\mathbf{m} = (m_1, \dots, m_{100})$, and the error bars mark $[\bar{m} - 1.96\text{std}(\mathbf{m})/\sqrt{100}, \bar{m} + 1.96\text{std}(\mathbf{m})/\sqrt{100}]$. Left: Maximin design, Right: Maximum projection design. SFD represents the SFDdesign package, MaxPro represents the MaxPro package and SLHD represents the SLHD package.

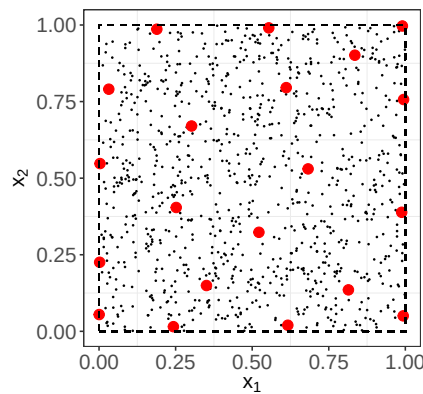


Figure 3: A maximin design of size 20 generated by `maximin.remove` from 1000 candidate points. Red points denote the design and the black dots are the candidates.

Maximum projection design

The major difference of `SFDdesign` and the existing `MaxPro` package (Ba and Joseph, 2018) is the addition of the deterministic swap refinements. Here, we demonstrate how to generate a MaxPro LHD using only the SA step:

```
R> n = 50
R> p = 5
R> D <- maxproLHD(n, p, method='sa')$design
```

This is equivalent to the `MaxProLHD` function in the `MaxPro` package. However, a deterministic swap of any LHD can be performed by the `SFDdesign` package to improve the MaxPro criterion:

```
R> D <- maxproLHD(D, method='deterministic')$design
```

The two steps can be grouped together by the following code:

```
R> result <- maxproLHD(n, p, method='full')
```

The trajectory of the criterion during optimization can be visualized by plotting `result$crit.hist`, as shown in Figure 5. This graph is particularly useful for selecting an appropriate initial temperature for

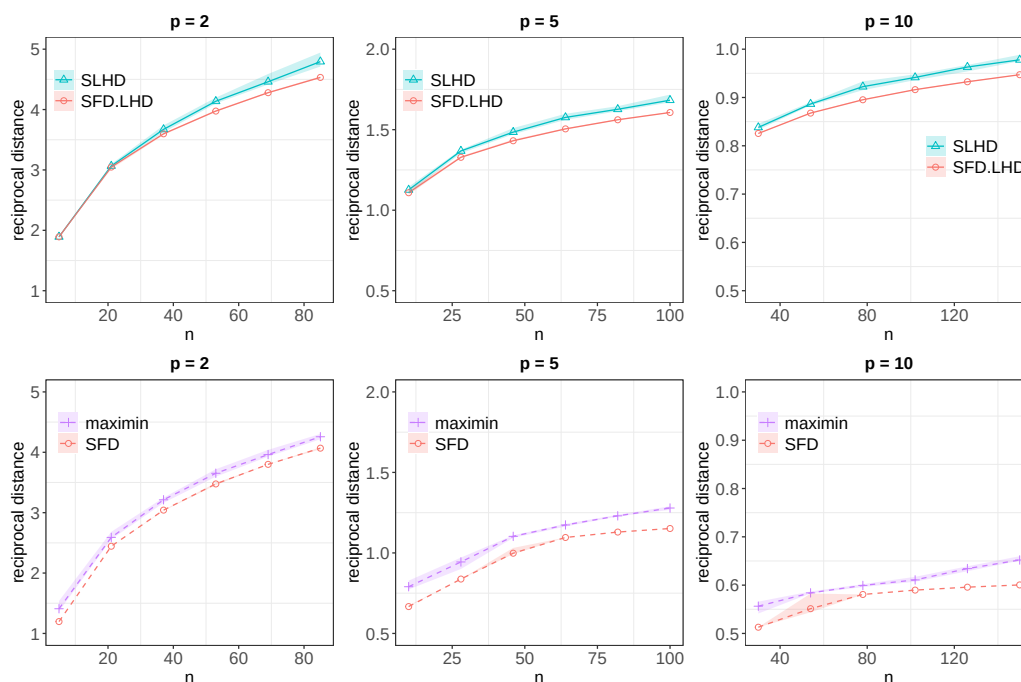


Figure 4: Comparison of the reciprocal distance criterion of designs (smaller the better). Top row: SLHD and SFD.LHD denote the maximin LHDs generated by **SLHD** and **SFDesign**, respectively; Bottom row: maximin and SFD denote the maximin designs without LHD constraints generated by **maximin** and **SFDesign**, respectively. Lines represent the median over 10 repetitions and the shaded bands mark the 5th and 95th quantiles. Solid and dashed lines represent designs with and without LHD restrictions, respectively.

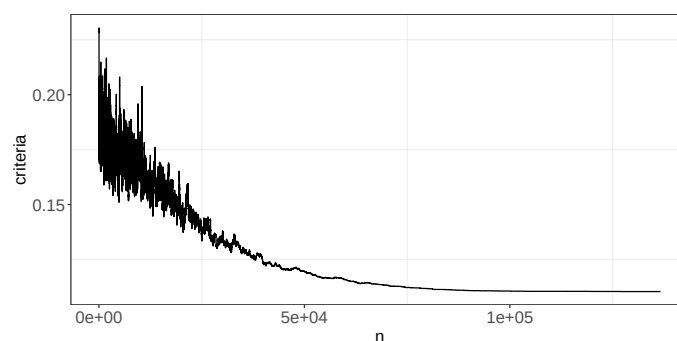


Figure 5: The trajectory of the MaxPro criterion during optimization.

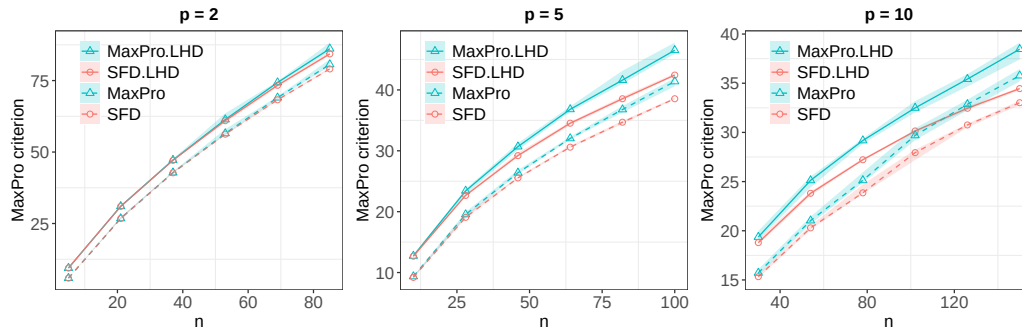


Figure 6: Comparison of the maximum projection criteria (smaller the better) of designs generated by **MaxPro** and **SFDDesign**. MaxPro.LHD and MaxPro are generated by the **MaxPro** package, with and without Latin hypercube (LHD) constraints, respectively. Similarly, SFD.LHD and SFD are designs generated by **SFDDesign** with and without LHD constraints. Lines represent the median over 10 repetitions and the shaded bands mark the 5th and 95th quantiles.

the SA step, especially for user-defined criteria discussed in Section 2.3.1, as their initial temperature is not computed automatically.

A follow-up continuous optimization can be performed to generate a design without LHD constraints:

```
R> D <- maxpro.optim(D)
```

The improvement achieved by deterministic swap is significant, as illustrated in Figure 6. In the figure, solid lines represent results for LHD and dashed lines correspond to designs without LHD constraints. For low-dimensional designs, the improvement is moderate; however, in higher dimensions (e.g., $p = 10$), the deterministic swap step can, in some cases, enhance the LHD to even outperform designs without LHD constraints generated by the **MaxPro** package.

Similar to the maximin LHD, the deterministic swap algorithm can improve the MaxPro LHD efficiently for the same number of swaps, as shown in the right panel in Figure 2. The total number of swaps is fixed at 60,000 for both **MaxPro** and **SFDDesign**. In our implementation, the number of swaps is fixed at 30,000 in the SA stage and 30,000 in the deterministic swapping stage. Once the algorithm enters the deterministic swapping stage, the MaxPro criterion clearly decreases faster for **SFDDesign**.

Uniform design

For uniform design, we optimize the wrap-around discrepancy (Equation 4). To generate an LHD with low discrepancy, the following code can be used:

```
R> D <- uniformLHD(n, p)$design
```

For continuous optimization, the gradient of wrap-around discrepancy with respect to each element in D_n can be derived as follows:

$$\frac{\partial \phi_{wa}}{\partial x_{ik}} = \frac{2}{n^2} \sum_{j \neq i} \prod_{t \neq k} \left[\frac{3}{2} - |x_{it} - x_{jt}|(1 - |x_{it} - x_{jt}|) \right] \left(-\text{sign}(x_{ik} - x_{jk}) + 2(x_{ik} - x_{jk}) \right). \quad (7)$$

The following code carries out the continuous optimization:

```
R> D <- uniform.optim(D)$design
```

Figure 7 presents a comparison with **DiceDesign**. The improvement is evident, particularly for high-dimensional designs. However, continuous optimization yields only marginal enhancements over the LHD version.

Besides Latin hyper cube design, we provide the functionality to generate a design of discrete factors with different number of levels. For example, suppose that we want to generate a design of size 9 on four factors with levels 3.

```
R> D = uniform.discrete(t = 3, p = 4, levels = rep(3, 4))
```

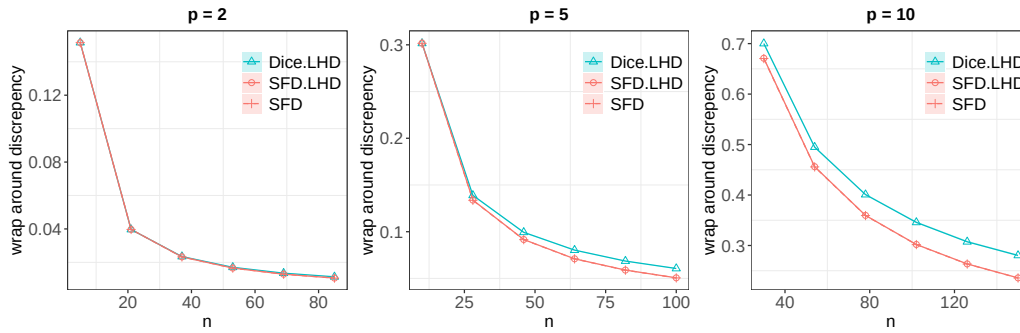


Figure 7: Comparison of the wrap-around discrepancy (smaller the better) of designs generated by **DiceDesign** and **SFDesign**. Dice.LHD denotes uniform LHDs generated by **DiceDesign**, and SFD.LHD and SFD are generated by **SFDesign**, with and without LHD constraint, respectively. Lines represent the median over 10 repetitions and the shaded bands mark the 5th and 95th quantiles.

The generated design is shown in Table 2, which actually is an orthogonal array $OA(9, 3^4)$ (Wu and Hamada, 2021). Although this need not be the case for larger designs, the generated designs can be close to optimal.

x_1	x_2	x_3	x_4
2	3	3	3
2	1	1	2
1	3	1	1
3	2	1	3
1	2	3	2
2	2	2	1
1	1	2	3
3	1	3	1
3	3	2	2

Table 2: Design generated by uniform.discrete.

User-defined criteria

The **SFDesign** provides the option to create an LHD based on a user-defined criterion.

```
R> D <- customLHD(compute.distance.matrix, compute.criterion,
  update.distance.matrix, temp, n, p)
```

The user should specify three functions: `compute.distance.matrix` which computes the distance of each pair of design points, `compute.criterion` which computes the user defined criterion from the distance matrix, and `update.distance.matrix` which will update the distance matrix after swapping one column of two rows of the design matrix. If simulated annealing algorithm is used in the optimization, a reasonable initial temperature `temp` should be specified as well so that the algorithm has a balance of exploration and exploitation.

Continuous optimization can also be performed once the user specify the initial design `D.ini` and the objective to minimize and the gradient with respect to the design matrix.

```
R> D <- continuous.optim(D.ini, objective, gradient)
```

3.2 Clustering-based design

Clustering-based designs tend to minimize the integrated prediction error of a Gaussian process model. Let $R(\mathbf{x}) = \exp(-\|\mathbf{x}\|^2/\theta^2)$ be the Gaussian correlation function and $\mathbf{r}(\mathbf{x}) = (R(\mathbf{x} - \mathbf{x}_1), \dots, R(\mathbf{x} - \mathbf{x}_n))'$. Then the root mean squared error for a new point is $\text{RMSE}(\mathbf{x}) = \sqrt{1 - \mathbf{r}(\mathbf{x})' \mathbf{R}^{-1} \mathbf{r}(\mathbf{x})}$. Let $\mathcal{X} = V_1 \cup \dots \cup V_n$ be a Voronoi diagram based on the design points. The integrated root mean squared

error (IRMSE) of a design D_n can be bounded as follows (Joseph, 2026, Ch. 4):

$$\begin{aligned}
 \text{IRMSE}(D_n) &= \int_{\mathcal{X}} \sqrt{1 - \mathbf{r}(x)' \mathbf{R}^{-1} \mathbf{r}(x)} dx \\
 &\leq \sum_{i=1}^n \int_{V_i} \sqrt{1 - R^2(x - x_i)} dx \\
 &= \sum_{i=1}^n \int_{V_i} \sqrt{1 - \exp(-2\|x - x_i\|^2/\theta^2)} dx \\
 &\approx \sum_{i=1}^n \int_{V_i} \sqrt{\|x - x_i\|^2/\theta^2} dx \quad (\text{assume } \|x - x_i\| \ll \theta) \\
 &= \frac{\sqrt{2}}{\theta} \sum_{i=1}^n \int_{V_i} \|x - x_i\| dx.
 \end{aligned}$$

Minimizing this bound is equivalent to the K -median clustering problem. A generalization of this criteria is the clustering criterion ϕ_{cluster} in (5). For example, when $\alpha = 2$, the clustering criterion is an upper bound for integrated mean square error $\text{IMSE}(D_n) = \int_{\mathcal{X}} \{1 - \mathbf{r}(x)' \mathbf{R}^{-1} \mathbf{r}(x)\} dx$. This package provides the option to generate designs based on the clustering criterion for any $\alpha > 0$, which is a functionality not provided by the existing packages. The design space is approximated using a large sample of Sobol points, S . We implement Lloyd's algorithm to generate cluster centers as design points, which iteratively refines an initial set of cluster centers by alternating between partitioning the design space into new Voronoi regions and updating the centers accordingly.

For $\alpha \leq 2$, we update each center by Weiszfeld's algorithm. Let $x_i^{(0)} = x_i$ denote the initial position of the i th center and let S_i represent the points within its Voronoi cell. The center is then updated as:

$$x_i^{(k+1)} = \left(\sum_{s \in S_i} \frac{s}{\|s - x_i^{(k)}\|_2^{2-\alpha}} \right) / \left(\sum_{s \in S_i} \frac{1}{\|s - x_i^{(k)}\|_2^{2-\alpha}} \right) \quad \text{for } k = 0, 1, \dots$$

For $\alpha > 2$, the centers are updated by accelerated gradient descent, similar to computing C_q -center in Mak and Joseph (2018).

Here, we mainly focus on $\alpha = 1$, which corresponds to minimizing the IRMSE.

```
R> D <- cluster.based.design(n, p, alpha=1)
```

Figure 8a shows a comparison with **Kmedians** (Godichon-Baggioni and Surendran, 2023) and **Gmedian** (Cardot, 2022). Since **SFDesign** employs the same algorithm as **Kmedians**, both exhibit similar performance, and both outperform **Gmedian**. However, the C++ implementation makes **SFDesign** more efficient, as shown in Figure 8b. Moreover, **SFDesign** enjoys the flexibility of different α . For example, choosing $\alpha = 10$ can lead to an approximate minimax design. The **Gmedian** package, which implements an averaged stochastic gradient algorithm, is designed to handle large samples in high-dimensional spaces efficiently (Cardot et al., 2012). When the design dimension is extremely high, using **Gmedian** may be advantageous.

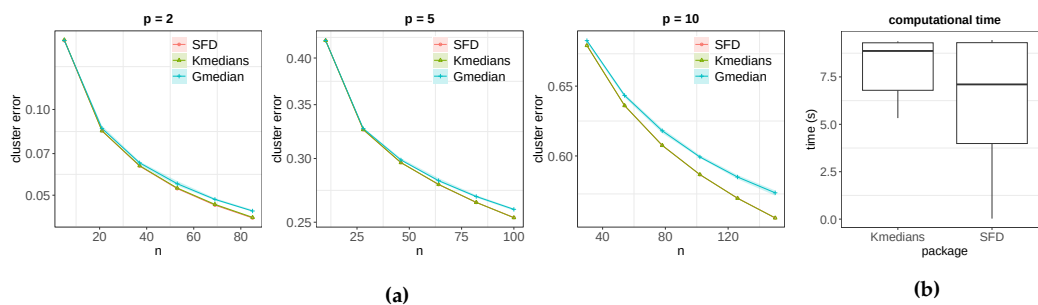


Figure 8: (a) Comparison of the clustering error of designs generated by **Gmedian**, **Kmedians** and **SFDesign** (denoted as SFD). The smaller the better. Lines represent the median over 10 repetitions and the shaded bands mark the 5th and 95th quantiles; (b) Comparison of computational time for generating fifty-point designs of 5 dimension. Boxplots are generated from 20 experiments.

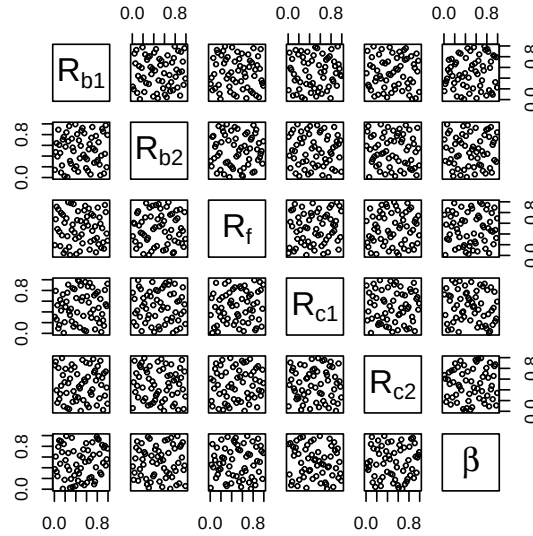


Figure 9: Maximum projection LHD generated by SFDesign

4 Case studies

4.1 Surrogate modeling

Here we consider the computer model for computing the midpoint voltage V_m of an output transformerless (OTL) push-pull circuit (Ben-Ari and Steinberg, 2007). There are six input variables: resistance b1 (K-Ohms) R_{b1} , resistance b2 (K-Ohms) R_{b2} , resistance f (K-Ohms) R_f , resistance c1 (K-Ohms) R_{c1} , resistance c2 R_{c2} (K-Ohms) and current gain (Amperes) β .

$$V_m(\mathbf{x}) = \frac{(V_{b1} + 0.74)\beta(R_{c2} + 9)}{\beta(R_{c2} + 9) + R_f} + \frac{11.35R_f}{\beta(R_{c2} + 9) + R_f} + \frac{0.74R_f\beta(R_{c2} + 9)}{(\beta(R_{c2} + 9) + R_f)R_{c1}}, \quad \text{where}$$

$$V_{b1} = \frac{12R_{b2}}{R_{b1} + R_{b2}}$$

Four inert input variables are added in order to show the importance of projection properties of designs. We tried two types of design with a design size of fifty: maximin LHD and MaxPro LHD. Figure 9 shows the six active variables (scaled to the unique hypercube) of MaxPro LHD generated by SFDesign.

Given the design and corresponding responses, we fit an ordinary kriging model using the **rkriging** package (Huang and Joseph, 2024). To evaluate the model's performance, we generate 10^6 Sobol points as the test dataset. The left panel of Figure 10 compares the MaxPro criterion for designs generated by MaxPro and SFDesign. Consistent with Figure 6, SFDesign demonstrates a notable improvement over MaxPro. Additionally, we compare the prediction root mean square error (RMSE) of the kriging models. The results indicate that ordinary kriging models fitted on MaxPro LHDs achieve lower prediction errors, with SFDesign providing more consistent improvements.

4.2 Uncertainty quantification

When the objective is to quantify the uncertainty of a computer code, uniform designs are desired. Here we consider a simulation of the piston motion within a cylinder (Kenett and Zacks, 1998). The cycle time in seconds is given by

$$C(\mathbf{x}) = 2\pi \sqrt{\frac{M}{k + S^2 \frac{P_0 V_0}{T_0} \frac{T_a}{V^2}}}, \quad \text{where}$$

$$V = \frac{S}{2k} \left(\sqrt{A^2 + 4k \frac{P_0 V_0}{T_0} T_a} - A \right)$$

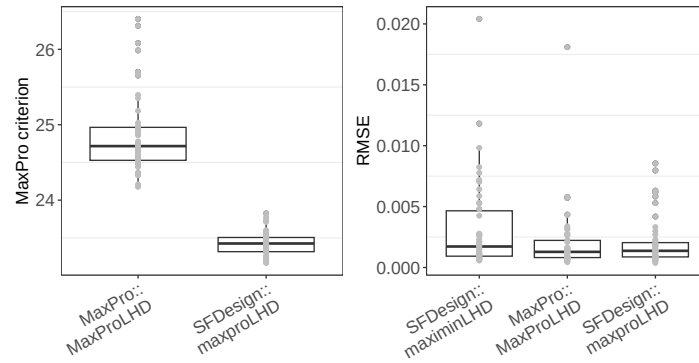


Figure 10: Left: Boxplots of maximum projection criterion of maximum projection LHDs generated by MaxPro and SFDesign. Right: Prediction RMSE of ordinary kriging fitted on maximin LHD designs generated by SFDesign and maximum projection LHDs generated by MaxPro and SFDesign. The result is generated from 20 random experiments.

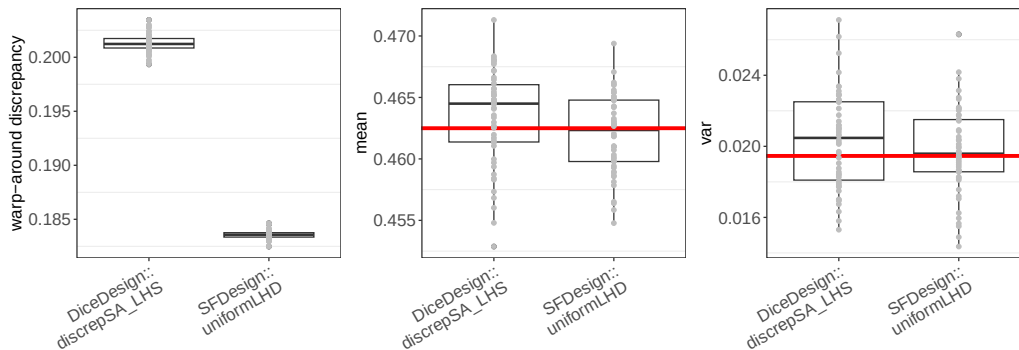


Figure 11: Boxplots of wrap-around discrepancy of designs generated by functions from **MaxPro** and **SFDesign**, and the mean and variance estimation based on the corresponding experimental responses. The red line is the true mean and variance of the response. The result is generated from 20 random experiments.

$$A = P_0 S + 19.62M - \frac{kV_0}{S}$$

where the piston weight $M \in [30, 60]$ kg, the piston surface area $S \in [0.005, 0.020]$ m², initial gas volume $V_0 \in [0.002, 0.010]$ m³, spring coefficient $k \in [1000, 5000]$ N/m, the atmospheric pressure $P_0 \in [90, 000, 110, 000]$ N/m², ambient temperature $T_a \in [290, 296]$ K, and filling gas temperature $T_0 \in [340, 360]$ K.

We are interested in computing the mean and variance of the cycle time when the inputs are uniformly distributed within their respective ranges. Figure 11 shows that the wrap-around discrepancy of uniform designs generated by SFDesign is lower, leading to more accurate mean and variance estimations that are closer to the true values.

5 Conclusion

In this paper, we introduce the **SFDesign** package, a comprehensive set of functions for constructing space-filling designs based on several widely used criteria. Through a combination of algorithmic advancements and software optimizations, our package achieves substantial improvements over existing packages, offering more efficient and effective design generation. In addition to supporting established space-filling criteria such as maximin, MaxPro, and uniformity-based discrepancy measures, the package provides a framework for incorporating customized criteria, enabling researchers and practitioners to adapt the design methodology to domain-specific applications. Future extensions may include support for additional criteria beyond those based on pairwise distance or clustering, expanding the scope of design optimization to accommodate alternative space-filling measures.

Acknowledgments

This work is supported by U.S. National Science Foundation grant DMS-2310637.

References

- S. Ba. *SLHD: Maximin-Distance (Sliced) Latin Hypercube Designs*, 2015. R package version 2.1-1. available at <https://CRAN.R-project.org/package=SLHD>. [p124, 129]
- S. Ba and V. R. Joseph. *MaxPro: Maximum Projection Designs*, 2018. R package version 4.1-2. available at <https://CRAN.R-project.org/package=MaxPro>. [p124, 130]
- E. N. Ben-Ari and D. M. Steinberg. Modeling data from computer experiments: an empirical comparison of kriging with MARS and projection pursuit regression. *Quality Engineering*, 19(4):327–338, 2007. [p135]
- H. Cardot. *Gmedian: Geometric Median, k-Medians Clustering and Robust Median PCA*, 2022. R package version 1.2.7. available at <https://CRAN.R-project.org/package=Gmedian>. [p134]
- H. Cardot, P. Cénac, and J.-M. Monnez. A fast and recursive algorithm for clustering large datasets with k-medians. *Computational Statistics & Data Analysis*, 56(6):1434–1449, 2012. [p134]
- R. Carnell. *lhs: Latin Hypercube Samples*, 2024. R package version 1.2.0. available at <https://CRAN.R-project.org/package=lhs>. [p124]
- J. H. Conway and N. J. A. Sloane. *Sphere packings, lattices and groups*, volume 290. Springer Science & Business Media, 2013. [p128]
- C. Currin, T. Mitchell, M. Morris, and D. Ylvisaker. Bayesian prediction of deterministic functions, with applications to the design and analysis of computer experiments. *Journal of the American Statistical Association*, 86(416):953–963, 1991. [p124]
- D. Dupuy, C. Helbert, and J. Franco. Dicedesign and diceeval: Two R packages for design and analysis of computer experiments. *Journal of Statistical Software*, 65:1–38, 2015. [p124]
- K.-T. Fang, D. K. Lin, P. Winker, and Y. Zhang. Uniform design: theory and application. *Technometrics*, 42(3):237–248, 2000. [p126]
- K.-T. Fang, R. Li, and A. Sudjianto. *Design and modeling for computer experiments*. Chapman and Hall/CRC, 2005. [p124]
- A. Godichon-Baggioni and S. Surendran. *Kmedians: K-Medians*, 2023. R package version 2.2.0. available at <https://CRAN.R-project.org/package=Kmedians>. [p134]
- R. B. Gramacy. *Surrogates: Gaussian process modeling, design, and optimization for the applied sciences*. Chapman and Hall/CRC, 2020. [p124]
- F. Hickernell. A generalized discrepancy and quadrature error bound. *Mathematics of computation*, 67(221):299–322, 1998. [p126]
- C. Huang and V. R. Joseph. *rkriging: Kriging Modeling*, 2024. R package version 1.0. available at <https://CRAN.R-project.org/package=rkriging>. [p135]
- M. E. Johnson, L. M. Moore, and D. Ylvisaker. Minimax and maximin distance designs. *Journal of statistical planning and inference*, 26(2):131–148, 1990. [p124]
- S. G. Johnson. *The NLOpt nonlinear-optimization package*, 2008. available at <https://github.com/stevengj/nlopt>. [p126]
- V. R. Joseph. Space-filling designs for computer experiments: A review (with discussions). *Quality Engineering*, 28(1):28–44, 2016. [p124]
- V. R. Joseph. *Experimental design for data science and engineering*. Chapman & Hall/CRC Press, ISBN: 9781041117520, 2026. [p124, 134]
- V. R. Joseph, E. Gul, and S. Ba. Maximum projection designs for computer experiments. *Biometrika*, 102(2):371–380, 2015. [p125, 128]
- R. Kenett and S. Zacks. *Modern industrial statistics: design and control of quality and reliability*. Duxbury Press, Pacific Grove, CA, 1998. [p135]

- D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989. [p126]
- S. Mak. *minimaxdesign: Minimax and Minimax Projection Designs*, 2021. URL <https://CRAN.R-project.org/package=minimaxdesign>. R package version 0.1.5. [p124]
- S. Mak and V. R. Joseph. Minimax and minimax projection designs using clustering. *Journal of Computational and Graphical Statistics*, 27(1):166–178, 2018. [p126, 134]
- M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979. [p124, 125]
- M. D. Morris and T. J. Mitchell. Exploratory designs for computational experiments. *Journal of statistical planning and inference*, 43(3):381–402, 1995. [p124, 125, 126]
- J. Sacks, S. B. Schiller, and W. J. Welch. Designs for computer experiments. *Technometrics*, 31(1):41–47, 1989. [p124]
- T. J. Santner, B. J. Williams, and W. I. Notz. *The Design and Analysis of Computer Experiments*. Springer, New York, NY, 2018. [p124]
- F. Sun and R. B. Gramacy. *maximin: Space-Filling Design under Maximin Distance*, 2024. R package version 1.0-5. available at <https://CRAN.R-project.org/package=maximin>. [p124, 129]
- E. R. van Dam. Two-dimensional minimax Latin hypercube designs. *Discrete Applied Mathematics*, 156(18):3483–3493, 2008. [p124]
- S. Wang and V. R. Joseph. *SFDesign: Space-Filling Designs*, 2025. R package version 0.1.0. [p124]
- C. F. J. Wu and M. S. Hamada. *Experiments: planning, analysis, and optimization*. John Wiley & Sons, 2021. [p133]

Shangkun Wang
H. Milton Stewart School of Industrial and Systems Engineering,
Georgia Institute of Technology
Atlanta, GA 30332
USA
shangkunwang01@gmail.com

Weijun Xie
H. Milton Stewart School of Industrial and Systems Engineering,
Georgia Institute of Technology
Atlanta, GA 30332
USA
wxie@gatech.edu

V. Roshan Joseph
H. Milton Stewart School of Industrial and Systems Engineering,
Georgia Institute of Technology
Atlanta, GA 30332
USA
roshan@gatech.edu

ProfileLadder: Functional-Based Reserving

by *Matúš Maciak, Rastislav Matúš, Ivan Mizera, and Michal Pešta*

Abstract An R package implementing a software environment for predicting loss reserves in insurance, including several novel, nonparametric, and functional-data-based algorithms, is introduced. These algorithms impute incomplete loss development profiles in run-off triangles without relying on traditional parametric assumptions used typically in actuarial practice. The package offers a flexible and computationally effective framework for point-wise and distributional reserve predictions and includes pertinent visualization and diagnostic tools through the S3 methods. It also provides accessor functions and real-world datasets to support exploratory analysis across insurance, operational risks, and other domains where triangular data structures arise, making modern, transparent, and extensible alternatives to classical approaches accessible in the insurance industry and academic research.

1 Introduction

One of the key components of insurance practice is maintaining sufficient financial reserves to cover the future claim payments; in non-life insurance, those are often spread into several installments spanning multiple time units (e.g., years). The possibility that an insurance company could default on its payments because of an insolvency caused by business failures is utterly unacceptable; this aspect is closely watched not only by the companies themselves, but also by their regulatory bodies. The companies have to abide by the rules set by the latter, and maintain sufficient funds to cover payments for past and future claims. An example of such a regulatory directive in the European Union is [European Parliament and Council \(2009\)](#), widely known as "Solvency II".

The insurance environment being inherently stochastic, an important task of actuarial science is to devise stochastic prediction techniques of such loss reserves. An important class of these methods works with aggregated data, organized into a structure known in actuarial science as a run-off triangle. Methods that could be in this context called classical are either quite simple, using so-called development factors ([Mack, 1993](#); [Renshaw and Verrall, 1998](#); [Clark, 2003](#); [Verdonck and Debruyne, 2011](#)), or rely on parametric assumptions justifying various regression models ([Verrall, 1996](#); [Maciak et al., 2021](#)). There is a growing interest in novel methods, offering better predictive performance, transparency, and robustness; an R package **ProfileLadder**, available from CRAN and introduced in this paper, chose to implement those proposed by [Maciak et al. \(2022\)](#).

Our motivation was and is to create and introduce the R package **ProfileLadder** as an extension and complement of other packages devoted to claims reserving—most importantly the ground-breaking package **ChainLadder** ([Germann et al., 2025](#)), featuring methods for non-life loss reserving that can be considered classical (Mack chain ladder, over-dispersed Poisson and other GLM models, Bornhuetter-Ferguson, etc.). Parametric reserving methods beyond those contained in **ChainLadder** can be found also in the package **NetSimR** ([Parizas, 2019](#)). On the other hand, our focus is rather nonparametric, inspired by functional data analysis; also, distributional predictions (which are generally considered compulsory, for instance by Solvency II) are done in this vein: via permutation bootstrap, an original technique introduced by [Maciak et al. \(2022\)](#). Classical techniques typically rely on parametric distributional assumptions, which we are determined to avoid; and our permutation bootstrap technique is applicable also to the classical methods of point prediction (which on the R level means that our `permuteReserve()` method can be applied also to the objects generated by the **ChainLadder**).

Certainly, R is not new to actuarial science: various other topics are covered by packages **actuaryr** ([Chmielewska, 2019](#), actuarial reporting); **lifecontingencies** ([Spedicato, 2013](#), life contingencies); **actuar** and **actuaRE** ([Dutang et al., 2008](#); [Campo, 2025](#), credibility models); and others. While **ProfileLadder** is specifically designed to meet practical needs of actuaries, the methods implemented therein can also be applied in other areas—for instance, to model the spread of infectious diseases, operational risks, or other data where incomplete functional profiles arise. The main methodological outcome, the nonparametric reconstruction of missing fragments of the data, here particularly suited to the structured missingness in run-off triangles, is adaptable also to other domains such as price curves ([Liebl and Rameseder, 2019](#)), growth curves ([Delaigle and Hall, 2013, 2016](#)), medical data ([Galmiche, 2016](#)), and geological or seismic activity profiles ([Bauer et al., 2021](#)).

The paper is organized as follows. Section 2 gives a brief summary of the implemented methods, including also some developments that were not available when [Maciak et al. \(2022\)](#) was published. In particular, these contributions beyond [Maciak et al. \(2022\)](#), are: a) theoretical—which includes a methodological approach for a run-off triangle exploratory, adaptation of the MACRAME algorithm beyond its data-driven version, methods for constructing one-year-ahead predictions (a new running diagonal) based on nonparametric approaches and, also, the permutation bootstrap applicable for other

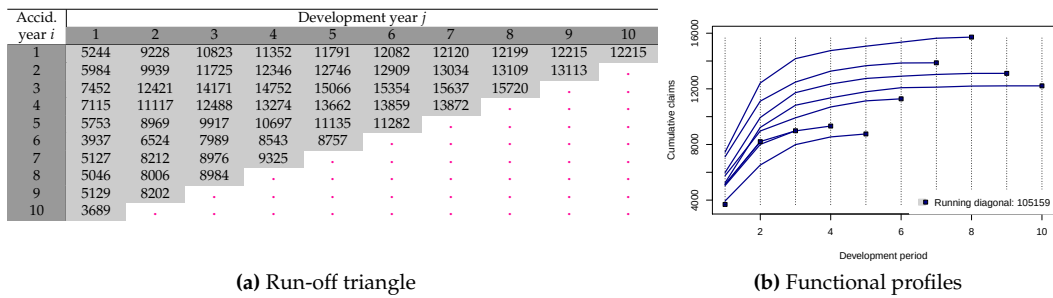


Figure 1: The cumulative run-off triangle (left) of an illustrative portfolio and its incomplete functional development profiles (right). The data, belonging to the Cameron Mutual Insurance Company, are taken from the National Association of Insurance Commissioners (NAIC) database (Meyers and Shi, 2011). The gray area figures are available to an actuary; the unknown future represented by dots is to be predicted.

than just the functional based prediction methods; and b) practical—mainly in terms of new actuarial datasets with much longer developments than typical and explicit utilization of the proposed methods beyond insurance applications. Section 3 describes the main features and implementation details of the **ProfileLadder** package, in the context of the sample session(s) featuring the main algorithms. Computational efficiency of the functional-based algorithms is illustrated and some comparisons with the classical parametric methods are also given there. Finally, Section 4 provides a summary and some outlook for possible applications beyond the actuarial context.

2 Description of the algorithms

The principal data structure used by claims reserving methods is the so-called run-off triangle. Formally, a *cumulative run-off triangle* is a dataset $\{Y_{i,j}; i = 1, \dots, n, j = 1, \dots, n+1-i\}$, where $i \in \{1, \dots, n\}$ corresponds to the *occurrence year* (when the insurance event happens) and $j \in \{1, \dots, n+1-i\}$ to the *development years* (when the ensuing claims are paid). An example is shown in the left-hand panel of Figure 1. The gray area exhibits figures which an actuary knows (observes) at a given moment of time; the dots in the white area represent the unknown future, which has to be predicted—the triangle is to be “completed”. It is assumed that the number of development years in each line is definitive; in particular, the first line of a triangle is already “completed”—no future payments are expected there.

At times, it is also useful to rephrase the same data in terms of an *incremental run-off triangle* $\{X_{i,j}; i = 1, \dots, n; j = 1, \dots, n+1-i\}$, with $X_{i,j} = Y_{i,j} - Y_{i,j-1}$, considering $Y_{i,0} = 0$. In what follows, random variables denoted by $Y_{i,j}$ ’s refer to cumulative run-off triangle(s) and random variables denoted by $X_{i,j}$ ’s to incremental run-off triangle(s).

The principal starting point of the algorithms proposed by Maciak et al. (2022) is the closely related concept of functional development profiles derived from aggregated data or run-off triangles respectively—the concept also known as the “*patterns of loss emergence*” (see Clark, 2003). Its graphical rendition can be seen in the right-hand panel of Figure 1. Each point-prediction algorithm provides a prediction of the overall reserve, the overall future losses/payments, by completing missing fragments within these observed patterns. Given that the structure of the missing values in the run-off triangle is predetermined by its specific form, standard imputation methods assuming missing completely at random (MCAR) situation do not apply (as was theoretically justified by Maciak et al., 2022).

2.1 PARALLAX (PARALLEL ApproXimation of missing fragments)

The idea of the first algorithm is to impute the missing parts of the run-off triangle by the most similar segments—triangle rows that can be found among the already observed development profiles. The name reflects the fact that in the functional development profiles, the imputation of the missing segments is always performed in a parallel way.

As described in Algorithm 1 scheme below, each step of the algorithm—the estimation of one unknown/unobserved quantity $Y_{i,j}$ respectively the imputation of one missing linear fragment between the points $[j-1, Y_{i,j-1}]$ and $[j, Y_{i,j}]$ for $i = 2, \dots, n$ and $j = n-i+2, \dots, n$, is performed via the minimization problem in (1). The overall reserve prediction of the true (but unknown) reserve $\mathcal{R}$ (the principal output of the algorithm at the point prediction stage) is $\hat{\mathcal{R}} = \sum_{i=2}^n \hat{Y}_{i,n} - \sum_{i=2}^n Y_{i,n+1-i}$, which is the difference between the (predicted) ultimate claim payments $\sum_{i=1}^n \hat{Y}_{i,j}$ and the claim payments

being already settled, the paid amount given by $\sum_{i=1}^n Y_{i,n+1-i}$ (the first row of the run-off triangle is typically assumed to be fully developed and, thus, $Y_{1,n} = \hat{Y}_{1,n}$). Besides the reserve estimate $\hat{\mathcal{R}}$, the algorithm also provides a full completion of the triangle into an $n \times n$ square—which can be used for a detailed inspection of the overall performance of the algorithm, but, more importantly, it is further needed for the estimation of the whole reserve distribution in terms of the permutation bootstrap resampling, as described in Section 2.4. Note the notation: the $Y_{i,j}$ stand for the observed quantities and $\hat{Y}_{i,j}$ for the predicted ones. The algorithm is implemented in the R function `parallelReserve()`; key features and certain implementation details are described in Section 3.1.

Algorithm 1: PARALLAX

```

1 Input: Observed (cumulative) run-off triangle  $\{Y_{i,j} : i = 1, \dots, n, j = 1, \dots, n+1-i\}$ 
2 begin
3   • Set the observed as the predicted,  $\hat{Y}_{i,j} = Y_{i,j}$  for  $i = 1, \dots, n$  and  $j = 1, \dots, n+1-i$ 
4   for  $i = 2, \dots, n$  do
5     for  $j = n+1-i, \dots, n-1$  do
6       • Find the most similar development profile
7         
$$\hat{\ell}_{i,j} = \arg \min_{\ell \in \{1, \dots, n-j\}} |\hat{Y}_{i,j} - Y_{\ell,j}| \quad (1)$$

8       • Predict the unobserved (future) value  $Y_{i,j+1}$  such that
9         
$$\hat{Y}_{i,j+1} = \hat{Y}_{i,j} + (Y_{\hat{\ell}_{i,j},j+1} - Y_{\hat{\ell}_{i,j},j})$$

10  Output: Completed square  $\{\hat{Y}_{i,j}\}_{i,j=1}^{n,n}$  and the reserve estimate  $\hat{\mathcal{R}} = \sum_{i=2}^n \hat{Y}_{i,n} - \sum_{i=2}^n Y_{i,n+1-i}$ 

```

2.2 REACT (approximation by the most REcent ACcident year)

This algorithm can be viewed as a simplified version of PARALLAX: instead of looking for the most similar parallel fragment among all existing fragments of the run-off triangle, the REACT algorithm relies—in the spirit of the classical development factors chain ladder methods—on the most recent segment in terms of the previous development year (which is, given the structure of the actuarial data, already observed). Formal description is given in Algorithm 2. Note that unlike Algorithm 1, the future payment prediction in Algorithm 2 is not defined in terms of any minimization problem any more. Therefore, the REACT algorithm is more straightforward and slightly easier to implement; due to evident similarities, both PARALLAX and REACT are implemented within the same R function `parallelReserve()`. See Section 3.1. A comprehensive discussion of various aspects of PARALLAX and REACT, addressing the pros and cons of both algorithms can be found in Maciak et al. (2022), as well as practical user recommendations and guidelines.

Algorithm 2: REACT

```

1 Input: Observed (cumulative) run-off triangle  $\{Y_{i,j} : i = 1, \dots, n, j = 1, \dots, n+1-i\}$ 
2 begin
3   • Set the observed as the predicted,  $\hat{Y}_{i,j} = Y_{i,j}$  for  $i = 1, \dots, n$  and  $j = 1, \dots, n+1-i$ 
4   for  $i = 2, \dots, n$  do
5     for  $j = n+1-i, \dots, n-1$  do
6       • Predict the unobserved (future) value  $Y_{i,j+1}$  such that
7         
$$\hat{Y}_{i,j+1} = \hat{Y}_{i,j} + (\hat{Y}_{i-1,j+1} - \hat{Y}_{i-1,j})$$

8   Output: Completed square  $\{\hat{Y}_{i,j}\}_{i,j=1}^{n,n}$  and the reserve estimate  $\hat{\mathcal{R}} = \sum_{i=2}^n \hat{Y}_{i,n} - \sum_{i=2}^n Y_{i,n+1-i}$ 

```

Note that for certain run-off triangles, triangles such that $(Y_{i,j} - Y_{i-1,j}) < (Y_{i,j} - Y_{k,j})$ for all $i = 3, \dots, n, j = n+1-i, \dots, n-1$, and $k = 1, \dots, i-2$, both algorithms, PARALLAX and REACT, provide the same reserve prediction and the same completion of the missing fragments: both algorithms are equivalent in such situations.

2.3 MACRAME (MARKov Chain fRAGMENT approximation)

The idea of this algorithm is to utilize a homogeneous Markov chain (MC) model to obtain the reserve prediction. This strategy is slightly different and more complex than the previous two; despite its additional mathematical complexity, however, it is quite intuitive and can be seen as a complex but rigorous generalization of the previous two algorithms described above. The underlying stochastic model invites additional user-defined adjustments and practically oriented modifications; unlike the MACRAME algorithm described in [Maciak et al. \(2022\)](#), where only a constrained and rather limited version of the algorithm was proposed, the present implementation in the R function `mcReserve()` in **ProfileLadder** allows for all this fine tuning and adaptations. See Section 3.2 details.

Firstly, unlike the previous two algorithms, MACRAME is based on the incremental run-off triangle $\{X_{i,j}; i = 1, \dots, n; j = 1, \dots, n+1-i\}$. Let us recall that $X_{i,j} = Y_{i,j} - Y_{i,j-1}$ and $Y_{i,0} = 0$ for all $i = 1, \dots, n$. In the first step, the algorithm transforms the incremental payments $\{X_{i,j}; i = 1, \dots, n; j = 1, \dots, n+1-i\}$ into a finite set of states of a homogeneous Markov chain, $\mathcal{S} = \{s_1, \dots, s_m\}$, and the observed run-off triangle is used to estimate the matrix of the corresponding transition probabilities $\mathbb{P} = (p(s_{i_1}, s_{i_2}))_{i_1=1, i_2=1}^{|\mathcal{S}|, |\mathcal{S}|}$, for $p(s_{i_1}, s_{i_2}) = P[U_{i,j+1} = s_{i_2} | U_{i,j} = s_{i_1}]$ where the increment $X_{i,j}$ is represented by the Markov state which is taken by $U_{i,j}$. Thus, each row $i \in \{1, \dots, n\}$ is assumed to be governed by an underlying Markov chain process $\{U_{i,j}; j \in \mathbb{N}\}_i$ and these processes are assumed to be independent for $i \in \mathbb{N}$.

The assignment of the observed increments $X_{i,j}$'s into the states in $\mathcal{S}$ is performed via *break points* (*breaks*) $-\infty = g_0 < g_1 < \dots < g_{m-1} < g_m = \infty$; each interval defined by the two adjacent grid points is represented by exactly one state s_k from $\mathcal{S}$. The transformation of the increments $X_{i,j}$'s into the Markov states is then straightforward: if $X_{i,j} \in [g_{k-1}, g_k)$ for some $k \in \{1, \dots, m\}$ then $U_{i,j} = s_k$ where $s_k \in \mathcal{S}$ and, also, $s_k \in [g_{k-1}, g_k)$.

A fully data-driven method for finding appropriate break points $\{g_k\}_{k=0}^m$ and the corresponding set of Markov states $\mathcal{S} = \{s_1, \dots, s_m\}$ was proposed in [Maciak et al. \(2022\)](#), and it is set as the default option for the `mcReserve()` function in the **ProfileLadder** package. The data-driven method is based on three crucial arguments:

- (i) to ensure a 1–1 transformation in time and the incremental amounts, the default number of Markov states is the same as the number of the development periods, $m = n$;
- (ii) the run-off triangle increments (excluding, by default, the first column of the run-off triangle) denoted as $\{x_{(i+j-3)(i+j-2)/2+i} := X_{i,j} : j > 1; i+j \leq n+1\}$ are ordered and split into m bins by using the set of grid points $\{g_m\}_{m=0}^m$, where $g_k := x_{(\lceil \frac{kn(n-1)}{2m} \rceil + 1)}$, for $k = 1, \dots, m-1$; and, in addition, $g_0 = -\infty$ and $g_m = \infty$;
- (iii) the corresponding Markov states $\mathcal{S} = \{s_1, \dots, s_m\}$ are taken as medians of the increments belonging to each bin, i.e., $\mathcal{S} := \{s_k := \text{median}(X_{i,j} \in [g_{k-1}, g_k) : j > 1, i+j \leq n+1), k = 1, \dots, m\}$, where the median of an empty set is omitted (and such bin is not considered).

For an illustration of this data-driven principle, consider a small 4×4 triangle formed from the portfolio in Figure 1. The cumulative triangle is converted into an incremental one and all increments (except the first column) are ordered. The third, fourth, and the sixth increments are used to define the break points g_1, g_2 , and g_3 as illustrated in the scheme below:

Accid. year i	Development j				Accid. year i	Development j				Number of states:	$m = n = 4$
	1	2	3	4		1	2	3	4	Ordered increments:	$\{529, 1595, 1786, 3955, 3984, 4969\}$
1	5244	9228	10823	11352	1	5244	3984	1595	592	Break point indexes:	$\left\{\left\lceil \frac{kn(n-1)}{2m} \right\rceil + 1; k = 1, 2, 3\right\} = \{3, 4, 6\}$
2	5984	9939	11725	.	2	5984	3955	1786	.	Break points $\{g_k\}_{k=0}^4$:	$\{-\infty, 1786, 3955, 4964, \infty\}$
3	7452	12421	.	.	3	7452	4946	.	.	Markov states:	medians of increments in bins $[g_{k-1}, g_k)$
4	7115	.	.	.	4	7115	.	.	.		

(a) Cumulative triangle → (b) Incremental triangle → (c) Increments, breaks, and MC states

Thus, the resulting Markov states are: $s_1 = 1062$ (i.e., the median of the first two increments, $\{529, 1595\}$, from the first bin, $[-\infty, 1786)$; $s_2 = 1786$ (which is the only increment in the second bin being $[1786, 3955)$); $s_3 = 3970$ (again the median of two increments, $\{3955, 3984\}$, from the third bin); and, finally, $s_4 = 4969$ which is the only increment in the last bin—the interval $[g_3, \infty)$. The transition matrix is estimated directly by counting the transitions between these four states.

However, a user can specify the set of states $\mathcal{S} = \{s_1, \dots, s_m\}$ and the breaks $\{g_k\}_{k=0}^m$ also differently; such alternative settings that lead to various user-defined modifications of the underlying Markov chain process are facilitated via the `mcReserve()` function and related functions. There are various reasons why such modifications could be of interest in practice. For atypical portfolios, standard parametric methods often do not apply: While functional-based techniques are able to handle those, the reserve prediction for portfolios with many zeros and small increments, for instance, would underestimate the true reserve if, particularly, some sudden claim amount increase is expected by actuarial experts (or vice versa). Thus, alternative settings provide useful tuning options for modeling rather atypical situations while also combining them with real expectations.

Algorithm 3: MACRAME

1 **Input:** Cumulative run-off triangle $\{Y_{i,j} : i = 1, \dots, n, j = 1, \dots, n+1-i\}$, the sequence of grid points $\{g_k\}_{k=1}^{m-1}$ (with $g_0 := -\infty, g_m := +\infty$), and the Markov states $\mathcal{S} = \{s_1, \dots, s_m\}$

2 **begin**

3 • Calculate incremental claims $X_{i,j} = Y_{i,j} - Y_{i,j-1}$ for all observed $Y_{i,j}$ (using $Y_{i,0} \equiv 0$)

4 • Set $\hat{X}_{i,j} = X_{i,j}$ and $\hat{Y}_{i,j} = Y_{i,j}$ for $i = 1, \dots, n$ and $j = 1, \dots, n+1-i$

5 • Use the states in $\mathcal{S}$ and transform $\hat{X}_{i,j}$ into $U_{i,j}$ for $i = 1, \dots, n$ and $j = 1, \dots, n+1-i$ such that $U_{i,j} = s$, if $\hat{X}_{i,j} \in [g_{k-1}, g_k)$ and $\mathcal{S} \ni s \in [g_{k-1}, g_k)$

6 **for** $s_1, s_2 \in \mathcal{S}$ **do**

7 • Calculate the transition probability estimates $\hat{p}(s_1, s_2)$ as

$$\hat{p}(s_1, s_2) = \frac{\sum_{j=1}^{n-1} \frac{1}{n-j} \sum_{i=1}^{n-j} \mathbb{1}\{U_{i,j} = s_1, U_{i,j+1} = s_2\}}{\sum_{j=1}^{n-1} \frac{1}{n-j} \sum_{i=1}^{n-j} \mathbb{1}\{U_{i,j} = s_1\}} \quad (2)$$

 and the estimated transition matrix $\hat{\mathbb{P}} = (\hat{p}(s_i, s_j))_{i=1, j=1}^{m,m}$

8 **for** $i = 2, \dots, n$ **do**

9 **for** $h = 1, \dots, i-1$ **do**

10 • Predict the unobserved value $X_{i,n+1-i+h}$ as $\hat{X}_{i,n+1-i+h} = \mathbf{c}(U_{i,n+1-i})^\top \hat{\mathbb{P}}^h \mathbf{s}$, where $\hat{\mathbb{P}}^h \equiv (\hat{\mathbb{P}})^h$ is the power matrix, $\mathbf{s} = (s_1, \dots, s_m)^\top$, and $\mathbf{c}(U_{i,n+1-i}) = (\mathbb{1}\{U_{i,n+1-i} = s_1\}, \dots, \mathbb{1}\{U_{i,n+1-i} = s_m\})^\top$

11 • Compute the corresponding predicted cumulative amount $\hat{Y}_{i,n+1-i+h} = \hat{X}_{i,n+1-i+h} + \hat{Y}_{i,n-i+h}$

12 **Output:** Completed square $\{\hat{Y}_{i,j}\}_{i,j=1}^{n,n}$ and the reserve estimate $\hat{\mathcal{R}} = \sum_{i=2}^n \hat{Y}_{i,n} - \sum_{i=2}^n Y_{i,n+1-i}$

Given the set of the Markov chain states in $\mathcal{S}$ one can use the above formula in (2) to construct the empirical estimate $\hat{\mathbb{P}}$ for the unknown theoretical transition probability matrix $\mathbb{P}$. In some situations (e.g., run-off triangles with extremely short developments—typical for certain portfolios related to material damages, for instance) it may be convenient to exaggerate the state in which the run-off triangle is already fully developed (i.e., there are only zero increments following after some development period). To this end, a transition probability matrix defined as a convex combination $\tilde{\mathbb{P}} := (1 - \delta_n)\hat{\mathbb{P}} + \delta_n \mathbb{I}_0$ can be used; $\hat{\mathbb{P}}$ is the estimated transition matrix and $\mathbb{I}_0$ is either a stochastic matrix with the column corresponding to the state $0 \in \mathcal{S}$ consisting of all ones and the remaining entries are zeros (if $0 \in \mathcal{S}$), or it is a zero matrix otherwise. The data-driven mixing coefficient is defined to be $\delta_n := \frac{1}{n} \sum_{s \in \mathcal{S}} \hat{p}(s, 0)$ if $0 \in \mathcal{S}$ and set to zero if $0 \notin \mathcal{S}$.

Note that for triangles with not fully developed pay-off profiles (non-zero increments and thus $0 \notin \mathcal{S}$, a common case in practice), $\delta_n = 0$ and therefore the estimate of the transition probability matrix $\hat{\mathbb{P}}$ in terms of (2) is used to drive the reserve estimation by the MACRAME algorithm.

2.4 Distributional predictions: Resampling via permutation bootstrap

The reserve prediction $\hat{\mathcal{R}}$ for the unknown claims reserve $\mathcal{R}$ provides only partial information in the entire loss reserving problem. In practice, a prediction of the whole reserve distribution is required too (for instance, by risk reserving assessment guidelines like Solvency II).

Classical techniques employ a residual (parametric or semiparametric) bootstrap approach (typically based on the back-fitted residuals) to emulate the distribution of interest—see, for instance, England and Verrall (1999), Pinheiro et al. (2003), or Maciak et al. (2022). However, for the functional-based claims reserving Maciak et al. (2022) proposed and theoretically justified a different strategy: permutation bootstrap. The prediction of the reserve distribution is still obtained via bootstrap resampling, but the algorithm avoids the use of residuals by resampling (permuting) the whole functional profiles. The rows of the completed run-off triangle produced by a certain algorithm—by each of those described above, but also any of the classical parametric reserving models implemented in the ChainLadder package—are treated as independent functional profiles. The completed triangle, the data matrix $\{\hat{Y}_{i,j}\}_{i,j=1}^{n,n}$, can be standardized: each row is divided by the first positive value within the row (considered from the left—which is, very likely, the first incremental payment in each row), as

Algorithm 4: Permutation bootstrap

```

1 Input: Completed run-off triangle  $\{\hat{Y}_{i,j} : i = 1, \dots, n; j = 1, \dots, n\}$  (output of PARALLAX,
   REACT, or MACRAME) and the number of permutation bootstrap resamples  $B \in \mathbb{N}$ 
2 begin
3   • Standardize the rows in  $\{\hat{Y}_{i,j}\}_{i,j=1}^n$  to obtain a standardized square  $\{\tilde{Y}_{i,j}\}_{i,j=1}^{n,n}$ , where
      
$$\tilde{Y}_{i,j} := \hat{Y}_{i,j} / \hat{Y}_{i,\ell_i}, \quad \ell_i = \min \{j \in \{1, \dots, n\} : \hat{Y}_{i,j} > 0\} \quad (3)$$

4   • Define a set  $\Pi(n) = \{\pi^{(b)}\}_{b=1}^{n!}$  of all distinct permutations of the set  $\{1, \dots, n\}$  where
       $\pi^{(b)} : (1, \dots, n) \mapsto (\pi^{(b)}(1), \dots, \pi^{(b)}(n))$ 
5   for  $b = 1, \dots, B \leq n!$  do
6     • Obtain a permuted square  $\{\tilde{Y}_{\pi^{(b)}(i),j}\}_{i,j=1}^{n,n}$ , where  $\pi^{(b)} \in \Pi(n)$ 
7     • Form the permuted run-off triangle  $\{\tilde{Y}_{\pi^{(b)}(i),j} : \pi^{(b)}(i) + j \leq n + 1\}$  by omitting the
       lower-triangular part—values  $\tilde{Y}_{\pi^{(b)}(i),j}$ , where  $\pi^{(b)}(i) + j > n + 1$ 
8     • Estimate the missing fragments in  $\{\tilde{Y}_{\pi^{(b)}(i),j} : \pi^{(b)}(i) + j \leq n + 1\}$  by using the
       pertinent algorithm and obtain the bootstrapped completed square
        $\{\hat{\tilde{Y}}_{i,j} : i = 1, \dots, n; j = 1, \dots, n\}$ 
9     • Apply the back-standardization  $\hat{\tilde{Y}}_{i,j} := \tilde{Y}_{i,j} \times \hat{Y}_{i,\ell_i}$ , for  $i, j = 1, \dots, n$ 
10    • Obtain the bootstrapped reserve  $\hat{\mathcal{R}}^{(b)} = \sum_{i=2}^n \hat{\tilde{Y}}_{i,j} - \sum_{i=2}^n \hat{Y}_{i,n-i+1}$ 
11 Output: Set  $\{\hat{\mathcal{R}}^{(b)}\}_{b=1}^B$  of bootstrapped reserves mimicking the distribution of interest

```

indicated in formula (3) in the description of Algorithm 4. The standardization step was proposed by Maciak et al. (2022) on the grounds that it is very typical in practice that the claims amounts paid in the first development period substantially increase over the years (possibly the effect of economical growth, inflation, more advanced or more expensive technology, etc.). The standardization is set as a default, but the user can suppress it if desiring so.

The resulting (standardized) square $\{\tilde{Y}_{i,j}\}_{i,j=1}^{n,n}$ is then resampled *without replacement*, in the row-wise manner. Formally speaking, for every permutation $\pi^{(b)} : (1, \dots, n) \mapsto (\pi^{(b)}(1), \dots, \pi^{(b)}(n))$, where $b = 1, \dots, B$ such that $\pi^{(b)} \neq \pi^{(s)}$ if $b \neq s$, there is a new permuted square $\{\tilde{Y}_{\pi^{(b)}(i),j}\}_{i,j=1}^{n,n}$ and the initial estimation algorithm is applied again to the permuted run-off triangle $\{\tilde{Y}_{\pi^{(b)}(i),j} : \pi^{(b)}(i) + j \leq n + 1\}$. After a possible back-standardization, the permuted bootstrap reserve $\hat{\mathcal{R}}^{(b)}$ is obtained and the set $\{\hat{\mathcal{R}}^{(b)}\}_{b=1}^B$ is used to estimate the overall reserve distribution. See Algorithm 4.

3 ProfileLadder package for R

The package provides a toolbox for nonparametric reserving techniques—the reserve point prediction algorithms PARALLAX, REACT, and MACRAME, and the permutation bootstrap algorithm for the prediction of the overall reserve distribution—together with a whole set of tools for handling and preprocessing the data, exploratory analysis, or confirmatory statistical inference. A list of key functions is given in Table 1; apart from those listed there, the package contains functions (such as `plot()`, `print()`, or `summary()`) corresponding to the (rather standard generic) S3 methods implemented in the package. An integral part of the package are numerous datasets—run-off triangles from actuarial practice, suitable for illustration, validation, and back-testing purposes. Some of the datasets are unique in this aspect: dataset GFCIB, provided by the Guarantee Fund of the Czech Insurers' Bureau (GFCIB) and datasets CZ.casco, CZ.liability, and CZ.property provided by a major and market-leading insurance company in the Czech Republic contain relatively large (compared to other publicly available actuarial data) run-off triangles (for example, GFCIP from the mandatory car insurance in the Czech Republic contains 60 origins/quarters and 60 development periods—quarters again). The triangles represent the claims developments from the beginning of 2008 (1Q) up to the end of 2022 (4Q) while distinguishing four different lines of business—i.e., four portfolios: bodily injuries, material damages, provision payments, and annuities. On the other hand, the datasets CZ.casco, CZ.liability, and CZ.property provide separate run-off triangles for gross paid amounts and RBNS reserves (all with the dimensions 18×18). For a complete overview refer to the package reference manual available on CRAN.

Function name	Function description
as.profileLadder()	function for defining an extended R class profileLadder to make work with the functional development profiles easier and more straightforward
parallelReserve()	function that implements two nonparametric, functional-based reserve prediction algorithms, PARALLAX and REACT
mcReserve()	implements the MACRAME algorithm while also allowing for various user-based modifications and explicitly defined Markov Chain process
permuteReserve()	function for estimating the overall reserve distribution in terms of the permutation bootstrap (for parametric and nonparametric reserving)
incrExplor()	provides an exploratory analysis of the run-off triangle increments in order to set the Markov chain states and breaks for mcReserve()
mcBreaks()	function to access the break points $\{g_k\}_{k=0}^m$ used to define the bins for the run-off triangle increments in the mcReserve() function
mcStates()	function to access the Markov chain states $S = \{s_1, \dots, s_m\}$ representing the bins with the run-off triangle increments in the mcReserve() function
mcTrans()	function to access the estimated transition probability matrix $\hat{P}$ used by the MACRAME algorithm implemented in mcReserve()

Table 1: Brief descriptions of the key functions in the R package `ProfileLadder`. More complex information can be obtained by using the R help session for each function—e.g., `help("as.profileLadder")`. Generic functions—such as `plot()`, `predict()`, `print()`, or `summary()` used for specific S3 method classes—are not provided in the table.

3.1 PARALLAX and REACT

Algorithms PARALLAX and REACT are similar and relatively straightforward. The same also applies for the `parallelReserve()` function that implements both these algorithms. Using the illustrative data from Figure 1, the reserve prediction provided by the PARALLAX or REACT algorithm can be obtained by the R commands

```
R> library("profileLadder")
R> data(CameronMutual, package = "profileLadder")

R> parallax <- parallelReserve(CameronMutual, method = "parallax")
R> react <- parallelReserve(CameronMutual, method = "react", residuals = TRUE)
```

The PARALLAX algorithm is used as the default option in `parallelReserve()` and, therefore, no specification in terms of `method = "parallax"` is actually needed. The output of the `parallelReserve()` function is a list (of the S3 method class `profileLadder`) with a few different elements (for a comprehensive description of all items we refer to the R help session that can be obtained in a standard way by typing `help("parallelReserve")`). Nevertheless, the most important part of the output is the predicted reserve $\hat{R}$ (the list element `$reserve`) and the full completed run-off triangle—contained in the list element `$FullTriangle` (note the analogy with the `ChainLadder` package). Both pieces of information are automatically provided by the `print()` method as the output:

```
R> parallax

PARALLAX Reserving
  Estimated Reserve   Estimated Ultimate   Paid Amount   True Reserve
           8540             113699         105159           7963

PARALLAX method (functional profile completion)
dev
origin  1    2    3    4    5    6    7    8    9   10
  1  5244  9228 10823 11352 11791 12082 12120 12199 12215 12215
  2  5984  9939 11725 12346 12746 12909 13034 13109 13113 13113
  3  7452 12421 14171 14752 15066 15354 15637 15720 15724 15724
  4  7115 11117 12488 13274 13662 13859 13872 13947 13951 13951
  5  5753  8969  9917 10697 11135 11282 11320 11399 11415 11415
  6  3937  6524  7989  8543  8757  8904  8942  9021  9037  9037
  7  5127  8212  8976  9325  9539  9686  9724  9803  9819  9819
  8  5046  8006  8984  9333  9547  9694  9732  9811  9827  9827
  9  5129  8202  8966  9315  9529  9676  9714  9793  9809  9809
 10 3689  6276  7741  8295  8509  8656  8694  8773  8789  8789
```

A fancy version of the print method is used by default which is particularly convenient when working with the run-off triangle shaped data—for instance, compare the following (outputs omitted)

```
R> print(as.profileLadder(observed(CameronMutual)), fancy.print = TRUE)
R> print(as.profileLadder(observed(CameronMutual)), fancy.print = FALSE)
```

but this option can be either altered by the command `set.fancy.print()`—for which we only refer to the R help session for more details—or it can be fully suppressed by using global options by calling the command

```
R> options(profileLadder.fancy = FALSE)
```

The overall (predicted) reserve, reported as Estimated Reserve, can be also obtained directly from the completed run-off triangle (in the output above) as $\hat{\mathcal{R}} = \sum_{i=2}^n \hat{Y}_{i,n} - \sum_{i=2}^n Y_{i,n+1-i}$. Some additional quantitative characteristics are also provided: Estimated Ultimate represents the sum of the last column—the ultimate cumulative payment developments $\sum_{i=1}^n \hat{Y}_{i,n}$; Paid Amount gives the amount that was already settled by the insurance company—i.e., the sum of the last running diagonal, $\sum_{i=1}^n Y_{i,n+1-i}$; Finally, the true reserve (provided only in situations in which a fully observed run-off triangle is available as an input) is reported as True Reserve. Analogous output, however with different predicted reserve (which is 8358) and different completed profiles, is obtained for the REACT algorithm (explicit output is, for brevity, omitted). For a comparison, the reserve prediction for the CameronMutual run-off triangle provided by the standard (parametric) over-dispersed Poisson model (ODP) implemented in the R function `glmReserve()` from the package [ChainLadder](#) is 8601. Thus, just in terms of the prediction accuracy, both nonparametric methods outperform the classical (benchmark) actuarial approach.

However, note that the REACT algorithm above is fitted with the additional parameter option `residuals = TRUE` (which works regardless of the method choice). This provides a set of residuals in the output. There are actually two sets of residuals that can be provided in the output (however, not simultaneously) depending on the input triangle type. Standard (incremental) residuals, defined as $X_{i,j} - \hat{X}_{i,j}$ for $i = 2, \dots, n$ and $j = n - i + 2, \dots, n$, where $X_{i,j}$ is the “future” increment (if known) and $\hat{X}_{i,j}$ is the predicted counterpart, are provided if a full triangle (data matrix) is available as the input (thus, the “future” payments are supposed to be known). On the other hand, if there is a standard run-off triangle provided as the input, the so-called “back-fitted” residuals are calculated using an approach that is well-known in the actuarial and finance circles as a *back-fitting* approach (see, for instance, [Popescu and Suci \(2020\)](#) or [Maciak et al. \(2022\)](#) for further details). The main idea behind the back-fitted residuals is to use a flipped¹ completed triangle as an input and to back-predict the run-off triangle by applying the same estimation procedure in a reverse manner so to say. Both sets of residuals can be used in a straightforward way for some further empirical inspection and analysis (which is typically performed and often also required by regulators when applying classical parametric methods).

The summary method applied to an R object of the class `profileLadder`—the output from the `parallelReserve()` function for instance—provides a complex overview (by adopting an analogous layout as the one already used in the main actuarial R package [ChainLadder](#)).

```
R> summary(react)
```

```
REACT reserve prediction (by origins)
      First Latest Dev.To.Date Ultimate IBNR
2      5984  13113  1.0000000    13113    0
3      7452  15720  0.9997456    15724    4
4      7115  13872  0.9937675    13959    87
5      5753  11282  0.9912142    11382   100
6      3937   8757  0.9725677     9004   247
7      5127   9325  0.9528919     9786   461
8      5046   8984  0.9172963     9794   810
9      5129   8202  0.8210210     9990  1788
10     3689   3689  0.4314620     8550  4861
total 49232  92944  0.9174942   101302  8358
```

¹The flipped run-off triangle is analogous to a transposition. However, unlike the matrix transposition performed with respect to the main diagonal, the flipped matrix is transposed with respect to the second diagonal—mathematically expressed, for a completed square $\{Y_{i,j}; i = 1, \dots, n; j = 1, \dots, n\}$, the “flipped triangle” is the square $\{\tilde{Y}_{i,j}; i = 1, \dots, n; j = 1, \dots, n\}$, where $\tilde{Y}_{i,j} = Y_{n+1-i, n+1-j}$.

Overall reserve summary

Est.Reserve	Est.Ultimate	Paid Amount	True Reserve	Reserve%
8358.00	113517.00	105159.00	7963.00	4.96

Residual summary (standard incremental residuals)

Min	1st Q.	Median	Mean	3rd Q.	Max	Std.Er.
-719	-49	-1	-9	34	300	143

Total number of residuals: 45, Total number of unique residuals: 41

Suspicious residuals (using 2sigma rule): 2, Outliers (3sigma rule): 1

The first part of the output is basically an analogy of the output provided by the `summary()` function when being applied to an output of some estimation method implemented in the R package [ChainLadder](#) (the Over-dispersed Poisson model in `glmReserve()` for instance). It provides a detailed overview of the ultimate payments and the corresponding reserves by origins.

```
R> library("ChainLadder")
R> summary(glmReserve(observed(CameronMutual)))
```

	Latest	Dev.To.Date	Ultimate	IBNR		S.E	CV
2	13113	1.0000000	13113	0	0.01035287		Inf
3	15720	0.9992372	15732	12	29.38975320	2.4491461	
4	13872	0.9934116	13964	92	73.46098788	0.7984890	
5	11282	0.9850694	11453	171	96.26787355	0.5629700	
6	8757	0.9688019	9039	282	120.58781735	0.4276164	
7	9325	0.9397360	9923	598	177.57019068	0.2969401	
8	8984	0.8905630	10088	1104	246.00693847	0.2228324	
9	8202	0.7789914	10529	2327	379.54811405	0.1631062	
10	3689	0.4788422	7704	4015	622.22022041	0.1549739	
total	92944	0.9152986	101545	8601	880.66723938	0.1023913	

The function `observed()` used above (implemented in the package [ProfileLadder](#)) transforms the full data matrix `CameronMutual` into a run-off triangle with the lower-right triangular part containing NA values only (as required by the `glmReserve()` function). Note that unlike `PARALLAX` and `REACT` the ODP model in `glmReserve()` imposes a distributional assumption and, therefore, the estimated standard errors (S.E.) and coefficients of variation (CV) can be provided in the output. On the other hand, the summary of the enriched `profileLadder` class object provides, in addition, the first cumulative payments in the first column which is often used by practitioners to assess the over-time stability of the given portfolio (and, as already mentioned above, it may play some role in the following permutation bootstrap resampling).

The second part of `summary(react)` above provides the same quantitative information about the estimated reserve as before (Estimated Reserve, Estimated Ultimate, Paid Amount and True Reserve) but, in addition, there is also a quantification of the prediction precision in terms of Reserve% defined in [Maciak et al. \(2022\)](#) as

$$\text{Reserve\%} = 100 \times \left| \frac{\hat{\mathcal{R}}}{\mathcal{R}} - 1 \right|.$$

Note that the value can be provided only in situations when the true reserve—the lower triangular part—is available. The last part of the summary output refers to (standard or backfitted) residuals—if the residuals are asked for in the output. Analogous summary is also provided for back-fitted residuals that are calculated in situations where only the run-off triangle itself is given as the input—which can be achieved by the `observed()` function as

```
R> summary(parallelReserve(observed(CameronMutual), method="react", residuals=TRUE))
```

The `summary()` function can be also used to plot a histogram of the residuals (using the parameter `plotOption = TRUE`) supplemented with a nonparametric estimate of the underlying density. A simple barplot of the paid amount, estimated ultimate, estimated reserve, and the true reserve is plotted if no residuals are available.

The R object of the class `profileLadder` can be visualized with a generic `plot()` function. Moreover, if the run-off triangle itself is also of the class `profileLadder` the `plot()` function can be also used to plot the functional profiles of the run-off triangle itself. The R code

```
R> plot(as.profileLadder(CameronMutual))
R> plot(parallelReserve(CameronMutual))
```

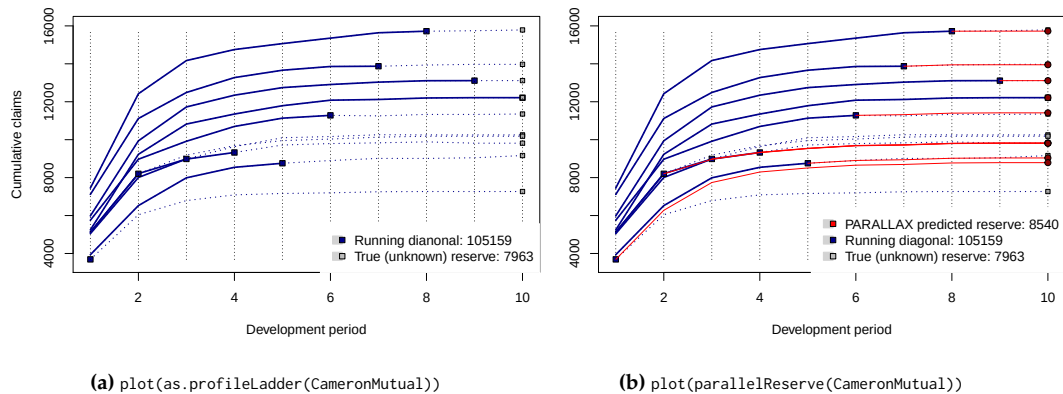


Figure 2: The observed functional development profiles (blue solid lines) plotted with the R method `plot()` applied to the object of the class `profileLadder`. For a "completed" triangle (data matrix), the function also plots the future "unobserved" fragments (blue dotted lines) and true "unknown" reserve. The predicted functional profiles (red solid lines in the right panel) are provided if `plot()` is applied to the output of some `parallelReserve()` or `mcReserve()`.

provides both plots in Figure 2. The same functionality of both generic R functions (`summary()` and `plot()`) is, of course, provided regardless of the prediction method that is used (PARALLAX or REACT) when calling the `parallelReserve()` function or the MACRAME algorithm implemented in the `mcReserve()` function—see below.

3.2 MACRAME

Let us recall that the MACRAME algorithm—as proposed in Maciak et al. (2022)—utilizes an underlying Markov chain mechanism to drive the prediction of the overall reserve $\mathcal{R}$. Due to the data availability (typically there are only 10 – 15 origin years available in run-off triangles in practice which means that there are only about 55 – 120 incremental payments in total), the underlying Markov chain is assumed to be a homogeneous stochastic process. However, there are still relatively many different parameters which need to be properly defined (e.g., the set of the increments, the states of the Markov chain, or the transition probability matrix) and some user-based interference can be of some interest here. The MACRAME algorithm, as originally proposed in Maciak et al. (2022), utilizes a fully automatic (data driven) approach to define the set of the Markov chain states $\mathcal{S} = \{s_1, \dots, s_m\}$ and the corresponding break points $\{g_k\}_{k=0}^m$. Nevertheless, the implementation of the algorithm in the R function `mcReserve()` provides users with various options—starting with a pre-specified number of the Markov states to be required, selecting the method how the run-off triangle increments are summarized into the states, or even providing a fully manual specification of the states $\mathcal{S} = \{s_1, \dots, s_m\}$, breaks $\{g_k\}_{k=0}^m$, or a subset of increments to be considered. These options are all summarized in Table 2. The default performance of the `mcReserve()` function corresponds with the data-driven selection of the states and breaks—in line with the original MACRAME algorithm proposed in Maciak et al. (2022). Considering again the `CameronMutual` dataset, the MACRAME predicted reserve is obtained by

```
R> macrame <- mcReserve(CameronMutual)
```

The output of the `mcReserve()` function is again the R object of the class `ProfileLadder` and it is analogous to the output of the `parallelReserve()` function already described before. For brevity, we omit repeating the same details again. However, specific information related to the underlying Markov chain in the MACRAME algorithm can be also obtained by using various Markov chain (mc) accessor functions (`mcBreaks()`, `mcStates()`, or `mcTrans()`). For instance,

```
R> mcBreaks(macrame)
[1] -Inf 75 147 288 388 554 780 1465 2587 3955 Inf
```

provides the (default) set of break points $\{g_k\}_{k=0}^m$ and

```
R> mcStates(macrame)
[1] 13.0 81.0 197.0 302.5 438.0 601.0 948.0 1672.5 3073.0 3993.0
```

gives the (default) set of the corresponding Markov chain states $\mathcal{S} = \{s_1, \dots, s_m\}$ such that exactly one state lies between two consecutive break points. The estimate of the transition matrix $\hat{\mathbb{P}}$ can be obtained by

```
R> mcTrans(macrame)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] 0.5000 0.5000 0.0000 0.0000 0.00 0.00 0.00 0.00 0 0
[2,] 0.6667 0.3333 0.0000 0.0000 0.00 0.00 0.00 0.00 0 0
[3,] 0.3333 0.6667 0.0000 0.0000 0.00 0.00 0.00 0.00 0 0
[4,] 0.3333 0.0000 0.3333 0.3333 0.00 0.00 0.00 0.00 0 0
[5,] 0.0000 0.0000 0.6000 0.2000 0.20 0.00 0.00 0.00 0 0
[6,] 0.0000 0.0000 0.2500 0.5000 0.25 0.00 0.00 0.00 0 0
[7,] 0.0000 0.0000 0.0000 0.0000 0.50 0.00 0.50 0.00 0 0
[8,] 0.0000 0.0000 0.0000 0.0000 0.25 0.75 0.00 0.00 0 0
[9,] 0.0000 0.0000 0.0000 0.0000 0.00 0.25 0.50 0.25 0 0
[10,] 0.0000 0.0000 0.0000 0.0000 0.00 0.00 0.25 0.75 0 0
```

Generic functions `plot()` and `summary()` can be both applied to the output in the same way as it was done in case of the `parallelReserve()` function.

In order to provide some useful insight into the structure of the incremental run-off triangle and to offer some visual inspection for various user-based modifications of the underlying Markov chain in the `mcReserve()` function, there is another software tool implemented in the [ProfileLadder](#) package. The R function `incrExplor()` takes the run-off triangle as an input (fully observed or not, incremental or cumulative) and returns a complex empirical exploration of the incremental payments. Recall that the original MACRAME algorithm (as proposed in [Maciak et al. \(2022\)](#)) does not take into account the first year payments, values $X_{i,1}$ for $i = 1, \dots, n$, when defining the grid points $\{g_k\}_{k=0}^m$ and the corresponding Markov states in $\mathcal{S}$. The reason is that most run-off triangles occurring in the real-world actuarial practice have substantially decreasing incremental payment profiles over the consecutive development periods (meaning that $X_{i,j} \rightarrow 0$ as $j \rightarrow \infty$) and, in order to achieve efficient reserve prediction, it is more substantial to model the tail increments within each row rather than capturing relatively large amounts of the first year (origin) payments. This can be changed by a user.

A default functionality of the `incrExplor()` function is to provide the data-driven set of bins (the break points respectively) and the corresponding Markov states for the underlying run-off triangle. The first year payments are not considered (i.e., `out = 1` is used by default). Taking the `CameronMutual` dataset again, the function provides the following output:

```
R> print(runoff.exploratory <- incrExplor(CameronMutual, out = 1))
```

Data-driven (default) setting of the Markov Chain in MACRAME

MC States: 13 81 197 302.5 438 601 948 1672.5 3073 3993

Corresponding bins for the run-off triangle increments

```
[1] "[-Inf, 75)" "[75, 147)" "[147, 288)" "[288, 388)" "[388, 554)"
[6] "[554, 780)" "[780, 1465)" "[1465, 2587)" "[2587, 3955)" "[3955, Inf)"
```

Note that all Markov states and also the bins for incremental payments correspond with the Markov states and the breaks already obtained by `mcStates()` and `mcBreaks()` from the output of the function `mcReserve()`. However, the output from the `incrExplor()` function is a complex object of the S3 class `mcSetup`. For more details we again refer to the R help session. Nevertheless, the summary method applied to the object of the class `mcSetup`—the output from `incrExplor()` provides the following details:

```
R> summary(runoff.exploratory)
```

Input triangle type: Cumulative

Summary of the increments

	Min	1st Q.	Median	Mean	3rd Q.	Max	Std.Er.
Raw increments	0	197.0000	529.0000	1126.0000	1595.000	4969	1340.0000
Std. increments	0	0.0396	0.1065	0.2267	0.321	1	0.2697

Total number of increments: 45, Total number of unique increments: 45

Number of suspicious increments (using 2sigma rule): 8, Outliers (3sigma rule): 1

Data-driven bins for the run-off triangle increments

```
[1] "[-Inf, 75)" "[75, 147)" "[147, 288)" "[288, 388)" "[388, 554)"
```

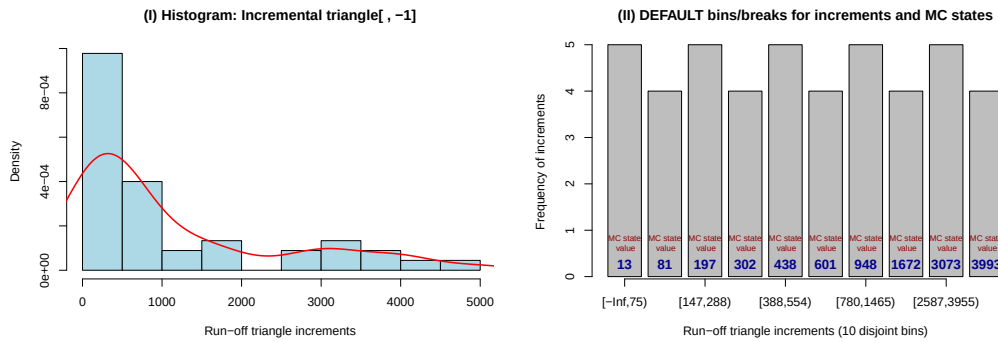



Figure 3: Visualization of the output of the `incrExplor()` function when applied to the `CameronMutual` dataset. The left-hand panel shows a standard histogram of the run-off triangle increments, values $X_{i,j}$, for $i = 1, \dots, n-1$ and $j = 2, \dots, n+1-i$. Note that the first year payments $\{X_{i,1}\}_{i=1}^n$ are not included. The right-hand panel shows the barplot of the run-off triangle increments distributed across disjoint bins defined by the break points $\{g_k\}_{k=0}^m$, where $g_0 = -\infty$ and $g_m = \infty$. Each bin is represented by the corresponding Markov chain state—blue bold values within the bars.

```
[6] "[554, 780)" "[780, 1465)" "[1465, 2587)" "[2587, 3955)" "[3955, Inf)"
```

Markov Chain states (medians of the increments within each bin)

```
[1] 13.0 81.0 197.0 302.5 438.0 601.0 948.0 1672.5 3073.0 3993.0
```

Graphical visualization is provided in a standard way using the `plot()` function. Both panels in Figure 3 are produced by the R command

```
R> plot(runoff.exploratory)
```

If there is a specific interest to also include the first year payments when defining the Markov chain states and the corresponding break points (or, alternatively, to exclude some other columns with the incremental payments), an additional parameter `out = 1` can be changed correspondingly—the DEFAULT value (one) stands for the first year increments that are typically not considered; the choice `out = 0` uses all available increments $X_{i,j}$, for $i = 1, \dots, n$ and $j = 1, \dots, n+1-i$; in order to exclude, for example, first three columns, the parameter can be specified as `out = c(1,2,3)`. The change of this parameter is also reflected by the output of `incrExplor()` which, in addition, contains analogous information as before, however, for a user-modified subset of the incremental payments.

```
R> print(runoff.exploratory.all <- incrExplor(CameronMutual, out = 0))
```

Data-driven (default) setting of the Markov Chain in MACRAME

MC States: 13 81 197 302.5 438 601 948 1672.5 3073 3993

Corresponding bins for the run-off triangle increments

```
[1] "[-Inf, 75)" "[75, 147)" "[147, 288)" "[288, 388)" "[388, 554)"
```

```
[6] "[554, 780)" "[780, 1465)" "[1465, 2587)" "[2587, 3955)" "[3955, Inf)"
```

User-modified MC setting

MC States: 14.5 125 285.5 400 601 978 2186.5 3689 5007.5 5984

Corresponding bins for the run-off triangle increments

```
[1] "[-Inf, 79)" "[79, 197)" "[197, 349)" "[349, 529)" "[529, 786)"
```

```
[6] "[786, 1595)" "[1595, 3085)" "[3085, 3984)" "[3984, 5244)" "[5244, Inf)"
```

Development periods (run-off triangle columns) not considered: 0

Method selected to summarize the increments within each bin: DEFAULT (median)

The `summary()` function applied on the `runoff.exploratory.all` object returns analogous information as before but the output refers to the user-defined subset of increments rather than the default set. For illustration, Figure 4 is obtained for all run-off triangle increments $\{X_{i,j}; i = 1, \dots, n, j = 1, \dots, n+1-i\}$, using the R command

```
R> plot(runoff.exploratory.all)
```

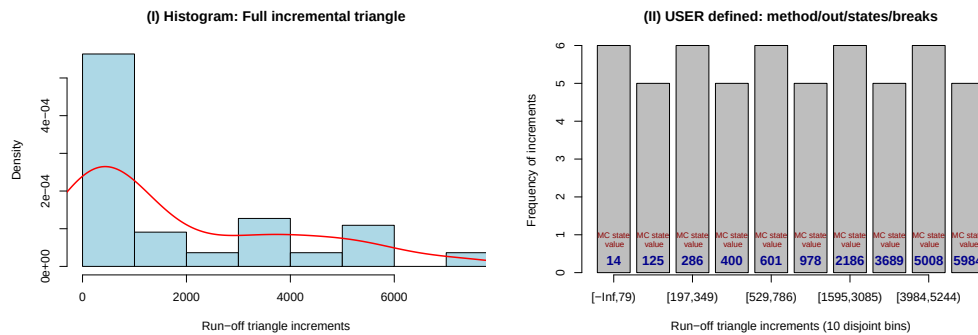


Figure 4: Visualization of the output of the `incrExplor()` function when applied to the `CameronMutual` dataset using the additional parameter `out = 0`. Instead of the default subset of 45 incremental payments summarized in the panels there are all 55 increments summarized in the same manner.

Note that the Markov chain states and the corresponding break points are also affected by the choice `out = 0` and this change is reflected now in Panel (II) in Figure 4 (compare with the same panel in Figure 3). The corresponding data-driven Markov states and the breaks are directly accessible by using the accessor methods

```
R> mcBreaks(runoff.exploratory.all)
[1] -Inf 79 197 349 529 786 1595 3085 3984 5244 Inf

R> mcStates(runoff.exploratory.all)
[1] 14.5 125.0 285.5 400.0 601.0 978.0 2186.5 3689.0 5007.5 5984.0
```

Three other parameters can be specified by a user in the `incrExplor()` function. The first one is the `method` parameter which defines the way how the increments within each bin are summarized to obtain the final Markov state values. By default, the Markov states are calculated as the medians of the increments within each bin (`method = "median"`). The method can be changed to the average (`method = "mean"`), the minimum increment from the bin (`method = "min"`), or the maximum increment respectively (`method = "max"`).

Another parameter that can be used to modify the underlying setting of the Markov chain behind the MACRAME prediction is the `parameter states` (which is set to `NULL` by default). This parameter sets the Markov states either by providing some specific integer value for the number of states (e.g., `states = 5`, as in Example 1) while everything else is determined in a data-driven manner proposed in Maciak et al., 2022 or an explicit vector of states can be given instead (e.g., `states = c(500, 1000, 1500, 2000, 2500)` in Example 2) to enforce not only five states but also their explicit values (with the corresponding break points being the midpoints between the provided states).

Finally, the last parameter that can be used within the `incrExplor()` function is the `breaks` parameter (set to `NULL` by default). This parameter can either take the default value `NULL` or it can be a vector of explicit break points $\{g_k\}_{k=0}^m$. Different combinations of the values of these parameters (`states` and `breaks` in particular) result in different performance of the `incrExplor()` function and, consequently, also the `mcReserve()` function. For clarity, a complex description is provided in Table 2. The same functionality of the parameters `method`, `states`, and `breaks` also applies when calling the `mcReserve()` function to predict the reserve by the MACRAME algorithm.

In practice, the `incrExplor()` function is supposed to be used to perform a detailed exploratory of various settings of the underlying Markov chain required by the `mcReserve()` function. Once the set of breaks is established and the corresponding Markov states are derived, the output from the `incrExplor()` function can be directly forwarded as a part of the input for the `mcReserve()` function. For illustration, the default performance of the MACRAME algorithm obtained by

```
R> mcReserve(CameronMutual)
```

and somehow more complex and more explicit specification of the states and breaks—however, all by using the default data-driven approach in the `incrExplor()` function—in terms of

```
R> user.states <- mcStates(incrExplor(CameronMutual))
R> user.breaks <- mcBreaks(incrExplor(CameronMutual))
```

MACRAME: USER based setting		performance
states = ...	breaks = ...	incrExplor() & mcReserve()
NULL	NULL	DEFAULT (fully data-driven)
1. numeric	any value for breaks is ignored	same as DEFAULT (data-driven) but the number of states is enforced
2. <code>c(...)</code>	NULL	Markov chain states explicitly defined by <code>states = c(...)</code> while the corresponding <i>bins for the run-off increments are set as the midpoints between the user-provided states</i>
3. NULL	<code>c(...)</code>	explicit breaks in <code>breaks = c(...)</code> are used to form the bins for the increments and the states are obtained by summarizing the increments within each bin by using the method option
4. <code>c(...)</code>	<code>c(...)</code>	Markov chain states explicitly defined by <code>states = c(...)</code> with the <i>bins for the run-off increments being also user-defined by</i> <code>breaks = c(...)</code> (each bin in breaks must contain one state)

Table 2: User-based modifications for the `incrExplor()` and `mcReserve()` functions provided by specifying certain values for additional parameters `states` and `breaks`. Any combination of these two parameters can be further used with one of four methods for summarizing the run-off triangle increments within the bins—the default method “median” or any of “mean”, “min”, and “max” respectively. The modifications 1–4 are illustrated in Examples 1–4.

and directly supplemented into the `mcReserve()` function by using the additional parameters `states` and `breaks` as

```
R> mcReserve(CameronMutual, states = user.states, breaks = user.breaks)
```

provide two identical outputs and the same reserve prediction $\hat{\mathcal{R}}$. For some more detailed illustration of different settings of the `incrExplor()` function and, consequently, the `mcReserve()` function there are four specific examples provided below together with the corresponding graphical outputs.

Example 1 Let us start with the simplest modification that can be used for both the `incrExplor()` and `mcReserve()` functions (Case 1 in Table 2). The number of the Markov chain states (by default, the number of states $m \in \mathbb{N}$ is equal to the number of origins, $n \in \mathbb{N}$) can be changed by the parameter `states`. In the R code below, there are five Markov chain states specified, the parameter `breaks` is ignored, and the break points and states are derived in a data-driven manner. The following code

```
R> plot(incrExplor(CameronMutual, states = 5))
R> plot(mcReserve(CameronMutual, states = 5))
```

performs the exploratory analysis of the run-off triangle increments (in terms of the standard output already described above) including the graphical visualization—see Figure 5a—and, consequently, the reserve is obtained by the `mcReserve()` function while using the same five states—see Figure 5b for the corresponding completed profiles. Note that five Markov chain states are enforced but, otherwise, the run-off triangle increments are summarized analogously—they are split uniformly into five bins and the corresponding Markov states are obtained by using the median values of the increments within each bin (default method). The predicted reserve is slightly worse than the one obtained by the default setup of the MACRAME algorithm (which gives the predicted reserve roughly $\hat{\mathcal{R}} = 8082$ while the prediction with only five Markov states produces $\hat{\mathcal{R}} = 8620$ with the true “unknown” reserve being $\mathcal{R} = 7963$). Nevertheless, it can be sometimes useful to manually control (and lower) the amount of the Markov states that are used for the prediction.

Example 2 The parameter `states` also allows for an explicit definition of the Markov states for both `incrExplor()` and `mcReserve()`. In such case (Scenario 2 in Table 2), the corresponding break points $\{g_k\}_{k=1}^{m-1}$ are defined as the mid-points between the specified state values (which must be all unique). In addition, it holds that $g_0 = -\infty$ and $g_m = \infty$. Similarly as in Example 1, the R code below provides the exploratory analysis of the run-off triangle increments, but instead of specifying just the number of states to be used, explicit Markov chain states are enforced by using `states = c(500, 1000, 1500, 2000, 2500)`. The MACRAME algorithm uses the same set of states (and breaks) for the consequent reserve prediction. Note that as far as the explicit states are provided, any option of the parameter `method = "..."` in the `incrExplor()` function is irrelevant and, therefore, ignored. Analogous graphical outputs provided in Figure 6 are produced by the R commands

```
R> user.states <- c(500, 1000, 1500, 2000, 2500)
R> plot(incrExplor(CameronMutual, states = user.states))
R> plot(mcReserve(CameronMutual, states = user.states))
```

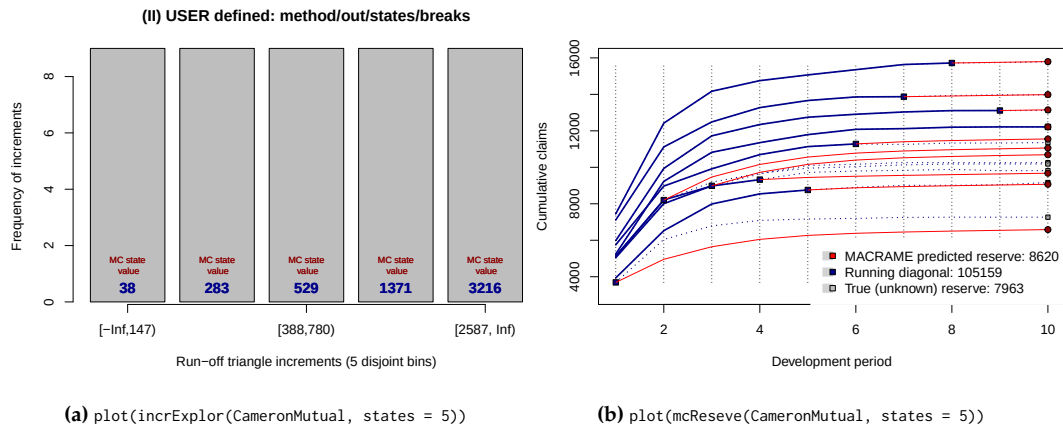


Figure 5: Partial plot of the output from `incrExplor()` (panel on the left) when applied to the `CameronMutual` dataset with the parameter choice `states = 5` (requiring five Markov states to be used). The run-off triangle increments are uniformly distributed into five bins and the corresponding Markov chain states are obtained by the default method—the medians of the increments within each bin are considered. The right panel shows the completed functional profiles provided by MACRAM with five Markov states used. The figure provides the observed, “unknown” future, and completed functional profiles. The predicted reserve, $\hat{R} = 8620$, is explicitly reported in the default legend (together with the paid amount and the target “unknown” reserve).

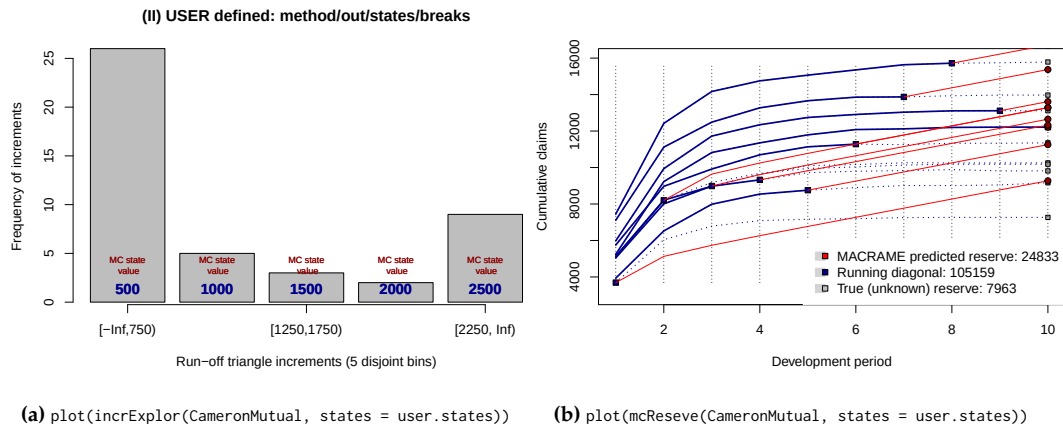


Figure 6: Partial plot of the output from the `incrExplor()` function (left) when applied to the `CameronMutual` dataset with an explicit specification of the states—`states = c(500, 1000, 1500, 2000, 2500)`. The increments are not distributed uniformly across the bins any more and relatively poor performance of the MACRAM algorithm is also observed in the completed profiles on the right (the predicted reserve heavily overestimates the true reserve due to almost no Markov chain states that would properly model the development tails).

Comparing Figure 5a (with `states = 5`) and Figure 6a (with `states = c(500, 1000, 1500, 2000, 2500)`), it is quite obvious that the numeric specification of the number of states preserves a uniform allocation of the increments in the bins (if possible). However, with the explicit states specification (and the break points $\{g_k\}_{k=1}^{m-1}$ being determined as the mid-points between the states) such uniform distribution is not (generally) possible any more and, therefore, relatively large variation is observed among the bars in Figure 6a. The Markov chain states in Figure 6a (blue bold values within the bars) indeed correspond with the declaration of the states provided by `states = c(500, 1000, 1500, 2000, 2500)`. Thus, there are again five bins for the run-off triangle increments but the bins are now more restricted. As a result, the development tails of the run-off triangle are approximated very roughly (note that there are more than 25 increments in the first bin) and, therefore, the reserve is (not surprisingly) heavily overestimated (compare Figure 6b with Figure 5b).

Example 3 Instead of enforcing the Markov chain states one can also pre-specify the set of breaks $\{g_k\}_{k=0}^m$ that are used to form the bins for the run-off triangle increments (Case 3 in Table 2). The corresponding states are obtained by summarizing the increments within the bins (either by using the default median method or by specifying other options with the `method` parameter). The following R code uses again five bins defined by four grid points and the corresponding Markov states are obtained as the medians of the increments within the bins.

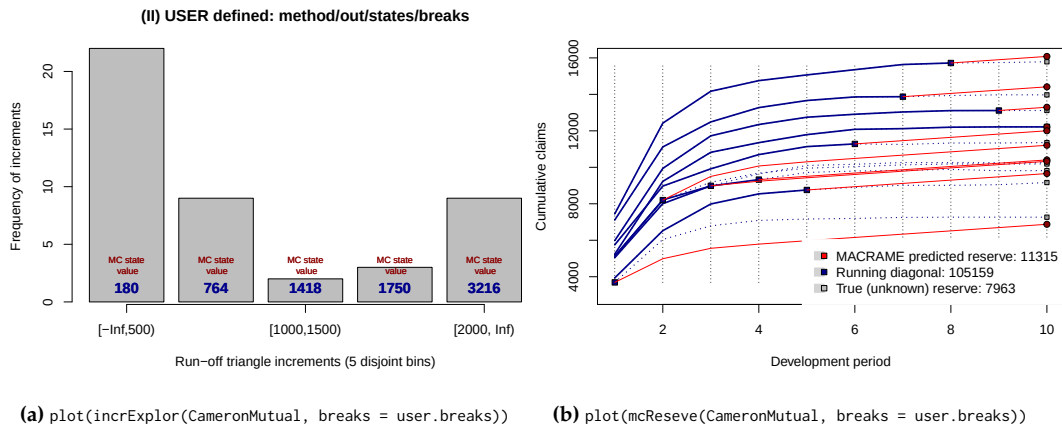


Figure 7: Partial plot of the output from the `incrExplor()` function (left) when applied to the `CameronMutual` dataset with an explicit specification of the breaks by using `breaks = c(500, 1000, 1500, 2000)`. The corresponding completed profiles are on the right-hand side. The predicted reserve is $\hat{\mathcal{R}} = 11315$.

```
R> user.breaks <- c(500, 1000, 1500, 2000)
R> plot(incrExplor(CameronMutual, breaks = user.breaks))
R> plot(mcReserve(CameronMutual, breaks = user.breaks))
```

Both plots are provided in Figure 7. It is also worth mentioning that if the specified break points $\{g_k\}_{k=1}^{m-1}$ define some empty bin with no increment within the bin, then two neighboring bins are automatically merged in a way that each bin contains at least one increment. More formally, if there is no increment in the bin $[g_k, g_{k+1})$ for some $k \in \{1, \dots, m-1\}$ then a new bin defined as $[g_{k-1}, g_{k+1})$ is used instead. Similarly, if there is no increment in the first bin (g_0, g_1) , where $g_0 = -\infty$, then the first bin is re-defined as $(-\infty, g)$, where $g = \min_{\{2 \leq k \leq m\}} \{g_k : g_k > \min\{X_{i,j} : 1 \leq i \leq n, 2 \leq j \leq n-i+1\}\}$. Thus, it is also possible, for some rather atypical run-off triangles, that there is only one bin used for all (typically zero) increments (no matter what is specified by states or breaks) and the underlying Markov chain process has only one (typically zero) state, i.e., $|\mathcal{S}| = |\{s\}| = 1$.

Example 4 Finally, we briefly address one more important modification of the underlying Markov chain. Note that the choice of the summary method (i.e., `method = c("median", "mean", "min", "max")`) is only applicable for the `incrExplor()` function. The `mcReserve()` function only allows one to specify the set of breaks and the set of states. This is however not limiting in practice.

The R code below specifies the user-based choice for breaks (i.e., `breaks = c(500, 1000, 1500, 2000)`) and, also, the user selected method how the increments within each bin are supposed to be summarized (i.e., `method = "min"`). The corresponding Markov chain states are calculated by the `incrExplor()` function and, together with the pre-specified breaks, they can be both forwarded to the `mcReserve()` function by using the `mcStates()` accessor and both parameters `breaks` and `states` (Case 4 in Table 2).

```
R> user.breaks <- c(500, 1000, 1500, 2000)
R> user.method <- incrExplor(CameronMutual, breaks = user.breaks, method = "min")

R> final.states <- mcStates(user.method)
R> mcReserve(CameronMutual, breaks = user.breaks, states = final.states)
```

The explicit outputs and plots are omitted for brevity. Note that if functions `incrExplor()` and `mcReserve()` are used with an explicit specification of both—the set of breaks² $\{g_k\}_{k=1}^{m-1}$ and the set of states $\mathcal{S} = \{s_1, \dots, s_m\}$ then the Markov states provided in `states = c( ... )` must represent a valid sequence of unique values such that exactly one state value belongs to exactly one bin determined by the break points $-\infty = g_0 < g_1 < \dots < g_{m-1} < g_m = \infty$.

3.3 Permutation bootstrap

The third key function implemented in the `ProfileLadder` package—providing the overall reserve distribution for functional-based reserving methods but also for traditional parametric approaches already

²The set of breaks $\{g_k\}_{k=1}^{m-1}$ can be specified differently but for a proper functionality of the MACRAME algorithm it is important that $g_0 = -\infty$ and $g_m = \infty$. The user can however either specify a sequence of breaks in a form $-\infty \neq g_1 < \dots < g_{m-1} \neq \infty$ and boundary values $g_0 = -\infty$ and $g_m = \infty$ are filled in automatically. Alternatively, the sequence of break points with one or both boundary points can be supplied.

For illustration, the overall reserve distribution for the CameronMutual1 dataset—provided by the MACRAME algorithm with 500 permutation bootstrap resamples can be obtained as

Two sets of values are listed in the output. The first part corresponds to the point prediction of the overall reserve (also discussed before) and the second part of the output summarizes the overall reserve distribution. The first two values in the second part of the output—the average (`Boot.Mean`) over $B \in \mathbb{N}$ bootstrap reserves $\{\hat{\mathcal{R}}^{(b)}\}_{b=1}^B$ and the corresponding standard error (`Std. Er.`)—provide some basic characterization of the location and shape of the estimated reserve distribution. In addition, two qualitative measures proposed in [Maciak et al. \(2022\)](#) are provided for some practical assessment of the resampling performance. The value of `BootCov%` expresses (in percentages) the coefficient of variation of the bootstrap distribution and the value of `BootVar.995` gives the 99.5% quantile (by default) of the distribution relative to the bootstrap mean (other choice of the quantile can be applied by changing the parameter `quantile = 0.995`). Both these quantities were defined, explained, and consequently also used for the empirical comparisons in [Maciak et al. \(2022\)](#) and more detailed discussion can be also found there. Note that the default quantile level of 0.995 is explicitly required by the Solvency II principle.

```
R> summary(distribution.macrame)
```

Overall reserve distribution

The MACRAME predicted reserve represents the 15.37% quantile of the distribution
Bootstrap simulated reserves beyond 2sigma rule: 20 (out of 500)

```
R> plot(distribution.macrame)
```


Prediction method	Point prediction	Permutation bootstrap	Residual bootstrap
PARALLAX	0.03s. (0.006)	15.01s. (0.341)	✗
REACT	0.01s. (0.001)	9.04s. (0.116)	✗
MACRAME	0.04s. (0.012)	48.75s. (11.51)	✗
Chainladder	0.03s. (0.008)	31.70s. (0.276)	✗
Mack model	0.04s. (0.002)	42.10s. (0.650)	✗
ODP model	3.86s. (0.043)	63.37m. (0.148)	∞ m. (NA*)
Tweedie formula	0.24s. (0.026)	4.19m. (0.307)	3.69m. (0.252)

Table 3: Computational efficiency in terms of the running times for different prediction algorithms from the **ProfileLadder** and **ChainLadder** package. Point predictions together with distributional predictions obtained by the proposed permutation bootstrap and the classical residual bootstrap (applied only to some (available) methods from the **ChainLadder** package) are compared (using 1000 bootstrap replicates). Three run-off triangles of the dimensions 60×60 from the dataset GFCIB from **ProfileLadder** are used and the running times (given either in seconds [s.] or minutes [m.] for brevity) are averaged (with the corresponding standard errors in brackets).

* The residual bootstrap for the ODP model was terminated after six days of running without any results.

`glmReserve()`; and Tweedie model implemented in `tweedieReserve()`) we used three large (60×60) run-off triangles from the GFCIB dataset (the fourth run-off triangle—the annuities—cannot be used with the chainladder method and the Mack model due to some fully zero rows) and the point predictions were obtained together with the distributional predictions (using the permutation bootstrap for each method and the residual based bootstrap implemented in **ChainLadder** for the ODP model (`glmReserve(...,mse.method = "bootstrap")`) and the Tweedie model (`tweedieReserve(...,bootstrap = 1)`). The number of bootstrap resamples was always set to $B = 1000$ and MacOS Architecture with 12 Apple M2 Pro chips (8 for performance and 4 for efficiency) with 32GB memory was used to get the results summarized in Table 3.

The limitations of the proposed methods are, on the other hand, mostly determined by the **ChainLadder** package rather than the functions from **ProfileLadder** themselves. In principle, there is no theoretical limitation for PARALLAX, REACT, or MACRAME to handle as minimal run-off triangles as 2×2 . However, unlike `parallelReserve()` which indeed handles such cases, `mcReserve()` does not (and the minimum dimensions required are 3×3). This is caused by the internal implementation of the `cum2incr()` function from the **ChainLadder** package which transforms cumulative triangles into incremental ones. On the other hand, there are no formal limitations for the maximum allowed dimensions that we would be aware of.

3.5 Other features of the R package **ProfileLadder**

The main core of the R package **ProfileLadder** consists of three key functions—`parallelReserve()` for applying the PARALLAX or REACT algorithm, `mcReserve()` implementing the MACRAME algorithm, and `permutedReserve()` providing the permutation bootstrap add-on. Other generic functions (for the R objects of the class `profileLadder`, `profilePredict`, `mcSetup`, and `permutedReserve`) are also implemented to facilitate a well structured summary of the outputs and graphical visualizations of the results. In addition, there are a few other helpful functions implemented in **ProfileLadder**. For a complex description and illustrative examples, we refer to the R help session by using the standard `help()` command. Below, we only provide a brief description of the three most relevant functions that in some sense interconnect our package **ProfileLadder** with the core actuarial package **ChainLadder** and some typical insurance standards.

- `as.profileLadder()` For the run-off triangle data type there is already an R class `triangle` implemented in the R package **ChainLadder** by Gesmann et al. (2025) with the corresponding R function `as.triangle()`. This is a generic S3 class method that eases the work with the triangle shaped matrix data. However, the generic summary and plot methods do not allow distinguishing the observed functional profiles and the unknown “future” segments when using this R class. Therefore, we introduce an enhanced run-off triangle class `profileLadder` that provides more appropriate organization suitable for the nonparametric functional-based reserving techniques. For illustration, compare the following R code that produces two plots in Figure 9.

```
### S3 class 'triangle' from the ChainLadder pkg
R> plot(as.triangle(CameronMutual))

### S3 class 'profileLadder' from the ProfileLadder pkg
R> plot(as.profileLadder(CameronMutual))
```

The new R class `ProfileLadder` also provides some additional information about the underlying

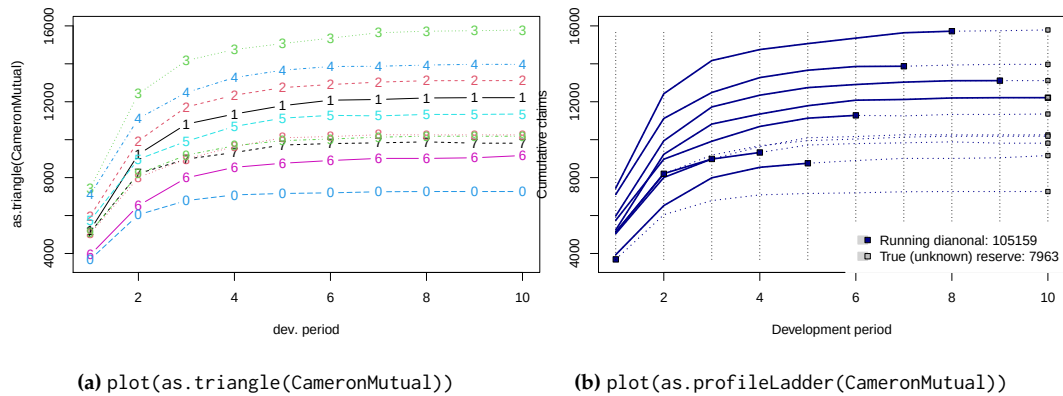


Figure 9: Graphical comparison of the performance of the generic R method `plot()` when applied to the same data set (CameronMutual) represented by different R classes. The left-hand panel shows the result for the `triangle` class defined in the package `ChainLadder`; the result for the `profileLadder` class is shown on the right-hand side.

run-off triangle (for instance, the true reserve amount if available, the triangle type—cumulative or incremental—and way more) and it also adapts various information provided by the functional-based algorithms (PARALLAX, REACT, or MACRAME);

- `observed()` This function provides practical layouts (of different types) to distinguish observed and unobserved functional profiles in the run-off triangle (mainly in situations when the triangle is provided in terms of a full data matrix). This R function also allows for an application of classical reserving approaches from the `ChainLadder` package (such as the `glmReserve()`, `tweedieReserve()`, or `MackChainLadder()`) to be easily used with the fully observed run-off triangles supplemented in the `ProfileLadder` package. The run-off triangles included as example datasets in the `ProfileLadder` package are (mostly) given in terms of full (squared) matrices (with the “unknown” future development profiles being also provided for an ex-post performance evaluation) while the parametric methods implemented in the `ChainLadder` package rely solely on the input data in form of the run-off triangle—meaning that $Y_{i,j} = X_{i,j} = \text{NA}$ for all $i + j > n + 1$. Compare, for instance, the following (where the first command results in an error message while the latter already works properly):

```
R> glmReserve(CameronMutual)
R> glmReserve(observed(CameronMutual))
```

The same reasoning also applies for other parametric reserving methods from the `ChainLadder` package (`tweedieModel()` and `MackChainLadder()` functions in particular).

- `predict()` A generic S3 class method applicable to the outputs of the `parallelReserve()` function and the `mcReserve()` function—the objects of the class `profileLadder`. For internal purposes of insurance companies, actuaries are at times interested in a one-step-ahead prediction only (typically a 1-year-ahead prediction say). Instead of completing the run-off triangle into a full square, the focus is then on predicting the so-called *next running diagonal*. The S3 method `predict.profileLadder()` performs this as

```
R> predict(parallelReserve(CameronMutual))
```

Unlike the standard run-off triangle completion where the output of `parallelReserve()` or `mcReserve()` is a matrix of the dimensions $n \times n$, the output of the `predict()` method is actually a new run-off triangle of the dimensions $n \times (n + 1)$ (the output is omitted for brevity).

4 Concluding remarks and summary

The `ProfileLadder` package is particularly developed to implement nonparametric methods into the actuarial risk assessment process performed by insurance companies (typically on a yearly or quarterly basis). Nevertheless, the underlying run-off triangle can be formally also represented in terms of an incomplete panel data scheme that is generally well known among statisticians and all types of practitioners. The statistical techniques and specific functions implemented in the R package

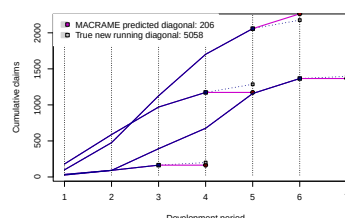
ProfileLadder can be also used for such data. Some explicit datasets (briefly described below) are included in the package to illustrate possible applications beyond the actuarial domain.

- **Health care & epidemiology** Triangular data, similar to run-off triangles, are common in tracking medical claims and receipts until resolution. Another example is epidemiological monitoring, such as tracking disease spread by new cases in specific cohorts over time. The dataset `covid19CZ` (see `help("covid19CZ")`) captures the spread of Covid-19 in Czechia over the first eight weeks using weekly cohort data by counties (grouped by the week of the first case occurrence). We use also this data to demonstrate the functionality of the `predict()` method mentioned at the end of the previous section. This function is not only useful for the actuaries, but it also has important practical utilization in any type of risk modeling. The S3 method `predict()` is implemented for the objects of the R class `profileLadder` that are created by the `parallelReserve()` function or the `mcReserve()` function (for details, we refer to the help session obtained by `help("predict.profileLadder")`). Instead of completing the run-off triangle into a full square (by one of the algorithms `PARALLAX`, `REACT`, or `MACRAME`), the `predict()` method only returns the prediction of the next running diagonal (also called a *1-step-ahead prediction*). The same algorithms are used for the diagonal prediction, only the output is not a completed square but rather a new (extended) run-off triangle of the dimensions $n \times (n + 1)$.

For illustration, the `predict()` method is applied below to the run-off triangle formed from the dataset `covid19CZ`, using all four cohorts (rows) and weeks 3 to 6 (i.e., columns 3 to 6), to allow for an ex post comparison between the diagonal prediction and the actual true outcome (the next running diagonal in `covid19CZ`). The output from the `predict()` method is another object of the R class `profilePredict` with the corresponding plot method, which provides a graphical illustration—see the R code and Figure 10 below.

```
R> diagonal <- predict(mcReserve(covid19CZ[,3:6]))
R> print(diagonal)
```

01/03 - 07/03	392	676	1158	1366	1396
08/03 - 14/03	1126	1702	2058	2264	.
15/03 - 21/03	970	1173	1173	.	.
22/03 - 28/03	164	164	.	.	.



```
R> plot(diagonal, trueProfiles = covid19CZ[,1:7])
```

Figure 10: 1-step-ahead prediction

Moreover, the S3 method `predict()` can be applied repeatedly to predict other consecutive diagonals as the output of the `predict()` function is a run-off triangle (although of a rectangular shape with dimensions $n \times (n + 1)$), and, thus, it can be forwarded again as the input for `parallelReserve()` or `mcReserve()` (considering all n rows but only last n columns).

- **Accounting & auditing** Tracking unpaid invoices or tax adjustments based on past years is common in business and government, and such data often form a triangle structure too. The dataset `xNetSubscribe` (see `help("xNetSubscribe")`) illustrates this with monthly income from a local internet provider, using cohorts of new subscribers by month;
- **Operational risk & banking** In general, the occurrence and development of events like fraud, IT failures, or compliance breaches can be viewed through the lens of operational risk. Such events are often tracked using development triangles, with rows indicating occurrence time and columns showing development periods;
- **Manufacturing & quality control** Triangular data also arise in various production processes. The triangular schemes are typically generated when tracking defects and failures within specific production batches in order to predict failure rates and to manage maintenance schedules.

From the overall point of view, the nonparametric point prediction methods implemented in the R package **ProfileLadder** (the algorithms `PARALLAX`, `REACT`, and `MACRAME`) are, despite the fact that all were developed specifically for the claims reserving purposes, widely applicable to any triangular data schemes arising in real-life applications. There are three core functions in the package: the first one, `parallelReserve()`, implements the `PARALLAX` algorithm and the `REACT` algorithm; the second one, `mcReserve()`, incorporates a more complex Markov chain prediction mechanism with a whole variety of user-based options; the third function, `permuteReserve()`, estimates for the overall reserve distribution—not exclusively for the proposed functional-based methods, but also for classical parametric reserving approaches. This function may thus serve as a practical alternative to a parametric (residual) bootstrap. Auxiliary R functions are also implemented to make the work with the triangular shaped data easier and more straightforward especially when interacting with the actuarial package **ChainLadder**. Finally, unique illustrative datasets from the actuarial practice and other practical real-world areas are featured as well.

Acknowledgment The work of Maciak, Mizera, and Pešta was supported by the Czech Science Foundation

grant GAČR 23–06461K. The work of Matúš was supported by MSD Czech Republic. The authors express sincere thanks to Kurt Hornik for his insight and some useful pieces of advice regarding the **ProfileLadder** package. The authors are also grateful to Petr Jedlička from the Czech Insurers' Bureau and Pavel Koudelka from Generali Česká pojišťovna for providing complex data from real-life insurance practice—not only for internal evaluation purposes but also for public access within the R package **ProfileLadder**. Last but not least, the authors appreciate the insight and help of Rob Hyndman and two anonymous reviewers—particularly for their suggestions and comments that helped to improve the overall quality of the package and the manuscript. The authors declare no conflict of interest.

References

- A. Bauer, F. Scheipl, H. Küchenhoff, and A.-A. Gabriel. Registration for incomplete non-Gaussian functional data, 2021. URL <https://arxiv.org/abs/2108.05634>. [p139]
- B. Campo. The **actuaRE** package: Handling hierarchically structured risk factors using random effects models, 2025. URL <https://CRAN.R-project.org/package=actuaRE>. R package version 0.1.3. [p139]
- Z. Chmielewska. **actuaryr**: Develop actuarial models, 2019. URL <https://CRAN.R-project.org/package=actuaryr>. R package version 1.1.1. [p139]
- D. R. Clark. LDF curve-fitting and stochastic reserving: A maximum likelihood approach. *Casualty Actuarial Society*, pages 41–92, 2003. URL https://www.casact.org/sites/default/files/database/forum_03ffforum_03ff041.pdf. E-Forum, Fall 2003. [p139, 140]
- A. Delaigle and P. Hall. Classification using censored functional data. *Journal of the American Statistical Association*, 108(504):1269–1283, 2013. doi: 10.1080/01621459.2013.824893. [p139]
- A. Delaigle and P. Hall. Approximating fragmented functional data by segments of Markov chains. *Biometrika*, 103(4):779–799, 2016. doi: 10.1093/biomet/asw040. [p139]
- C. Dutang, V. Goulet, and M. Pigeon. **actuar**: An R package for actuarial science. *Journal of Statistical Software*, 25(7): 1–37, 2008. URL <https://www.jstatsoft.org/index.php/jss/article/view/v025i07>. [p139]
- P. D. England and R. J. Verrall. Analytic and bootstrap estimates of prediction errors in claims reserving. *Insurance: Mathematics and Economics*, 25(3):281–293, 1999. doi: 10.1016/S0167-6687(99)00016-5. [p143]
- European Parliament and Council. Directive 2009/138/EC of the European Parliament and of the Council of 25 November 2009 on the taking-up and pursuit of the business of Insurance and Reinsurance (Solvency II). *Official Journal of the European Union*, 52(L 335):1–155, 2009. URL <https://eur-lex.europa.eu/eli/dir/2009/138/oj/eng>. [p139]
- J. P. O. Galmiche. *Fragmented Functional Data*. Master Thesis, Politecnico di Milano, Italy, 2016. URL <https://www.politesi.polimi.it/handle/10589/135829>. [p139]
- M. Gesmann, D. Murphy, Y. Zhang, A. Carrato, M. Wuthrich, F. Concina, and E. Dal Moro. **ChainLadder**: Statistical methods and models for claims reserving in general insurance, 2025. URL <https://cran.rstudio.com/web/packages/ChainLadder/vignettes/ChainLadder.html>. R package version 0.2.20. [p139, 157]
- D. Liebl and S. Rameseder. Partially observed functional data: The case of systematically missing parts. *Computational Statistics and Data Analysis*, 131:104–115, 2019. doi: 10.1016/j.csda.2018.08.011. [p139]
- M. Maciak, M. Pešta, and O. Okhrin. Infinitely stochastic micro reserving. *Insurance: Mathematics and Economics*, 100:30–58, 2021. doi: 10.1016/j.insmatheco.2021.04.007. [p139]
- M. Maciak, I. Mizera, and M. Pešta. Functional profile techniques for claims reserving. *ASTIN Bulletin*, 52(2): 449–482, 2022. doi: 10.1017/asb.2022.4. [p139, 140, 141, 142, 143, 144, 146, 147, 148, 149, 151, 155]
- T. Mack. Distribution-free calculation of the standard error of chain ladder reserve estimates. *ASTIN Bulletin*, 23(2): 213–225, 1993. doi: 10.2143/AST.23.2.2005092. [p139]
- G. G. Meyers and P. Shi. Loss reserving data pulled from NAIC Schedule P, 2011. URL <https://www.casact.org/publications-research/research-resources/loss-reserving-data-pulled-naic-schedule-p>. [Accessed June 10, 2014]. [p140]
- Y. Parizas. **NetSimR**: Actuarial functions for non-life insurance modelling, 2019. URL <https://CRAN.R-project.org/package=NetSimR>. R package version 0.1.5. [p139]
- P. J. R. Pinheiro, J. M. Andrade e Silva, and M. De Lourdes Centeno. Bootstrap methodology in claim reserving. *Journal of Risk and Insurance*, 70(4):701–714, 2003. doi: 10.1046/j.0022-4367.2003.00071.x. [p143]
- R. Popescu and D. Suci. Glancing back. *The Actuary*, 2020(July):20–22, 2020. URL <https://www.theactuary.com/issues/2020/07/july-2020>. [p146]
- A. E. Renshaw and R. J. Verrall. A stochastic model underlying the chain-ladder technique. *British Actuarial Journal*, 4:903–923, 1998. doi: 10.1017/S1357321700000222. [p139]
- G. A. Spedicato. The **lifecontingencies** package: Performing financial and actuarial mathematics calculations in R. *Journal of Statistical Software*, 55(10):1–36, 2013. doi: 10.18637/jss.v055.i10. [p139]
- T. Verdonck and M. Debruyne. The influence of individual claims on the chain-ladder estimates: Analysis and diagnostic tool. *Insurance: Mathematics and Economics*, 48(1):85–98, 2011. doi: 10.1016/j.insmatheco.2010.10.001. [p139]
- R. J. Verrall. Claims reserving and generalised additive models. *Insurance: Mathematics and Economics*, 19(1):31–43, 1996. doi: 10.1016/S0167-6687(96)00000-5. [p139]

Matúš Maciak, Ivan Mizera, Michal Pešta

Charles University, Faculty of Mathematics and Physics
Sokolovská 49/83, Prague, 186 00
Czech Republic

maciak|mizera|pesta@karlin.mff.cuni.cz

Rastislav Matúš

Merck Sharp & Dohme s.r.o.
Na Valentince 3336/4, Prague, 150 00
Czech Republic

rastislav.matus@merck.com

blocking: An R Package for Blocking of Records for Record Linkage and Deduplication

by Maciej Beręsewicz and Adam Struzik

Abstract Entity resolution (probabilistic record linkage, deduplication) is essential for bringing together data needed for estimation or other analyses based on multiple sources. It aims to link records that refer to the same entity (e.g., person, company) in the absence of unique identifiers. Without identifiers, researchers must specify which records to compare to calculate matching probability and reduce computational complexity. Traditional deterministic blocking uses common variables like first letters of names, sex, or dates of birth, but assumes error-free, complete data. To address this limitation, we developed the R package **blocking**, which uses approximate nearest neighbour search and graph algorithms to reduce the number of comparisons. This paper presents the package design, functionalities, and two case studies.

1 Introduction

1.1 Blocking for record linkage

Entity resolution (probabilistic record linkage, deduplication) is essential for bringing together data needed for estimation or other analyses based on multiple sources (cf. Fellegi and Sunter, 1969; Binette and Steorts, 2022). The goal is to link records that refer to the same entity (e.g., person, company, job position) in the absence of unique identifiers. This situation is frequently observed in administrative records, e.g., for foreign-national populations. For instance, the Social Insurance Institution register in Poland at the end of 2023 included 1.206 million records referring to approximately 1.105 million individuals, of which about 10% had missing information in the personal identifier (PESEL) and about 50% had missing address details. Note that the exact number of individuals is certainly lower than 1.105 million, as the 10% with missing identifiers may include duplicates (cf. Beręsewicz, 2025).

This drives the need to link records without unique identifiers, which often requires certain assumptions about how to reduce the large number of possible comparisons, as it is not feasible to compare all pairs of records in large datasets (e.g., the aforementioned example would require over 700 billion comparisons). Consequently, researchers aim to reduce the number of comparisons in various ways prior to the record linkage/deduplication stage. The rationale is twofold: computational resource constraints and clerical review workload (cf. Christen, 2012b).

Reducing the number of comparisons is accomplished through blocking, a method that limits possible comparisons by assuming that certain variables must match exactly or that combinations of variables should match above a specified threshold. For instance, a standard approach assumes that sex or birth year must match exactly, while other record characteristics may vary. Another method employs phonetic algorithms such as Soundex (cf. Knuth, 1973) or its adaptations for non-English languages (cf. Howard II, 2020) to block records that sound similar but are spelt differently (e.g., Smith and Smyth, or Anna and Ania). Furthermore, with the growing popularity of large language models (both closed and open-source), one may consider using embeddings (Mikolov et al., 2013) to identify nearest neighbours and treat these as potential comparison pairs. For a comprehensive review of blocking methods, see Christen (2012a), Steorts et al. (2014), or Papadakis et al. (2020). Section 1.2 discusses existing R packages that implement blocking methods.

Reducing the number of pairs has inherent costs: missed comparisons lead to increased false positive rates (FPR) and false negative rates (FNR) in linkage studies. To validate the blocking strategy, a subset of true pairs should be provided or simulation studies of proposed methods should be conducted. Alternatively, one may consider approaches proposed by Dasylyva and Goussanou (2021) and Dasylyva and Goussanou (2022), who demonstrated how to estimate FPR and FNR without access to an audit sample.

1.2 Existing software and our contribution

The R ecosystem offers several packages that implement various blocking techniques, which we have grouped according to the following classification:

- **Deterministic blocking:**

- **reclin2** (van der Laan, 2024, 2022) allows pairing records using `pair_blocking()` with a prespecified list of columns in a `data.frame`, and the `pair_minsim()` function, which allows specifying the minimal similarity score (e.g., 1 out of 3 variable values must match exactly).
 - **RecordLinkage** (Sariyar and Borg, 2025, 2010) allows specifying blocking variables in the `blockfld` parameter of either `compare.dedup()` or `compare.linkage()` functions as a vector (either character or numeric).
 - **fastLink** (Enamorado et al., 2023, 2019) implements various blocking methods via the `blockData()` function, including exact matching, window matching (e.g., no more than a 2-year difference between birth years), and *k*-means clustering. Notably, **fastLink** returns datasets split into separate lists, while **reclin2** and **RecordLinkage** packages create a single dataset.
 - **PPRL** (Schnell and Rukasz, 2025) focuses on privacy-preserving deterministic and probabilistic record linkage and implements exact blocking via the `SelectBlockingFunction()` function.
- **Phonetic blocking:**
 - **RecordLinkage** allows direct specification of phonetic comparison via the `phonetic` argument in `compare.dedup()` or `compare.linkage()` functions using the `soundex()` function.
 - Additionally, **stringdist** (van der Loo, 2014) implements the Soundex algorithm, while **phonics** (Howard II, 2021) implements various phonetic algorithms that can be applied prior to the blocking procedure (e.g., to create a new column).
 - **Probabilistic blocking:**
 - **klsh** (Steorts, 2020) is the only R package that implements probabilistic blocking using the *k*-means variant of locality sensitive hashing. The main `klsh()` function implements this approach, and the resulting object is a list containing row identifiers for the prespecified number of blocks (via the `num.blocks` argument).

In practice, the situation is more complicated, as missing data may be present in blocking/matching variables (such as birth dates) or typos may occur in names and surnames. Therefore, we developed **blocking**, which leverages approximate nearest neighbour (ANN) algorithms and graphs to create numerous small blocks that can be used in subsequent analysis. ANN algorithms are methods designed to efficiently find the closest matches to a query point in a large dataset by trading off search accuracy for computational speed. Unlike exact nearest neighbour search, ANN algorithms do not guarantee finding the true nearest neighbours but are able to identify close approximations substantially faster (cf. Indyk and Motwani, 1998; Wang et al., 2021), making them naturally suited to noisy or incomplete data. This approach is similar to micro-clustering (cf. Johndrow et al., 2018), but we do not aim at providing the final linkage of units between or within sources.

The basic workflow of the **blocking** package consists of the following steps as shown in Figure 1. These steps can be explained as follows:

1. Create shingles (n-character sequences, e.g. "home" will be `c("ho", "om", "me")`) of the input character vectors using either **tokenizers** (Mullen et al., 2018) or **text2vec** (Selivanov et al., 2023). The former tokenises strings into character n-grams, while the latter additionally provides term-frequency weighting. Alternatively, the user may provide a matrix of vector representations of the input strings. Such representations, known as embeddings, encode semantic or contextual similarity between strings as dense numeric vectors. Embeddings can be obtained, for example, via **ragnar** (Kalinowski and Falbel, 2025), a package that provides an interface to large language models for generating text embeddings (either static or dynamic). An example using embeddings is provided in the Supplementary Materials.
2. Search for nearest neighbours using ANN algorithms implemented in **rnndescent** (the only one allowing for sparse matrices, cf. Melville, 2024b), **RcppHNSW** (Melville, 2024a), **mlpack** (Curtin et al., 2023; Singh Parihar et al., 2025), and **RcppAnnoy** (Eddelbuettel, 2024).
3. Create final blocks using **igraph** (Csárdi et al., 2025; Csárdi and Nepusz, 2006).

This is the only package in the R ecosystem that readily applies modern ANN algorithms to reduce the number of comparisons and substantially accelerate record linkage and deduplication tasks. Additionally, we have developed the `pair_ann()` function for seamless integration with the **reclin2** package, as described in the package vignette entitled "Integration with existing packages". The **blocking** package is not limited to the **reclin2** and can be applied in various record linkage or deduplication contexts. Furthermore, one can combine results from our package with deterministic methods, e.g., by blocking on the first letter of a variable in addition to the ANN-based candidate pairs.

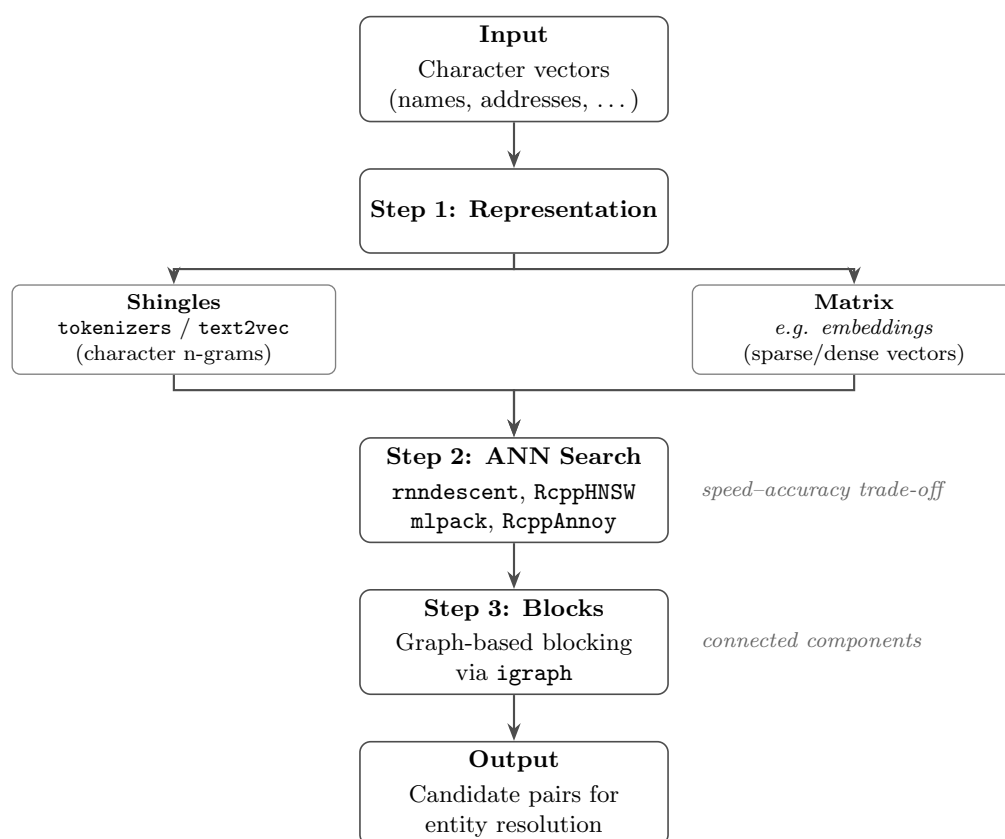


Figure 1: Workflow of the 'blocking()' function

1.3 Outline of the article

This paper is structured as follows. Section 2 provides a description of the main functionalities of the **blocking** package and how results can be evaluated. Section 3 presents two case studies: probabilistic record linkage and deduplication. These examples demonstrate how our package can improve the entity resolution pipeline and integrate with existing R packages.

2 Blocking of records using blocking() function

2.1 The main function

The main functionality is available via the `blocking()` function, which contains the following key arguments:

- `x`, `y` – vectors of keys used for linkage, where `y = NULL` indicates a deduplication task;
- `representation` – whether `x` and `y` should be represented as shingles, a custom matrix, or vectors (e.g., provided by the user via the `model` argument);
- `ann` – which ANN algorithm should be applied (by default, we use the `rnnodescent` package as it supports sparse matrices);
- `distance` – which distance metric should be applied (default is cosine distance);
- `graph` – whether a plot of the graph showing connected records should be returned (default `FALSE`);
- `true_blocks` – if a subset of true blocks is available, it can be provided here so that quality measures, presented in the next section, are returned;
- `n_threads` – number of threads used for computation;
- `control_txt` – controls provided via `controls_txt()` specifying how `x`, `y` are processed;
- `control_ann` – controls provided via `controls_ann()` allowing users to fine-tune the ANN algorithm (see documentation for the `controls_ann()` function and `control_*` functions with names referring to specific algorithms, e.g., `control_nnd()` for the NND algorithm).

This function returns an object of class `blocking` containing the following elements:

- `result` – a `data.table` with indices (rows) of `x`, `y`, `block`, and distance between points;
- `method` – name of the ANN algorithm used;
- `deduplication` – information about whether deduplication was applied;
- `representation` – information about whether shingles, a custom matrix, or vectors were used;
- `metrics` – quality assessment metrics, if `true_blocks` is provided;
- `confusion` – confusion matrix, if `true_blocks` is provided;
- `colnames` – variable names (`colnames`) used for search;
- `graph` – an `igraph` class object.

2.2 Assessment of results

The package implements several measures that can be used to assess results. The first is the *reduction ratio* (RR, cf. Christen and Goiser, 2007), which indicates the reduction in comparison pairs within the given blocks. It has a value between $[0, 1]$, where 1 indicates perfect reduction while values close to 0 indicate poor reduction. The RR indicator for deduplication has the following form:

$$RR_{\text{dedup}} = 1 - \frac{\sum_{i=1}^k \binom{|B_i|}{2}}{\binom{n}{2}},$$

where k is the total number of blocks, n is the total number of records in the dataset, and $|B_i|$ is the number of records in the i -th block. $\sum_{i=1}^k \binom{|B_i|}{2}$ is the number of comparisons after blocking, while $\binom{n}{2}$ is the total number of possible comparisons without blocking. For record linkage, the reduction ratio is defined as follows:

$$RR_{\text{reclin}} = 1 - \frac{\sum_{i=1}^k |B_{i,x}| \cdot |B_{i,y}|}{m \cdot n},$$

where m and n are the sizes of datasets X and Y , and k is the total number of blocks. The term $|B_{i,x}|$ is the number of records from dataset X in the i -th block, while $|B_{i,y}|$ is the number of records from dataset Y in the i -th block. The expression $\sum_{i=1}^k |B_{i,x}| \cdot |B_{i,y}|$ represents the number of comparisons after blocking.

Another way to assess blocking is to examine the confusion matrix at the *block* level, i.e., blocking results are compared with ground-truth *blocks* in a pairwise manner (e.g., one true positive pair occurs when both records from the comparison pair belong to the same predicted *block* and to the same ground-truth *block* in the evaluation data.frame). The values in this table are defined as follows:

- True Positive (TP): Record pairs correctly matched in the same block.
- False Positive (FP): Record pairs identified as matches that are not true matches in the same block.
- True Negative (TN): Record pairs correctly identified as non-matches (different blocks).
- False Negative (FN): Record pairs identified as non-matches that are true matches in the same block.

Metrics calculated based on this confusion matrix are defined in Table 1. In the context of blocking, these metrics are interpreted as follows:

- Accuracy: The ability to discriminate true pairs (both matches and non-matches).
- Precision (Positive Predictive Value, or Pairs Quality): The proportion of true matches among the candidate pairs. In blocking, it is expected to be lower than in final linkage.
- False Positive Rate (FPR): The share of non-matches incorrectly included in the blocks. It is expected to be low, as pairs of non-links should be spotted as True Negatives.
- False Negative Rate (FNR): The share of matches missed by blocking. It is expected to be low, as blocking should discover all pairs one wants to compare in entity resolution.
- F1 Score: Combines precision and FNR. However, it may be less relevant in the context of blocking, since low precision and low FNR are expected. For a critique of the F1 score in the context of record linkage see Hand and Christen (2018), and for an alternative F^* measure see Hand et al. (2021) (not yet implemented).

Table 1: Evaluation metrics

Metric	Formula	Metric	Formula
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	Precision	$\frac{TP}{TP+FP}$
False Positive Rate	$\frac{FP}{FP+TN}$	False Negative Rate	$\frac{FN}{FN+TP}$
F1 Score	$2 \cdot \frac{\text{Precision} \cdot (1-\text{FNR})}{\text{Precision} + (1-\text{FNR})}$		

The `blocking()` function also returns recall (sensitivity, or pairs completeness) and specificity, defined as $\text{Recall} = 1 - \text{FNR}$ and $\text{Specificity} = 1 - \text{FPR}$.

3 Case studies

3.1 An example of blocking for record linkage

Let us first load the required packages.

```
library("blocking")
library("data.table")
library("reclin2")
```

We demonstrate the use of the `blocking()` function for record linkage on the foreigners dataset included in the package. This fictional representation of the foreign population in Poland was generated based on publicly available information, preserving the distributions from administrative registers. It contains 110,000 rows with 100,000 entities (thus containing 10,000 duplicates). Each row represents one record, with the following columns: `fname` – first name, `sname` – second name, `surname` – surname, `date` – date of birth, `region` – region (county), `country` – country, and `true_id` – a person identifier.

Next, we load the data and examine the first six records.

```
data("foreigners")
setDT(foreigners)
head(foreigners)

#>   fname sname  surname      date region country true_id
#>   <char> <char>   <char>   <char> <char>  <char>   <num>
#> 1:  emin          imanov 1998/02/05      031      0
#> 2: nurlan      suleymanli 2000/08/01      031      1
#> 3:  amio      maharrsmov 1939/03/08      031      2
#> 4:  amik      maharramof 1939/03/08      031      2
#> 5:  amil      maharramov 1993/03/08      031      2
#> 6: gadir      jahangirov 1991/08/29      031      3
```

In the next step, we split the dataset into two separate `data.frames`: one containing the first appearance of each entity in the foreigners dataset, and the other containing subsequent appearances. We then add row identifiers (`x` and `y`).

```
foreigners_1 <- foreigners[!duplicated(foreigners$true_id), ]
foreigners_1[, x := 1:.N]
foreigners_2 <- foreigners[duplicated(foreigners$true_id), ]
foreigners_2[, y := 1:.N]
```

Now, in both datasets we remove separators from the date column and create a new character column that concatenates information from all columns (excluding `true_id`) in each row. In computer science, this is called schema-agnostic blocking (cf. Papadakis et al., 2016). Information stored in the `txt` column will be used for blocking records in the `blocking()` function.

```
foreigners_1[, txt := paste0(fname, sname, surname, gsub("/", "", date),
                             region, country)]
foreigners_2[, txt := paste0(fname, sname, surname, gsub("/", "", date),
                             region, country)]
head(foreigners_1[, .(true_id, txt)])
```

```
#>   true_id      txt
#>   <num>    <char>
#> 1:      0   eminimanov19980205031
#> 2:      1  nurlansuleymanli20000801031
#> 3:      2   amiomaharrsmov19390308031
#> 4:      3   gadirjahangirov19910829031
#> 5:      4  zaurbayramova1996100601261031
#> 6:      5   asifmammadov19970726031
```

The default algorithm is the Nearest Neighbour Descent Method (Dong et al., 2011) implemented in the `nndescent` package. Note that a default parameter of the `blocking()` function is `seed = 2023`, which sets the random seed.

```
result_reclin <- blocking(x = foreigners_1$txt,
                        y = foreigners_2$txt)
```

Now, we can examine the results by printing the `result_reclin` object. In this example, we have created 6,470 blocks utilizing 1,232 2-character shingles. Blocks are small, as we have 3,920 blocks of 2 elements, 1,599 blocks of 3 elements, ..., 2 blocks of 7 elements.

```
result_reclin

#> =====
#> Blocking based on the nnd method.
#> Number of blocks: 6470.
#> Number of columns used for blocking: 1232.
#> Reduction ratio: 0.9999.
#> =====
#> Distribution of the size of the blocks:
#>    2    3    4    5    6    7
#> 3920 1599  928   19    2    2
```

To access the result, one should use `result_reclin$result`. The resulting data.table has four columns (as presented below):

- `x` – reference dataset (i.e., `foreigners_1`) – this may not contain all units of `foreigners_1`;
- `y` – query (each row of `foreigners_2`) – this will contain all units of `foreigners_2`;
- `block` – the block identifier;
- `dist` – distance between pairs.

```
head(result_reclin$result)
```

```
#>      x      y block      dist
#>   <int> <int> <num>   <num>
#> 1:     3     1     1 0.2216882
#> 2:     3     2     1 0.2122737
#> 3:    21     3     2 0.1172652
#> 4:    57     4     3 0.1863238
#> 5:    57     5     3 0.1379310
#> 6:    61     6     4 0.2307692
```

Let's examine the first block. Clearly, there are typos in the `fname` and `surname`. Nevertheless, all records refer to the same entity (as denoted by `true_id`).

```
rbind(foreigners_1[3, 1:7], foreigners_2[1:2, 1:7])
#>   fname  surname  date region country true_id
#>   <char> <char>   <char> <char> <char>  <num>
#> 1:  amio      maharrsmov 1939/03/08      031      2
#> 2:  amik      maharramof 1939/03/08      031      2
#> 3:  amil      maharramov 1993/03/08      031      2
```

Now we use the `true_id` column to evaluate our approach.


```

matches <- merge(x = foreigners_1[, .(x, true_id)],
                 y = foreigners_2[, .(y, true_id)],
                 by = "true_id")
matches[, block := rleid(x)]
head(matches)

```

```

#> Key: <true_id>
#>   true_id     x     y block
#>   <num> <int> <int> <int>
#> 1:      2      3      1     1
#> 2:      2      3      2     1
#> 3:     20     21      3     2
#> 4:     56     57      4     3
#> 5:     56     57      5     3
#> 6:     60     61      6     4

```

We have 10,000 matched pairs, which can be used in the `true_blocks` argument of the `blocking()` function to specify the true block assignments. This data frame must contain three columns: `x`, `y`, and `block`.

We obtain quality metrics for the assessment of record linkage.

```

res_reclin <- blocking(x = foreigners_1$txt,
                      y = foreigners_2$txt,
                      true_blocks = matches[, .(x, y, block)])

```

```
res_reclin
```

```

#> =====
#> Blocking based on the nnd method.
#> Number of blocks: 6470.
#> Number of columns used for blocking: 1232.
#> Reduction ratio: 0.9999.
#> =====
#> Distribution of the size of the blocks:
#>    2    3    4    5    6    7
#> 3920 1599  928   19    2    2
#> =====
#> Evaluation metrics (standard, in %):
#>   recall  precision      fpr      fnr  accuracy specificity
#>   96.7532   78.6700    0.0038   3.2468   99.9957   99.9962
#>   f1_score
#>   86.7795

```

For example, our approach results in a 3.25% FNR. To improve this, we can increase the `epsilon` parameter of the NND method from 0.1 to 0.5. To do so, we configure the `control_ann` parameter in the `blocking()` function using the `controls_ann()` and `control_nnd()` functions.

```

res_reclin2 <- blocking(x = foreigners_1$txt,
                       y = foreigners_2$txt,
                       true_blocks = matches[, .(x, y, block)],
                       control_ann = controls_ann(nnd = control_nnd(epsilon = 0.5)))

```

```

res_reclin2

#> =====
#> Blocking based on the nnd method.
#> Number of blocks: 6394.
#> Number of columns used for blocking: 1232.
#> Reduction ratio: 0.9999.
#> =====
#> Distribution of the size of the blocks:
#>    2    3    4    5    7
#> 3800 1615  954   21    4
#> =====
#> Evaluation metrics (standard, in %):
#>      recall  precision      fpr      fnr  accuracy specificity
#>      96.8776   80.0500   0.0036   3.1224   99.9960   99.9964
#>      f1_score
#>      87.6636

```

That decreases the FNR to 3.12%.

Now, to use the result in the record linkage process, we add this information to both datasets and specify it in the appropriate argument of a given function. Below, we present an example using the `reclin2` package, employing the Expectation-Maximization algorithm by [Fellegi and Sunter \(1969\)](#).

```

foreigners_1[res_reclin2$result, on = "x", block:= i.block]
foreigners_2[res_reclin2$result, on = "y", block:= i.block]

pairs <- pair_blocking(x = foreigners_1,
  y = foreigners_2, on = "block") |>
  compare_pairs(c("fname", "surname", "date"))
model <- problink_em(~ fname + surname + date, data = pairs)
predict(model, pairs = pairs, add = TRUE, type = "all") |>
  head(n = 4)

#> First data set: 100 000 records
#> Second data set: 10 000 records
#> Total number of pairs: 4 pairs
#> Blocking on: 'block'
#>
#>      .x      .y  fname surname  date      mprob      uprob      mpost      upost
#>    <int> <int> <lgcl> <lgcl> <lgcl>      <num>      <num>      <num>      <num>
#> 1:      3      1 FALSE  FALSE  TRUE 0.24865523 0.217235 0.12223165 0.8777683
#> 2:      3      2 FALSE  FALSE FALSE 0.09215071 0.563555 0.01950491 0.9804951
#> 3:     21      3 FALSE  FALSE  TRUE 0.24865523 0.217235 0.12223165 0.8777683
#> 4:     57      4 FALSE  FALSE FALSE 0.09215071 0.563555 0.01950491 0.9804951
#>      weight
#>      <num>
#> 1: 0.1350875
#> 2: -1.8108395
#> 3: 0.1350875
#> 4: -1.8108395

```

3.2 An example of blocking for deduplication

In this section, we demonstrate a deduplication application using the `blocking()` function on the `RLdata500` dataset from the `RecordLinkage` package. Note that the dataset is included in the `blocking` package. It contains artificial personal data, and fifty records have been duplicated with randomly generated errors. Each row represents one record, with the following columns: `fname_c1` – first name, `fname_c2` – second name, `lname_c1` – last name, `lname_c2` – last name (second component), `by`, `bm`, `bd` – year, month, and day of birth, `rec_id` – record ID, and `ent_id` – entity ID.

```
data("RLdata500")
setDT(RLdata500)
head(RLdata500)

#>   fname_c1 fname_c2 lname_c1 lname_c2   by   bm   bd rec_id ent_id
#>   <char>   <char>   <char>   <char> <int> <int> <int> <int> <int>
#> 1:  CARSTEN          MEIER      1949    7   22     1     34
#> 2:    GERD          BAUER      1968    7   27     2     51
#> 3:  ROBERT      HARTMANN      1930    4   30     3    115
#> 4:  STEFAN          WOLFF      1957    9    2     4    189
#> 5:    RALF      KRUEGER      1966    1   13     5     72
#> 6:  JUERGEN      FRANKE      1929    7    4     6    142
```

For the purpose of this example, we create a new column (`id_count`) that indicates how many times a given unit occurs, and then add leading zeros to the `bm` and `bd` columns. Finally, we create a new string column that concatenates information from all columns (excluding `rec_id`, `ent_id`, and `id_count`), as presented below.

```
RLdata500[, id_count := .N, ent_id]
RLdata500[, txt:=tolower(paste0(fname_c1,fname_c2,lname_c1,lname_c2,by,
                                sprintf("%02d", bm),sprintf("%02d", bd)))]
head(RLdata500[, .(rec_id, id_count, txt)])
```

```
#>   rec_id id_count          txt
#>   <int>   <int>         <char>
#> 1:     1       1 carstenmeier19490722
#> 2:     2       2  gerdbauer19680727
#> 3:     3       1 roberthartmann19300430
#> 4:     4       1 stefanwolff19570902
#> 5:     5       1 ralfkrueger19660113
#> 6:     6       1 juergenfranke19290704
```

As in the previous example, we use the `txt` column in the `blocking()` function. This time, we set `ann = "hsw"` to use the Hierarchical Navigable Small World (HNSW, [Malkov and Yashunin, 2018](#)) algorithm from the [RcppHNSW](#) package.

```
res_dedup <- blocking(x = RLdata500$txt,
                      ann = "hsw",
                      verbose = 1)

#> ===== creating tokens =====
#> ===== starting search (hsw, x, y: 500, 500, t: 429) =====
#> ===== creating graph =====
```

The results are as follows. This time, the HNSW algorithm provided blocks varying from 2 to 17 units.

```
res_dedup

#> =====
#> Blocking based on the hsw method.
#> Number of blocks: 134.
#> Number of columns used for blocking: 429.
#> Reduction ratio: 0.9917.
#> =====
#> Distribution of the size of the blocks:
#>  2  3  4  5  6  7  8  9 10 11 12 17
#> 47 35 23  9  6  5  2  3  1  1  1  1
```

Next, we create a long data table with information on blocks and units from the original dataset. We add the block information to the final dataset. We can check in how many blocks the same entities (`ent_id`) are observed. In our example, all identical entities are in the same blocks.

```
# Reshape to long format
df_block_melted <- melt(res_dedup$result, id.vars = c("block", "dist"))
# Unique record-block pairs
df_block_melted_rec_block <- unique(df_block_melted[, .(rec_id=value, block)])
# Join block IDs
RLdata500[df_block_melted_rec_block, on = "rec_id", block_id := i.block]
# Check block consistency per entity
RLdata500[, .(uniq_blocks = uniqueN(block_id)), .(ent_id)][, .N, uniq_blocks]

#>      uniq_blocks      N
#>      <int> <int>
#> 1:           1    450
```

Additionally, Figure 2 visualises the results based on whether a block contains matches or not. The distribution of distance for matches is bimodal. There is a group of units that are true matches where the distance between them is small (less than 0.2), while for the second group, the distance is similar to true non-matches (between 0.4 and 0.6). This distance may be used as additional information for deduplication (and record linkage) studies.

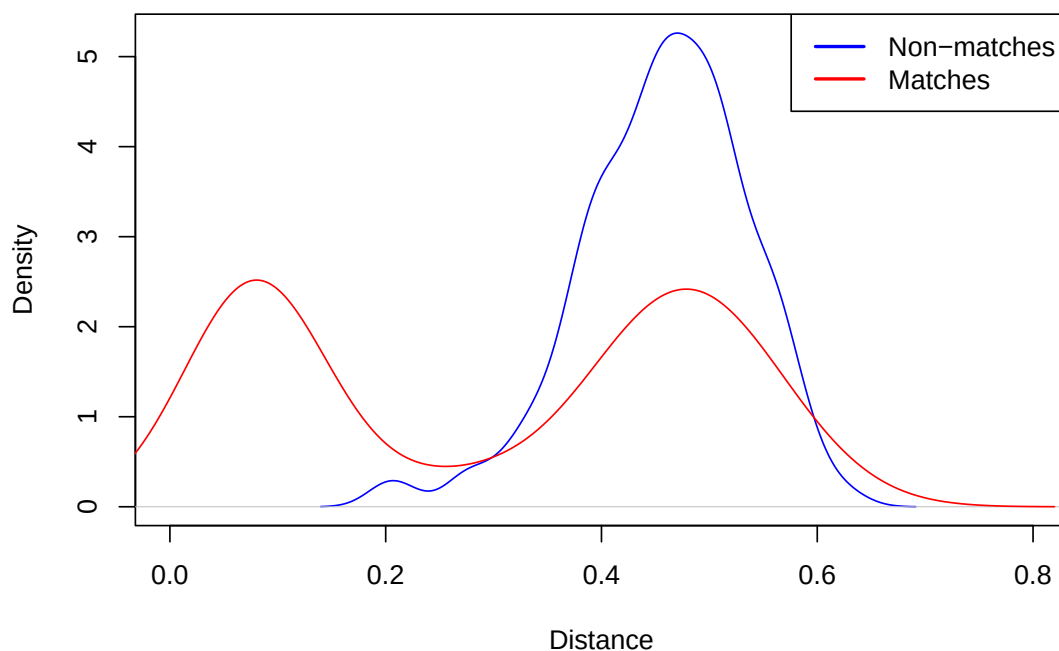


Figure 2: Distribution of distances between true matches and non-matches within blocks

Finally, we compare the evaluation metrics across all ANN algorithms supported by the `blocking()` function, i.e., NND, HNSW, Annoy (from the [RcppAnnoy](#) package), Locality-Sensitive Hashing (LSH, from the [mlpack](#) package), and k -Nearest Neighbours (kNN – denoted as “kd”, from the [mlpack](#) package). We use the `rec_id` and `ent_id` columns from the `RLdata500` dataset to specify the true blocks and then calculate evaluation metrics for all algorithms.

We compare our package with the `klsh()` function from the [klsh](#) package, configured to create 10 blocks (denoted as `klsh_10`) and 100 blocks (denoted as `klsh_100`), respectively. In both settings, we use 20 random projections and 2-character shingles. The results are presented in Table 2, and the code generating it (which also outputs confusion matrices) is included in the Supplementary materials.

The results of this small-scale comparison are broadly consistent with the existing literature on ANN-based blocking. The tree-based methods (NND, HNSW, Annoy, and kNN) achieve 0% FNR while maintaining FPR below 1%, indicating tight blocks that miss no true matches. Their precision and F1 scores (around 5–10%) are expected, as blocking is not intended to provide the final linkage.

In contrast, the LSH-based methods show more variable behaviour: the [mlpack](#) LSH implementation achieves 2% FNR with a higher FPR (3.74%), while the [klsh](#) results illustrate a trade-off between block granularity and recall: `klsh_10` with only 10 blocks yields 16% FNR and 10.13% FPR, whereas `klsh_100` with 100 blocks improves to 10% FNR and 0.94% FPR.

We note that this comparison is illustrative rather than exhaustive; the dataset is small and results may differ on larger or noisier data. Nevertheless, the pattern that modern ANN algorithms such

Table 2: Comparison of various approximate nearest neighbour algorithms implemented in the **blocking** and the **klsh** packages for creation of blocks for deduplication (values in percentages)

	precision	fpr	fnr	accuracy	f1_score
nnd	5.14	0.74	0	99.26	9.78
hnsu	4.80	0.79	0	99.21	9.17
annoy	4.80	0.79	0	99.21	9.17
lsh	1.04	3.74	2	96.26	2.06
kd	5.19	0.73	0	99.27	9.87
klsh_10	0.33	10.13	16	89.87	0.66
klsh_100	3.72	0.94	10	99.06	7.14

as NND, HNSW, and Annoy produce high-recall, low-FPR blocks with minimal configuration is consistent with findings reported elsewhere in the blocking literature (cf. Papadakis et al., 2020).

4 Summary

In this paper, we have demonstrated the basic use cases of the **blocking** package. We believe that the software will be useful for researchers working in various fields where integration of multiple sources is an important aspect. This is certainly of interest in the field of official statistics, where register-based statistics rely on high-quality linkage of administrative datasets, or medical studies, where assessment of health statistics relies on correct linkage of medical history with treatment outcomes or mortality records.

The package does not currently support Privacy-Preserving Record Linkage; however, the flexibility of the representation argument (which accepts user-supplied matrices) provides a natural entry point for integrating encodings such as Bloom filters. Developing and evaluating dedicated PPRL workflows is a direction for future work.

Finally, to extend the package’s ability to block large datasets, we could allow users to use DuckDB (Raasveldt and Mühleisen, 2019) via the **duckdb** package (Mühleisen and Raasveldt, 2026). DuckDB smoothly handles large datasets and allows for extensions that implement ANN algorithms (see core and community extensions at <https://duckdb.org/>).

5 Acknowledgements

We thank the three reviewers for their comments. We also thank the participants of the uRos 2024 conference for valuable comments and discussion. Work on this package was supported by the National Science Centre, OPUS 20 grant no. 2020/39/B/HS4/00941.

We have also developed a Python version of the package, **BlockingPy**, which is available through PyPI. It has a similar structure but offers more ANN algorithms (e.g., FAISS) and enables the use of embeddings. For more details, see: Strojny and Beręsewicz (2026).

References

- M. Beręsewicz. Estimation of the number of foreigners in Poland [Szacunek liczby cudzoziemców w Polsce]. Presentation at: Resident, But Who? – Measuring the Unmeasurable [Mieszkaniec, czyli kto? – zmierzyć niemierzalne], March 2025. URL https://dsp.stat.gov.pl/dsp2025/slides/sesja_3_problemy_pomiaru/2_beresewicz.m.pdf. March 24-25, 2025. [p162]
- O. Binette and R. C. Steorts. (almost) all of entity resolution. *Science Advances*, 8(12):eabi8021, 2022. doi: 10.1126/sciadv.abi8021. URL <https://www.science.org/doi/abs/10.1126/sciadv.abi8021>. [p162]
- P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(9):1537–1555, 2012a. doi: 10.1109/TKDE.2011.127. [p162]
- P. Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer, 2012b. URL <https://link.springer.com/book/10.1007/978-3-642-31164-2>. [p162]

- P. Christen and K. Goiser. Quality and complexity measures for data linkage and deduplication. In F. Guillet and H. Hamilton, editors, *Quality Measures in Data Mining*, pages 127–151. Springer, 2007. doi: 10.1007/978-3-540-44918-8_6. [p165]
- G. Csárdi and T. Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*:1695, 2006. URL <https://igraph.org>. [p163]
- G. Csárdi, T. Nepusz, V. Traag, S. Horvát, F. Zanini, D. Noom, and K. Müller. *igraph: Network Analysis and Visualization in R*, 2025. URL <https://CRAN.R-project.org/package=igraph>. R package version 2.1.4. [p163]
- R. R. Curtin, M. Edel, O. Shrit, S. Agrawal, S. Basak, J. J. Balamuta, R. Birmingham, K. Dutt, D. Eddelbuettel, R. Garg, S. Jaiswal, A. Kaushik, S. Kim, A. Mukherjee, N. G. Sai, N. Sharma, Y. S. Parihar, R. Swain, and C. Sanderson. mlpack 4: a fast, header-only C++ machine learning library. *Journal of Open Source Software*, 8(82), 2023. doi: 10.21105/joss.05026. [p163]
- A. Dasylyva and A. Goussanou. Estimating the false negatives due to blocking in record linkage. *Survey Methodology*, 47(2):299–312, 2021. URL <https://www150.statcan.gc.ca/n1/pub/12-001-x/2021002/article/00002-eng.pdf>. [p162]
- A. Dasylyva and A. Goussanou. On the consistent estimation of linkage errors without training data. *Japanese Journal of Statistics and Data Science*, 5(1):181–216, 2022. doi: 10.1007/s42081-022-00153-3. [p162]
- W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web, WWW '11*, page 577–586, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306324. doi: 10.1145/1963405.1963487. URL <https://doi.org/10.1145/1963405.1963487>. [p167]
- D. Eddelbuettel. *RcppAnnoy: 'Rcpp' Bindings for 'Annoy', a Library for Approximate Nearest Neighbors*, 2024. URL <https://CRAN.R-project.org/package=RcppAnnoy>. R package version 0.0.22. [p163]
- T. Enamorado, B. Fifield, and K. Imai. Using a probabilistic model to assist merging of large-scale administrative records. *American Political Science Review*, 113(2):353–371, 2019. doi: 10.1017/S0003055418000783. [p163]
- T. Enamorado, B. Fifield, and K. Imai. *fastLink: Fast Probabilistic Record Linkage with Missing Data*, 2023. URL <https://CRAN.R-project.org/package=fastLink>. R package version 0.6.1. [p163]
- I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969. doi: 10.1080/01621459.1969.10501049. [p162, 169]
- D. J. Hand and P. Christen. A note on using the F-measure for evaluating record linkage algorithms. *Statistics and Computing*, 28(3):539–547, 2018. doi: 10.1007/s11222-017-9746-6. [p165]
- D. J. Hand, P. Christen, and N. Kirielle. F*: an interpretable transformation of the F-measure. *Machine Learning*, 110(3):451–456, 2021. doi: 10.1007/s10994-021-05964-1. [p165]
- J. P. Howard II. Phonetic spelling algorithm implementations for R. *Journal of Statistical Software*, 95(8): 1–21, 2020. doi: 10.18637/jss.v095.i08. URL <https://www.jstatsoft.org/index.php/jss/article/view/v095i08>. [p162]
- J. P. Howard II. *phonics: Phonetic Spelling Algorithms in R*, 2021. URL <https://jameshoward.us/phonics-in-r/>. R package version 1.3.10. [p163]
- P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 604–613, 1998. doi: 10.1145/276698.276876. [p163]
- J. Johndrow, K. Lum, and D. Dunson. Theoretical limits of microclustering for record linkage. *Biometrika*, 105(2):431–446, 2018. doi: 10.1093/biomet/asv003. [p163]
- T. Kalinowski and D. Falbel. *ragnar: Retrieval-Augmented Generation (RAG) Workflows*, 2025. URL <https://CRAN.R-project.org/package=ragnar>. R package version 0.2.0. [p163]
- D. E. Knuth. *The Art of Computer Programming, Vol. 3: Sorting and Searching*. Addison-Wesley, 1973. URL <https://dl.acm.org/doi/10.5555/280635>. [p162]

- Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018. doi: 10.1109/tpami.2018.2889473. [p170]
- J. Melville. *RcppHNSW: ‘Rcpp’ Bindings for ‘hnsplib’, a Library for Approximate Nearest Neighbors*, 2024a. URL <https://CRAN.R-project.org/package=RcppHNSW>. R package version 0.6.0. [p163]
- J. Melville. *rnndescent: Nearest Neighbor Descent Method for Approximate Nearest Neighbors*, 2024b. URL <https://CRAN.R-project.org/package=rnndescent>. R package version 0.1.6. [p163]
- T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. URL <https://arxiv.org/abs/1301.3781>. [p162]
- L. A. Mullen, K. Benoit, O. Keyes, D. Selivanov, and J. Arnold. Fast, consistent tokenization of natural language text. *Journal of Open Source Software*, 3(23):655, 2018. doi: 10.21105/joss.00655. URL <https://doi.org/10.21105/joss.00655>. [p163]
- H. Mühleisen and M. Raasveldt. *duckdb: DBI Package for the DuckDB Database Management System*, 2026. URL <https://CRAN.R-project.org/package=duckdb>. R package version 1.4.4. [p172]
- G. Papadakis, G. Papastefanatos, T. Palpanas, and M. Koubarakis. Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. *Proceedings of the VLDB Endowment*, 9(4):312–323, 2016. doi: 10.14778/2856318.2856326. [p166]
- G. Papadakis, D. Skoutas, E. Thanos, and T. Palpanas. Blocking and filtering techniques for entity resolution: A survey. *ACM Comput. Surv.*, 53(2), Mar. 2020. ISSN 0360-0300. doi: 10.1145/3377455. URL <https://doi.org/10.1145/3377455>. [p162, 172]
- M. Raasveldt and H. Mühleisen. DuckDB: An embeddable analytical database. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD ’19*, New York, NY, USA, 2019. ACM. doi: 10.1145/3299869.3320212. [p172]
- M. Sariyar and A. Borg. The RecordLinkage package: Detecting errors in data. *The R Journal*, 2(2): 61–67, 2010. doi: 10.32614/RJ-2010-017. URL <https://doi.org/10.32614/RJ-2010-017>. [p163]
- M. Sariyar and A. Borg. *RecordLinkage: Record Linkage Functions for Linking and Deduplicating Data Sets*, 2025. URL <https://CRAN.R-project.org/package=RecordLinkage>. R package version 0.4-12.5. [p163]
- R. Schnell and D. Rukasz. *PPRL: Privacy Preserving Record Linkage*, 2025. URL <https://CRAN.R-project.org/package=PPRL>. R package version 0.3.9. [p163]
- D. Selivanov, M. Bickel, and Q. Wang. *text2vec: Modern Text Mining Framework for R*, 2023. URL <https://CRAN.R-project.org/package=text2vec>. R package version 0.6.4. [p163]
- Y. Singh Parihar, R. Curtin, D. Eddebuettel, and J. Balamuta. *mlpack: ‘Rcpp’ Integration for the ‘mlpack’ Library*, 2025. URL <https://CRAN.R-project.org/package=mlpack>. R package version 4.6.2. [p163]
- R. Steorts. *klsh: Blocking for Record Linkage*, 2020. URL <https://CRAN.R-project.org/package=klsh>. R package version 0.1.0. [p163]
- R. C. Steorts, S. L. Ventura, M. Sadinle, and S. E. Fienberg. A comparison of blocking methods for record linkage. In J. Domingo-Ferrer, editor, *Privacy in Statistical Databases*, pages 253–268, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11257-2. doi: 10.1007/978-3-319-11257-2_20. [p162]
- T. Strojny and M. Beresewicz. BlockingPy: approximate nearest neighbours for blocking of records for entity resolution. *SoftwareX*, 34:102583, 2026. doi: 10.1016/j.softx.2026.102583. URL <https://doi.org/10.1016/j.softx.2026.102583>. [p172]
- J. van der Laan. reclin2: a toolkit for record linkage and deduplication. *The R Journal*, 14(2), 2022. URL <https://journal.r-project.org/articles/RJ-2022-038/>. [p163]
- J. van der Laan. *reclin2: Record Linkage Toolkit*, 2024. URL <https://CRAN.R-project.org/package=reclin2>. R package version 0.5.0. [p163]
- M. van der Loo. The stringdist package for approximate string matching. *The R Journal*, 6(1):111–122, 2014. URL <https://CRAN.R-project.org/package=stringdist>. [p163]

M. Wang, X. Xu, Q. Yue, and Y. Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proceedings of the VLDB Endowment*, 14(11): 1964–1978, 2021. doi: 10.14778/3476249.3476255. [p163]

Maciej Beręsewicz

Department of Statistics, Poznań University of Economics and Business (Poland)

Centre for the Methodology of Population Studies, Statistical Office in Poznań (Poland)

Al. Niepodległości 10

Poznań 61-875, Poland

<https://maciejberesewicz.com>

ORCID: 0000-0002-8281-4301

maciej.beresewicz@ue.poznan.pl

Adam Struzik

Faculty of Mathematics and Computer Science, Adam Mickiewicz University in Poznań (Poland)

Centre for Urban Statistics, Statistical Office in Poznań (Poland)

Ul. Uniwersytetu Poznańskiego 4

Poznań 61-614, Poland

ORCID: 0009-0002-2547-482X

adastr5@st.amu.edu.pl

NMAforest: An R Package for Generating Forest Plots in Network Meta-Analysis with Decomposition of Direct, Indirect, and Study-Level Evidence Contributions

by Yanqi Zhang, Chong Wang, Annette O'Connor

Abstract We present **NMAforest**, an R package designed to visualize results from network meta-analysis (NMA) by producing forest plots that explicitly decompose and visualize the contributions of direct and indirect evidence to treatment effect sizes. While existing packages provide tools to summarize treatment effect estimates and confidence intervals in NMA, they typically do not attribute how much each study, comparison, or path contributes to these results. **NMAforest** addresses this limitation by integrating effect estimation summaries with the treatment network's structural information and the Hat matrix, enabling a transparent decomposition of evidence flows. The package computes effect estimates for direct comparisons, indirect routes through specific network paths, and the overall NMA estimate, accompanied by 95% confidence intervals. The forest plot enables visualization of the relative contribution of each evidence component using proportion bars aligned with the estimates, facilitating a clearer interpretation of the strength and source of evidence.

1 Introduction

Network meta-analysis (NMA) extends pairwise meta-analysis by allowing simultaneous comparisons among multiple treatments, even in the absence of head-to-head comparisons. This framework has become essential in comparative effectiveness research across disciplines such as clinical medicine, epidemiology, and public health, where evidence networks are often sparse or partially connected.

While NMA enables indirect estimations and an approach to combining direct and indirect evidence, it also introduces substantial complexity in interpreting how treatment effect estimates are derived. Specifically, each estimate typically combines direct evidence from studies that directly compared the treatments and indirect evidence accumulated via intermediate comparisons across the network. Understanding this composition is crucial for evaluating the robustness of conclusions, detecting influential studies or comparisons, and assessing the consistency between direct and indirect evidence.

One approach to quantifying evidence contribution is the projection matrix (commonly referred to as the Hat matrix), which arises from the linear model formulation of NMA introduced by Lu et al. (2011). This approach formalizes the combination of evidence and enables estimation of both treatment effects and their associated uncertainty. Building upon this, König et al. (2013) demonstrated how the Hat matrix can be interpreted in terms of evidence flow, assigning proportions of each treatment effect estimate to different direct comparisons. More recently, Papakonstantinou et al. (2018) extended this interpretation by introducing the concept of evidence streams: directed paths through the treatment network that quantify how evidence propagates indirectly between treatments. Their algorithms use flow decomposition to allocate contributions from direct comparisons across all paths in a way that is consistent with the underlying model assumptions.

Despite this theoretical progress, practical tools for decomposing and visualizing evidence contributions remain limited. For example, the popular **netmeta** R package (Rücker et al., 2023) implements frequentist NMA and computes Hat matrices but does not provide built-in methods to enumerate evidence paths or produce contribution-aware forest plots. General-purpose visualization tools, such as the R package **forestplot** (Gordon, 2023), are designed only for plotting and do not perform statistical calculations or differentiate between sources or relative contributions of evidence. This lack of transparency can limit interpretability and hinder effective communication of NMA results to stakeholders.

To address these challenges, we introduce the **NMAforest** package, which integrates statistical estimates, evidence flow decomposition, and contribution visualization in a unified workflow. By combining network structure and Hat matrix-based proportions with standard forest plot representations, **NMAforest** enables researchers to generate richly annotated plots that clearly delineate the origins of each estimate and the influence of each study and path within the network. The **NMAforest** package is available from CRAN at <https://CRAN.R-project.org/package=NMAforest>.

2 Motivation

The motivation for developing **NMAforest** stems from the practical challenges faced by researchers and decision-makers who need to interpret and communicate NMA results. Pairwise meta-analysis and corresponding forest plots are effective for presenting treatment effect estimates and their uncertainty. Pairwise meta-analysis and forest plots also provide the weight that individual studies provide to treatment effect estimates. However, NMA is limited in the detail it conveys about where the evidence comes from. In practice, this often leads to questions such as: How much of the estimated effect relies on direct comparisons? Which indirect paths contribute the most information? Could the result be dominated by a small subset of studies?

These questions are particularly important in applications where evidence transparency and reproducibility are critical. For example, health technology assessments, guideline development, and policy decisions often require a clear audit trail showing how each piece of evidence informs the final estimate. While the Hat matrix and evidence stream decomposition methods provide a rigorous statistical basis for such attribution, extracting and interpreting these contributions typically requires multiple manual steps and specialized code.

NMAforest was designed to fill this gap by offering an integrated set of tools that connect statistical estimation with intuitive visualization. The package builds directly upon the outputs of **netmeta**, leveraging its estimates and Hat matrix computations to break down treatment comparisons into three main components: direct estimates derived from head-to-head studies, indirect estimates based on evidence propagating through specific paths in the network, and the overall NMA estimate that combines all available evidence.

Beyond estimating and displaying these components, **NMAforest** provides a structured approach to display them graphically. In each forest plot, the effect size estimates for the direct, indirect, and NMA components are aligned with horizontal bars that represent the relative contribution of each evidence path. This combination of quantitative and visual decomposition improves transparency, facilitates critical appraisal, and supports sensitivity analyses by making the influence of each evidence path explicit.

In this way, the package aims to make advanced evidence attribution methods accessible to a broader audience and to support a more informed interpretation of complex NMA results.

3 Functionality

The main function provided by the package is:

- **NMAforest()**: Creates a forest plot that decomposes evidence contributions for a specified treatment comparison. The function also returns a structured statistical output containing study labels, treatment effect estimates for direct, indirect, and NMA comparisons, together with their standard errors, confidence intervals, and relative weights. Importantly, the output also includes proportion values that attribute contributions across overall estimates, paths, and individual studies.

This function supports the following arguments:

data - A data frame in long format containing the input data. Required columns include study (study label), t (treatment), and appropriate outcome data:

- For binary outcomes: r (events) and N (sample size)
- For continuous outcomes: y (mean), sd (standard deviation), and N (sample size)

sm - Summary measure for the treatment effect. Supported values include "OR" (odds ratio), "RR" (risk ratio), "MD" (mean difference), and "SMD" (standardized mean difference), etc.

reference - The reference treatment against which others are compared.

model - Type of NMA model to use. The options are "random" or "fixed".

comparison - A character vector of length two, indicating the pair of treatments to be compared.

study - Column name identifying the study label.

treat - Column name for treatment assignment.

event - Column name for event counts (used for binary outcomes).

N - Column name for total sample size (used for binary outcomes and continuous outcomes).

mean - Column name for mean values (used for continuous outcomes).

- sd** - Column name for standard deviations (used for continuous outcomes).
- study_id** - Column name used to uniquely identify each study arm (default = "study_id"). If the column is not present in the original dataset, the updated data frame will be returned with this column added.
- study_path** - Logical argument. If TRUE, the function decomposes indirect path contributions further into study-level components. If FALSE, only the overall indirect effect and its individual paths are shown, without breaking them down to individual study combinations. For large or highly connected networks, setting study_path = FALSE is recommended to suppress study-level path decomposition and generate a cleaner, more interpretable summary plot.

4 Method

The **NMAforest** package implements a structured approach to quantify and visualize the evidence contributing to network meta-analysis (NMA) estimates. It decomposes each comparison into direct and indirect evidence, identifies study-level contributions, and displays these as aligned effect estimates and proportion bars. The following subsections describe the computations used to generate each component of the plot.

All computations implemented in the **NMAforest** package are based on the standard frequentist network meta-analysis model as implemented in **netmeta**. Consequently, the proposed decomposition and visualization of evidence rely on the assumptions of **transitivity** and **consistency** (Ahn and Kang, 2021). Transitivity assumes that studies comparing different treatment pairs are sufficiently comparable with respect to effect-modifying factors, while consistency assumes agreement between direct and indirect evidence for the same treatment contrast.

4.1 Overall Direct Effect and Study-Level Contributions

The overall direct effect for a given treatment comparison (e.g., x vs. y) is estimated using the `netmeta()` function. This produces an effect estimate $\hat{\theta}_{xy}^D$ with an associated standard error and a 95% confidence interval. The superscript notation is used to distinguish direct effect estimates from indirect ($\hat{\theta}_{xy}^I$) estimates considered elsewhere.

To determine how much each study contributes to the direct estimate, we use the **hat matrix** H , derived from the linear projection of NMA comparisons onto the space of observed direct effects:

$$H = Y(X^\top (V^D)^{-1} X)^{-1} X^\top (V^D)^{-1}$$

where:

- X is a $D \times (T - 1)$ matrix describing observed direct comparisons, where:
 - D is the number of treatment comparisons informed by direct evidence,
 - T is the total number of treatments in the network.
- Y is a $\binom{T}{2} \times (T - 1)$ matrix for all treatment contrasts,
- V^D is a $D \times D$ covariance matrix of direct estimates.

The total contribution (flow) of the xy direct comparison to the overall NMA estimate is captured by the corresponding diagonal entry of the hat matrix, denoted as $|H_{xy}^{xy}|$ (Papakonstantinou et al., 2018).

In a **random-effects model**, between-study variability is captured using the heterogeneity parameter τ^2 (tau-squared). This represents the variance of true treatment effects across studies.

Each study i contributing to the xy direct comparison receives a weight:

$$w_i^{xy} = \frac{1}{(SE_i^{xy})^2 + \tau^2}$$

The proportional contribution of study i to the overall direct estimate for the comparison x vs. y is then (Riley et al., 2018):

$$\text{Proportion}_i^{xy} = \frac{w_i^{xy}}{\sum_{i' \in S_{xy}} w_{i'}^{xy}} \cdot |H_{xy}^{xy}|$$

where:

- S_{xy} is the set of studies with direct data on the xy comparison,
- $\text{Proportion}_{i}^{xy}$ is the contribution of study i to the NMA estimate via the xy direct comparison.

4.2 Indirect Paths and Study-Combination Contributions

Indirect evidence for comparison x vs. y arises via intermediate treatments. All indirect paths (e.g., $x \rightarrow v \rightarrow y$) are identified using `igraph::all_simple_paths()` (Csárdi and Nepusz, 2006; Csárdi et al., 2025). Each path is treated as a sequence of direct comparisons, and the indirect estimate for one certain path is calculated as:

$$\hat{\theta}_{xy}^I = \hat{\theta}_{xv}^D + \hat{\theta}_{vy}^D$$

The variance of this estimate accounts for the variances of the contributing comparisons and their covariance. If any two comparisons along the path originate from the same study, the covariance between them is approximated using a constant correlation approach:

$$\text{Cov}(\hat{\theta}_{xv}^D, \hat{\theta}_{vy}^D) = \rho \cdot \text{SE}_i^{xv} \cdot \text{SE}_j^{vy}$$

where $\rho = 0.5$ by default. If no study includes both comparisons, the covariance term is set to zero.

In multi-arm trials, treatment comparisons within the same study are statistically correlated and require specification of within-study correlations to construct the variance–covariance matrix. Following standard practice in `netmeta`, we adopt a constant-correlation assumption with $\rho = 0.5$ (Lu and Ades, 2006). This assumption affects the estimated variances and covariances of treatment effects and therefore influences the width of confidence intervals, while leaving point estimates unchanged.

The total variance of the indirect estimate is then:

$$\text{Var}(\hat{\theta}_{xy}^I) = \text{Var}(\hat{\theta}_{xv}^D) + \text{Var}(\hat{\theta}_{vy}^D) + 2 \cdot \text{Cov}(\hat{\theta}_{xv}^D, \hat{\theta}_{vy}^D)$$

Each indirect path is assigned a **flow value** Flow_k^{xy} , which quantifies the contribution of that specific indirect path to the overall NMA estimate. These flow values are computed using an adapted `comparisonStreams()` function originally developed by Papakonstantinou et al. (2018). These flow values satisfy the constraint:

$$\sum_{\text{indirect paths } k} \text{Flow}_k^{xy} = 1 - |H_{xy}^{xy}|$$

Within each path, we identify contributing study combination (i, j) , where:

- i is a study contributing to comparison xv ,
- j is a study contributing to comparison vy .

The weights for these studies are defined as:

$$w_i^{xv} = \frac{1}{(\text{SE}_i^{xv})^2 + \tau^2}, \quad w_j^{vy} = \frac{1}{(\text{SE}_j^{vy})^2 + \tau^2}$$

Then, the proportional contribution of study combination (i, j) to the indirect path is:

$$\text{Proportion}_{(i,j)}^{xy} = \left(\frac{w_i^{xv}}{\sum_{j' \in S_{xv}} w_{j'}^{xv}} \right) \cdot \left(\frac{w_j^{vy}}{\sum_{j' \in S_{vy}} w_{j'}^{vy}} \right) \cdot \text{Flow}_k^{xy}$$

where:

- S_{xv} is the set of studies with direct data on the xv comparison,
- S_{vy} is the set of studies with direct data on the vy comparison,
- $\text{Proportion}_{(i,j)}^{xy}$ is the contribution of study combination (i, j) to the NMA estimate via the xy direct comparison.

These combination-level contributions appear as individual rows under each path in the expanded forest plot, along with their corresponding effect estimates and proportion bars.

4.3 Evidence Flow Algorithm

This subsection briefly describes how indirect evidence is represented and partitioned across paths using the adapted `comparisonStreams()` procedure implemented in **NMAforest**. For a target comparison (e.g., x vs. y), the procedure starts from the hat matrix obtained from the **netmeta** model, which quantifies the contribution of each observed direct comparison to the network estimate.

Indirect evidence is represented through paths that connect x to y via one or more intermediate treatments. Each indirect path is defined as an ordered sequence of direct comparisons (for example, $x \rightarrow v \rightarrow y$). The absolute values of the corresponding hat-matrix entries are treated as available contribution along each direct comparison involved in the path.

The algorithm iteratively identifies admissible indirect paths from x to y and assigns each path a contribution determined by the smallest remaining contribution among the direct comparisons forming that path. Once a path-level contribution is assigned, it is removed from the available contribution of all comparisons along the path. This ensures that indirect evidence is allocated across paths without double counting.

The final output is a collection of indirect paths, each associated with a nonnegative contribution weight. These path-level contributions are used by **NMAforest** to display indirect evidence in the forest plot, allowing indirect information to be interpreted in terms of specific treatment sequences.

4.4 Overall NMA Effect and Evidence Attribution

The overall NMA effect $\hat{\theta}_{xy}^{\text{NMA}}$ and its confidence interval are computed using `netmeta()`, combining all direct and indirect evidence for comparison x vs. y .

The contribution bar displayed in the NMA effect row represents a full decomposition of all sources of evidence. It is constructed by summing the proportion contributed by the direct comparison and the proportions from all indirect paths, which ensures that 100% of the evidence contributing to the NMA estimate is accounted for. As a result, users can interpret the overall estimate in terms of clearly attributed contributions from specific direct studies and indirect study combinations, facilitating transparent and interpretable visualizations.

5 Usage

5.1 Required data format

The `NMAforest()` function expects the input data to be in long format, with one row per study arm. The following columns are required, either using the default names listed below or specified explicitly via function arguments:

- `treat`: treatment label for each arm;
- `study`: study identifier or label;
- `n`: sample size for each arm;
- `event`: number of events per arm (for binary outcomes);
- `mean, sd`: mean and standard deviation (for continuous outcomes).

An optional column `study_id` may be provided to uniquely identify studies. If `study_id` is not supplied, `NMAforest()` automatically generates this column and returns the updated data frame as part of the output.

5.2 Output Structure

The `NMAforest()` function returns a structured list that includes both a contribution-annotated forest plot and numerical outputs summarizing treatment effects and evidence contributions. The first element of this list is a `ggplot2` object named `$plot`, which visualizes the decomposition of evidence. The forest plot consists of two aligned panels:

- **Effect Size Panel (Left)**: Displays point estimates and 95% confidence intervals for:
 - The overall direct effect;
 - Individual study-level direct comparisons;

- The overall indirect effect;
- Each indirect path;
- Optional: study combinations within indirect paths;
- The overall NMA effect estimate.
- **Visual weighting:** The size of each point estimate is scaled in proportion to its corresponding proportion value, so that larger contributions appear as larger points.
- **Proportion Panel (Right):** Displays horizontal bars aligned with each effect size row, representing the relative contribution (proportion) of each evidence source:
 - Direct proportions are computed using the Hat matrix (H);
 - Indirect proportions are derived from the `comparisonStreams()` function [Papakonstantinou et al. \(2018\)](#), adapted from the `flow_contribution` package;
 - When `study_path = TRUE`, the function further disaggregates flow across combinations of studies along each path.

In addition to the plot, `NMAforest()` returns the following components as part of the output list:

- `$output`: A data frame containing study label names, treatment effect estimates for direct, indirect, and NMA comparisons, along with standard errors, confidence intervals, relative weights, and proportion values for overall estimates, paths, and studies;
- `$updated_df`: The input data frame with a unique `study_id` column added (if not already present), used for internal processing and traceability.

These outputs allow users not only to visualize the structure and sources of evidence but also to extract, inspect, and reuse the underlying quantitative components in downstream analyses.

5.3 Example Case

Example 1: Binary outcome data

```
data(example_data)
```

```
NMAforest(
  data = example_data,
  sm = "OR",
  reference = "x",
  model = "random",
  comparison = c("x", "y"),
  study = "study",
  treat = "t",
  event = "r",
  N = "n",
  study_id = "id",
  study_path = TRUE
)
```

Example 2: Continuous outcome data

```
data("parkinson", package = "pcnetmeta")
```

```
NMAforest(
  data = parkinson,
  sm = "MD",
  reference = "1",
  model = "random",
  comparison = c("1", "3"),
  study = "s.id",
  treat = "t.id",
  mean = "mean",
  sd = "sd",
  N = "n",
  study_path = TRUE
)
```

5.4 Example Datasets

To demonstrate that **NMAforest** operates consistently across different outcome types, we present two example datasets in arm-level format: one with binary outcomes and one with continuous outcomes. In both cases, each row corresponds to a study arm and provides the quantities required by **NMAforest** to perform network meta-analysis and visualize evidence decomposition.

The first example dataset contains binary outcome data from 13 clinical trials comparing four treatments: x, y, u, and v. Each row records the study label (study), treatment label (t), number of events (r), total sample size (n), and a study identifier (id). These variables correspond directly to the arguments study, treat, event, N, and study_id used in **NMAforest**.

The second example dataset uses the parkinson data from the **pcnetmeta** package and contains continuous outcome summary statistics from 7 clinical trials comparing five treatments labeled 1–5. Each row includes the study identifier (s.id), treatment identifier (t.id), arm-level sample mean (mean), standard deviation (sd), and sample size (n). These correspond to the arguments study, treat, mean, sd, and N required for continuous outcomes.

In both examples, the treatment networks are connected and include multi-arm studies, allowing the target comparisons to be informed by both direct and indirect evidence.

5.5 Binary Data Example Plot and Output

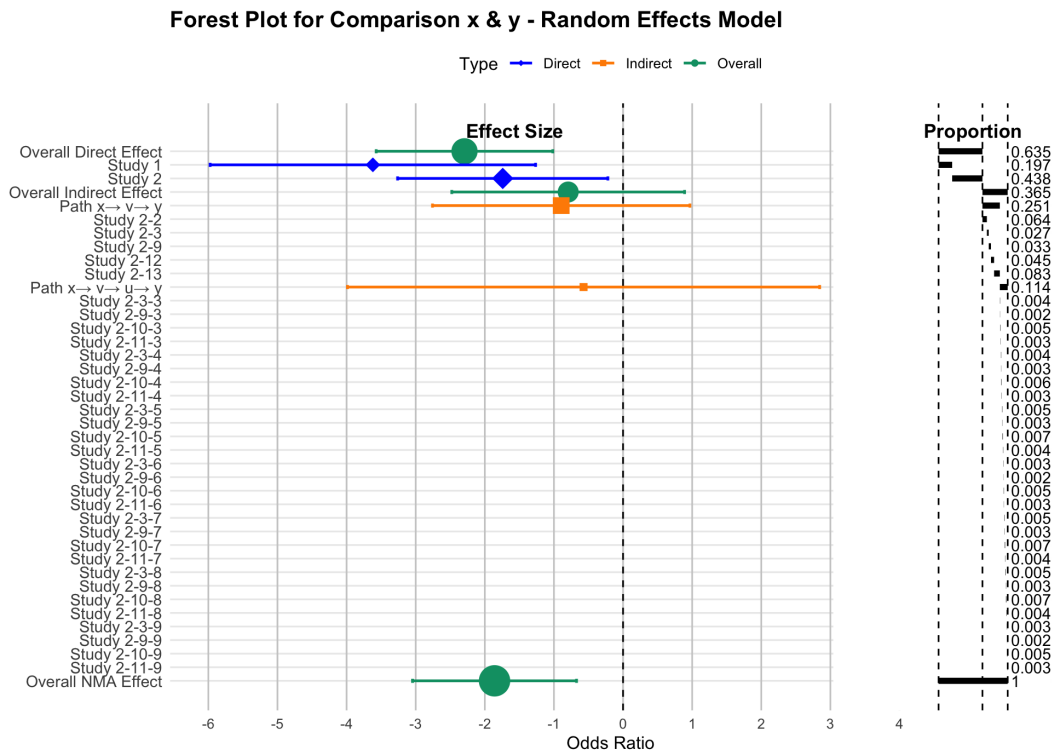


Figure 1: Expanded forest plot for comparison x vs. y, displaying direct, indirect, and overall effects with study-level decomposition of direct and indirect evidence contributions.

\$output		Label	EffectSize	LowerCI	UpperCI	SE	weight	proportion
1	Overall Direct Effect		-2.294	-3.572	-1.017	0.652	2.353	0.635
2	Study 1		-3.62	-5.974	-1.265	1.201	0.693	0.197
3	Study 2		-1.741	-3.262	-0.22	0.776	1.66	0.438
4	Overall Indirect Effect		-0.794	-2.478	0.89	0.859	1.354	0.365
5	Path x→ v→ y		-0.895	-2.757	0.967	0.95	1.108	0.251
6	Study 2-2							0.064
7	Study 2-3							0.027
8	Study 2-9							0.033

9	Study 2-12						0.045
10	Study 2-13						0.083
11	Path x→ v→ u→ y	-0.571	-3.987	2.845	1.743	0.329	0.114
12	Study 2-3-3						0.004
13	Study 2-9-3						0.002
14	Study 2-10-3						0.005
15	Study 2-11-3						0.003
16	Study 2-3-4						0.004
17	Study 2-9-4						0.003
18	Study 2-10-4						0.006
19	Study 2-11-4						0.003
20	Study 2-3-5						0.005
21	Study 2-9-5						0.003
22	Study 2-10-5						0.007
23	Study 2-11-5						0.004
24	Study 2-3-6						0.003
25	Study 2-9-6						0.002
26	Study 2-10-6						0.005
27	Study 2-11-6						0.003
28	Study 2-3-7						0.005
29	Study 2-9-7						0.003
30	Study 2-10-7						0.007
31	Study 2-11-7						0.004
32	Study 2-3-8						0.005
33	Study 2-9-8						0.003
34	Study 2-10-8						0.007
35	Study 2-11-8						0.004
36	Study 2-3-9						0.003
37	Study 2-9-9						0.002
38	Study 2-10-9						0.005
39	Study 2-11-9						0.003
40	Overall NMA Effect	-1.859	-3.046	-0.673	0.605	2.729	1.000

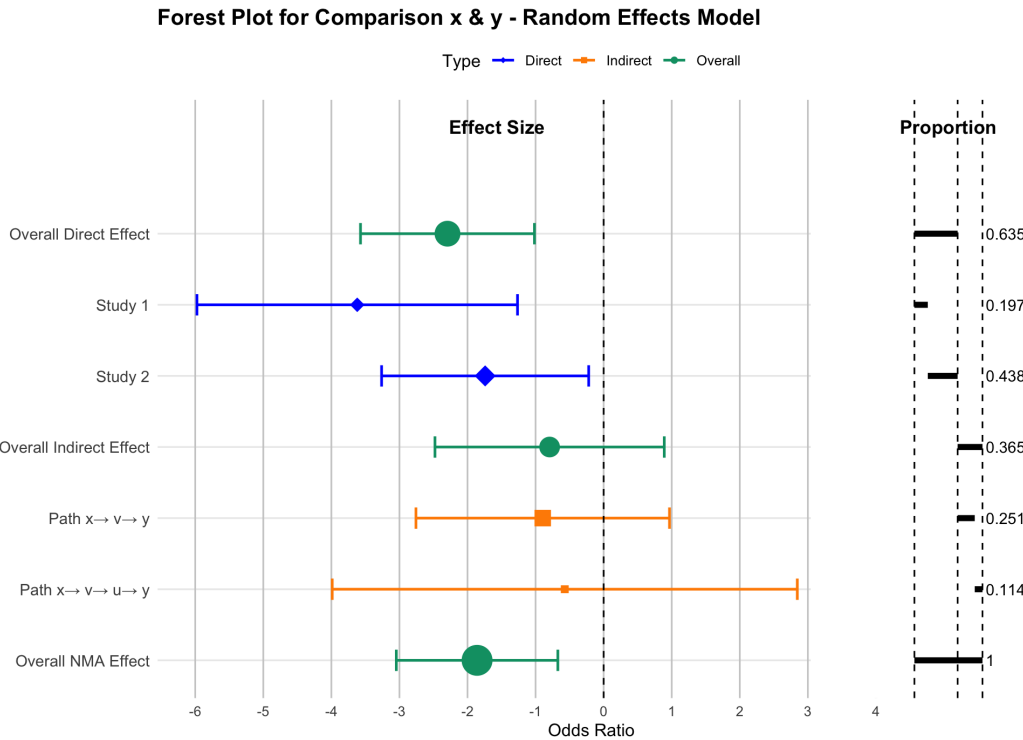


Figure 2: Summary forest plot for comparison x vs. y, displaying direct, indirect, and overall effects without study-combination decomposition. This summary plot corresponds to setting `study_path = FALSE` and is particularly useful for large networks where displaying study-combination-level contributions may result in overly dense visualizations.

\$output		Label	EffectSize	LowerCI	UpperCI	SE	weight	proportion
1	Overall Direct Effect		-2.294	-3.572	-1.017	0.652	2.353	0.635
2	Study 1		-3.620	-5.974	-1.265	1.201	0.693	0.197
3	Study 2		-1.741	-3.262	-0.220	0.776	1.660	0.438
4	Overall Indirect Effect		-0.794	-2.478	0.89	0.859	1.354	0.365
5	Path x→ v→ y		-0.895	-2.757	0.967	0.950	1.108	0.251
6	Path x→ v→ u→ y		-0.571	-3.987	2.845	1.743	0.329	0.114
7	Overall NMA Effect		-1.859	-3.046	-0.673	0.605	2.729	1.000

Note For binary outcomes, the summary measure is specified as the odds ratio (sm = "OR" in [netmeta](#)). However, estimation is performed on the log-odds ratio scale. Accordingly, effect estimates and confidence intervals displayed in the output tables and forest plots are presented on the log scale.

5.6 Continuous Data Example Plot and Output

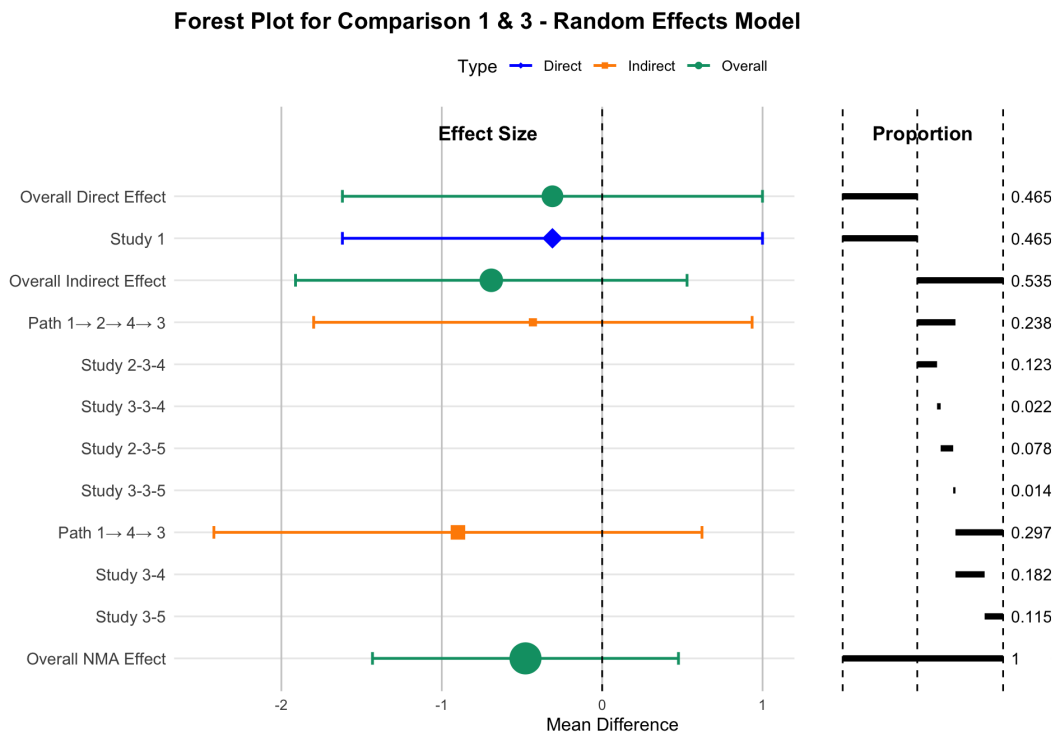


Figure 3: Expanded forest plot for comparison 1 vs. 3, displaying direct, indirect, and overall effects with study-level decomposition of direct and indirect evidence contributions.

\$output		Label	EffectSize	LowerCI	UpperCI	SE	weight	proportion
1	Overall Direct Effect		-0.31	-1.619	0.999	0.668	2.241	0.465
2	Study 1		-0.31	-1.619	0.999	0.668	2.24	0.465
3	Overall Indirect Effect		-0.691	-1.912	0.529	0.623	2.579	0.535
4	Path 1→ 2→ 4→ 3		-0.432	-1.799	0.935	0.697	2.057	0.238
5	Study 2-3-4							0.123
6	Study 3-3-4							0.022
7	Study 2-3-5							0.078
8	Study 3-3-5							0.014
9	Path 1→ 4→ 3		-0.899	-2.421	0.622	0.776	1.66	0.297
10	Study 3-4							0.182
11	Study 3-5							0.115
12	Overall NMA Effect		-0.478	-1.432	0.476	0.487	4.223	1.000

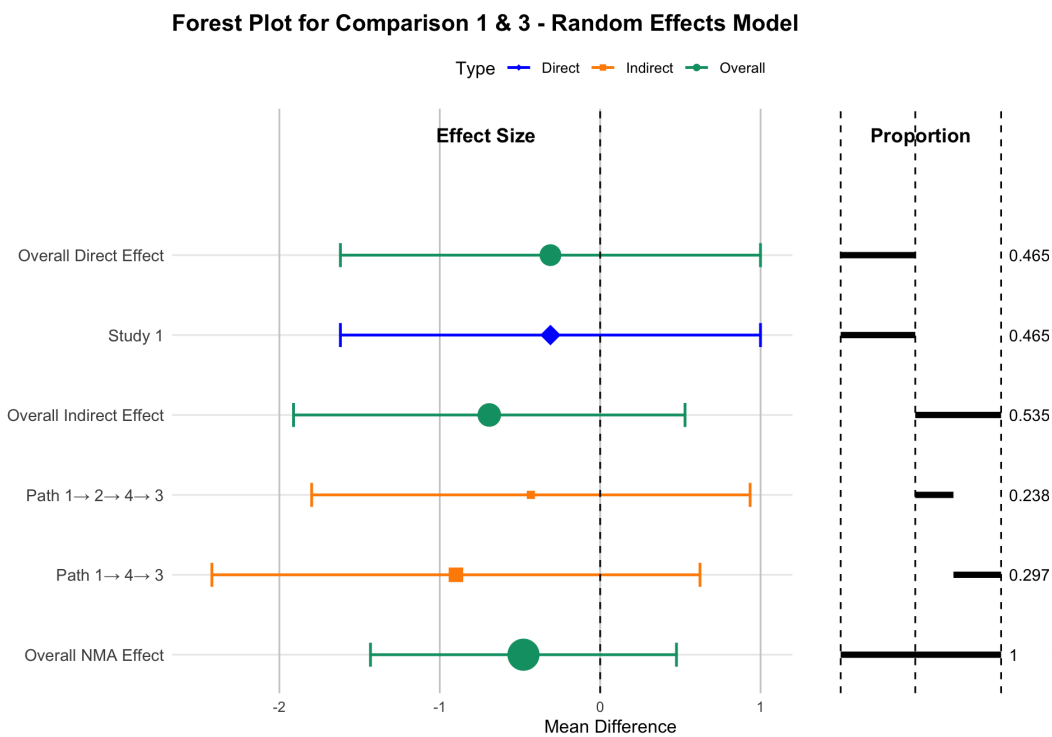


Figure 4: Summary forest plot for comparison 1 vs. 3, displaying direct, indirect, and overall effects without study-combination decomposition. This summary plot corresponds to setting `study_path = FALSE` and is particularly useful for large networks where displaying study-combination-level contributions may result in overly dense visualizations.

\$output							
	Label	EffectSize	LowerCI	UpperCI	SE	weight	proportion
1	Overall Direct Effect	-0.310	-1.619	0.999	0.668	2.241	0.465
2	Study 1	-0.310	-1.619	0.999	0.668	2.240	0.465
3	Overall Indirect Effect	-0.691	-1.912	0.529	0.623	2.579	0.535
4	Path 1+ 2+ 4+ 3	-0.432	-1.799	0.935	0.697	2.057	0.238
5	Path 1+ 4+ 3	-0.899	-2.421	0.622	0.776	1.660	0.297
6	Overall NMA Effect	-0.478	-1.432	0.476	0.487	4.223	1.000

Computational details All analyses and example figures reported in this article were generated using R version 4.4.1. The following main packages were used: [netmeta](#) (version 3.2.0), [igraph](#) (version 2.1.4), and [NMAforest](#) (version 0.1.3).

5.7 Visualizing Decomposition of Evidence

The [NMAforest](#) package provides two styles of forest plots to visualize the composition of treatment effect estimates in a network meta-analysis: an **expanded version** with full study-level breakdown and a **summary version** with aggregate-level detail. Both plots help clarify how direct and indirect evidence combine to form the final NMA effect, and how much each evidence component contributes to the result.

The first plot (Figure 1) illustrates a detailed breakdown for the comparison between treatments *x* and *y*. It displays:

- **Overall direct effect estimate** with 95% confidence interval (CI);
- **Study-level direct effect estimates** with 95% CIs that contribute to the overall direct effect;
- **Overall indirect effect estimate** with 95% CI;
- **Path-specific indirect effect estimates** with 95% CIs (e.g., $x \rightarrow v \rightarrow y$, $x \rightarrow v \rightarrow u \rightarrow y$);
- **Study-combination contribution rows** under each path (proportion-only; they share the path’s effect estimate, so SE/CI are omitted);

- **Right-panel proportion bars** indicating the relative contribution of each study, path, or overall component to the effect estimate in the same row.

This expanded visualization is especially useful for understanding how indirect comparisons are constructed and which studies dominate specific routes of evidence. Within each indirect path, all study combinations inherit the same path-level effect estimate and uncertainty; therefore, individual effect sizes and confidence intervals are not displayed at the study-combination level.

The second plot (Figure 2) is designed for contexts where clarity and brevity are prioritized. It retains the essential decomposition structure—distinguishing between direct, indirect, and overall NMA effect estimates—while omitting study-level details. It includes:

- **Overall direct effect estimate** with 95% CI;
- **Study-level direct effect estimates** with 95% CIs contributing to the overall direct effect;
- **Overall indirect effect estimate** with 95% CI, with each indirect path displayed as a single row, without further decomposition into study-combination rows;
- **Overall NMA effect estimate** with 95% CI;
- **Right-panel proportion bars** aligned with each overall, path-, or study-level estimate, without displaying study-combination contributions.

This summary plot is more compact and is well-suited for presentations where space is limited but key decomposition information still needs to be shown, especially in cases where study-level decomposition would be too dense or overwhelming to display in full. In such cases, users are encouraged to set `study_path = FALSE`, which suppresses study-level path detail and focuses visualization on aggregated indirect paths, improving clarity without sacrificing key decomposition information.

Both visualizations use color-coded point estimates (blue for study-level direct effect, orange for indirect path effect, and green for overall effect) and horizontal proportion bars. This dual-panel format enables clear interpretation of both statistical magnitude and structural evidence contributions.

6 Discussion

The **NMAforest** package provides a structured and transparent framework for decomposing evidence contributions in network meta-analysis. By combining frequentist estimation outputs from **netmeta** with graph-based decomposition of indirect evidence flows, it fills a critical gap left by existing tools, which often summarize effect estimates without revealing their sources. This detailed attribution supports users in assessing the robustness and interpretability of treatment comparisons, especially in networks where indirect evidence constitutes a substantial proportion of the overall information.

While the package offers an additional perspective beyond conventional forest plots, several limitations merit consideration. First, the current implementation relies on a constant correlation coefficient to approximate covariance between indirect path segments, which may oversimplify dependencies, particularly in the presence of multi-arm studies or sparse evidence structures. Second, although the visualization of study combinations within indirect paths provides a high level of granularity, it can result in complex or crowded displays when networks are large. To address this, the package includes an option that allows users to suppress the study-combination detail and display only the aggregated contributions of each indirect path, thereby balancing interpretability and comprehensiveness according to the needs of the analysis.

Future enhancements could further improve flexibility by exploring alternative methods for covariance estimation, incorporating Bayesian models of network meta-analysis, and developing interactive visualization features that allow users to filter, collapse, or highlight specific evidence contributions dynamically.

7 Conclusion

The **NMAforest** package provides a clear and practical approach for linking treatment effect estimates with their contributing studies and indirect evidence paths in network meta-analysis. By offering contribution-aware visualizations alongside standard effect summaries, the package supports more transparent reporting and helps users interpret the origins of their results with greater confidence. We believe this approach will facilitate clearer communication of evidence synthesis findings across research and applied decision-making contexts.

References

- E. Ahn and H. Kang. Concepts and emerging issues of network meta-analysis. *Korean Journal of Anesthesiology*, 74(5):371–382, 2021. doi: 10.4097/kja.21358. [p178]
- G. Csárdi and T. Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695, 2006. URL <https://igraph.org>. [p179]
- G. Csárdi, T. Nepusz, V. Traag, S. Horvát, A. Zanini, D. Noom, and K. Müller. *igraph: Network Analysis and Visualization in R*, 2025. URL <https://CRAN.R-project.org/package=igraph>. R package version 2.1.4. [p179]
- M. Gordon. forestplot: Advanced forest plot using ‘grid’ graphics. <https://CRAN.R-project.org/package=forestplot>, 2023. R package version 2.0.1. [p176]
- J. König, U. Krahn, and H. Binder. Visualizing the flow of evidence in network meta-analysis and characterizing mixed treatment comparisons. *Statistics in Medicine*, 32(30):5414–5429, 2013. doi: 10.1002/sim.6001. URL <https://doi.org/10.1002/sim.6001>. [p176]
- G. Lu and A. E. Ades. Assessing evidence inconsistency in mixed treatment comparisons. *Journal of the American Statistical Association*, 101(474):447–459, 2006. doi: 10.1198/016214505000001302. [p179]
- G. Lu, N. J. Welton, J. P. T. Higgins, I. R. White, and A. E. Ades. Linear inference for mixed treatment comparison meta-analysis: A two-stage approach. *Research Synthesis Methods*, 2(1):43–60, 2011. doi: 10.1002/jrsm.34. URL <https://doi.org/10.1002/jrsm.34>. [p176]
- T. Papakonstantinou, A. Nikolakopoulou, G. Schwarzer, G. Rücker, and G. Salanti. *Estimating the Contribution of Studies in Network Meta-Analysis: Paths, Flows and Streams*. F1000Research, 2018. URL <https://doi.org/10.12688/f1000research.14770.3>. Version 3. [p176, 178, 179, 181]
- R. D. Riley, J. Ensor, D. Jackson, and D. L. Burke. Deriving percentage study weights in multi-parameter meta-analysis models: with application to meta-regression, network meta-analysis and one-stage individual participant data models. *Statistical Methods in Medical Research*, 27(10):2885–2905, 2018. doi: 10.1177/0962280216688033. URL <https://doi.org/10.1177/0962280216688033>. [p178]
- G. Rücker, G. Schwarzer, T. Rücker, and J. Beyersmann. netmeta: Network meta-analysis using frequentist methods. *Journal of Statistical Software*, 106(2):1–40, 2023. doi: 10.18637/jss.v106.i02. URL <https://www.jstatsoft.org/article/view/v106i02>. [p176]

Yanqi Zhang
Department of Statistics
Iowa State University
2438 Osborn Dr Ames, IA 50011-1090
United States
zyq1998@iastate.edu

Chong Wang
Professor, Department of Veterinary Diagnostic and Production Animal Medicine, Department of Statistics
Iowa State University
2239 Lloyd Vet Med Ctr Ames, IA 50011-3619
United States
chwang@iastate.edu

Annette O'Connor
Professor, Department of Large Animal Clinical Sciences
Michigan State University
Veterinary Medicine Center 736 Wilson Road, Room D-202 East Lansing, MI 48824
United States
oonconn445@msu.edu

Kerasnip: A Bridge Between Keras and Tidymodels

by David Díaz

Abstract The `kerasnip` R package bridges the `keras3` R package and the `tidymodels` framework, allowing users to define Keras models as `parsnip` model specifications. This seamless integration enables the use of Keras models within the `tidymodels` ecosystem for preprocessing, tuning, and evaluation. `kerasnip` dynamically generates `parsnip` specifications from user-defined layer blocks, simplifying the process of building and tuning complex Keras models. This enables the definition of non-linear, functional architectures by providing a clear way to map connections between layers. This article introduces the package's architecture and demonstrates its use with a case study.

1 Introduction

Deep learning (Goodfellow et al., 2016) has reshaped applied machine learning, and Keras [Keras3; Chollet (2015)] remains one of the most popular APIs for composing modern neural networks. In R, the `keras3` package faithfully mirrors the Keras experience, while the `tidymodels` framework (Kuhn and Wickham, 2020) has become the de facto standard interface for preprocessing, tuning, and evaluating models. Although `parsnip` (a `tidymodels` component) ships with a basic Keras engine, it only covers a handful of predefined multilayer perceptron templates.

The difficulty emerges when practitioners need to move beyond those templates. Keras models are constructed by freely composing layers, whereas `parsnip` specifications expect a fixed set of hyperparameters that map directly to arguments. This mismatch prevents R users from expressing arbitrary Keras architectures, repeating blocks, or wiring multi-branch graphs inside a `tidymodels` workflow.

`kerasnip` addresses this impedance mismatch by generating `parsnip` specifications on the fly from user-defined Keras blocks. The package surfaces the entire `tidymodels` toolchain—`recipes`, `tune`, `workflows`, and `workflowsets`—for any model that can be described with the Keras sequential or functional APIs. Functional pipelines leverage the `inp_spec()` helper to declare connections explicitly, keeping even complex graphs readable. The package name itself blends `keras` and `parsnip`, underscoring this bridging goal.

The remainder of the paper is structured as follows. Section 2 positions `kerasnip` relative to existing tooling and summarizes its main advantages. Section 3 details the code generation approach that connects Keras blocks with `parsnip`. Section 4 walks through a quick-start example, and Section 5 presents a full case study on the Ames Housing data. The paper concludes by outlining the current limitations and near-term roadmap.

2 Ecosystem context and advantages

2.1 Related Work

The `kerasnip` package builds upon a rich ecosystem of packages for deep learning and machine learning in R. This section provides an overview of the most relevant packages and positions `kerasnip` in relation to them.

reticulate: The `reticulate` package (Ushey et al., 2023) is the foundation for R-Python interoperability. It allows R users to call Python code from R, and it is the package that the `keras3` package uses to interface with the Python Keras library. While `reticulate` is essential for using `keras3` in R, it does not provide any direct integration with the `tidymodels` framework.

keras3: The `keras3` package [Keras3] provides a direct R interface to the Keras library, allowing R users to define and train Keras models using a familiar R syntax. While `parsnip` offers a basic Keras engine, the `keras3` package itself does not provide a native mechanism for integrating custom or dynamically defined Keras architectures into `tidymodels` workflows, which is the specific challenge `kerasnip` addresses. Keras offers a high-level, user-friendly API, widespread adoption, and multi-backend support (TensorFlow (Abadi et al., 2015), JAX (Bradbury et al., 2018), PyTorch (Paszke et al., 2019)), making it a powerful and versatile choice. `kerasnip` thus serves as a valuable bridge for users preferring the Keras ecosystem.

luiz and brulee: The `luiz` (Falbel, 2023) and `brulee` packages provide a `tidymodels`-like interface

for the **torch** deep learning framework. They are excellent tools for R users who want to use **torch** with **tidymodels**. **kerasnip** works by extending the existing **parsnip** keras engine, enabling a similar level of integration and flexibility for models built with the **keras3** R package as **luz** and **brulee** do for **torch** models.

While both **kerasnip** and **luz/brulee** integrate deep learning frameworks with **tidymodels**, they cater to different backends and user preferences. **luz** and **brulee** offer an R-native experience with **torch**, potentially providing lower overhead and direct control. In contrast, **kerasnip** leverages the high-level Keras API, built on optimized backends (TensorFlow, JAX, PyTorch), offering user-friendliness and rapid prototyping for those familiar with Keras. While **luz/brulee** might offer more granular control, **kerasnip** prioritizes abstraction and ease of use. Crucially, **kerasnip** uniquely enables dynamic definition and architectural tuning of Keras models directly within **tidymodels**, a capability not fully offered by other existing solutions.

A key design divergence in **kerasnip** compared to **luz/brulee** is its emphasis on dynamic model generation through user-defined `layer_blocks` and the `inp_spec()` helper. While **luz/brulee** provide a **tidymodels**-like interface for predefined **torch** modules, **kerasnip** empowers users to construct arbitrary Keras architectures on the fly, directly within the **parsnip** specification. This architectural flexibility, particularly the ability to tune the *number* of layers or define complex functional connections via `inp_spec()`, is a deliberate choice to leverage Keras's compositional nature and extend **tidymodels'** tuning capabilities beyond just hyperparameters to the model's very structure. This approach allows for a more comprehensive architectural search within the **tidymodels** framework, which is a unique offering of **kerasnip**.

kerasnip extends the **parsnip** ecosystem not by adding a new engine, but by enabling the creation of dynamic **parsnip** model specifications that use the existing keras engine. This is important to distinguish from **parsnip**'s default Keras integration, such as `parsnip::keras_mlp()`, which offers limited architectural flexibility. While `keras_mlp()` is convenient for predefined MLP architectures, **kerasnip**'s core strength is its ability to dynamically generate specifications from user-defined `layer_blocks`. This allows users to integrate custom and complex model architectures into **tidymodels** workflows and enables comprehensive architectural search and tuning (e.g., varying the number of layers or defining complex functional connections), making it powerful for optimizing the very structure of a Keras model, not just its hyperparameters.

In summary, while several packages provide access to deep learning libraries from R, **kerasnip** is unique in its focus on providing a dedicated, flexible, and **tidymodels**-native solution for **keras3** models. It is designed to be a lightweight and intuitive bridge that allows R users to seamlessly integrate the power of **keras3** into their **tidymodels** workflows, particularly benefiting those already invested in the Keras ecosystem or requiring Keras-specific features not available in other frameworks. Its block-based architecture empowers users to define and tune complex deep learning models with minimal boilerplate, making advanced deep learning techniques more accessible within the familiar **tidymodels** paradigm.

2.2 Key Advantages of **kerasnip**

kerasnip offers several practitioner-oriented advantages that complement the ecosystem context above:

- **Reduced boilerplate and full workflow parity:** dynamically generated **parsnip** specifications eliminate bespoke wrappers. Any generated model can immediately plug into **recipes**, **tune**, **workflows**, or **workflowsets**, so Keras users receive the exact same ergonomics that **tidymodels** users expect for **glmnet**, **xgboost**, or **torch** engines.
- **Architectural tuning, not just hyperparameters:** arguments of the form `num_{block_name}` control how many times a block is repeated, enabling structural search in addition to parameter search. Figure “Conceptual diagram illustrating num block tuning” visualizes how varying `num_dense` alters the network depth, and the accompanying code sample shows how to expose `num_dense` to `tune()`. Repeated blocks intentionally share the same hyperparameters; to tune each hidden layer independently, users can register separate block functions (e.g., `dense_head`, `dense_tail`) and assign distinct tuning parameters to each.
- **Extensibility to advanced architectures:** because layer blocks can wrap any Keras layer, the current release already supports CNNs, RNNs, Transformers, or residual graphs—users simply describe the building blocks they need. Future work will focus on shipping higher-level templates and validation helpers for these architectures, not on unlocking the underlying capability.
- **Enhanced reproducibility and performance:** encapsulating the architecture definition in R functions makes it straightforward to version-control both data preprocessing and network topology, while the heavy computation is still executed by optimized TensorFlow, JAX, or

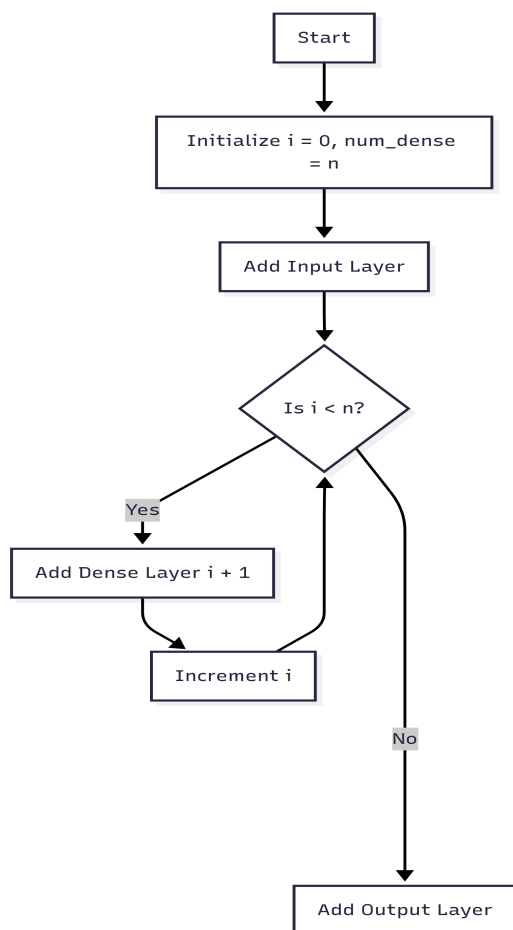


Figure 1: Conceptual diagram illustrating num block tuning

PyTorch backends.

- **Debugging support:** helpers such as `compile_keras_grid()` summarize the dynamically generated graphs, and every fitted specification can expose the underlying Keras model for inspection via `summary()` or `plot()`.

```

# Define a tunable model specification
tune_spec <- tunable_mlp(
  num_dense = tune(),
  dense_units = tune(),
  fit_epochs = 10
) |>
  set_engine("keras")

# Define the tuning grid
params <- extract_parameter_set_dials(tune_spec) |>
  update(
    num_dense = dials::num_terms(c(1, 3)),
    dense_units = dials::hidden_units(c(8, 64))
  )
grid <- grid_regular(params, levels = 3)

# Run the tuning
# ...

```

3 Bridging the gap: The kerasnip architecture

The core of **kerasnip** is a code generation engine that bridges the gap between the flexible, layer-based definition of **keras3** models and the structured, argument-based definition of **parsnip** models. It achieves this by dynamically creating a new **parsnip** model specification from a set of user-defined “layer blocks”.

When a user calls `create_keras_sequential_spec()` to create a Keras model using the sequential API or `create_keras_functional_spec()` to create a model using the functional API, **kerasnip** inspects the functions provided in the `layer_blocks` argument to identify their formal arguments. It then generates a new function that represents the **parsnip** model specification. This new function has a unified set of arguments that can be used to control the architecture and hyperparameters of the **keras3** model.

To illustrate this, let’s consider a simplified example of the code that **kerasnip** generates for a sequential MLP. When you call `create_keras_sequential_spec("my_mlp", ...)` with input, dense, and output layer blocks, **kerasnip** generates a function similar to this:

```
my_mlp <- function(mode = "regression", num_dense = 1, dense_units = 8, ...) {
  args <- list(
    num_dense = num_dense,
    dense_units = dense_units,
    ...
  )
  new_model_spec(
    "my_mlp",
    args = args,
    eng_args = NULL,
    mode = mode,
    engine = "keras"
  )
}
```

This generated function, `my_mlp()`, is a **parsnip** model specification. It has arguments to control the number of dense layers (`num_dense`) and the number of units in each dense layer (`dense_units`). When you call this function, it returns a **parsnip** model specification that can be used in a **tidymodels** workflow.

kerasnip uses a simple naming convention to map the arguments of the generated function to the arguments of the layer blocks. For example, an argument named `dense_units` in the `my_mlp()` function is mapped to the `units` argument of the `dense_block` function. This allows for a flexible and intuitive way to control the architecture of the **keras3** model from the **parsnip** model specification.

For functional API models, **kerasnip** provides the `inp_spec()` helper function to define the connections between layers. You use it to specify which layer outputs (tensors) should be fed into which arguments of the next layer’s function. This explicit mapping is crucial for building non-linear, multi-input/multi-output architectures where the flow of data is not a simple sequential chain.

3.1 Using `inp_spec()`

The `inp_spec()` function takes two main arguments:

1. The layer block function to be called (e.g., `concat_block`).
2. A character vector specifying the inputs. The **names** of the vector must match the argument names of the layer block function (e.g., `a` and `b` in `concat_block`), and each value refers to the upstream layer block whose tensor should be supplied to that argument (e.g., `input_a` and `input_b`).

To illustrate, consider a simple functional model with two inputs that are concatenated and then passed to a dense layer:

```
input_a_block <- function(input_shape) {
  layer_input(shape = input_shape, name = "input_a")
}
input_b_block <- function(input_shape) {
  layer_input(shape = input_shape, name = "input_b")
}
```



```

}
concat_block <- function(a, b) {
  layer_concatenate(list(a, b))
}
output_block <- function(tensor) {
  layer_dense(tensor, units = 1)
}
create_keras_functional_spec(
  "my_functional_model",
  list(
    input_a = input_a_block,
    input_b = input_b_block,
    concatenated = inp_spec(concat_block, c(a = "input_a", b = "input_b")),
    output = inp_spec(output_block, "concatenated")
  )
)

```

In this example:

- `inp_spec(concat_block, c(a = "input_a", b = "input_b"))` specifies that the `concat_block` function should be called.
- The output of the `input_a` block will be passed to the `a` argument of `concat_block`.
- The output of the `input_b` block will be passed to the `b` argument of `concat_block`.
- If the input is a single tensor, you can simply provide the name of the upstream block as a character string, as in `inp_spec(output_block, "concatenated")`.

This explicit way of defining connections allows for the creation of complex, non-linear model architectures.

Clarity on `inp_spec()` for Complex Scenarios

`inp_spec()` is essential for creating complex functional Keras models, enabling intricate data flows like parallel branches or residual connections (e.g., in ResNets). For a residual connection, `inp_spec()` directs tensors from both the main path and a skip connection into a subsequent block, such as one using `layer_add()`. This explicit mapping of outputs to inputs grants fine-grained control, facilitating highly customized and complex network topologies.

Internally, **kerasnip** parses the `inp_spec()` calls to understand the intended data flow and connections between layer blocks. It then leverages `rlang::exec()` and the `!!!` operator for dynamic function execution, enabling the flexible construction of Keras models from user-defined blocks, effectively translating the `inp_spec()` graph into the Keras functional API.

4 A quick start example

To quickly illustrate the core functionality of **kerasnip**, a simple example of a sequential model for classifying the iris dataset is presented.

First, the layer blocks for the model are defined:

```

# Define layer blocks for a simple sequential model
input_block <- function(model, input_shape) {
  keras_model_sequential(input_shape = input_shape)
}
hidden_block <- function(model, units = 32) {
  model |> layer_dense(units = units, activation = "relu")
}
output_block <- function(model, num_classes) {
  model |> layer_dense(units = num_classes, activation = "softmax")
}

```

Next, the **parsnip** model specification is created using `create_keras_sequential_spec()`:

```

# Create the model specification
create_keras_sequential_spec(

```

```

model_name = "my_iris_mlp",
layer_blocks = list(
  input = input_block,
  hidden = hidden_block,
  output = output_block
),
mode = "classification"
)

```

The `my_iris_mlp()` function can now be used as a [parsnip](#) model.

```

# Define the model and fit it
iris_fit <- my_iris_mlp(
  num_hidden = 2,
  hidden_units = 16,
  fit_epochs = 10
) |>
  set_engine("keras") |>
  fit(Species ~ ., data = iris)

```

This simple example demonstrates how [kerasnip](#) can be used to quickly create and train a [keras3](#) model within the [tidymodels](#) framework.

5 Case study: Predicting house prices with a multi-input functional model

To demonstrate the capabilities of [kerasnip](#), the Ames Housing dataset will be used to predict house prices. This case study highlights a complete [tidymodels](#) workflow for a regression task using a [keras3](#) functional model with multiple inputs, where numerical and categorical features are processed through separate input branches.

5.1 Data Preparation

First, the necessary packages are loaded and the data prepared. The Ames Housing dataset from the [modeldata](#) package will be used, relevant columns selected, and the data split into training and testing sets.

```

library(kerasnip)
library(tidymodels)
library(keras3)
library(dplyr) # For data manipulation
library(ggplot2) # For plotting
library(finetune) # For racing
library(workflowsets) # For comparing models

# Select relevant columns and remove rows with missing values
ames_df <- ames |>
  select(
    Sale_Price,
    Gr_Liv_Area,
    Year_Built,
    Neighborhood,
    Bldg_Type,
    Overall_Cond,
    Total_Bsmt_SF,
    contains("SF")
  ) |>
  na.omit()

# Split data into training and testing sets
set.seed(123)

```

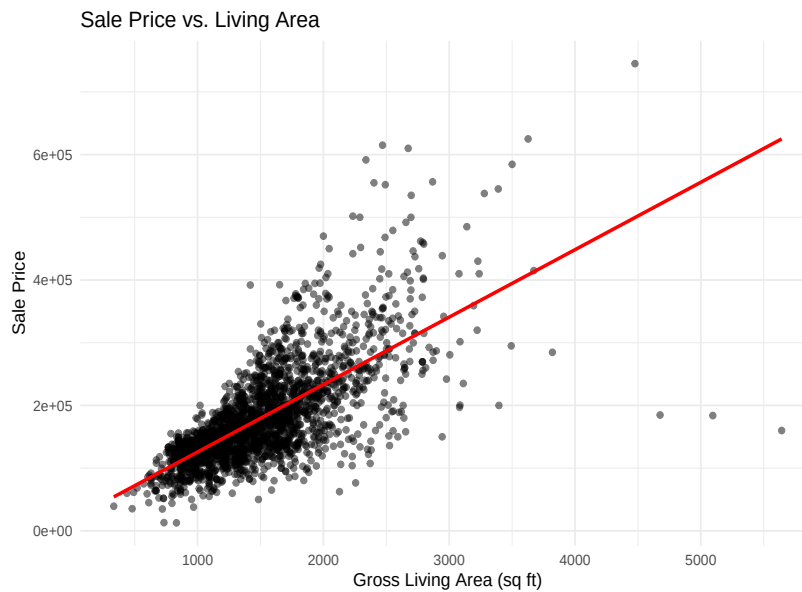


Figure 2: Data Exploration Plot for Ames Housing Dataset

```
ames_split <- initial_split(ames_df, prop = 0.8, strata = Sale_Price)
ames_train <- training(ames_split)
ames_test <- testing(ames_split)
```

```
# Create cross-validation folds for tuning
ames_folds <- vfold_cv(ames_train, v = 5, strata = Sale_Price)
```

To gain initial insights into the dataset, Figure 2 scatters `Sale_Price` against `Gr_Liv_Area`, illustrating the relationship between these key variables.

5.2 Recipe for Preprocessing

A `recipes` object is created to normalize numerical predictors, one-hot encode categorical predictors, and then collapse each group of predictors (e.g., all dummy variables associated with a factor) into a single matrix column via `step_collapse()`. This preparation is crucial for multi-input `keras3` models because every branch in the functional API expects a single tensor as input.

```
ames_recipe <- recipe(Sale_Price ~ ., data = ames_train) |>
  step_normalize(all_numeric_predictors()) |>
  step_collapse(
    all_numeric_predictors(),
    new_col = "numerical_input"
  ) |>
  step_dummy(Neighborhood) |>
  step_collapse(
    starts_with("Neighborhood"),
    new_col = "neighborhood_input"
  ) |>
  step_dummy(Bldg_Type) |>
  step_collapse(
    starts_with("Bldg_Type"),
    new_col = "bldg_input"
  ) |>
  step_dummy(Overall_Cond) |>
  step_collapse(
    starts_with("Overall_Cond"),
    new_col = "condition_input"
  )
```

5.3 Define Keras Functional Model with `kerasnip`

Next, the Keras functional model is defined using `kerasnip`'s layer blocks. This model will have four distinct input layers: one for numerical features and three for categorical features. These branches will be processed separately and then concatenated before the final output layer.

Define input layer blocks:

```
# Define input layer blocks for multi-input functional model.
# Each function represents an input block in the Keras model graph.
input_numerical <- function(input_shape) {
  layer_input(shape = input_shape, name = "numerical_input")
}
input_neighborhood <- function(input_shape) {
  layer_input(shape = input_shape, name = "neighborhood_input")
}
input_bldg <- function(input_shape) {
  layer_input(shape = input_shape, name = "bldg_input")
}
input_condition <- function(input_shape) {
  layer_input(shape = input_shape, name = "condition_input")
}
```

Then, the dense layer blocks that will process the numerical and categorical inputs are defined:

```
# Define dense layer blocks for processing numerical and categorical inputs.
dense_numerical <- function(tensor, units = 32, activation = "relu") {
  tensor |>
    layer_dense(units = units, activation = activation)
}
dense_categorical <- function(tensor, units = 16, activation = "relu") {
  tensor |>
    layer_dense(units = units, activation = activation)
}
```

Finally, the concatenation layer block to combine the processed features and the output layer block for regression are defined:

```
# Define concatenation and output layer blocks.
concatenate_features <- function(numeric, neighborhood, bldg, condition) {
  layer_concatenate(list(numeric, neighborhood, bldg, condition))
}
dense_combined <- function(
  tensor,
  units = 64,
  activation = "relu",
  dropout = 0.2) {
  tensor |>
    layer_dense(units = units, activation = activation) |>
    layer_dropout(rate = dropout)
}
output_regression <- function(tensor) {
  layer_dense(tensor, units = 1, name = "output")
}
```

Note that `concatenate_features()` is merely a thin wrapper around `layer_concatenate()`. The helper collects the four upstream tensors into a list because `layer_concatenate()` expects a single inputs argument; wrapping it keeps the `layer_blocks` definition explicit while still allowing any built-in Keras layer to be used directly when its signature already matches the desired inputs.

With all the individual layer blocks defined, the `kerasnip` functional model specification is created. This specification ties together the input, processing, and output blocks, defining the flow of data through the model:

```
# Create the kerasnip model specification function.
```

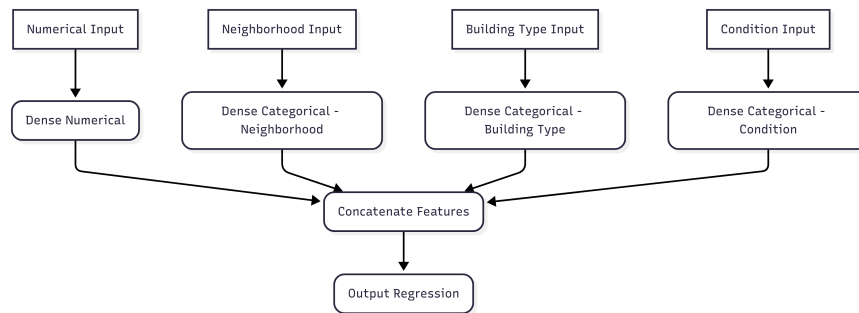


Figure 3: Functional Model Architecture for Ames Housing Dataset

```

# This function will generate a new parsnip model called
# `ames_functional_mlp`.
create_keras_functional_spec(
  model_name = "ames_functional_mlp",
  layer_blocks = list(
    # Define the four input layers of the model.
    numerical_input = input_numerical,
    neighborhood_input = input_neighborhood,
    bldg_input = input_bldg,
    condition_input = input_condition,
    # Process each input branch with a dense layer.
    # `inp_spec()` defines the input tensor for each block.
    processed_numerical = inp_spec(dense_numerical, "numerical_input"),
    processed_neighborhood = inp_spec(
      dense_categorical,
      "neighborhood_input"
    ),
    processed_bldg = inp_spec(dense_categorical, "bldg_input"),
    processed_condition = inp_spec(dense_categorical, "condition_input"),
    # Concatenate the processed branches.
    # The named character vector in `inp_spec()` lists the argument names of
    # `concatenate_features` and assigns each to the upstream block that
    # supplies its tensor.
    combined_features = inp_spec(
      concatenate_features,
      c(
        numeric = "processed_numerical",
        neighborhood = "processed_neighborhood",
        bldg = "processed_bldg",
        condition = "processed_condition"
      )
    ),
    # Add a dense layer after concatenation
    processed_combined = inp_spec(dense_combined, "combined_features"),
    # Define the final output layer.
    output = inp_spec(output_regression, "processed_combined")
  ),
  mode = "regression"
)

```

Figure 3 visualizes the resulting graph:

5.4 Model Specification and Workflow

The `ames_functional_mlp` model specification is defined, setting some hyperparameters to `tune()`. Then a workflow that combines the recipe and the model specification is created.

```

# Define the tunable model specification.
# The `ames_functional_mlp()` function that was created is used.

```

```
# The arguments of the layer blocks can be tuned by prefixing them with the
# block name (e.g., `processed_numerical_units`).
functional_mlp_spec <- ames_functional_mlp(
  processed_numerical_units = tune(id = "numerical_units"),
  processed_neighborhood_units = 256,
  processed_bldg_units = 256,
  processed_condition_units = 256,
  num_processed_combined = tune(id = "num_combined_layers"),
  processed_combined_units = tune(id = "combined_units"),
  processed_combined_dropout = tune(id = "combined_dropout"),
  # compile and fit arguments can also be passed for the \CRANpkg{keras3} model.
  compile_loss = "mean_squared_error",
  compile_optimizer = tune(),
  compile_metrics = c("mean_absolute_error"),
  learn_rate = tune(),
  fit_epochs = 100,
  fit_batch_size = 32,
  fit_validation_split = 0.2,
  fit_callbacks = list(
    callback_early_stopping(monitor = "val_loss", patience = 10)
  )
) |>
  set_engine("keras")

# Create a workflow with the recipe and the model specification.
ames_wf <- workflow() |>
  add_recipe(ames_recipe) |>
  add_model(functional_mlp_spec)
```

5.5 Tune Model

A tuning grid is defined and `tune_race_anova()` is used to perform cross-validation and find the best hyperparameters.

```
# Define the tuning grid
params <- extract_parameter_set_dials(ames_wf) |>
  update(
    numerical_units = hidden_units(range = c(64, 128)),
    num_combined_layers = num_terms(range = c(3, 5)),
    combined_units = hidden_units(range = c(64, 256)),
    combined_dropout = dropout(range = c(0.1, 0.5)),
    learn_rate = learn_rate(range = c(0.01, 0.05), trans = NULL),
    compile_optimizer = optimizer_function(
      values = c("adam", "rmsprop")
    )
  )

functional_mlp_grid <- grid_regular(params, levels = 2)

ames_tune_results <- tune_race_anova(
  ames_wf,
  resamples = ames_folds,
  grid = functional_mlp_grid,
  metrics = metric_set(rmse, mae, rsq),
  control = control_race(save_pred = TRUE, save_workflow = TRUE)
)
```

5.6 Visualizing Tuning Results

Table 1 reports the learning rate, optimizer, and number of combined layers for the best-performing configurations.

Table 1: Tuning Results Summary (Top RMSE Candidates)

learning_rate	optimizer	num_units	combined_layers	comb_units	comb_dropout	rmse
0.01	adam	64	3	64	0.1	37595.64
		64	3	64	0.5	39586.93
		64	3	256	0.1	35693.79
		64	3	256	0.5	43446.65
		64	5	64	0.1	41047.63
		64	5	256	0.1	36576.71
		64	5	256	0.5	42987.74
		128	3	64	0.1	37038.40
		128	3	64	0.5	42904.82
		128	3	256	0.1	37546.45
		128	3	256	0.5	36806.73
		128	5	64	0.1	39004.30
		128	5	256	0.1	36746.76
	rmsprop	64	3	64	0.1	43236.38
		64	3	64	0.5	46536.41
		64	3	256	0.5	42205.63
		64	5	256	0.1	45557.59
		128	3	64	0.1	38122.71
		128	3	64	0.5	40911.50
		128	5	64	0.1	45079.44
		128	5	256	0.1	35540.96
0.05	adam	64	3	64	0.1	39109.81
		64	3	64	0.5	46394.76
		64	3	256	0.1	36851.49
		64	3	256	0.5	41796.49
		128	3	64	0.1	39974.52
		128	3	64	0.5	47511.90
	rmsprop	128	3	256	0.1	32858.04
		128	3	64	0.1	47010.48

5.7 Comparing Models with `workflowsets`

To demonstrate the flexibility of `kerasnip` within the broader `tidymodels` ecosystem, `ames_functional_mlp` can be compared with a traditional statistical model using `workflowsets`. This allows for a systematic comparison of different modeling approaches on the same dataset.

First, a simple linear regression model and a corresponding recipe are defined.

```
# Define a linear regression model for comparison
linear_reg_spec <- linear_reg() |>
  set_engine("lm")

# Create a simpler recipe for the linear regression model
# (without step_collapse, as it's not needed for single-input models)
linear_reg_recipe <- recipe(Sale_Price ~ ., data = ames_train) |>
  step_normalize(all_numeric_predictors()) |>
  step_dummy(all_nominal_predictors())
```

Next, a workflowset containing both the `kerasnip` workflow and the linear regression workflow is created.

```
ames_workflowset <- workflow_set(
  preproc = list(
    kerasnip = ames_recipe,
    linear_reg = linear_reg_recipe
  ),
  models = list(
    kerasnip_mlp = functional_mlp_spec,
    linear_model = linear_reg_spec
  )
)

# Combine the workflows into a workflowset
ames_workflowset <- ames_workflowset |>
  filter(
```

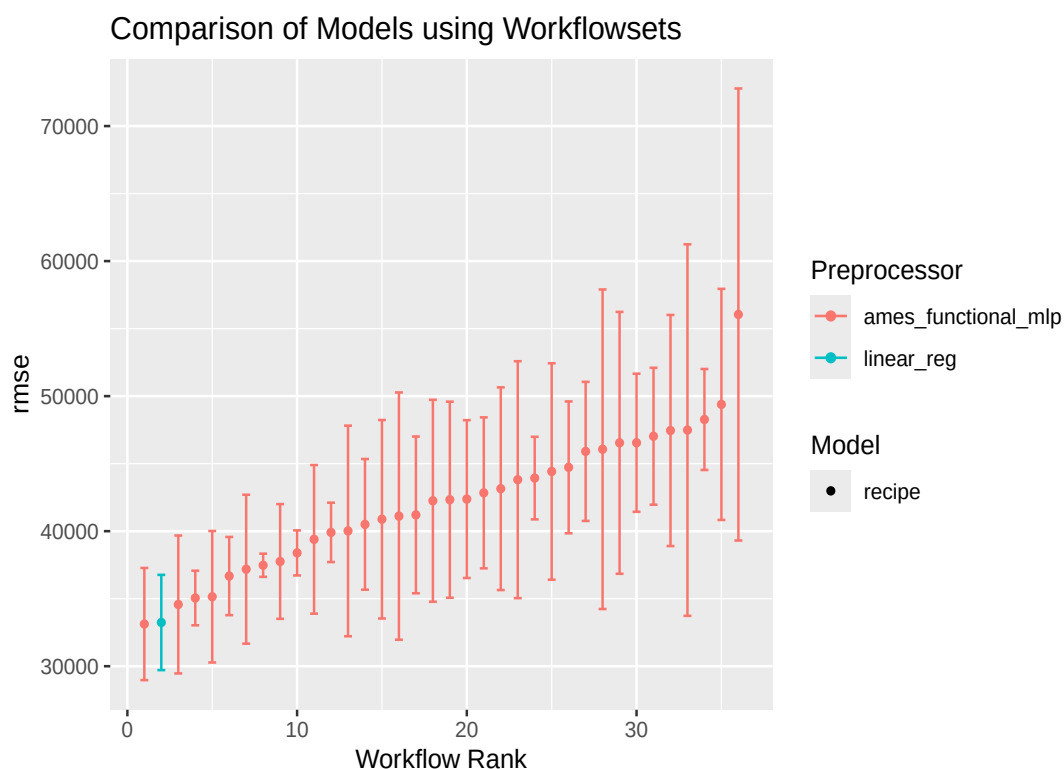


Figure 4: Workflowset Results - RMSE across Different Models

```
wflow_id %in% c("kerasnip_kerasnip_mlp", "linear_reg_linear_model")
) |>
option_add(grid = functional_mlp_grid, id = "kerasnip_kerasnip_mlp") |>
option_add(grid = 10, id = "linear_reg_linear_model")
```

Now, the workflowset can be tuned using `workflow_map()`.

```
ames_workflowset_results <- ames_workflowset |>
  workflow_map(
    "tune_race_anova",
    resamples = ames_folds,
    metrics = metric_set(rmse, mae, rsq),
    control = control_race(save_pred = TRUE, save_workflow = TRUE)
  )
```

Figure 4 visualizes the workflowset comparison so the relative performance of the baseline linear model and the functional MLP can be assessed side by side.

5.8 Finalize Workflow and Evaluate

```
best_functional_mlp_params <- select_best(ames_tune_results, metric = "rmse")

final_ames_wf <- finalize_workflow(ames_wf, best_functional_mlp_params)

final_ames_fit <- fit(final_ames_wf, data = ames_train)

ames_test_pred <- predict(
  final_ames_fit,
  new_data = ames_test
)
```

The performance of the final model on the test set is shown in the following table.

wflow_id	.metric	mean	std_err	.config
kerasnip_kerasnip_mlp	rmse	33122.90	2525.470	pre0_mod11_post0
linear_reg_linear_model	rmse	33238.93	2144.440	pre0_mod0_post0
kerasnip_kerasnip_mlp	rmse	34572.16	3104.013	pre0_mod41_post0
kerasnip_kerasnip_mlp	rmse	35051.24	1228.393	pre0_mod57_post0
kerasnip_kerasnip_mlp	rmse	35144.00	2959.411	pre0_mod25_post0
kerasnip_kerasnip_mlp	rmse	36676.42	1759.823	pre0_mod01_post0
kerasnip_kerasnip_mlp	rmse	37184.60	3352.894	pre0_mod27_post0
kerasnip_kerasnip_mlp	rmse	37476.51	521.139	pre0_mod19_post0
kerasnip_kerasnip_mlp	rmse	37757.21	2583.135	pre0_mod09_post0
kerasnip_kerasnip_mlp	rmse	38390.95	1014.105	pre0_mod17_post0
kerasnip_kerasnip_mlp	rmse	39395.43	3345.039	pre0_mod03_post0
kerasnip_kerasnip_mlp	rmse	39909.96	1335.792	pre0_mod12_post0
kerasnip_kerasnip_mlp	rmse	40016.36	4740.274	pre0_mod26_post0
kerasnip_kerasnip_mlp	rmse	40504.98	2942.000	pre0_mod05_post0
kerasnip_kerasnip_mlp	rmse	40883.88	4468.054	pre0_mod49_post0
kerasnip_kerasnip_mlp	rmse	41116.34	5566.620	pre0_mod02_post0
kerasnip_kerasnip_mlp	rmse	41205.46	3528.358	pre0_mod29_post0
kerasnip_kerasnip_mlp	rmse	42250.80	4545.248	pre0_mod15_post0
kerasnip_kerasnip_mlp	rmse	42328.54	4414.458	pre0_mod50_post0
kerasnip_kerasnip_mlp	rmse	42376.67	3553.134	pre0_mod18_post0
kerasnip_kerasnip_mlp	rmse	42837.85	3398.218	pre0_mod14_post0
kerasnip_kerasnip_mlp	rmse	43145.28	4562.742	pre0_mod33_post0
kerasnip_kerasnip_mlp	rmse	43812.70	5332.857	pre0_mod10_post0
kerasnip_kerasnip_mlp	rmse	43932.14	1858.352	pre0_mod13_post0
kerasnip_kerasnip_mlp	rmse	44420.38	4872.205	pre0_mod45_post0
kerasnip_kerasnip_mlp	rmse	44726.03	2968.202	pre0_mod22_post0
kerasnip_kerasnip_mlp	rmse	45907.85	3129.222	pre0_mod21_post0
kerasnip_kerasnip_mlp	rmse	46066.50	7192.835	pre0_mod04_post0
kerasnip_kerasnip_mlp	rmse	46541.57	5894.981	pre0_mod08_post0
kerasnip_kerasnip_mlp	rmse	46548.22	3108.154	pre0_mod51_post0
kerasnip_kerasnip_mlp	rmse	47028.88	3081.750	pre0_mod30_post0
kerasnip_kerasnip_mlp	rmse	47456.52	5201.905	pre0_mod31_post0
kerasnip_kerasnip_mlp	rmse	47488.55	8362.792	pre0_mod34_post0
kerasnip_kerasnip_mlp	rmse	48270.46	2273.806	pre0_mod06_post0
kerasnip_kerasnip_mlp	rmse	49387.63	5202.067	pre0_mod46_post0
kerasnip_kerasnip_mlp	rmse	56045.59	10175.920	pre0_mod16_post0

.metric	.estimator	.estimate
rmse	standard	36820.125073
mae	standard	21806.668965
rsq	standard	0.841205



Figure 5: Predicted vs. Actual Sale Prices on the Test Set

compile_optimizer	learn_rate	units nu com	combined_dropout	num_combined_layers	mean	std_err
adam	0.05	128 256	0.1	3	32858.04	2087.124
rmsprop	0.01	128 256	0.1	5	35540.96	3230.442
adam	0.01	64 256	0.1	3	35693.79	1521.243
adam	0.01	64 256	0.1	5	36576.71	3205.070
adam	0.01	128 256	0.1	5	36746.76	5275.331

5.9 Visualizing Model Performance

Figure 5 contrasts predictions with observed prices to illustrate how well the tuned workflow generalizes to unseen data.

This case study demonstrates how `kerasnip` can be used to define, tune, and evaluate a complex, multi-input `keras3` model within a standard `tidymodels` workflow. The ability to seamlessly integrate such models opens up new possibilities for tackling complex machine learning problems in R.

6 Conclusion

In this paper, `kerasnip` was introduced as an R package that bridges `keras3` and `tidymodels`. The examples and the Ames Housing case study demonstrated how practitioners can define, preprocess, tune, and evaluate complex `keras3` models inside reproducible `tidymodels` workflows. `kerasnip`’s core innovation lies in dynamic code generation, enabling architectural tuning and democratizing deep learning access in R while remaining faithful to `tidyverse` design principles. Future work will focus on shipping higher-level helpers for popular architectures (e.g., CNN templates, Transformer scaffolds), enhancing `tfdatasets` integration, and smoothing workflows for fine-tuning pre-trained `keras3` applications—capabilities that build on top of, rather than unlock, the flexible building blocks already present. `kerasnip` is envisioned as a cornerstone for the R deep learning community, fostering innovation.

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 <http://tensorflow.org/>. Software available from tensorflow.org. [p188]

- J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL <http://github.com/google/jax>. [p188]
- F. Chollet. Keras. <https://github.com/fchollet/keras>, 2015. [p188]
- D. Falbel. *luz: Higher-Level Interface for torch*, 2023. URL <https://github.com/mlverse/luz>. R package version 0.3.0. [p188]
- I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>. [p188]
- M. Kuhn and H. Wickham. Tidymodels: a framework for modeling and machine learning in R. *The R Journal*, 2020. URL <https://tidymodels.org>. R package version 0.1.0. [p188]
- A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. 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 <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>. [p188]
- K. Ushey, J. J. Allaire, and Y. Tang. *reticulate: R Interface to Python*, 2023. URL <https://CRAN.R-project.org/package=reticulate>. R package version 1.30. [p188]

David Díaz
Independent Researcher

ORCID: 0000-0002-0927-9795
daviddrsch@gmail.com

Conformal Prediction Tools in R

by Jesus Uriel Diaz-Martinez, Xiaoqian Wang, and Paula Moraga

Abstract Conformal prediction constitutes a flexible family of methods for predictive uncertainty quantification, which has recently drawn significant attention in the statistical literature because of its distribution-free, finite-sample, and model-agnostic coverage guarantees. While numerous methodological advances have extended the original conformal framework to broader settings, the availability of related software in R remains limited. In this paper, we present an overview of several conformal prediction methods and exemplify the use of available R packages, with a focus on regression for continuous responses given covariates and its extensions to time series forecasting. We further identify current limitations and outline opportunities for future improvements.

1 Introduction

Conformal prediction is a family of methods to construct prediction intervals for point predictions. These methods have been popularized recently in the statistical literature due to their model-agnostic and finite-sample coverage guarantees, which require mild assumptions on the distribution of the data. Such properties make the framework appealing for a wide range of tasks such as regression and time series forecasting. The model-agnostic property allows conformal prediction to be wrapped around several models. On the other hand, the finite-sample property is related to conservatively marginal coverage guarantees. Specifically, for a future observation of interest Y_{n+1} , the resulting prediction interval, denoted by Γ^α , will satisfy $P(Y_{n+1} \in \Gamma^\alpha) \geq 1 - \alpha$ for any significance level $\alpha \in (0, 1)$. These notable features are particularly valuable for providing predictive uncertainty quantification in models that inherently lack it, in particular black-box prediction models. Standard references on conformal prediction include the book *Algorithmic Learning in a Random World* by Vovk et al. (2022) with a recent edition (Vovk et al., 2022), and a tutorial by Shafer and Vovk (2008).

Conformal prediction has emerged as a reliable framework for uncertainty quantification, with applications spanning a wide range of domains. In the medical sciences, it has been applied in breast cancer diagnosis, survivability prediction, and symptom-based condition identification (Vazquez and Facelli, 2022). In the energy sector, conformal prediction has been used to measure prediction uncertainty in simulation-based forecasting, energy and gas demand prediction, and electricity price prediction (e.g., Mendil et al., 2022). More broadly, conformal prediction has also been used to construct prediction intervals for the Direction-of-Arrival (DOA) predictions in geophysics and underwater acoustics, as well as enhancing reliability in aircraft landing systems (Vilfroy et al., 2024).

On the methodological advances, significant progress has been made to extend the scope and applicability of conformal prediction. Lei et al. (2018) demonstrated that conformal prediction methods are not only conservatively valid but also admit model-independent upper bounds on coverage under additional assumptions on the residuals. Many attempts have been made to broaden the framework to more general settings. For example, Tibshirani et al. (2019) proposed a method to address covariate shift, while Romano et al. (2019) tackled the problem of variance heterogeneity in regression settings using flexible quantile regression. Barber et al. (2023) proposed a general theory based on weighted quantiles, extending conformal prediction to settings previously far from the scope of the original formulation. Chen et al. (2018) investigated computationally efficient methods to implement conformal prediction in regression tasks. In the context of time series forecasting, specialized methods based on online updating of prediction intervals have been pioneered by Gibbs and Candes (2021), Zaffran et al. (2022), Bhatnagar et al. (2023) and recently by Wang and Hyndman (2026).

In the R ecosystem, several CRAN packages provide implementations of conformal prediction for both regression and time series forecasting. Available packages for regression include **pintervals** (Randahl, 2026), **marginalEffects** (Arel-Bundock et al., 2024), and **probably** (Kuhn et al., 2025). For time series forecasting, the **caretForecast** (Akay, 2026) and **conformalForecast** (Wang and Hyndman, 2025) are two packages for conformal prediction of time series forecasting. These packages generally adopt a flexible framework that can be applied to diverse underlying models.

Other available packages offer model-specific implementations, among them are **conformalInference.multi** (Diquigiovanni et al., 2025), designed for multivariate regression; and **ConformalSmallest** (Yang, 2021), which optimizes conformal prediction intervals generated with quantile and ridge regression. While these packages provide valuable contributions, they also present some limitations. For instance, some implementations restrict conformal prediction to models from specific packages due to dependencies on custom objects generated by those ecosystems.

In other languages, like Python, libraries such as **MAPIE** (Cordier et al., 2023), **crepes** (Boström, 2024) and **torchCP** (Huang et al., 2025) provide implementations of conformal prediction algorithms that augment regression and classification algorithms available in **scikit-learn** (Pedregosa et al., 2011) and **pytorch** (Paszke et al., 2019), which are two major machine learning libraries.

In this paper, we demonstrate the use of the **pintervals**, **marginaleffects**, **probably** and **conformal-Forecast** packages in R to the problem of constructing uncertainty intervals in two settings: regression with covariates and time series forecasting. In the regression context, the methods are applied to predict a continuous response variable using a number of covariates, exemplified through a wind farm energy production dataset. For the time series domain, where multi-step-ahead forecasting is of particular interest, we demonstrate the approach using data on the number of active cases of Dengue in Brazil.

The remainder of the paper is as follows. In Section 2 we review the basics of conformal prediction, from the classical perspective, and its adaptations to regression and time series forecasting. In Sections 3 and 4, we provide code examples for regression and time series forecasting, respectively. Section 5 provides a discussion from the perspective of our examples, while Section 6 states our conclusions.

2 Background on conformal prediction

In this section, we provide an overview of the technical aspects of conformal prediction required to understand the examples and the code. For an in-depth overview, the readers are encouraged to consult the work by Vovk et al. (2022), Shafer and Vovk (2008), Lei et al. (2018), Romano et al. (2019), and Wang and Hyndman (2026), just to mention a few.

2.1 The setting for conformal prediction

The basic setting for conformal prediction assumes that there exists an *example space* $\mathbf{Z}$, from which data points $Z_1, Z_2, Z_3, \dots$ can be drawn according to a data-generating process. In practice, only a finite sample $\{Z_1, Z_2, \dots, Z_n\}$ is available; this collection will be denoted by $\mathbf{Z}^{(*)}$. The goal of conformal prediction methods is to construct prediction sets, Γ^α , for a future observation $Z_{n+1} \in \mathbf{Z}$ such that the coverage guarantee $P(Z_{n+1} \in \Gamma^\alpha) \geq 1 - \alpha$ holds for a predefined significance level $\alpha \in (0, 1)$. This guarantee is known as *conservative validity*. A key distributional assumption to achieve this goal is that the data generating process is exchangeable, i.e., the process produces an exchangeable collection of random variables. Specifically, a collection of (possibly vector-valued) random variables, $(Z_1, Z_2, \dots, Z_n)$, is said to be exchangeable if for any permutation π of the set $\{1, \dots, n\}$, the joint distribution of $(Z_{\pi(1)}, Z_{\pi(2)}, \dots, Z_{\pi(n)})$ is the same as the joint distribution of the original collection. This assumption is weaker, and hence more flexible, than the usual i.i.d. assumption since exchangeable random variables are not necessarily independent.

The practical construction of the prediction set relies on the so-called *nonconformity measure*, which is a function $S : \mathbf{Z}^{(*)} \times \mathbf{Z} \rightarrow [-\infty, +\infty]$, to quantify how different a new example $z \in \mathbf{Z}$ is relative to the available examples in $\mathbf{Z}^{(*)}$. From this function, one obtains nonconformity scores; the corresponding score for each observation Z_i is defined as $V_i := S(\mathbf{Z}^{(*)}, Z_i)$. In several tasks, such as regression and time series forecasting, there are *natural* choices of the nonconformity measure, for example the absolute residuals for regression and the signed residuals for time series forecasting. This does not imply that they must be the unique choice, as the choice usually depends on the context of the problem (Vovk et al., 2022). A detailed discussion and review of nonconformity measures in the context of regression is provided in Kato et al. (2023).

2.2 Full conformal prediction

One of the earliest and more general approaches is full conformal prediction, which serves as a conceptual starting point for other methods in this framework for uncertainty quantification. In this method, the collection of available examples is augmented with a potential value, $z \in \mathbf{Z}$, for Z_{n+1} . Let $\mathbf{Z}_z^{(*)} := \{Z_1, Z_2, \dots, Z_n, z\}$ be the collection of augmented examples; note that for each potential value in $\mathbf{Z}$ the augmented set of available examples will be different. Define $V_j := S(\mathbf{Z}_z^{(*)}, Z_j)$ for $j = 1, 2, \dots, n$, $V_{n+1}^z := S(\mathbf{Z}_z^{(*)}, z)$, and let α be a significance level. The p-value for the new potential observation will be defined as

$$p_{n+1}^z = \frac{|\{j = 1, \dots, n+1 : V_j \geq V_{n+1}^z\}|}{n+1}, \quad (1)$$

and the full conformal prediction set is defined as

$$\Gamma^\alpha := \{z \in \mathbf{Z} : p_{n+1}^z > \alpha\}. \quad (2)$$

Intuitively, full conformal prediction evaluates how plausible a candidate future observation would be if it were included together with the training data. Candidate values that produce nonconformity scores similar to those observed in the data will have larger p-values and will be included in the prediction set, whereas those that appear atypical relative to the observed data will have smaller p-values and will be excluded. Therefore, p-values play a crucial role in conformal prediction, determining whether a candidate value is included in the prediction set Γ^α . As noted by Vovk et al. (2022), the choice of nonconformity measure directly affects the resulting prediction set.

A practical difficulty with full conformal prediction is that, to construct the prediction set, one needs to compute the nonconformity scores for all candidate elements in the set $\mathbf{Z}$, as every element can be potentially included. In terms of computations, there are particular algorithms for which the expensive computations can be avoided due to the structure of the prediction algorithm, such as conformalized ridge regression and nearest neighbor regression (Vovk et al., 2022). Nevertheless, as highlighted by Lei et al. (2018), the construction of full conformal prediction sets remains computationally expensive in general. This challenge has motivated the development of alternative methods with lower computational cost, a widely used alternative is described in the following subsection.

2.3 Inductive conformal prediction

A widely used and computationally efficient variant of conformal prediction is inductive conformal prediction, also known as split conformal prediction. It is conceptually similar to the full conformal framework; however, the key difference lies in how p-values are constructed. In split conformal prediction, model fitting and prediction set construction are separated, eliminating the need to recompute scores for each candidate value, thereby substantially reducing the computational cost. Instead of augmenting the available examples $\mathbf{Z}^{(*)}$ with potential values from the example space, the available data is split into two disjoint subsets: a proper training set $\mathbf{Z}_{\text{Train}}^{(*)}$ and a proper calibration set $\mathbf{Z}_{\text{Cal}}^{(*)}$, containing l and m elements, respectively, such that $l + m = n$. The key idea in split conformal is to use only the data in the proper training set to fit the model and then use the examples in the calibration set to construct the prediction intervals. Therefore, the nonconformity scores V_j are only defined for calibration samples, with $V_j := S(\mathbf{Z}_{\text{Train}}^{(*)}, Z_j)$, $Z_j \in \mathbf{Z}_{\text{Cal}}^{(*)}$, and for a candidate $z \in \mathbf{Z}$ representing a potential value of Z_{n+1} , $V_{n+1}^z := S(\mathbf{Z}_{\text{Train}}^{(*)}, z)$. Using these nonconformity scores, nonconformity of a candidate value is assessed by comparing its score against the empirical distribution of calibration scores. The p-value for inductive (split) conformal prediction is then defined by

$$p_{n+1}^z = \frac{|\{j \in I_{\text{Cal}} : V_j \geq V_{n+1}^z\}|}{m + 1}, \quad (3)$$

where I_{Cal} is the index set of the calibration set, with $|I_{\text{Cal}}| = m$. The corresponding inductive prediction interval is constructed as

$$\Gamma^\alpha := \{z \in \mathbf{Z} : p_{n+1}^z > \alpha\}. \quad (4)$$

For split conformal prediction, an alternative expression to Equation (4) exists in terms of the empirical quantiles induced by the nonconformity scores on the calibration set. The prediction interval is constructed according to Equation (5), which is particularly advantageous for computer implementations and theoretical analysis.

$$\Gamma^\alpha = \left\{z \in \mathbf{Z} : V_{n+1}^z \leq \text{Quantile}\left(1 - \alpha; \{V_j\}_{j \in I_{\text{Cal}}}\right)\right\}. \quad (5)$$

2.4 Applications to regression and time series forecasting

In general, the regression setting can be described as the task of estimating a function $\hat{f} : X \rightarrow Y$, where X corresponds to the space where the predictors take values, typically a subset of $\mathbb{R}^d$ with d representing the number of *predictors*; the space Y , which is usually a subset of $\mathbb{R}$, contains the *response* of interest. For the application of conformal prediction to regression, the example space is defined as $\mathbf{Z} = X \times Y$. It is assumed that a future value of the predictor X_{n+1} is observed in order to produce a prediction for the response Y_{n+1} given by $\hat{Y}_{n+1} = \hat{f}(X_{n+1})$.

In the particular case of full conformal prediction, the data augmentation step will be performed as $\mathbf{Z}_y^{(*)} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n), (x_{n+1}, y)\}$, for some $y \in Y$. The nonconformity scores will be computed as $V_i = S(\mathbf{Z}_y^{(*)}, y_i)$ and $V_{n+1}^y = S(\mathbf{Z}_y^{(*)}, y)$; then the p-value and the prediction interval will follow a similar definition as in Equations (1) and (2), respectively, with minor notational changes to make explicit the dependence only on the potential value y for Y_{n+1} :

$$\Gamma^\alpha := \{y \in Y : p_{n+1}^y \geq \alpha\}. \quad (6)$$

The theoretical properties of this approach have been investigated by [Lei et al. \(2018\)](#), who established an upper bound on the coverage probability in finite sample settings. This property, also known as anti-conservativeness, is independent of the predictive model or any distributional assumptions, depending only on the sample size of the dataset. The result is stated below.

Result 1 *Under the exchangeability assumption the conformal prediction method for regression satisfies that for a new pair of observation (X_{n+1}, Y_{n+1}) it holds that $1 - \alpha \leq P(Y_{n+1} \in \Gamma^\alpha) \leq 1 - \alpha + \frac{1}{n+1}$.*

It is important to remark that this result was proven using the absolute deviation in Equation (7) as nonconformity measure, although the result holds with any valid nonconformity measure as long as the data is exchangeable and the algorithm is symmetric.

$$S(Y_i, \hat{Y}_i) := |Y_i - \hat{Y}_i|. \quad (7)$$

Split conformal prediction extends naturally to the regression setting by using the same ideas as full conformal prediction, but with nonconformity scores computed exclusively on the calibration set. This yields the following coverage guarantee.

Result 2 *Under the exchangeability assumption the split conformal prediction method for regression, constructed with a calibration set with m examples, satisfies that for a new pair of observations (X_{n+1}, Y_{n+1}) , $1 - \alpha \leq P(Y_{n+1} \in \Gamma^\alpha) \leq 1 - \alpha + \frac{1}{m+1}$.*

Using Result 2, one can solve for the size of the calibration set m to get

$$m \geq \frac{2 - (\nu + \alpha)}{\nu + \alpha - 1}, \quad (8)$$

where ν is a positive scalar that satisfies $1 - \alpha < \nu \leq 1$. The equation above does not imply that using the derived calibration set size in practice we will observe an empirical coverage around ν , as this upper bound is derived based on a worst-case analysis and not a coverage calibration. However, we can use it to choose the size of the calibration set and we will use it in the next section.

When the nonconformity measure defined in Equation (7) is used in the split conformal prediction framework, the resulting prediction intervals take the following simple form:

$$[\hat{Y}_{n+1} - \hat{Q}_\alpha, \hat{Y}_{n+1} + \hat{Q}_\alpha], \quad (9)$$

where $\hat{Q}_\alpha$ is the $\lceil (m+1)(1-\alpha) \rceil$ -th empirical quantile of the nonconformity scores computed on the calibration set. The prediction interval from Equation (9) therefore has a constant width across all values of the predictors.

The coverage guarantees of the methods described above rely on the validity of exchangeability in the data and on the prediction algorithm being symmetric. However, in the context of time series, the exchangeability is not a realistic assumption for the data and hence coverage guarantees of usual conformal prediction methods do not apply. One of the main features of time series is the existence of temporal dependence, with observations ordered in time and often displaying autocorrelation. In addition, several methods for time series modeling and prediction are not symmetric, again, because of the temporal structure inherent to the data. In practice, applications of classic conformal prediction methods to time series forecasting can lead to systematic under coverage, meaning the obtained prediction interval does not achieve the nominal coverage level. In time series forecasting, two scenarios are of particular interest: one-step ahead and multi-step ahead forecasting. In the following, time series will be denoted by $\{Y_t\}$, where each $Y_t \in Y$. An h -step ahead forecast is denoted by $\hat{Y}_{t+h}$, where $H \geq 1$ denotes the forecasting horizon. If covariates are used to produce the forecast, they will be denoted by $\mathbf{x}_t \in X$.

Several methods have been developed to extend conformal prediction to time series forecasting. Of particular interest has been to address the violation of data exchangeability and to allow for non-symmetric prediction algorithms. For example, the NexCP (Non Exchangeable Conformal Prediction)

method of Barber et al. (2023) extends full and split conformal prediction by introducing weights to the construction of quantiles, thereby allowing more “trusted” observations to play a greater role in the prediction step. The approach provides approximate coverage when exchangeability does not hold and exact coverage when it does. This method is general and applies not only to time series but other instances where dependence plays a major role in the data and must not be ignored. Other approaches exist such as the EnbPI method proposed by Xu and Xie (2021), which employs ensembles of bootstrap models to construct prediction intervals, one notable advantage of this method is the fact that it is suitable for multi-step forecasting tasks.

Beyond weighted or bootstrap-based extensions, another line of research known as online conformal prediction, is inspired by the sequential nature of time series. Online conformal prediction methods produce intervals dynamically as new data becomes available, refining the significance level over time to adapt to changes in data distribution or updates to the model based on the most recent information. The first such method was developed by Gibbs and Candes (2021), under the assumption that an optimal significance level exists for constructing the desired $1 - \alpha$ prediction interval for the point forecast $\hat{Y}_{t+1}$. Hence the authors propose a method to estimate this time-varying significance level α_{t+1} , and use it to construct prediction intervals. Further enhancements of the update rule include Zaffran et al. (2022), Bhatnagar et al. (2023), Angelopoulos et al. (2023), and recently Wang and Hyndman (2026) to construct prediction intervals for multi-step ahead predictions. Wang and Hyndman (2026) also developed the **conformalForecast** R package (Wang and Hyndman, 2025), providing a practical implementation of their approaches.

Online conformal prediction methods take into account the temporal ordering of the data by introducing the idea of sequential splits. At each time point $t = N$ where $N \gg H$, the available data are split into a proper training set of size t_r and a proper calibration set of size $N - t_r > H$. For notational convenience, the data at time t is denoted by z_t , to accommodate both the univariate case, where only the time series of interest is used to fit the model (hence $z_t = y_t$), and the multivariate case, where covariates are included so that $z_t = (x_t, y_t)$. In the latter, the covariates may include not only information up to the current time but also information of the subsequent h steps. Within this framework, the training set is defined as a moving window of size t_r and the nonconformity measure is defined for each h -step-ahead forecast at time $t + h$ as follows

$$S(\{Z_j\}_{j=t-t_r+1}^t, Y_{t+h}), \quad \text{for } t_r \leq t \leq N; h = 1, \dots, H,$$

with corresponding score expressed as

$$V_{t+h|t} := S(\{z_j\}_{j=t-t_r+1}^t, y_{t+h}), \quad \text{for } t_r \leq t \leq N; h = 1, \dots, H.$$

In the particular case of the work by Wang and Hyndman (2026), the nonconformity measure is given by

$$S(\{Z_j\}_{j=t-t_r+1}^t, Y_{t+h}) := Y_{t+h} - \hat{Y}_{t+h|t},$$

where $\hat{Y}_{t+h|t}$ is the prediction of Y_{t+h} using data in the training window corresponding to time t , i.e., $\{Z_j\}_{j=t-t_r+1}^t$.

The steps for the multi-step-ahead conformal prediction are outlined as follows:

1. Initialization: Obtain the initial proper training set $\{z_k\}_{k=1+t-t_r}^t$ with $t = t_r$, and fit the forecasting model. Then obtain H -step-ahead point forecasts $\{\hat{y}_{t+h|t}\}_{h=1}^H$ and compute the corresponding nonconformity scores $\{V_{t+h|t}\}_{h=1}^H$.
2. Rolling procedure: Roll the training window of size t_r forward by one observation, i.e., set $t \rightarrow t + 1$, for $t_r \leq t \leq N$. Then repeat Step 1 until the collection of nonconformity scores, $\{V_{t+h|t}\}_{t=t_r}^{N-H}$, $h = 1, \dots, H$, have been computed across the entire calibration set.
3. Prediction interval construction: Use the nonconformity scores obtained from Step 2 to compute quantile estimates $\hat{q}_{t+h|t}$ for each forecasting horizon $h = 1, \dots, H$. Conformal prediction intervals are then given by $\Gamma_{t+h|t}(\hat{q}_{t+h|t}) := \{y \in Y : V_{t+h|t}^y \leq \hat{q}_{t+h|t}\}$, with $t = N$ and for each horizon $h = 1, \dots, H$.
4. Online update: As new observations arrive, update the calibration and the training sets by rolling the data forward one data point at a time. Recompute the nonconformity scores based on the new calibration set and then repeat Step 3 to update prediction intervals across time.

The quantile $\hat{q}_{t+h|t}$ in Step 3 can be estimated using the methods proposed by Wang and Hyndman (2026). In particular, the update rule for the proposed Autocorrelated Multi-step Conformal Prediction (AcMCP) method is defined as

$$\hat{q}_{t+h|t} = \hat{q}_{t+h-1|t-1} + \eta \left(\text{err}_{t|t-h} - \alpha \right) + r_t \left(\sum_{i=h+1}^t \left(\text{err}_{i|i-h} - \alpha \right) \right) + \tilde{e}_{t+h|t}, \quad (10)$$

where $\tilde{e}_{t+h|t}$ is a term that accounts for autocorrelation in the nonconformity scores and is the combination of a MA($h-1$) model trained on the h -step-ahead forecast errors and a linear regression model trained to regress the h -step-ahead error from the past $h-1$ errors. $\text{err}_{t|t-h}$ is defined as

$$\text{err}_{t|t-h} := \begin{cases} 1, & \text{if } y_t \notin \Gamma_{t|t-h}(\hat{q}_{t|t-h}) \\ 0, & \text{otherwise} \end{cases}.$$

$\eta > 0$ is a constant learning rate, and r_t is a saturation function that satisfies the conditions

$$x \geq c \cdot g(t-h) \Rightarrow r_t(x) \geq b, \quad \text{and} \quad x \leq -c \cdot g(t-h) \Rightarrow r_t(x) \leq -b,$$

for some constant $b, c > 0$ and a function g that is sublinear, nonnegative and nondecreasing. These conditions are required to establish the asymptotic coverage guarantees.

In the following sections, we show how to use available implementations of conformal prediction methods for regression and time series forecasting. It is important to note that for the regression part, the space $Y = \{y \in \mathbb{R} : y \geq 0\}$, and the covariate space is $X = \{(1, x) : x \geq 0\}$. For time series forecasting, the response Y remains the same.

3 Conformal prediction methods for regression in R

In this section, split conformal prediction will be illustrated using the [SCADA dataset](#) obtained from Kaggle ([Erisen, 2019](#)). SCADA, short for Supervisory Control and Data Acquisition systems, is widely used to monitor wind energy production. The dataset consists of 39,692 registries of electricity production (KWh), wind speed (m/s), theoretical electricity production (KWh), and wind direction collected from a wind turbine's SCADA system in Turkey. Our objective is to predict the production of a windmill using wind speed as a covariate. Such datasets present challenges for modeling, as system failures can generate anomalous observations that deviate from expected behavior. Therefore, the models used for prediction should be robust and capable of quantifying uncertainty in the presence of outliers, which may otherwise compromise the reliability of prediction intervals.

3.1 Data preparation

First, we load the dataset and inspect its first rows. Then, we create a data frame `d_scada` for the analysis, containing the response y , representing active power (`active_power`), and the covariate x , representing wind speed (`wind_speed`). A simple summary of the data indicates that active power (y) ranges from 0 to 3618.7 (kW), while wind speed ranges from 1.2 to 25.2 (m/s).

```
library(readr)
library(curl)
url_scada <- "https://raw.githubusercontent.com/uriel278/tutorials-cp/main/data/scada.csv"
ds_scada <- read_csv(curl(url_scada))
head(ds_scada)

#> # A tibble: 6 x 4
#>   active_power wind_speed theoretical_power wind_direction
#>   <dbl>         <dbl>         <dbl>         <dbl>
#> 1     380.         5.31           416.           260.
#> 2     454.         5.67           520.           269.
#> 3     306.         5.22           391.           273.
#> 4     420.         5.66           516.           271.
#> 5     381.         5.58           492.           266.
#> 6     402.         5.60           499.           265.

d_scada <- data.frame(y = ds_scada$active_power, x = ds_scada$wind_speed)
```

As split conformal prediction requires a calibration set and we need a proper training set for model fitting, we generate the data splits by using the function `split(d_scada, idx)`, where the second argument `idx` is a vector of the same length as the number of observations and contains one of the

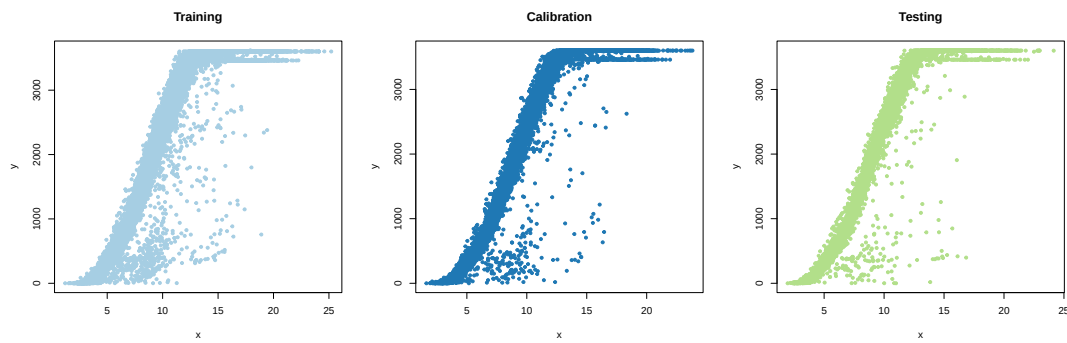


Figure 1: Splits of the original dataset. Left: Proper training data to fit each model. Middle: Proper calibration data for computation of nonconformity scores. Right: Test split to evaluate empirical coverage of each prediction interval.

strings from the vector `c("train", "calib", "test")` selected randomly by the `sample()` function with arguments `prob = c(0.6, 0.252, 0.148)`, `size = nrow(d_scada)`, and `replace = TRUE`; this results in a split that allocates 60% of the data to the proper training set, 25.2% to the proper calibration set, and the remaining 14.8% to the testing set. In this example, the choice of allocating 60% of the data for training is expected to preserve the response-covariate relationship for reliable model training, the size of the calibration set was determined by setting $\nu = 0.95001$ in Equation (8).

```
set.seed(141019)
idx <- sample(x = c("train", "calib", "test"), prob = c(0.6, 0.252, 0.148),
             size = nrow(d_scada), replace = TRUE)
scada <- split(d_scada, idx)
```

Next, we visualize in Figure 1 the obtained data splits.

```
cols <- c(
  palette.colors(3, palette = "Paired"),
  palette.colors(2, palette = "Okabe-Ito")
)
par(mfrow = c(1, 3))
with(scada, {
  plot(y ~ x, data = train,
       col = cols[1], xlab = "x", ylab = "y", main = "Training", pch = 20)
  plot(y ~ x, data = calib,
       col = cols[2], xlab = "x", ylab = "y", main = "Calibration", pch = 20)
  plot(y ~ x, data = test,
       col = cols[3], xlab = "x", ylab = "y", main = "Testing", pch = 20)
})
```

3.2 Modeling

In this subsection the models to perform predictions for the data set will be introduced. Three models will be fitted to the data to exemplify different workflows for conformal prediction. The first model is a linear model with quadratic term and will be implemented with the well known `lm()` function, the second model will be a monotone generalized additive model fitted using the `scam()` function from the `scam` package, and for users familiar with the `tidymodels` framework, the third model will be a random forest regression model fitted using functions from the `ecosystem`.

Model: Linear model

For simplicity, we first model the response variable using a simple linear regression (SLR) model with a quadratic term for the covariate using the training data.

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon, \quad \epsilon \sim N(0, \sigma^2).$$

In R, the following call fits the model described above.

```
linear_model <- lm(y ~ x + I(x^2), data = scada$train)
```


Model: SCAM

Given the nonlinear shape of the relationship between the response and the covariate of interest, we also fit an alternative model, namely a Shape Constrained Additive Model (SCAM), which offers greater flexibility in capturing the form of the regression function and includes the option to add monotonicity constraints to the fitted regression function. The model is of the form

$$Y = f(X) + \epsilon, \quad \epsilon \sim N(0, \sigma^2),$$

where $f(X) = \sum_{j=1}^k b_j(X)\beta_j$ and $b_j(\cdot)$ denotes the basis function, taken here to be cubic splines.

To fit the model above, we use the functions `s()` and `scam()` from the **scam** package (Pyra, 2025) to specify the smooth term and fit the model to the training data, respectively. To specify the monotone increasing smooth term, we call the `s(x, bs = "mpi")` in the right hand side of the formula; here the argument `bs = "mpi"` indicates the monotonicity constraint. Additionally, we have specified the argument `family = Gamma(link = "identity")` to select the Gamma distribution as the likelihood for the data with identity link, as energy production is non-negative.

```
library(scam)
scam_model <- scam(y ~ s(x, bs = "mpi"), family = Gamma(link = "identity"),
  data = scada$train)
```

Model: Random forest regression

As a non-parametric alternative that makes fewer assumptions about the underlying distribution and functional form, we also consider a Random Forest (RF) regression model. Random Forest is an ensemble learning method that captures complex, non-linear interactions by aggregating the predictions of multiple decision trees. This model can be expressed as

$$Y = \frac{1}{B} \sum_{b=1}^B T_b(X, \theta_b) + \epsilon$$

Here, the regression function has the form $f(X) = \frac{1}{B} \sum_{b=1}^B T_b(X, \theta_b)$ where each $T_b(\cdot)$ denotes an individual decision tree fitted using a bootstrap sample of the training data. To fit this model, we will illustrate the usage of the **tidymodels** framework for those users familiar with it. In this framework, we will use the **parsnip** package to specify the model, we will set the engine as **ranger** to specify such package as backend for model fitting.

```
library(tidymodels)
library(parsnip)
rf_spec <- rand_forest(trees = 154, min_n = 813) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("regression")
rf_wf <- workflow() %>%
  add_model(rf_spec) %>%
  add_formula(y ~ x)
rf_fit <- rf_wf %>% fit(data = scada$train)
```

The parameters `trees = 154` and `min_n = 813` were selected via 5-fold cross-validation on the training split. The argument `min_n` was optimized first by setting the `trees` argument to its default (`trees = 500`). After the optimal value was found, the `trees` parameter was optimized by setting `min_n = 813`, the optimal value from the previous step.

Prediction intervals

For the computation of prediction intervals powered by conformal prediction, we will use the packages **pintervals** and **marginaleffects** for all the fitted models, while the **probably** package will be used for the model fitted with the **tidyverse** framework. We will focus on split conformal prediction to reduce the computational load. This will help us illustrate the differences between the two packages.

The **pintervals** package follows a package agnostic implementation for the computation of the intervals. The function for split conformal prediction is `pinterval_conformal()`, which requires as arguments a vector of predictions made by a model on the data for which the intervals will be computed, a vector of predictions on the calibration set, the vector of true values of the response on

the calibration set, and the confidence level. These arguments correspond to the argument `pred`, `calib`, `calib_truth`, and `alpha`. The `pinterval_conformal` handles the computation of the nonconformity scores and returns the prediction intervals as a tibble.

```
# This step takes on average 2 minutes on a MacBook Pro machine with 16 GB of
# RAM and a 2.3 GHz 8-Core Intel(R) Core(TM) i9-9889H CPU.
library(pintervals)
lm_pintervals <- pinterval_conformal(
  pred = predict(linear_model, newdata = scada$test),
  calib = predict(linear_model, newdata = scada$calib),
  calib_truth = scada$calib$y
)
scam_pintervals <- pinterval_conformal(
  pred = predict(scam_model, newdata = scada$test),
  calib = predict(scam_model, newdata = scada$calib),
  calib_truth = scada$calib$y
)
rf_pintervals <- pinterval_conformal(
  pred = predict(rf_fit, new_data = scada$test)$pred,
  calib = predict(rf_fit, new_data = scada$calib)$pred,
  calib_truth = scada$calib$y
)
```

When using the **marginalEffects** package to compute the prediction intervals, we need to use the predictions and inferences functions from the package. The arguments to the first function should be a fitted model and covariate values at which predictions will be computed in our case they should be specified by `model = scam_model` and `newdata = scada$test`. The inferences function is used to specify the method employed to compute uncertainty estimates around the predictions from the model. The three required arguments by this function are the method, in our case `method = "conformal_split"`, the training data and the calibration data specified by `data_train = scada$train` and `data_calib = scada$calib`.

```
library(marginalEffects)
linear_meint <- predictions(model = linear_model, newdata = scada$test) |>
  inferences(
    method = "conformal_split",
    data_train = scada$train,
    data_calib = scada$calib
  )

scam_meint <- predictions(model = scam_model, newdata = scada$test) |>
  inferences(
    method = "conformal_split",
    data_train = scada$train,
    data_calib = scada$calib
  )
```

Finally, the **probably** package can be used to obtain prediction intervals for models fitted with the **tidymodels** framework. The nonconformity scores for split conformal prediction are computed using the `int_conformal_split()` function. The arguments of this function include the fitted model workflow and the calibration split. Then, to retrieve the prediction intervals we use the `predict()` function by providing the nonconformity scores obtained by `int_conformal_split()` function, the predictor values for which we want to get prediction intervals, and the desired confidence level.

```
library(probably)
rf_pintervals <- rf_fit |>
  int_conformal_split(cal_data = scada$calib) |>
  predict(new_data = scada$test, level = 0.95)
```

Plot predictions and prediction intervals

After the results for each method have been obtained, plots of the estimated regression lines together with their corresponding prediction intervals are shown in Figure 2. The code used to generate the figure is provided below.

```

plot_pred <- function(pred, pred.low, pred.high, newdata, label = "Split", model = "LM") {
  p <- data.frame("x" = newdata$x, "y" = newdata$y, "pred" = pred,
                 "pred.low" = pred.low, "pred.high" = pred.high)

  p <- p[order(p$x), ]
  with(p, {
    coverage <- mean(y >= pred.low & y <= pred.high) * 100
    width <- median(pred.high - pred.low)
    caption <- sprintf("%s + %s\nCoverage = %.2f%\nMedian width = %.2f",
                      model, label, coverage, width)
    plot(y~x, col = cols[3], pch = 19, cex = .5, main = caption)
    lines(pred~x)
    lines(x, pred.high, lwd = 1.5, col = cols[5])
    lines(x, pred.low, lwd = 1.5, col = cols[5])
  })
}

```

4 Conformal prediction methods for time series forecasting

The R package **conformalForecast** (Wang and Hyndman, 2025) implements the AcMCP method developed in Wang and Hyndman (2026) and some extensions of several other popular conformal prediction methods for multi-step forecasting. An attractive feature of the package design is the attempt to separate the model specification and fitting from the conformal prediction step. This structure reflects the core principle of conformal prediction—its flexibility and ability to wrap around almost any model. In particular, the functions for conformal prediction require the user to provide a function capable of generating predictions (a model). This function is then used to obtain predictions on the (online) calibration set, from which conformal prediction intervals are constructed using any of the methods implemented in the package.

In this example, a time series of the weekly number of [dengue cases](#) in São Paulo, Brazil is considered. The data has been obtained from the surveillance system [InfoDengue](#) (Codeco et al., 2018) and contains the number of dengue cases spanning 678 weeks from 2010 to 2022 (52 weeks per year, with the exception of 2012 and 2017, which each contains 53 weeks).

4.1 Data preparation

We begin by reading the dengue time series data with `read.csv()`, and then we create a time series object with the `ts()` function using the weekly number of cases. As mentioned above, there are two years in which the number of observations is 53 instead of 52. For visualization purposes, we specify `frequency = 52` in the `ts()` function, which simplifies the visualization despite these irregularities. An alternative approach could involve using `frequency = 52.28571` to account for leap years, or imputing the 53rd week's data by averaging it with the 52nd week, if appropriate for the context of the problem and data at hand. For this illustration, however, we prioritize simplicity and clarity in illustrating how to calculate intervals, so we proceed with `frequency = 52`. Importantly, in the model building step (see section [Modeling and prediction](#)), the `frequency` argument is not used, as seasonality is explicitly defined in the model. If a model were to depend on this argument, the user would need to adjust the frequency accordingly to ensure valid results.

A summary of the data shows weekly dengue cases range from 38 to 12,367 cases.

```

library(readr)
library(curl)
url_dengue <- "https://raw.githubusercontent.com/uriel278/tutorials-cp/main/data/dengue.csv"
d_dengue <- read_csv(curl(url_dengue))
d_dengue <- ts(d_dengue$x, start = c(2010, 1), frequency = 52)
summary(d_dengue)

#>      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>      38.0  126.5   232.0   563.8  459.0 12367.0

```

For this time series, the data from 2010-2013 will play the role of the initial training set, the data from 2014-2017 will be used for calibration, and the data from 2018-2022 will serve as the testing set for evaluation of empirical coverage. The **conformalForecast** package splits the data automatically, as

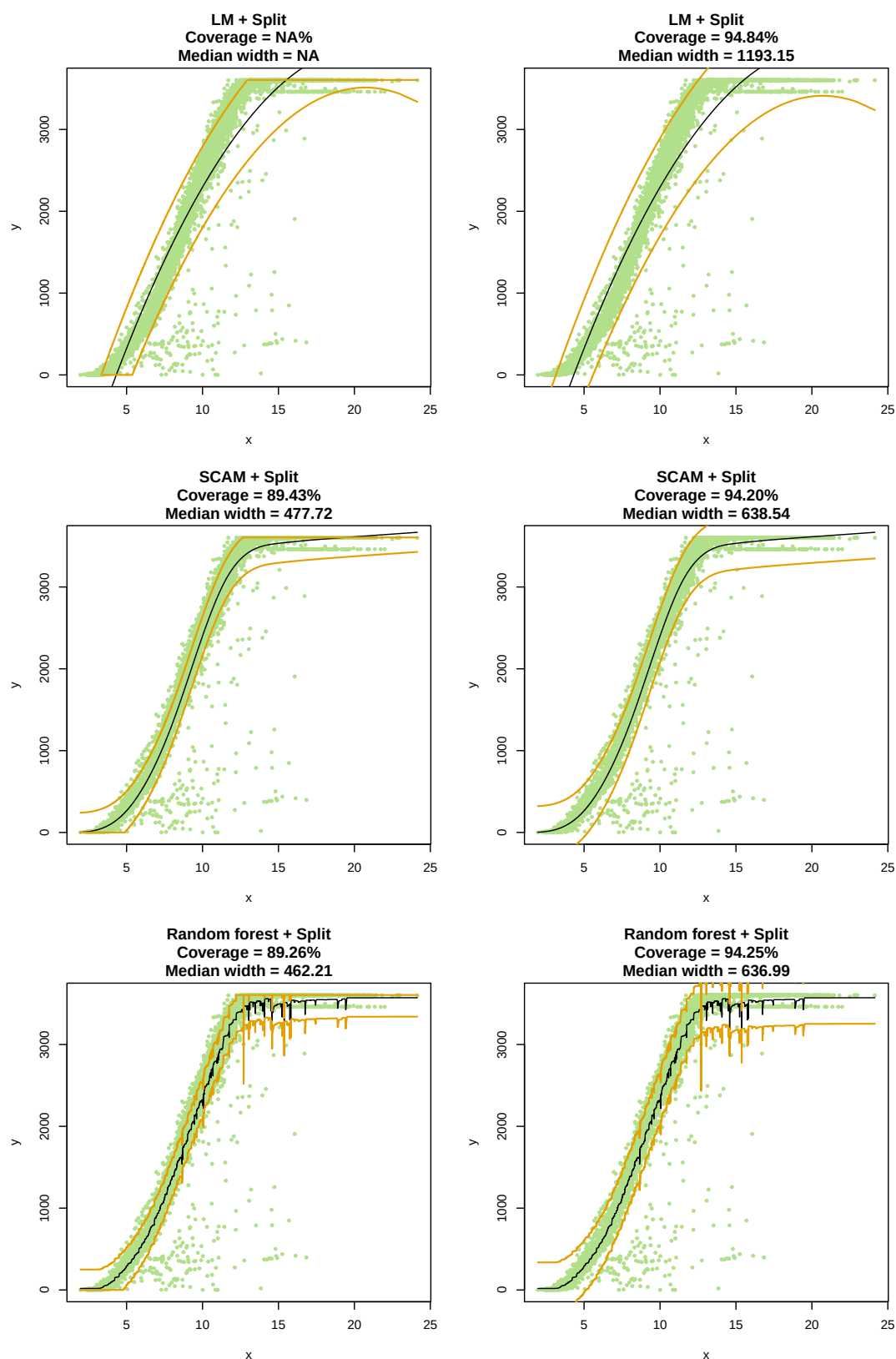


Figure 2: Point predictors depicted with black color and its corresponding 95% prediction intervals in dark orange. Top row: Prediction intervals computed with the pintervals package for the linear model with quadratic term (left), and the margineffects package (right). Middle row: Conformal prediction intervals for the SCAM model using pintervals (left), and margineffects (right). Bottom row: Prediction intervals for the random forest model produced by the pintervals package (left), and the probably package (right).

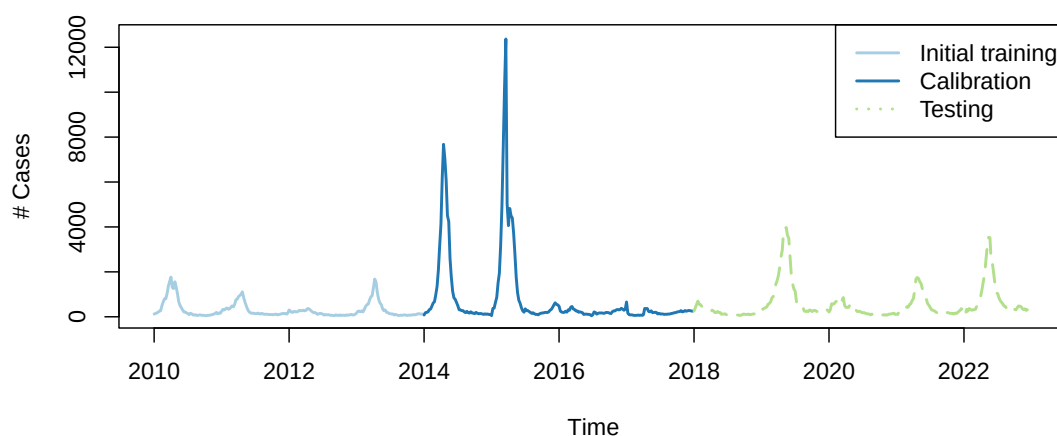


Figure 3: Splits of the dengue time series for the example. The initial training split (in light blue), calibration set (in dark blue) and the testing set in dashed green lines.

described in section [Prediction intervals](#) below, ensuring consistency with the setting illustrated in Figure 3.

For visualization purposes, the initial training, calibration, and testing splits can be created manually using the `window()` function by providing the start and end dates of each segment. The resulting initial splits are displayed in the Figure 3, each represented with a different color.

```
dtrain_dengue <- window(d_dengue, start = c(2010, 1), end = c(2013, 52))
dcal_dengue <- window(d_dengue, start = c(2014, 1), end = c(2017, 52))
dtest_dengue <- window(d_dengue, start = c(2018, 1))
```

4.2 Modeling and prediction

The **conformalForecast** package separates the model specification and fitting from the process of computing conformal prediction intervals. Users provide a forecasting function that generates point forecasts for the calibration set. These forecasts are then used to compute nonconformity scores and to construct conformal prediction intervals.

It is important to remark that the main function from the **conformalForecast** package, `cvforecast()`, does not require the user to pass pre-defined data splits. The splits generated above are only for visualization purposes. Instead, the `cvforecast()` function generates splits following the sequential split framework described in Section Multi-step Conformal Prediction.

The `cvforecast()` function

The `cvforecast()` function is the main function of the **conformalForecast** package. As arguments it requires a time series object via the `y` argument, a forecasting function with the `forecastfun` argument, the forecasting horizon `h`, and the confidence level(s) `level` for the prediction interval. The `level` argument accepts either a single numerical value or a vector of numerical values to compute prediction intervals at multiple confidence levels. However, caution is required when providing a vector, as some conformal prediction methods available in the package cannot handle multiple significance values and may return NA values for prediction intervals without warning. All methods work correctly when a single numerical input is provided, although this limitation is not documented. Additional arguments are available, and readers are encouraged to see the package documentation for details.

The `cvforecast()` function applies the `forecastfun` to the data, computes predictions for each forecasting horizon given sequential information based on sequential splits, and computes nonconformity scores (in this case, residuals). The returned object is of class `cvforecast` and `forecast`.

This function only computes predictions that are required by the conformal prediction methods available in the package, it does not compute any conformal prediction intervals directly. However, if the forecasting function supplied via `forecastfun` produces its own prediction intervals, these will be stored in the returned object.

An example of a call to the `cvforecast()` function is as follows.

```
pred <- cvforecast(y = d_dengue, forecastfun = forecastfunction,
```

```
h = 4, level = 95, window = 52*4)
```

Here, the data `d_dengue` contains the dengue time series. The forecasting function is provided through the argument `forecastfun` (details are explained in the following subsection). The horizon `h = 4` requests four-step-ahead forecasts, and `level = 95` specifies 95% prediction intervals from the forecasting function. The argument `window = 52*4` restricts the training window to four years of data.

Model specification and the `forecastfun` argument

Here, we specify the function used to obtain point forecasts. In this illustration example, we use a Seasonal ARIMA (SARIMA) model, as in the plots we can observe some temporal patterns that repeat over time. The SARIMA model for a time series Y_t is of the form:

$$\Phi_P(B)\phi_p(B)\nabla^d\nabla_S^DY_t = \Theta_Q(B)\theta_q(B)W_t,$$

where B is a lag operator, $\Phi(\cdot)$ and $\phi(\cdot)$ are the seasonal autoregressive and autoregressive polynomials of orders P and p , respectively; $\Theta_Q(\cdot)$ and $\theta_q(\cdot)$ are the seasonal moving average and moving average polynomials of orders Q and q , respectively; ∇_S^D and ∇^d are the seasonal difference and difference operators of orders (D, S) and d , respectively, and W_t is a white noise process.

The code below shows how to define a forecasting function, `forecastfunction()`, using the `Arima()` function of the **forecast** R package. This function will later be passed to `cvforecast()` to compute forecasts as the `forecastfun` argument. To be compatible, any forecasting function defined here must accept three arguments: a time series `x`, the forecasting horizon `h`, and the confidence level `level`. Failure to do so will result in errors when calling the `cvforecast()` function. Note that this requirement does not pose a limitation in the forecasting models that can be used, as the user can always create a wrapping function mapping these arguments to the corresponding inputs of the model of interest. In our example, the wrapper combines the `Arima()` function, which estimates the model given a dataset and specification arguments, with the forecast function, which produces multi-step forecasts and confidence intervals. Furthermore, note that before the prediction/forecast step we fit the model. This is not a mandatory step, we may have an already fitted model and can call only the corresponding prediction function. A complete example of the `forecastfunction()` is shown below.

```
library(forecast)
forecastfunction <- function(x, h, level){
  Arima(x, order = c(1, 0, 1),
        seasonal = list(order = c(0, 1, 0), period = 52),
        method = "ML") |>
    forecast(h = h, level = level)
}
```

We can verify that this function is working well by applying it to the training set `dtrain_dengue` to obtain four-step-ahead forecasts with 95% prediction intervals. These intervals are generated by the `forecast()` function of the **forecast** package, applied to the object returned by `Arima()`, and are not conformal prediction intervals.

```
forecastfunction(x = dtrain_dengue, h = 4, level = 95)
```

```
#>      Point Forecast      Lo 95      Hi 95
#> 2014.000      173.5596    11.67358 335.4457
#> 2014.019      249.0351   -48.40245 546.4726
#> 2014.038      259.5708  -110.87368 630.0154
#> 2014.058      226.1600  -192.66605 644.9861
```

Despite being very flexible, the **conformalForecast** requires objects of type `forecast` with a certain structure to be able to work properly. In the following, using the **caretForecast**, we will illustrate how to go over the issues by extracting the required fields from a model, and creating a similar object for the **conformalForecast** package to work with. The second model will be fitted using nonnegative least squares, which restrict the fitted regression function to be nonnegative; in the **caretForecast** package all the models have an autoregressive structure.

To fit the model, we specify the `predfun` as follows

```
library(caretForecast)
```



```

nnlsreg <- function(x, h, level = 0.95){
  fit <- ARml(x, max_lag = 52, caret_method = "nnls",
             verbose = FALSE, seasonal = FALSE, fixed_window = TRUE,
             cv_horizon = h, initial_window = 52)
  fc <- forecast(fit, h = h, level = level)
  caretToforecast(fc)
}

```

The function `caretToforecast()` is specified in the following, where the fields of the forecast object produced by the resulting fit from the **caretForecast** are parsed to the expected format of the **conformalForecast**, in this case the lower and upper fields must have as column names the coverage specified by the user.

```

caretToforecast <- function(caretForecastObj){
  level <- caretForecastObj$level
  colnames(caretForecastObj$lower) <- paste0(level, "%")
  colnames(caretForecastObj$upper) <- paste0(level, "%")
  standardize_fc <- structure(
    list(
      "method" = caretForecastObj$method,
      "level" = caretForecastObj$level,
      "mean" = caretForecastObj$mean,
      "lower" = caretForecastObj$lower,
      "upper" = caretForecastObj$upper,
      "fitted" = caretForecastObj$fitted
    ),
    class = "forecast"
  )
  return(standardize_fc)
}

```

4.3 Prediction intervals

Now that the `cvforecast()` function and the `forecastfun` argument have been introduced, we proceed to obtain prediction intervals with conformal prediction. The first step is to call the `cvforecast()` function with the required arguments, as shown below.

```

# This step takes on average 4 minutes and 20 seconds on a MacBook Pro machine with 16 GB of
# RAM and a 2.3 GHz 8-Core Intel(R) Core(TM) i9-9889H CPU.
library(conformalForecast)
pred_arima <- cvforecast(d_dengue, forecastfunction,
                       h = 4, level = 95, initial = 1, window = 52*4)
pred_nnls <- cvforecast(d_dengue, nnlsreg,
                      h = 4, level = 95, window = 52*4)

```

Once forecasts have been obtained using `cvforecast()`, prediction intervals can be generated using conformal prediction methods implemented in **conformalForecast** package, see [Wang and Hyndman \(2026\)](#) for more details and a complete list of available algorithms.

In this tutorial, we focus on the AcMCP method developed by the authors, implemented via the `acmcp()` function, and the extension of the Adaptive Conformal Prediction (ACP) method implemented in the function `acp()`.

The AcMCP method applies the update rule described in Equation (10). Its first argument is the object of class `cvforecast` returned by the `cvforecast()` function, which in our case is stored as `pred`. The calibration set size should also be specified; we set `ncal = 52*4` to match the four years of calibration data (2014–2017). In addition, we set `rolling = TRUE` to use a rolling window for quantile construction. With these settings, the initial training set covers 2010–2013, while the initial calibration set corresponds to 2014–2014, as described earlier in subsection [Data preparation](#).

The `acmcp()` function also accepts several additional parameters, which are documented in detail in [Wang and Hyndman \(2026\)](#) and in the package documentation. Briefly, the learning rate parameter `lr` controls the step size of the update and is set here to `lr = 0.1`. The parameter `KI` — specific to this conformal method — is a positive constant in order to place the integrator on the same scale as the nonconformity scores. If prior knowledge about the scale of the scores is available, it can be used to set

KI; in this example, we set $KI = 2000$. Finally, the parameter $Csat$ is a positive constant ensuring that by time T , the absolute coverage guarantee is at least $1 - \alpha - \delta$, where δ reduces the desired coverage guarantee $1 - \alpha$ because, unlike traditional conformal prediction methods that are conservative in finite samples, the coverage guarantees here are asymptotic. This reduction δ is *controlled* by the argument δ and can be interpreted as the amount of coverage one is willing to forgo given that we cannot observe the process for an infinite period of time. The expression for $Csat$ in the `acmcp` function is defined in Appendix C of [Angelopoulos et al. \(2023\)](#).

```
# This step takes on average 16 seconds on a MacBook Pro machine with 16 GB of RAM and a
# 2.3 GHz 8-Core Intel(R) Core(TM) i9-9889H CPU.
arima_acmcp <- acmcp(pred_arima, ncal = 52*4, rolling = TRUE,
                    lr = 0.1, Tg = length(d_dengue), delta = 0.01, KI = 2000)
nnls_acmcp <- acmcp(pred_nnls, ncal = 52*4, rolling = TRUE,
                    lr = 0.1, Tg = length(d_dengue), delta = 0.01, KI = 2000)
```

For the ACP method, we require fewer parameters. In addition to the nonconformity scores, only two parameters are necessary: a learning rate γ , similar to lr , and the size of the calibration set $ncal$.

```
# This step takes on average 7 seconds on a MacBook Pro machine with 16 GB of RAM and a
# 2.3 GHz 8-Core Intel(R) Core(TM) i9-9889H CPU.
arima_acp <- acp(pred_arima, gamma = 0.1, ncal = 52*4)
nnls_acp <- acp(pred_nnls, gamma = 0.1, ncal = 52*4)
```

Plot predictions and prediction intervals

We plot the prediction intervals for each forecasting horizon as follows, and the result is displayed in Figure 4. We can observe that the width of the prediction intervals grows as the forecasting horizon increases, this is an expected behavior as extending the forecast horizon generally increases forecast uncertainty.

```
plot_cpForecast <- function(obj, plot_h, ts_train, ts_cal, ts_test, ts,
                           model, label = "AcMCP"){
  coverage_mean <- coverage(obj, level = 95, window = 52)
  width_mean <- width(obj, window = 52)
  for(h in plot_h){
    caption <- sprintf("%s + %s, h = %s\nCoverage = %.2f%\nMean width = %.2f",
                      model, label, h, coverage_mean$mean[h], width_mean$mean[h])
    plot(ts*NA, ylim = c(-500, 12500), ylab = "# Cases",
         main = caption)
    lines(ts_train, col = cols[1], lwd = 2)
    lines(ts_cal, col = cols[2], lwd = 2)
    lines(ts_test, col = cols[3], lwd = 2, lty = 3)
    lines(obj$LOWER$`95%`[, h], col = cols[5])
    lines(obj$UPPER$`95%`[, h], col = cols[5])
  }
}
```

5 Discussion

The examples presented in this work demonstrate how the packages **pintervals**, **marginaleffects**, **probably**, and **conformalForecast** enable the practical implementation of conformal prediction for regression and time series forecasting, respectively. The main strength of these packages is their modular design, which facilitates their integration with existing R packages, and broader modeling ecosystems such as **tidymodels**. This modularity allows practitioners to incorporate conformal prediction into existing workflows with minimal changes, making distribution-free uncertainty quantification more accessible in applied settings.

This modularity allows practitioners to incorporate conformal prediction into existing workflows with minimal disruption, making distribution-free uncertainty quantification more accessible in applied settings. The **pintervals** package shows progress in this direction by including multiple nonconformity measures and allowing users to define custom ones, providing a pathway toward

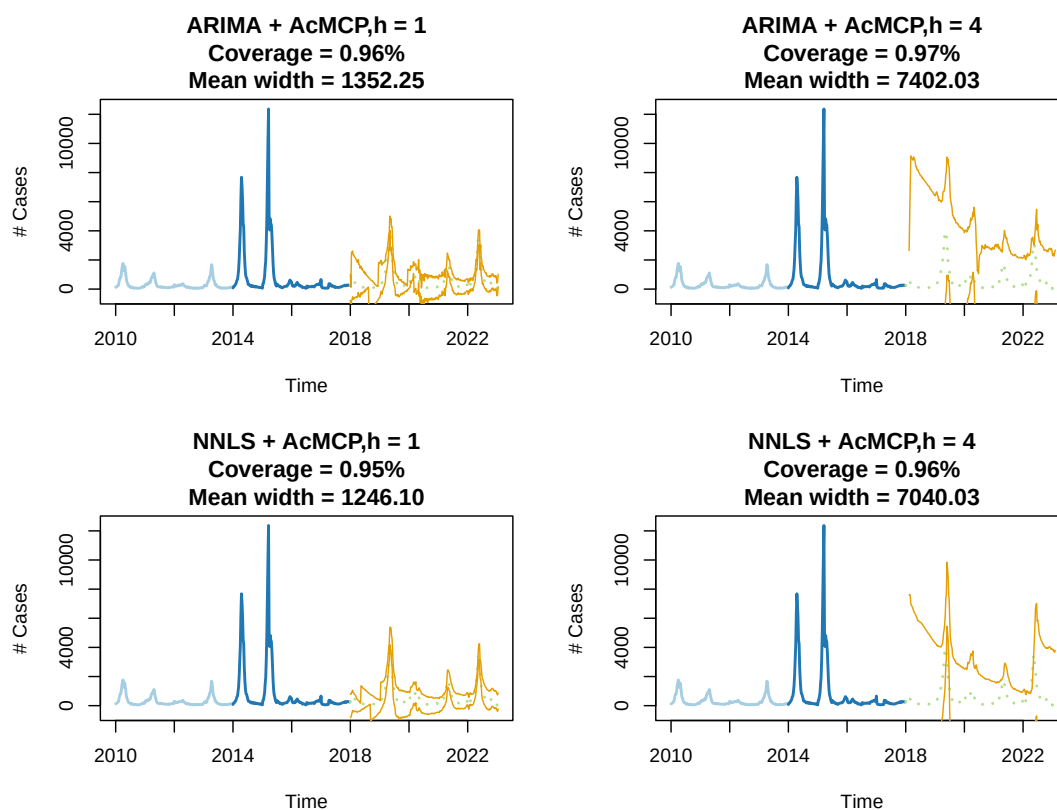


Figure 4: Prediction intervals (in orange) using AcMCP on the dengue data for two forecasting horizon for the ARIMA (first row) and NNLS models (second row).

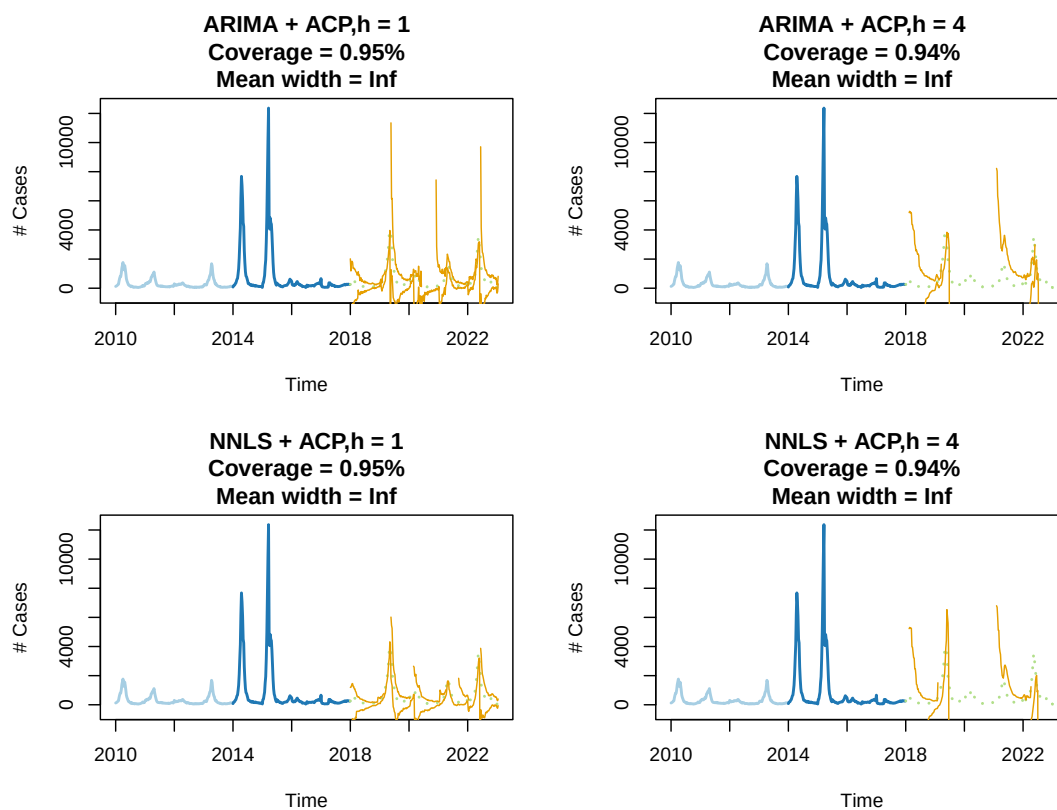


Figure 5: Prediction intervals (in orange) using ACP on the dengue data for two forecasting horizon for the ARIMA (first row) and NNLS models (second row).

more general and adaptable tools, according to their documentation. During the development of this tutorial, however, we identified practical limitations. In particular, computing split conformal prediction intervals can be computationally expensive in some settings, and the package may return missing values for prediction intervals without providing clear warnings or diagnostic messages. Addressing these issues in future releases would further improve reliability for both practitioners and researchers.

The **marginaleffects** and **probably** packages provide efficient implementations of split conformal prediction through functions designed to integrate naturally into their respective frameworks. These tools fill an important gap by enabling conformal prediction within widely used modeling pipelines. The **marginaleffects** package may be particularly well suited for users working in local computing environments, while **probably** may be more suitable for production-oriented workflows. Because both packages are part of larger modeling ecosystems with standardized objects and interfaces, they support a wide range of models and simplify the computation of nonconformity scores and prediction intervals. However, this framework-based design can also introduce limitations. Models outside these ecosystems may require additional development effort to be compatible, which can pose challenges for users without experience extending these frameworks. Future developments could focus on improving extensibility and reducing the technical barrier for integrating custom models.

For time series forecasting, the **conformalForecast** package offers a broad set of algorithms for online conformal prediction. The package includes standardized functions for computing, storing, and retrieving results, facilitating reproducible workflows. Currently, the package relies primarily on the forecast modeling framework. While this does not represent a major practical limitation, as users can construct compatible objects manually as demonstrated in our example, the development of native interfaces to additional ecosystems such as **fable** (O'Hara-Wild et al., 2026) would further enhance usability and adoption.

Beyond the packages considered in this tutorial, several specialized conformal prediction packages are available on CRAN, including **conformalbayes** (McCartan, 2025), **conformalInference.fd** (Diquigiovanni et al., 2022), **conformalInference.multi** (Diquigiovanni et al., 2025), **conformalpvalue** (Tyagi, 2023), **ConformalSmallest** (Yang, 2021), **fabPrediction** (Bersson, 2024), **ClusTorus** (Jung et al., 2022). These packages often provide computationally efficient implementations tailored to specific conformal prediction variants, though sometimes at the expense of model-agnostic flexibility. Together, they reflect the rapid evolution of conformal prediction methodology and software tools.

From a practical standpoint, users seeking seamless integration of conformal prediction into existing regression workflows may benefit most from packages embedded in large modeling ecosystems. In contrast, users requiring specialized conformal prediction variants or optimized implementations for large-scale problems may prefer dedicated conformal prediction packages, even if this reduces model flexibility. The choice of tool therefore depends strongly on the intended application, computational constraints, and existing modeling infrastructure. Looking forward, further progress will likely depend on the development of standardized data structures for storing intermediate results across the multiple stages of conformal prediction. Equally important will be the design of interfaces that clearly define interactions between predictive models, nonconformity measures, score computation, and prediction interval construction. Advances in these areas could significantly improve flexibility while maintaining coherence across software implementations.

6 Conclusions

In this paper, we showcase the **pintervals**, **marginaleffects**, **probably**, and **conformalForecast** packages for regression and time series forecasting, respectively. A key contribution of these tools is the separation of predictive model fitting from conformal inference procedures, allowing users to construct prediction intervals for a wide range of models without relying on distributional assumptions. These packages are designed for practical use and integrate naturally with existing tools in the R ecosystem.

Together, these packages highlight the importance of modular software development for statistical methods and conformal prediction naturally lends itself to such design principles, since multiple interacting components — the predictive model, the choice of nonconformity measure, the computation of nonconformity scores, and the construction of prediction intervals — must work together in a flexible yet coherent way, their design sets them apart from other existing packages that restrict the applicability of conformal prediction to a narrow class of models and offers practitioners a straightforward path to add distribution-free uncertainty quantification to their existing prediction and forecasting workflows.

A current limitation shared by these packages, and by many conformal prediction tools more broadly, is the restricted set of built-in nonconformity measures. Future development could focus on expanding the available measures and improving support for user-defined nonconformity functions, further increasing flexibility across application domains.

Overall, these four packages represent an important step toward making reliable, model-agnostic prediction intervals more accessible to the R community. As software ecosystems continue to mature, conformal prediction is likely to become an increasingly standard component of predictive modeling workflows, lowering the barrier for statisticians and data scientists to incorporate principled uncertainty quantification into applied prediction and forecasting tasks.

7 Acknowledgments

Xiaoqian Wang was supported by the Presidential Foundation of the Academy of Mathematics and Systems Science, Chinese Academy of Sciences, China (No. E555930101).

References

- R. Akay. *caretForecast: Conformal Time Series Forecasting Using State of Art Machine Learning Algorithms*, 2026. URL <https://CRAN.R-project.org/package=caretForecast>. R package version 0.1.3. [p203]
- A. Angelopoulos, E. Candes, and R. J. Tibshirani. Conformal PID control for time series prediction. *Advances in Neural Information Processing Systems*, 36:23047–23074, 2023. [p207, 217]
- V. Arel-Bundock, N. Greifer, and A. Heiss. How to interpret statistical models using marginaeffects for R and Python. *Journal of Statistical Software*, 111:1–32, 2024. [p203]
- R. F. Barber, E. J. Candès, A. Ramdas, and R. J. Tibshirani. Conformal prediction beyond exchangeability. *The Annals of Statistics*, 51(2), 04 2023. doi: 10.1214/23-AOS2276. [p203, 207]
- E. Bersson. *fabPrediction: Compute FAB (Frequentist and Bayes) Conformal Prediction Intervals*, 2024. URL <https://CRAN.R-project.org/package=fabPrediction>. R package version 1.0.4. [p219]
- A. Bhatnagar, H. Wang, C. Xiong, and Y. Bai. Improved online conformal prediction via strongly adaptive online learning. In *International Conference on Machine Learning*, pages 2337–2363. PMLR, 2023. [p203, 207]
- H. Boström. Conformal prediction in python with crepes. *Proceedings of Machine Learning Research*, 230: 236–249, 2024. [p204]
- W. Chen, K.-J. Chun, and R. F. Barber. Discretized conformal prediction for efficient distribution-free inference. *Stat*, 7(1):e173, 2018. [p203]
- C. Codeco, F. Coelho, O. Cruz, S. Oliveira, T. Castro, and L. Bastos. Infodengue: A nowcasting system for the surveillance of arboviruses in brazil. *Revue d’Épidémiologie et de Santé Publique*, 66:S386, 2018. [p212]
- T. Cordier, V. Blot, L. Lacombe, T. Morzadec, A. Capitaine, and N. Brunel. Flexible and systematic uncertainty estimation with conformal prediction via the mapie library. In *Conformal and Probabilistic Prediction with Applications*, pages 549–581. PMLR, 2023. [p204]
- J. Diquigiovanni, M. Fontana, A. Solari, S. Vantini, P. Vergottini, and R. Tibshirani. *conformalInference.fd: Tools for Conformal Inference for Regression in Multivariate Functional Setting*, 2022. URL <https://CRAN.R-project.org/package=conformalInference.fd>. R package version 1.1.1. [p219]
- J. Diquigiovanni, M. Fontana, A. Solari, S. Vantini, P. Vergottini, and R. Tibshirani. *conformalInference.multi: Conformal Inference Tools for Regression with Multivariate Response*, 2025. URL <https://CRAN.R-project.org/package=conformalInference.multi>. R package version 1.1.2. [p203, 219]
- B. Erisen. Wind turbine scada dataset, 2019. URL <https://www.kaggle.com/datasets/berkerisen/wind-turbine-scada-dataset>. [p208]
- I. Gibbs and E. Candes. Adaptive conformal inference under distribution shift. volume 34, pages 1660–1672, 2021. [p203, 207]
- J. Huang, J. Song, X. Zhou, B. Jing, and H. Wei. Torchcp: A python library for conformal prediction. *Journal of Machine Learning Research*, 26(266):1–25, 2025. [p204]
- S. Jung, S. Hong, K. Park, and B. Kim. *ClusTorus: Prediction and Clustering on the Torus by Conformal Prediction*, 2022. URL <https://CRAN.R-project.org/package=ClusTorus>. R package version 0.2.2. [p219]

- Y. Kato, D. M. Tax, and M. Loog. A review of nonconformity measures for conformal prediction in regression. *Conformal and Probabilistic Prediction with Applications*, pages 369–383, 2023. [p204]
- M. Kuhn, D. Vaughan, and E. Ruiz. *probably: Tools for Post-Processing Predicted Values*, 2025. URL <https://CRAN.R-project.org/package=probably>. R package version 1.2.0. [p203]
- J. Lei, M. G'Sell, A. Rinaldo, R. J. Tibshirani, and L. Wasserman. Distribution-free predictive inference for regression. *Journal of the American Statistical Association*, 113(523):1094–1111, 07 2018. [p203, 204, 205, 206]
- C. McCartan. *conformalbayes: Jackknife(+) Predictive Intervals for Bayesian Models*, 2025. URL <https://CRAN.R-project.org/package=conformalbayes>. [p219]
- M. Mendil, L. Mossina, M. Nabhan, and K. Pasini. Robust gas demand forecasting with conformal prediction. In *Conformal and Probabilistic Prediction with Applications*, pages 169–187. PMLR, 2022. [p203]
- M. O'Hara-Wild, R. Hyndman, and E. Wang. *fable: Forecasting Models for Tidy Time Series*, 2026. URL <https://CRAN.R-project.org/package=fable>. R package version 0.5.0. [p219]
- A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019. [p204]
- 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. [p204]
- N. Pya. *scam: Shape Constrained Additive Models*, 2025. URL <https://CRAN.R-project.org/package=scam>. R package version 1.2-20. [p210]
- D. Randahl. *pintervals: Model Agnostic Prediction Intervals*, 2026. URL <https://CRAN.R-project.org/package=pintervals>. R package version 1.1.1. [p203]
- Y. Romano, E. Patterson, and E. Candes. Conformalized quantile regression. *Advances in Neural Information Processing Systems*, 32, 2019. [p203, 204]
- G. Shafer and V. Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(12): 371–421, 2008. [p203, 204]
- R. J. Tibshirani, R. Foygel Barber, E. Candes, and A. Ramdas. Conformal prediction under covariate shift. *Advances in Neural Information Processing Systems*, 32, 2019. [p203]
- C. Tyagi. *conformalpvalue: Computes Conformal p-Values*, 2023. URL <https://CRAN.R-project.org/package=conformalpvalue>. R package version 0.1.0. [p219]
- J. Vazquez and J. C. Facelli. Conformal prediction in clinical medical sciences. *Journal of Healthcare Informatics Research*, 6(3):241–252, 2022. [p203]
- S. Vilfroy, L. Bombrun, T. Urruty, F. De Grancey, J.-P. Lebrat, and P. Carré. Conformal prediction for regression models with asymmetrically distributed errors: application to aircraft navigation during landing maneuver. *Machine Learning*, 113(10):7841–7866, 09 2024. [p203]
- V. Vovk, A. Gammerman, and G. Shafer. *Algorithmic learning in a random world*. Springer, second edition, 2022. [p203, 204, 205]
- X. Wang and R. Hyndman. *conformalForecast: Conformal Prediction Methods for Multistep-Ahead Time Series Forecasting*, 2025. URL <https://CRAN.R-project.org/package=conformalForecast>. R package version 0.1.0. [p203, 207, 212]
- X. Wang and R. J. Hyndman. Online conformal inference for multi-step time series forecasting, 2026. URL <https://arxiv.org/abs/2410.13115>. [p203, 204, 207, 212, 216]
- C. Xu and Y. Xie. Conformal prediction interval for dynamic time-series. In *International Conference on Machine Learning*, pages 11559–11569. PMLR, 2021. [p207]
- Y. Yang. *ConformalSmallest: Efficient Tuning-Free Conformal Prediction*, 2021. URL <https://CRAN.R-project.org/package=ConformalSmallest>. R package version 1.0. [p203, 219]

M. Zaffran, O. Féron, Y. Goude, J. Josse, and A. Dieuleveut. Adaptive conformal predictions for time series. In *International Conference on Machine Learning*, pages 25834–25866. PMLR, 2022. [p203, 207]

Jesus Uriel Diaz-Martinez

King Abdullah University of Science and Technology (KAUST)

Computer, Electrical and Mathematical Sciences and Engineering Division

Department of Statistics

Thuwal, 23955-6900, Kingdom of Saudi Arabia

jesus.diazmartinez@kaust.edu.sa

Xiaoqian Wang

Academy of Mathematics and Systems Science, Chinese Academy of Sciences

The Center for Forecasting Science (corresponding author)

Beijing, China

<https://xqnwang.rbind.io/>

ORCID: 0000-0003-4827-496X

xiaoqian.wang@amss.ac.cn

Paula Moraga

King Abdullah University of Science and Technology (KAUST)

Computer, Electrical and Mathematical Sciences and Engineering Division

Department of Statistics

Thuwal, 23955-6900, Kingdom of Saudi Arabia

<http://www.paulamoraga.com>

ORCID: 0000-0001-5266-0201

paula.moraga@kaust.edu.sa

CovEsts: Nonparametric Autocovariance Estimation and Analysis

by Adam Bilchouris and Andriy Olenko

Abstract The paper introduces the R package **CovEsts**, which implements several nonparametric estimators for the autocovariance function. First, it presents the theoretical foundations of the implemented estimators, their properties, assumptions, and potential limitations. Next, it outlines the structure of the package and its key functions, including several methods for estimating autocovariance functions, constructing corresponding bootstrap confidence regions, and correcting the provided estimators. The package also includes diagnostic tools, such as several metrics for comparing estimators, and additional functions for broader use. The article illustrates a high degree of flexibility of the package in the selection of function parameters and the tuning of the estimators. Applications of selected estimators and package functions are illustrated using simulated data, yearly sunspot counts, and US unemployment increments data.

1 Introduction

For a stationary time series $X(j)$, $j = 0, 1, \dots$, the autocovariance function is used to measure dependencies between observations separated by a lag h . This function appears in various theoretical results and statistical applications. For example, in the Gaussian case, the mean function and autocovariance function describe the time series entirely. Furthermore, the autocovariance function is used in the estimation of parameters for an autoregressive time series (Brockwell and Davis, 1991, Chapter 8.1), model identification, and Kriging (Cressie, 1993, Chapter 3.2). It is also utilised in various non-statistical fields such as digital signal processing, image analysis (Bull and Zhang, 2021, Chapter 3.2), and quality assessment (Massart et al., 2003), just to name a few. Therefore, accurate estimation of the autocovariance function is an important problem in numerous statistical and data science applications.

Surprisingly, despite several other approaches in the literature, the classical standard estimator of the autocovariance function remains the predominant choice among practitioners. Many of them are unaware of its limitations or of more accurate alternatives developed in the recent literature. This is evident in statistical software, where the majority of R and Python packages only provide a wrapper function for the classical estimator. For example, the classical autocovariance estimator realised by the `stats::acf` function in R, is used by **TSA** (Chan and Ripley, 2022 and Cryer and Chan, 2008), **astsa** (Stoffer and Poison, 2024, and Shumway and Stoffer, 2025), **sarima** (Boshnakov and Halliday, 2025) and **forecast** (Hyndman and Khandakar, 2008). The package **forecast** also contains a tapered autocovariance estimator, `taperedacf()`, which is based on the results in McMurry and Politis (2010); Hyndman (2015).

The motivation for the package **CovEsts** (Bilchouris and Olenko, 2025b) was that many nonparametric autocovariance estimators appeared only in research papers with no R or Python code for their computation. Also, as will be discussed in Section 2, different estimators have different theoretical properties, which influence when they should be used. We have not found any packages dealing with potential issues regarding estimation and their corrections, apart from **ncf** (Bjornstad, 2022) implementing the positive-definite adjustment from Hall et al. (1994). Further, no existing packages provide a unified interface and consistent data/output structure across various autocovariance estimators.

Nonparametric autocovariance estimators are particularly important due to their minimal reliance on distributional assumptions, offering greater flexibility compared to parametric approaches. Additionally, they are often used as a preliminary step in parametric modelling, where a restricted set of predefined autocovariance models is fitted to a nonparametric estimate. For example, **gstat** (Gräler et al., 2016) does this for the (semi-)variogram. Parametric estimation was not included in **CovEsts**, as this functionality for specific parametric models is already provided by existing packages.

We intend this package to be used by practitioners working with time series and spatial statistics, with applications in areas such as finance, signal processing, and climate research. We assume basic knowledge of R for using the functional interface of the package. However, more specialised R knowledge may be required for advanced inference and working with user-defined objects.

In the article, we will interchangeably use the terms random process and time series, where the former is for samples at arbitrary time locations, and the latter is used when sampled values are on a uniform sampling grid.

The paper is structured as follows. Section 2 introduces the nonparametric autocovariance estimators and some general approaches for modifying autocovariance estimators. It discusses the

theoretical properties of the estimators and the drawbacks one should be aware of. Section 3 presents the structure of the package and selected main functions. Section 4 provides applications to three data examples: a simulated Gaussian process, yearly sunspot counts, and unemployment data, sourced from the US Bureau of Labor Statistics. Example 4 empirically studies the computational complexity of the estimators by comparing their memory and time usage.

2 Nonparametric estimators of autocovariance functions

There are several nonparametric autocovariance function estimators in the literature, see the reviews in [Cressie \(1993, Chapter 2.4\)](#), [Hall et al. \(1994\)](#), [Cuevas et al. \(2013\)](#), [Dürre et al. \(2015\)](#) and [Bilchouris and Olenko \(2025a\)](#). This section briefly introduces the estimators that were implemented in the package and mentions some of the theoretical drawbacks to deal with when applying them.

For a weakly stationary real-valued time series $X(j)$, $j = 1, 2, \dots$, the autocovariance function is defined as

$$C(h) := E[(X(j) - \bar{X})(X(j+h) - \bar{X})].$$

and the corresponding autocorrelation at lag h is $\rho(h) := C(h)/C(0)$.

Another related function used in the analysis of dependencies is the semivariogram. It uses the second-order moments of increments,

$$\gamma(h) := \frac{1}{2} \text{Var}[X(j) - X(j+h)],$$

The package [CovEsts](#) mainly focuses on estimating $C(\cdot)$ as, under the assumption of weak stationarity, the semivariogram can be determined using the autocovariance function through the following relation

$$\gamma(h) = C(0) - C(h). \quad (1)$$

This relation is not necessarily true when considering the estimated functions or if the semivariogram is unbounded ([Chilès and Delfiner, 2012](#) and [Bilchouris and Olenko, 2025a](#)).

Another function used in time series analysis is the partial autocorrelation function, $\phi_{h,h}$. Unlike the autocorrelation function, it measures the dependencies between observations $X(t)$ and $X(t+h)$ after removing the effects of the intermediary lags. The partial autocorrelation function at lag h can be computed using the Durbin-Levinson algorithm ([Durbin, 1960](#))

$$\phi_{h,h} := \frac{\rho(h) - \sum_{k=1}^{h-1} \phi_{h-1,k} \rho(h-k)}{1 - \sum_{k=1}^{h-1} \phi_{h-1,k} \rho(k)} \quad (2)$$

where $\phi_{h,k} := \phi_{h-1,k} - \phi_{h,h} \phi_{h-1,h-k}$ for $k = 1, 2, \dots, h-1$, and $\phi_{1,1} := \rho(1)$. Unlike the autocorrelation function, there is no zero lag for the partial autocorrelation function. In [stats::pacf](#), the formula (2) uses the classical autocorrelation estimator, but any autocorrelation estimate can be supplied in the package [CovEsts](#).

2.1 Standard estimators

The first two estimators implemented in [CovEsts](#) are well-known and the most widely used. They differ only by the normalising constants:

$$\hat{C}^*(h) := \frac{1}{N-h} \sum_{j=1}^{N-h} (X(j) - \bar{X})(X(j+h) - \bar{X}) \quad (3)$$

and

$$\hat{C}^{**}(h) := \frac{1}{N} \sum_{j=1}^{N-h} (X(j) - \bar{X})(X(j+h) - \bar{X}), \quad (4)$$

where $X(j)$ represents values of an observed time series, $\bar{X}$ is the sample mean of the time series, N is the length of the observation period and $0 \leq h \leq N-1$ is the lag for which the autocovariance function is estimated at ([Yaglom, 1987, Section 3.17](#)). These estimators are typically used for equally sampled values at integer time moments or those with a constant time difference.

Even for these classical estimators, there are several issues that are not expected when applying autocovariance functions in theoretical statistical inference. The first estimator (3) is unbiased if the mean $E[X(j)]$ is known and used instead of $\bar{X}$, but is biased otherwise ([Brockwell and Davis, 2016](#)). This estimator is not positive-definite. The estimator (4) is biased but is positive-definite. This means

that the second estimator gives a true autocovariance function, whilst the first only gives an estimate of a function similar to an autocovariance function.

Another, less-known drawback is that when considering the sum over all lags for the estimated autocorrelations in (4), the sum is always equal to $-1/2$, regardless of the sampled values, see [Hassani \(2009\)](#), [Hassani et al. \(2012\)](#) and [Bilchouris and Olenko \(2025a\)](#). This causes a problem, especially when considering estimation over long time intervals or modelling long-range dependence, as the sum of estimated autocorrelation values is always equal to $-1/2$. Even though the estimators have desirable theoretical properties for a fixed h , when $N \rightarrow \infty$, the constant sum necessitates substantial departure of the estimates from the true autocovariance values if a large range of h is considered for a fixed N .

2.2 Kernel regression estimators

The next estimator, proposed in [Hall and Patil \(1994\)](#) and [Hall et al. \(1994\)](#), applies the kernel regression of pairwise autocovariances to estimate the autocovariance function,

$$\hat{C}_H(t) := \frac{\sum_{i=1}^N \sum_{j=1}^N \check{X}_{ij} K\left(\left(t - (t_i - t_j)\right)/b\right)}{\sum_{i=1}^N \sum_{j=1}^N K\left(\left(t - (t_i - t_j)\right)/b\right)}, \quad (5)$$

where $t, t_i, t_j \in \mathbb{R}$, $\check{X}_{ij} := (X(t_i) - \bar{X})(X(t_j) - \bar{X})$, $i, j = 1, \dots, N$, $K(\cdot)$ is a kernel which has the properties of a symmetric probability density and $b > 0$ is some bandwidth. A variant of this estimator, proposed in [Hall et al. \(1994\)](#), brings the initial estimator down to zero linearly between time moments $T_1 > 0$ and $T_2 > T_1$,

$$\hat{C}_1(t) := \begin{cases} \hat{C}_H(t), & 0 \leq t \leq T_1 \\ \hat{C}_H(T_1) (T_2 - t) (T_2 - T_1)^{-1}, & T_1 < t \leq T_2 \\ 0, & t > T_2. \end{cases} \quad (6)$$

This estimator cannot be used for long-memory time series as it vanishes at T_2 , but is a better choice when estimating a short-range dependent autocovariance function, as the estimate will go to zero. Unlike estimators (3) and (4), these estimators can be applied for arbitrary observation grids and lags.

The estimators given by (5) and (6) are not necessarily positive-definite, and thus not valid autocovariance functions. Two corrections to make them positive-definite were proposed in [Hall and Patil \(1994\)](#) and [Hall et al. \(1994\)](#).

- (i) The first correction method computes the Fourier transform of (5) to obtain the corresponding spectral density. Then, it makes any negative values in the spectral density equal to zero, and performs the inverse Fourier transform to obtain a positive-definite estimate of the autocovariance function. The modification of the spectral density that corresponds to $\hat{C}_H(\cdot)$ can be expressed as $\tilde{F}(\theta) := \max(\hat{F}(\theta), 0)$ for every frequency θ , where $\hat{F}(\cdot)$ and $\tilde{F}(\cdot)$ denote the original and modified spectral densities.
- (ii) The second correction method considers manipulating the Fourier transform again. However, it finds the smallest frequency corresponding to a negative value in the spectral density. Then, it sets all values in the spectrum to zero whose corresponding frequencies are larger than the smallest frequency. Then, the inverse Fourier transform is taken. The process of selecting the frequency and modifying the spectrum is as follows. Let $\hat{\theta} := \inf \left\{ \theta > 0 : \hat{F}(\theta) < 0 \right\}$. Then the spectral density is modified as follows, $\tilde{F}(\theta) := \hat{F}(\theta) \mathbf{1}(\theta < \hat{\theta})$, where $\mathbf{1}(A)$ is the indicator function of a set A . The drawback is that this correction can fail to produce a meaningful result if $\hat{\theta}$ is a small frequency, as the modified estimator of the autocovariance function consists of a sum of only a few cosines. Both of these correction methods can be applied to any estimator of the autocovariance function, one is not restricted to just estimators (5) and (6).

2.3 Tapered estimators

The following estimator, proposed in [Dahlhaus and Künsch \(1987\)](#), takes the edge effect into account. The edge effect reflects the situation that points closer to the boundaries of an observation region can have neighbouring observations that are outside of the study region. Thus, some dependencies may not be properly reflected, which can introduce bias. To deal with this, the estimator assigns weights

for each location depending on how close it is to the boundary, where lower weights are given to locations closer to the boundaries:

$$\hat{C}_N^a(h) := \frac{\sum_{j=1}^{N-h} (X(j) - \bar{X})(X(j+h) - \bar{X}) a((j-1/2)/N; \rho) a((j+h-1/2)/N; \rho)}{H_{2,N}(0)}, \quad (7)$$

where the normalising factor is

$$H_{2,N}(0) := \sum_{s=1}^N a((s-1/2)/N; \rho)^2,$$

$a(\cdot; \cdot)$ is a taper function on the interval $[0, 1]$ with the smoothness parameter $\rho \in (0, 1]$,

$$a(u; \rho) := \begin{cases} w(2u/\rho), & 0 \leq u < \frac{1}{2}\rho, \\ 1, & \frac{1}{2}\rho \leq u \leq \frac{1}{2}, \\ a(1-u; \rho), & \frac{1}{2} < u \leq 1, \end{cases}$$

and $w(\cdot)$ is a continuous nondecreasing function on $[0, 1]$ with $w(0) = 0$ and $w(1) = 1$. We will refer to $w(\cdot)$ as a window function. The estimator (7) is positive-definite and biased, but the bias is negligible asymptotically. This estimator is applied to observations on an integer grid.

2.4 Splines estimators

Unlike the other estimators, the following estimator does not use observations directly. It is based on an approximation of $\hat{C}(\cdot)$ by completely monotone basis functions proposed in [Choi et al. \(2013\)](#)

$$\hat{C}^B(h) := \sum_{j=1}^{m+p} \beta_j f_j^{(p-1)}(h^2), \quad (8)$$

where $\beta_j \geq 0$, $f_j^{(l)}(x) := \int_0^1 (m+1)t^x B_{j+1}^{(l)}(t) dt$, $B_j^{(l)}(\cdot)$ is the j^{th} B-spline of order l and $j = 1, 2, \dots, m+p$. The constants β_j are chosen via weighted least squares, where the objective function is

$$\sum_{i=1}^L w_i \left(\hat{C}(h_i) - \sum_{j=1}^{m+p} \beta_j f_j^{(p-1)}(h_i^2) \right)^2,$$

$\hat{C}(\cdot)$ is an autocovariance function estimator, $\{h_1, \dots, h_L\}$ is a set of lags and $\{w_1, \dots, w_L\}$ is a set of weights. [Choi et al. \(2013\)](#) uses (3) as $\hat{C}(\cdot)$, however, one can use any autocovariance estimator. For the choice of weights $w_i = (N - h_i) / (1 - \hat{C}(h_i))^2$ was proposed in [Cressie \(1985\)](#).

This estimator can calculate the estimated autocovariance at an arbitrary lag once the fitting process is done. However, the lags used during the fitting process should be chosen such that they are the same as in $\hat{C}(\cdot)$. As estimator (8) is constructed using completely monotone basis functions, and $\beta_j \geq 0$, it is nonnegative. This estimator is a positive-definite function. Further, it is also bounded from below by zero, meaning this estimator cannot be used when the autocovariance is below zero, and due to the monotonicity, it cannot be used if cyclicity is present.

2.5 Kernel correction estimators

The next estimator is simply a modification of any estimator. It was proposed to remove estimation wave artefacts ([Yaglom, 1987](#), Section 3.17). The waves are present in various estimated autocovariance functions and are more prominent as the estimation lag increases, as fewer points are available to compute the autocovariance function. This estimator also helps to reduce constant summation effects, which were discussed earlier for estimators (3) and (4). It is defined as

$$\hat{C}_T^{(a)}(h) := a_T(h) \hat{C}(h), \quad (9)$$

where $a_T(\cdot)$ is a kernel function, which approaches zero as $|h|$ increases and $\hat{C}(\cdot)$ is any autocovariance estimator ([Yaglom, 1987](#), Section 3.17). Typically, $a_T(h) := a(h/N_T)$, where N_T is some constant, usually 10% of the number of observations. If $a_T(\cdot)$ is chosen as positive-definite and $\hat{C}(\cdot)$ is positive-definite, then $\hat{C}_T^{(a)}$ will also be positive-definite.

A combination of (4) and (9) can also be used to reduce the waves in the estimate and guarantee a finite range of dependencies:

$$\widehat{C}_T^{**}(h) = a_T(h)\widehat{C}^{**}(h). \quad (10)$$

2.6 Linear shrinking

A linear shrinkage correction method, introduced by [Devlin et al. \(1975\)](#), adjusts the estimated autocorrelation matrix R through the following transformation

$$\widetilde{R} := \lambda R + (1 - \lambda)I_p, \quad (11)$$

where $\widetilde{R}$ is the shrunk autocorrelation matrix, $\lambda \in [0, 1]$ is the shrinking coefficient and I_p is the $p \times p$ identity matrix, often called the shrinkage target ([Rousseeuw and Molenberghs, 1993](#)). λ is chosen as the maximal value for which $\widetilde{R}$ remains positive-definite.

2.7 Block bootstrap

The moving block bootstrap, independently developed by [Künsch \(1989\)](#) and [Liu and Singh \(1992\)](#), allows for the resampling of time series data with dependencies without relying on any parametric assumptions ([Lahiri, 2003](#), Chapter 2.5). For a time series $X(1), \dots, X(n)$, first, construct $n - \ell + 1$ blocks of length ℓ , $\mathcal{B}_i = (X(i), \dots, X(i + \ell - 1))$, for $i = 1, \dots, n - \ell + 1$. Then, the blocks are sampled in the following way. Let $I_1, \dots, I_k$ be k independent and identically sampled values, from the discrete uniform distribution on $\{1, \dots, n - \ell + 1\}$, which are the block indices, that is $\mathcal{B}_{I_i}$. To construct a bootstrapped time series, join the randomly sampled blocks $\mathcal{B}_{I_1}^*, \dots, \mathcal{B}_{I_k}^*$, resulting in the time series $X^*(1), \dots, X^*(k\ell)$, where $*$ denotes the sampled versions of the blocks and time series ([Lahiri, 2003](#), Chapter 2.5). If $k\ell > n$, the moving block sampled time series is truncated at n .

The moving block bootstrap suffers from a boundary effect, where lesser weights are given to observations at the beginning and end of the sampled time series ([Lahiri, 2003](#), Chapter 2.7). A modified method to construct bootstrap samples is the circular bootstrap, proposed by [Politis and Romano \(1992\)](#), which addresses this issue. Instead of the time series $X(1), \dots, X(n)$ being observed on the line, it is considered to be observed on the circle. This results in the observation $X(n + k)$ being the same as $X(k)$ for $k = 1, \dots, n$. For $i = 1, \dots, n$, blocks are constructed in a similar, but circular fashion. For example, the block $\mathcal{B}_{n-\ell+2} = (X(n - \ell + 2), \dots, X(n), X(n + 1))$ is the same as $(X(n - \ell + 2), \dots, X(n), X(1))$. The procedure to construct a circular bootstrap time series is the same as for the moving block bootstrap. However, the block indices are instead sampled from $\{1, \dots, n\}$.

The block length parameter choice is flexible, although some guidelines exist, see [Hall et al. \(1995\)](#), [Politis and White \(2004\)](#) and [Arteche \(2024\)](#). For example, a block length $\ell \sim Cn^{1/k}$, $k = 3, 4, 5$, is recommended in ([Hall et al., 1995](#)). For a time series with long memory, the block bootstrap may give inconsistent results, see [Lahiri \(1993\)](#). In such cases, a larger block size should be chosen to capture the dependency structure.

In the context of autocovariance estimation in this paper, first, a block bootstrap method is used to sample multiple time series. Then, a specified autocovariance estimation method is applied to compute the estimated autocovariance for each sampled time series. The bootstrap estimate is the average of all individual estimators, and the corresponding bootstrap confidence region is constructed from the bootstrap confidence intervals at each lag t of the considered autocovariance estimate.

Table 1 summarises some properties of the estimators presented in this section, including their input requirements, assumptions of grid regularity, positive-definiteness, computational efficiency, suitability for long-memory processes, and disadvantages that applied users should be aware of.

Additional theoretical results and formulas will be introduced later when needed.

3 Package structure and function overview

This section provides a high-level overview of the package, including the types of functions and their relations, function structures, and the parameters of selected functions.

3.1 Package interface and S3 objects overview

The package [CovEsts](#) consists of several functions to compute the autocovariance estimators discussed in the previous section. For the [CovEsts](#) package, we deliberately aimed to keep the number of package dependencies minimal, so it only uses packages that come with base R, [stats](#) for `acf`, `optim`

Estimator	Input	Grid	P.d.	Computational efficiency	Suitable for long memory	Disadvantages
Standard (3)	Data	Regular	No	Fast	No	Constant Cov.est. summation
Standard (4)	Data	Regular	Yes	Fast	No	Constant Cov.est. summation
Hall's (5)	Data	Both	No	Slow/unstable	Yes	Slow for long time series
Hall's Truncated (6)	Data	Both	No	Slow/unstable	No	Slow for long time series
Hall's Correction (i)	Cov.est.	Regular	Yes	Fast	No	Inflates est.variance
Hall's Correction (ii)	Cov.est.	Regular	Yes	Fast	No	May degenerate to few cosines
Tapered (7)	Data	Regular	Yes	Fast	Yes	Tapering may cause bias
Splines (8)	Data & Cov.est.	From Cov.est.	Yes	Slow/unstable	No	Cov.est. is always nonnegative
Kernel Correction (9)	Cov.est.	Regular	No	Fast	No	Removes long memory nature
Kernel Correction (10)	Cov.est.	Regular	No	Fast	No	Removes long memory nature
Shrinkage Correction (11)	Data	Regular	No	From Cov.est.	Target dependent	High bias if wrong target
Moving block bootstrap average	Data	From Cov.est.	No	From Cov.est.	From Cov.est.	Weighting boundary effect
Circular bootstrap average	Data	From Cov.est.	No	From Cov.est.	From Cov.est.	Duplicated information, Edge jumps

Table 1: Summary of selected properties of the estimators

and `fft`, **graphics** for legend, lines and polygon, and **parallel** for `parLapply`, `makePSOCKcluster`, `clusterExport` and `stopCluster`. This was done to make the package self-contained.

Figure 1 groups the main functions depending on their functionality and relations to the autocovariance estimation functions. The main group, *Autocovariance Estimators*, consists of the autocovariance function estimators discussed in Section 2, whilst the remaining groups are used for intermediary and complementary calculations, except those that compare autocovariance estimates. Note that this figure does not include all functions in the package. In particular, non-user-facing functions are omitted.

In the main group, there are nine functions that estimate the autocovariance function or its modifications. As the parameters are often shared or similar between the estimator functions, to avoid repetition, only one set of parameters is shown in Table 2. As can be seen from the table, the functions are vectorised.

The estimator functions are called through a single function, accepting a time series as the first argument, and additional arguments depending on the estimation method. As the first argument is the time series, the pipe operator `|>` can be used for all estimators, for example, `x |> truncated_est` (other arguments). All autocovariance estimator functions return a **CovEsts** S3 object. These S3 objects are lists containing four elements,

- `acf`: the estimated autocovariance/autocorrelation/partial autocorrelation values,
- `lags`: the lag indices used to compute the estimates on,
- `est_type`: the type of estimate, namely 'autocorrelation', 'autocovariance' or 'partial',
- `est_used`: the estimator function used.

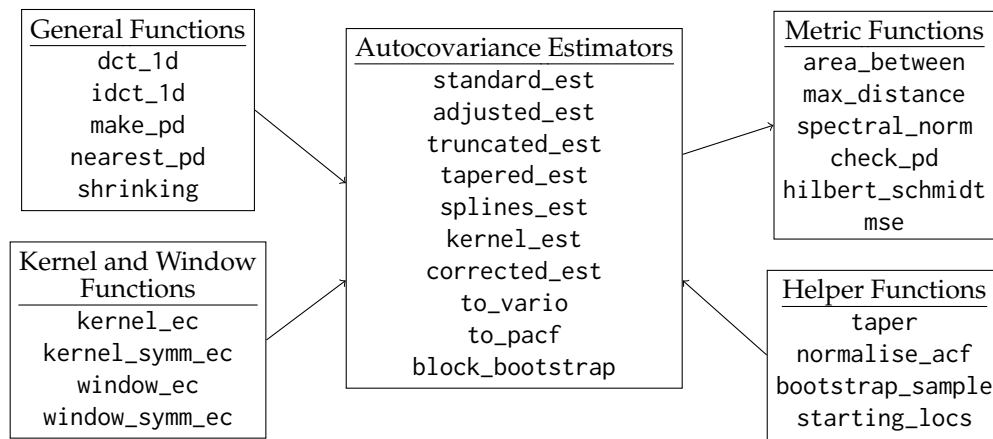


Figure 1: Types and links between main package functions.

Parameter	Explanation
X	A vector representing observed values of the time series.
x	A vector of lags.
t	The arguments at which the autocovariance function is calculated.
T1	The first truncation point, $T_1 > 0$.
T2	The second truncation point, $T_2 > T_1 > 0$.
b	Bandwidth parameter, greater than 0.
kernel_name	The name of the symmetric kernel function to be used. Possible values are: gaussian, wave, rational_quadratic, and bessel_j. Alternatively, a custom kernel function can be provided.
kernel_params	A vector of parameters of the kernel function.
custom_kernel	If a custom kernel is to be used or not. Defaults to FALSE.
pd	Whether a positive-definite estimate should be used. Defaults to TRUE.
type	Compute either the 'autocovariance' or 'autocorrelation'. Defaults to 'autocovariance'.
meanX	The average value of X. Defaults to mean(X).
parallel	Whether or not the computations should be done in parallel or not. Defaults to FALSE.
ncores	The number of cores to be used in the parallel computations. Defaults to the number cores - 1 (threads if hyperthreading is available), calculated from <code>parallel::detectCores() - 1</code> .
cl_export	A vector of any additional functions or variables to export for parallel computations. This may be required if estimator is not within the package. Defaults to NULL.
cl	An optional cluster object created by <code>parallel::makeCluster</code> . Defaults to NULL, which creates a temporary PSOCK cluster.
Return	A <code>CovEsts</code> S3 object containing the truncated kernel regression estimates, the lags, the estimated type and the estimator used.

Table 2: Example of parameters for `truncated_est`

In addition to the **CovEsts** S3 objects, there are also the **VarioEsts** and **BootEsts** S3 objects for variograms computed using `to_vario` and for block bootstrap estimates. The **VarioEsts** object has the following elements,

- `acf`: the estimated variogram values,
- `lags`: the lag indices used to compute the estimates on,
- `est_type`: 'to_vario'.

The **BootEsts** has several more elements,

- `avg_acf`: the average bootstrap autocovariance/autocorrelation function estimate,

- `lags`: the lag indices used to compute the estimates on,
- `acf_orig`: the nonbootstrapped autocovariance/autocorrelation estimate,
- `acf_mat`: a matrix of all of the bootstrap estimates,
- `conf_lower`: the lower bounds for the estimated pointwise confidence interval,
- `conf_upper`: the upper bounds for the estimated pointwise confidence interval,
- `est_type`: the type of estimate, namely 'autocorrelation', 'autocovariance',
- `est_used`: the estimator function used,
- `boot_type` is either 'moving' or 'circular' depending on the type of block bootstrap used,
- `alpha` is the α value used to compute the confidence intervals.

All these S3 objects have `print`, `plot` and `lines` methods. The `lines` method allows easy comparison between the obtained estimates. For **CovEsts** and **VarioEsts** it overlays a line directly on the plot. The `lines` method for **BootEsts** plots a single averaged estimate across all bootstrap samples, rather than displaying the individual estimates corresponding to each bootstrap sample.

S3 objects were chosen over S4 and R6 due to their simplicity, both internally and in terms of user-facing flexibility. Namely, they allow plotting and printing the obtained estimates as ordinary R objects, without requiring inspection of the underlying list structure. This is particularly useful when plotting multiple results together, enabling an easy comparison. Further, if a user wishes to extract the estimated values or any other elements, this can be done straightforwardly.

3.2 Main estimators

To compute the standard estimator (3), one uses the function

```
standard_est(X, pd = TRUE, maxLag = length(X) - 1, x = 0:length(X),
            type = c("autocovariance", "autocorrelation"), meanX = mean(X)) ,
```

where $\text{maxLag} \leq N - 1$ is the maximum lag for which the estimated autocovariance function is computed and `x` are the indices for which the time series was observed on. To obtain the estimate (4), `pd = TRUE` should be used instead. As can be seen, this function assumes the basic case of equally spaced observations on a consecutive grid of points, such as the integers or those with a constant difference of the observation period. This function, along with all other estimator functions with the "`_est`" suffix, returns a **CovEsts** S3 object.

To provide examples of other function calls, we consider estimators (5) and (6) that can be computed through the commands

```
adjusted_est(X, x, t, b,
            kernel_name = c("gaussian", "wave", "rational_quadratic", "bessel_j"),
            kernel_params=c(), pd = TRUE, type = c("autocovariance", "autocorrelation"),
            meanX = mean(X), custom_kernel = FALSE, parallel = FALSE,
            ncores = parallel::detectCores() - 1, cl_export = NULL, cl = NULL)

truncated_est(X, x, t, T1, T2, b,
            kernel_name = c("gaussian", "wave", "rational_quadratic", "bessel_j"),
            kernel_params = c(), pd = TRUE, type = c("autocovariance", "autocorrelation"),
            meanX = mean(X), custom_kernel = FALSE, parallel = FALSE,
            ncores = parallel::detectCores() - 1, cl_export = NULL, cl = NULL) .
```

These estimators can use sampled values on an arbitrary grid and set of lags. For the parameters for these two estimators, refer to Table 2.

As several estimators include options for kernel smoothing and adjustment, a list of possible symmetric kernels can be found in the discussion of the kernels and window functions later in this section and in Table 3. For the considered estimators, if a custom kernel is chosen, it must have the properties of a symmetric probability density.

These two functions also allow for parallelism. The other estimators either do not support parallelisation or do not benefit from it, as the call-based R functions are sufficiently fast. Parallelisation is done over the argument `t`, which represents the lags. The user can pass their own cluster, otherwise, a temporary PSOCK cluster is created. This can be useful if the user is on an operating system that supports `fork()`, allowing a cluster created using `parallel::makeForkCluster` to be used instead.

Another example, the splines estimator, defined by (8), is called as

```
splines_est(X, x, estCov, p, m, maxLag = length(X) - 1,
            type = c("autocovariance", "autocorrelation"), initial_pars = c(),
            control = list('maxit' = 1000)) ,
```

where `estCov` is an estimated autocovariance function used during the fitting process, which can either be a numeric vector or a **CovEsts** S3 object generated by one of the other autocovariance function estimators. The parameter `p` is the order of the splines, `m` is the number of nonboundary knots, `initial_pars` and `control` are optional parameters used during the optimisation process (see **stats::optim** for `control`), where `initial_pars` is an $m + p$ vector whose default values are 0.5. This estimator can fail to produce an output if neither optimisation algorithm converges. The optimisation algorithms used are Nelder-Mead and L-BFGS-B.

Many of the estimators considered in this package require non-trivial multistep calculations. As an example, the high-level structure of the function `splines_est` and the steps required to compute it are shown in Figure 2. Arrows represent function calls to other functions for obtaining values from them. The function calls for `optim` are repeated as it is called twice using different optimisation algorithms, where the one with the lowest error is selected. A whole number placed near an arrow indicates the corresponding step of the computation in which the function operates. The number after the whole number is the order in which a function is called by the level preceding it.

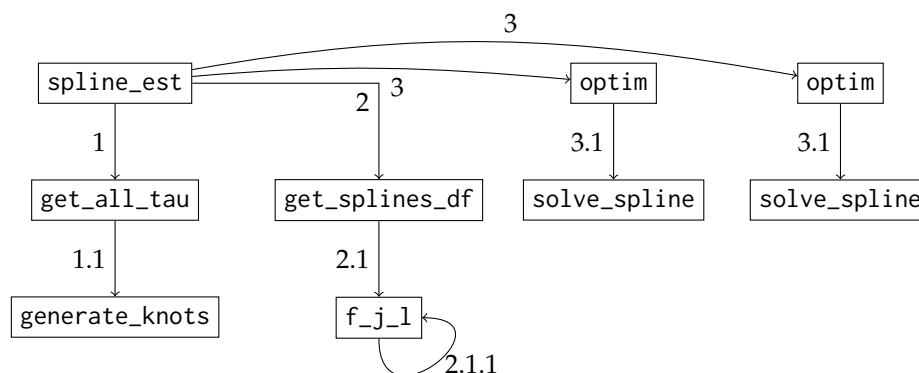


Figure 2: Example of a high-level structure of `splines_est` and the functions it calls.

The semivariogram estimate can be computed from an autocovariance function estimate `estCov`, either a numeric vector or a **CovEsts** S3 object, as per (1), using the command

```
to_vario(estCov) .
```

The partial autocorrelation for any estimated autocovariance or autocorrelation function `estCov`, either a numeric vector or a **CovEsts** S3 object, can be computed using

```
to_pacf(estCov) .
```

Block bootstrap can be performed using the following command

```
block_bootstrap(X, maxLag, x = 0:length(X), n_bootstrap = 100,
               l = ceiling(length(X)^(1/3)), estimator = standard_est,
               type = c("autocovariance", "autocorrelation"), alpha = 0.05,
               boot_type = c("moving", "circular"), parallel = FALSE,
               ncores = parallel::detectCores() - 1, cl_export = NULL,
               boot_seed = NULL, cl = NULL, ...) ,
```

where `n_bootstrap` is the number of times to run block bootstrap, `l` is the block length, `estimator` is the estimator function one wishes to use, where `estimator` (4) is chosen by default, `alpha` is the level of significance for the pointwise confidence intervals, `boot_type` is either 'moving' or 'circular'. This function supports parallelism, see Table 2 for the relevant parameters. "..." are any other parameters that are to be passed into `estimator`. The parallelism is performed over `n_bootstrap` iterations.

3.3 Kernels, windows and associated estimators

Several standard kernels, symmetric kernels and window functions available in the literature are realised in the package. We also provided the option for user-defined kernels, symmetric kernels,

window functions and symmetric window functions. Tables 3 and 4 provide lists of the main available kernels and window functions in the package. The symmetric kernels and symmetric window functions will be mentioned only briefly, as they are modifications of the main kernels.

Kernel Name	Equation $a(x; \theta, \nu, d, \alpha, \beta)$	Constraints
gaussian	$\exp(-x^2/\theta)$	
exponential	$\exp(-x/\theta)$	
wave	$\begin{cases} \frac{\theta}{x} \sin(x/\theta), & x \neq 0 \\ 0, & x = 0 \end{cases}$	
rational_quadratic	$1 - x^2/(x^2 + \theta)$	
spherical	$\begin{cases} 1 - 3x/(2\theta) + (x/\theta)^3/2, & x < \theta \\ 0, & \text{otherwise} \end{cases}$	
circular	$\begin{cases} \frac{2}{\pi} \arccos(x/\theta) - \frac{2x}{\pi\theta} \sqrt{1 - (x/\theta)^2}, & x < \theta \\ 0, & \text{otherwise} \end{cases}$	
matern	$(\sqrt{2\nu}x/\theta)^{\nu} (2^{\nu-1}\Gamma(\nu))^{-1} K_{\nu}(\sqrt{2\nu}x/\theta)$	$\nu > 0$
bessel_j	$2^{\nu}\Gamma(\nu+1)J_{\nu}(x/\theta)(x/\theta)^{-\nu}$	$\nu \geq d/2 - 1$
cauchy	$(1 + (x/\theta)^{\alpha})^{-(\beta/\alpha)}$	$\alpha \in (0, 2], \beta \geq 0$

Table 3: List of main kernels

The kernel and window functions are called through

```
kernel_ec(x, name = c("gaussian", "exponential", "wave", "rational_quadratic",
  "spherical", "circular", "bessel_j", "matern", "cauchy"), params=c(1)) ,
kernel_symm_ec(x, name = c("gaussian", "wave", "rational_quadratic", "bessel_j"),
  params=c(1)) ,
window_ec(x, name = c("tukey", "triangular", "sine", "power_sine", "blackman",
  "hann_poisson", "welch"), params=c(1)) ,
window_symm_ec(x, name = c("tukey", "triangular", "sine", "power_sine", "blackman",
  "hann_poisson", "welch"), params=c(1)) ,
```

where the parameters are given in the formulas in Tables 3 and 4.

The symmetric kernels (gaussian, wave, rational_quadratic and bessel_j) are symmetric versions of the kernels in Table 3, but they are standardised so that their area is 1. For the symmetric window functions, all options are the same as the window functions, however, they are defined as $1 - w(|x|; a)$ for $x \in [-1, 1]$ and 0 elsewhere, where $w(x; a)$ is a window function. For kernel_ec, its argument x is nonnegative, for kernel_symm_ec and window_symm_ec, their arguments x can be negative, and for window_ec, it must be within 0 and 1.

Kernel Name	Equation $w(x; a)$	Constraints
tukey	$\frac{1}{2} - \frac{1}{2} \cos(\pi x)$	
triangular	x	
sine	$\sin(\pi x/2)$	
power_sine	$\sin^a(\pi x/2)$	$a > 0$
blackman	$((1-a)/2) - \frac{1}{2} \cos(\pi x) + \frac{a}{2} \cos(2\pi x)$	$ a \leq 0.25$
hann_poisson	$(1/2)(1 - \cos(\pi x)) \exp(-(a 1-x))$	$a \in \mathbb{R}$
welch	$1 - (x-1)^2$	

Table 4: List of main window functions, where $x \in [0, 1]$

The package offers several functions to deal with the potential issues in the estimated autocovariance functions, which were discussed in Section 2. These corrections can be applied to any given autocovariance estimate. A general method to correct any estimator is provided by (9) and is called as

```
kernel_est(estCov, kernel_name = c("gaussian", "exponential", "wave",
  "rational_quadratic", "spherical", "circular", "bessel_j", "matern", "cauchy"),
kernel_params = c(), N_T = 0.1 * length(estCov), maxLag = length(estCov) - 1,
```

```
x = 0:length(X), type = c("autocovariance", "autocorrelation"),
custom_kernel = FALSE) ,
```

where `kernel_name` and `kernel_params` are given in Table 5, `N_T` is the rate at which the kernel function vanishes and is recommended to be of order $0.1N$ when considering all lags (Yaglom, 1987, Section 3.17). As with `splines_est`, `estCov` can be a numeric vector or a **CovEsts** S3 object, obtained from another autocovariance estimator.

Parameter	Explanation
<code>x</code>	A vector or matrix of arguments of at least length 1.
<code>name</code>	The name of the kernel. Options are: gaussian, exponential, wave, rational_quadratic, spherical, circular, <code>bessel_j</code> , <code>matern</code> , and <code>cauchy</code> .
<code>params</code>	A vector of parameters for the kernel. See Table 3 for the position of the parameters. All kernels will have a scale parameter as the first value in the vector.
Return	A vector of values.

Table 5: Example of parameters for kernel functions

In addition to this, kernel-corrected versions of (3) and (4) are provided, called by

```
corrected_est(X, kernel_name = c("gaussian", "exponential", "wave",
    "rational_quadratic", "spherical", "circular", "bessel_j", "matern", "cauchy"),
kernel_params = c(), N_T = 0.1 * length(X), pd = TRUE, maxLag = length(X) - 1,
x = 0:(maxLag - 1), type = c("autocovariance", "autocorrelation"),
meanX = mean(X), custom_kernel = FALSE) ,
```

where the parameters are the same as for `kernel_corrected_est` with the addition of having a positive-definite estimator. If a custom kernel is used that is not positive-definite, then the estimator will no longer be positive-definite. For both kernel correction estimators, unlike the kernel regression estimators (recall estimators (5) and (6)), the custom kernel is not required to have the properties of a symmetric probability density.

The function that computes the taper corrected estimator (7) is called as

```
tapered_est(X, rho, window_name = c("tukey", "triangular", "sine", "power_sine",
    "blackman", "hann_poisson", "welch"), window_params = c(1),
maxLag = length(X) - 1, x = 0:length(X),
type = c("autocovariance", "autocorrelation"), meanX = mean(X),
custom_window = FALSE) ,
```

where $\rho \in (0, 1]$ is a scale parameter, `window_ec` and `window_params` are given in Table 4, and the option `custom_window` serves the same purpose as `custom_kernel`. For the custom window function, it should be a nondecreasing function on $[0, 1]$ with $w(0) = 0$ and $w(1) = 1$.

3.4 Estimator adjustment/modification

The remaining groups in Figure 1 fall into two categories, helper functions and *Metric Functions*. Helper functions are used during autocovariance function estimation, whilst metric functions are used to compare estimated autocovariance functions, which will be discussed in the next subsection.

The helper functions are further split into two categories, *Helper Functions* and *General Functions*. The discussion of the *Helper Functions* is omitted as these functions exist only for auxiliary calculations. *General Functions* can also be useful for other applications, for example, the forward and inverse type-II discrete cosine transforms can be called the following functions

```
dct_1d(X) ,
idct_1d(X) ,
```

where `X` is a vector of values for which the discrete cosine transform is being computed.

Also, one can make any estimator positive-definite by using the positive-definite corrections from Section 2. In the package, the first correction method (see Section 2 (i)) can be called by the function

```
make_pd(x, method.1 = TRUE) ,
```


where x can be either a numeric vector of estimated autocovariance values or a **CovEsts** S3 object generated by one of the other autocovariance functions. If the second correction (see Section 2 (ii)) is to be done, `method.1 = FALSE` is used instead.

Another method to make an estimated autocorrelation function positive-definite is to use linear shrinking, recall (11), which can be computed using

```
shrinking(estCov, return_matrix = FALSE, target = NULL) ,
```

where `estCov` is an estimated autocovariance or autocorrelation function, either a numeric vector or a **CovEst** S3 object, `return_matrix` is a boolean, determining if the shrunken matrix is to be returned or not. The default value returns a shrunken autocorrelation function instead of an autocorrelation matrix. `target` is the target matrix, which defaults to the identity matrix.

For a matrix A that is not necessarily positive-definite, the following procedure can be used to find the nearest positive-definite matrix (Higham, 1988, Th. 2.1). First, a new matrix is constructed, $B = (A + A^T)$ and then compute the polar decomposition $B = UH$, where U is an orthogonal matrix and H is a positive-definite matrix. The nearest positive-definite matrix to A is then $A_F = (B + H) / 2$. The function that computes this matrix is called using

```
nearest_pd(X, return_matrix = FALSE) ,
```

where X is either a numeric vector, a numeric square matrix or a **CovEsts** S3 object. If a numeric vector or **CovEsts** is supplied, a matrix similar to (12) will be constructed where $D(\cdot)$ is replaced by X or the autocovariance values within the **CovEsts** object.

3.5 Metrics

Six *Metric Functions* are provided in the package to compare different autocovariance estimates.

For two continuous autocovariance function estimates, $\hat{C}_1(\cdot)$ and $\hat{C}_2(\cdot)$, the area between them can be expressed as $\int_0^{h_n} |D(h)| dh$, where $D(h) = \hat{C}_1(h) - \hat{C}_2(h)$ and h_n denotes the maximum lag at which the functions are estimated up to. However, as the estimates are computed on a discrete grid, the integral is approximated using the trapezoid rule, that is,

$$\int_0^{h_n} |D(h)| dh \approx \sum_{i=1}^n \frac{|D(h_{i-1})| + |D(h_i)|}{2} (h_i - h_{i-1}),$$

where the estimates are given over the set of lags $\{h_0, h_1, \dots, h_{N-1}, h_n\}$, and h_0 is assumed to be 0.

The area between two estimated autocovariance functions can be calculated as

```
area_between(estCov1, estCov2, lags = c(), plot = FALSE) ,
```

where `estCov1` and `estCov2` are estimated values of autocovariance functions given over the same set of lags, either numeric vectors or **CovEsts** S3 objects. The parameter `lags` is optional and defaults to a vector starting at 0, increasing by 1 until `length(estCov1)`. Another optional parameter, `plot`, determines whether a plot should be created showing the area between the two estimated autocovariance functions (for example, see Figure 9).

The maximum distance can be expressed as follows, $\sup_{h \in [0, h_n]} |D(h)|$, where $\sup$ is replaced by $\max$ over the set of lags in the discrete case.

The spectral norm is the largest eigenvalue of the matrix

$$\begin{bmatrix} D(h_0) & D(h_1) & \cdots & D(h_{n-1}) & D(h_n) \\ D(h_1) & D(h_0) & \cdots & D(h_{n-2}) & D(h_{n-1}) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ D(h_{n-1}) & D(h_{n-2}) & \cdots & D(h_0) & D(h_1) \\ D(h_n) & D(h_{n-1}) & \cdots & D(h_1) & D(h_0) \end{bmatrix}. \quad (12)$$

The maximum vertical distance and spectral norm have similar arguments to `area_between` and are called as

```
max_distance(est1, est2, lags = c(), plot = FALSE) ,
spectral_norm(est1, est2) ,
```

where the plot for the maximum distance shows the magnitude of the vertical distances for each lag, see an example in Figure 10. Like with `area_between`, `estCov1` and `estCov2` are estimated values of autocovariance functions given over the same set of lags, either numeric vectors or **CovEsts** S3 objects, which is the case for all metric functions.

The Hilbert-Schmidt metric for the matrix (12) is defined as $\sqrt{\sum_{i,j=1}^n d_{i,j}^2}$ and can be called as `hilbert_schmidt(est1, est2)`.

The closely related mean-square error/difference (MSE) between two estimated autocovariance functions is given by $n^{-1} \sum_{i=0}^n D(h_i)^2$. If the estimate is to be compared against a theoretical model, $\hat{C}_2(\cdot)$ can be replaced with the corresponding theoretical autocovariance function. The MSE can be called as

`mse(est1, est2)`.

To check that the autocovariance function estimate is positive-definite, first one can construct a covariance matrix that is similar to (12), where $\hat{C}(\cdot)$ replaces $D(\cdot)$. Then, if all eigenvalues of this matrix are positive, the estimate is positive-definite. This can be checked by using the function

`check_pd(est)`,

where `est` is a vector of numeric values of an estimated autocovariance function or a **CovEsts** S3 object.

4 Examples

This section provides an example utilisation of some main functions and computational time and memory usage analysis. It will demonstrate applications of the estimators to a simulated data set, the sunspot.year data set in the **datasets** package and unemployment data from the US Bureau of Labor Statistics.

The typical usage of the kernel and window functions, defined only for nonnegative values of their argument `x` (in the `window_ec` case, it is further restricted to the interval $[0, 1]$) is as follows:

```
library(CovEsts)
x <- c(0.2, 0.4, 0.6)
theta <- 0.9
kernel_ec(x, "gaussian", c(theta))
[1] 0.9565287 0.8371284 0.6703200
nu <- 1
dim <- 1
kernel_ec(x, "bessel_j", c(theta, nu, dim))
[1] 0.9938398 0.9755110 0.9454638

window_ec(x, "tukey")
[1] 0.0954915 0.3454915 0.6545085
window_ec(x, "blackman", c(0.16))
[1] 0.04021286 0.20077014 0.50978714
```

When using the `bessel_j` option, the dimension `dim` is passed, and the function additionally verifies that the kernel is valid for the specified dimension and parameters. For the Blackman window function, one should use $|a| \leq 0.25$ to ensure that it is nondecreasing on $[0, 1]$.

`kernel_symm_ec` and `window_symm_ec` are called in a similar way to `kernel_ec` and `window_ec`, but can be applied to negative values of `x`, for example,

```
x <- c(-0.4, -0.2, 0, 0.2, 0.4)
kernel_symm_ec(x, "gaussian", c(theta))
[1] 0.4978470 0.5688553 0.5947080 0.5688553 0.4978470

window_symm_ec(x, "blackman", c(0.16))
[1] 0.7992299 0.9597871 1.0000000 0.9597871 0.7992299
```

Figures 3, 4, 5 and 6 plot examples of the main available kernels, symmetric kernels, window functions, and symmetric window functions, respectively.

4.1 Example 1

To illustrate the application of the estimators and their accuracy, we will consider a simulated Gaussian time series on a uniform grid on $[0, 40]$ with a spacing of 0.02, having a short-range dependent Gaussian autocovariance model, $\exp(-x^2)$. We will only estimate the autocovariance function to a lag of 5, to use the recommended 10-20% of total samples (Yaglom, 1987, Section 3.17).

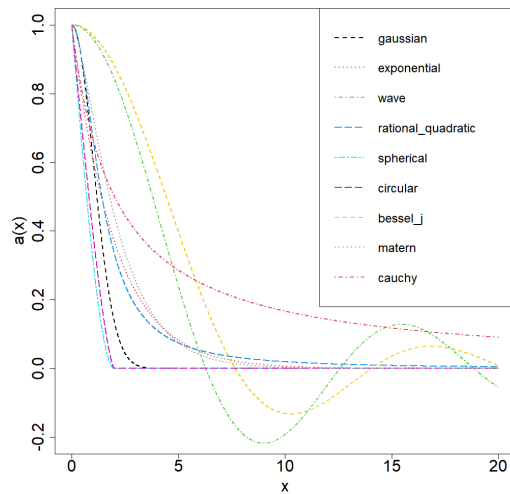


Figure 3: Examples of main available kernels where $\theta = 2, \nu = d = \alpha = \beta = 1$.

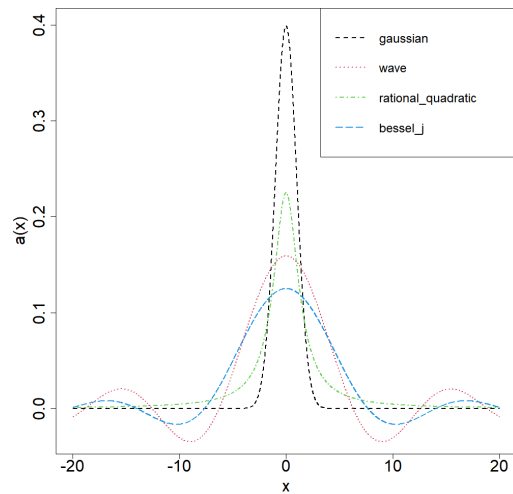


Figure 4: Examples of main available symmetric kernels where $\theta = 2, \nu = d = \alpha = \beta = 1$.

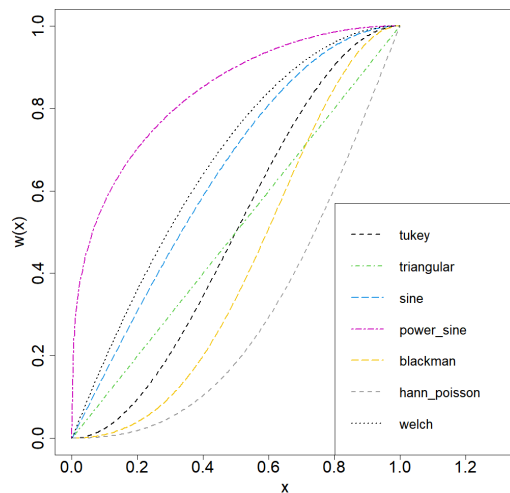


Figure 5: Examples of main available window functions where $a = 0.3, 0.16, 1$ for the power sine, Blackman and Hann-Poisson window functions, respectively.

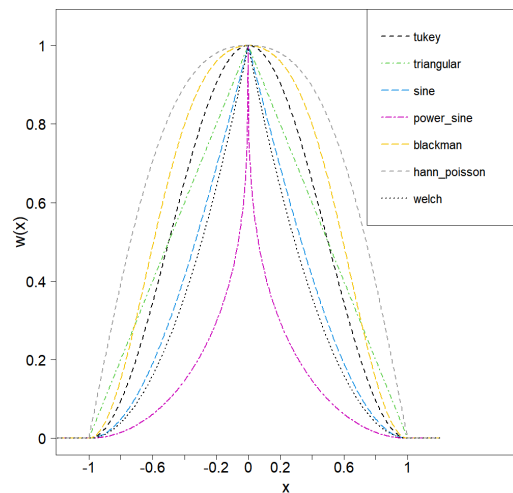


Figure 6: Examples of main available symmetric window functions where $a = 0.3, 0.16, 1$ for the power sine, Blackman and Hann-Poisson window functions, respectively.

The realisations were generated using the approach based on eigendecomposition of the autocovariance matrix. Below is the code to simulate the process and obtain the estimates based on the realisation in Figure 7, which are then plotted in Figure 8.

```
set.seed(135)
N <- 2001
x <- seq(0, 40, length.out = N)
Z <- rnorm(N)
dist_mat <- abs(outer(x, x, '-'))
cov_mat <- exp(-(dist_mat^2))
eig <- eigen(cov_mat)
X <- as.vector((eig$vectors %*% sqrt(diag(zapsmall(eig$values)))) %*% Z)

maxLag <- 251
t <- x[1:maxLag]

# standard estimators
Cs <- standard_est(X, maxLag = maxLag - 1, pd = FALSE, x = x)
Css <- standard_est(X, maxLag = maxLag - 1, pd = TRUE, x = x,
  type = "autocorrelation")
```

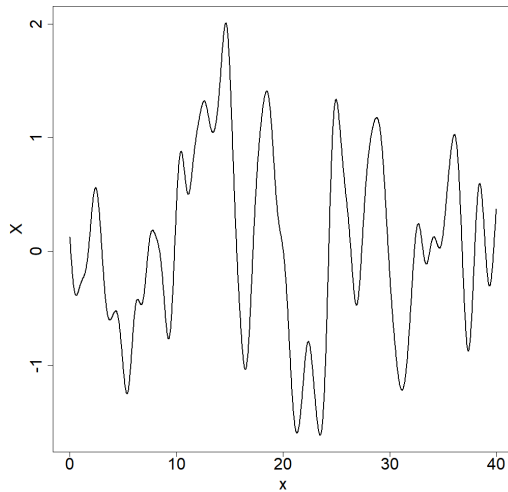


Figure 7: Realisation of short-range dependent Gaussian time series.

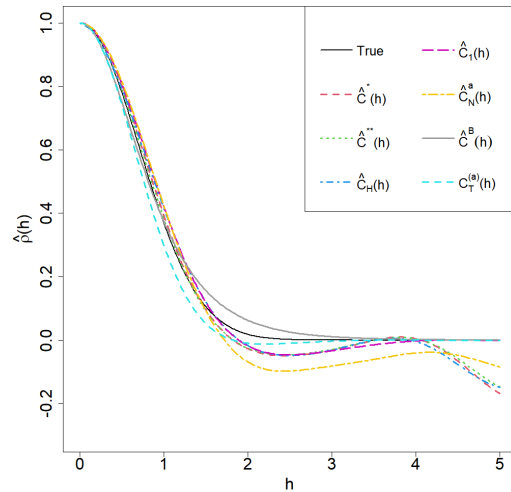


Figure 8: Estimated autocorrelation functions for a short-range dependent time series.

```
# Hall's estimators
hall_1 <- adjusted_est(X, x, t, 0.1, "gaussian", type = "autocorrelation")
hall_2 <- truncated_est(X, x, t, 3, 4, 0.1, "gaussian", type = "autocorrelation")

# tapered
tapered <- tapered_est(X, 1, "tukey", maxLag = maxLag - 1, x = x,
  type = "autocorrelation")

# splines
splines <- splines_est(X, x, Cs, 3, 2, maxLag = maxLag - 1, type = "autocorrelation")
Cs <- normalise_acf(Cs)

# Correction
corrected <- corrected_est(X, "gaussian", N_T=5*length(X), maxLag = maxLag - 1, x = x,
  type = "autocorrelation")

# Plot
par(mar=c(4,5.25,0.25,0.25)+.1)
plot(x[1:maxLag], exp(-x[1:maxLag]^2), type='l', lwd=2, ylim=c(-0.3, 1),
  xlab=expression(h), ylab=expression(hat(rho)*'(h)'), cex.axis=2,
  cex.lab=2)
lines(Cs, lwd=3, lty=2, col=2)
lines(Css, lwd=3, lty=3, col=3)
lines(hall_1, lwd=3, lty=4, col=4)
lines(hall_2, lwd=3, lty=5, col=6)
lines(tapered, lwd=3, lty=6, col=7)
lines(splines, lwd=3, lty=7, col=8)
lines(corrected, lwd=3, lty=8, col=13)

legend('topright', c('True', expression(hat('C')^'*'*(h)'),
  expression(hat('C')^'*'*(h)'), expression(hat('C')[H]*'(h)'),
  expression(hat('C')[1]*'(h)'), expression(hat('C')[N]^'a'*(h)'),
  expression(hat('C')^'B'*(h)'), expression('C'[T]^'(a)'*(h)'),
  col=c(1, 2, 3, 4, 6, 7, 8, 13), lty=c(1, 2, 3, 4, 5, 6, 7, 8),
  lwd=c(2, rep(3, 7)), y.intersp=1, cex=2, ncol = 2)
```

Whilst for short distances, the majority of the estimated autocorrelation functions are consistent and adequately reflect the true autocorrelation function, it is clear that for increasing distances the waves start to overwhelm the estimators, seen in Figure 8. The waves are eliminated when applying the estimators $\hat{C}_1(h)$ and $C_T^{(a)}(h)$, given by (6) and (10), respectively.

For this and the following examples, we consider the estimated autocorrelation functions instead of the estimated autocovariance functions to allow for easier comparison.

Figure 9 plots the area (in grey) between estimators given by (5) (red) and (6) (green). As the green curve is forced to zero after $h = 4$, the area between the two estimates increases after that point. The estimated area between them is approximately 0.061. This difference can also be seen in the plot of distances between the two estimates in Figure 10. For $h < 3$, the estimates are rather close. However, the distance starts to increase after this point dramatically. The maximum distance between the two estimates is 0.1. These values and plots can be produced as follows:

```
area_between(hall_1, hall_2, plot=T)
max_distance(hall_1, hall_2, plot=T)
```

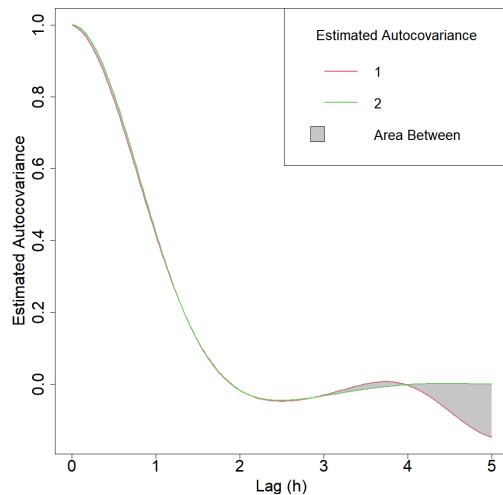


Figure 9: Area between estimated autocovariance functions.

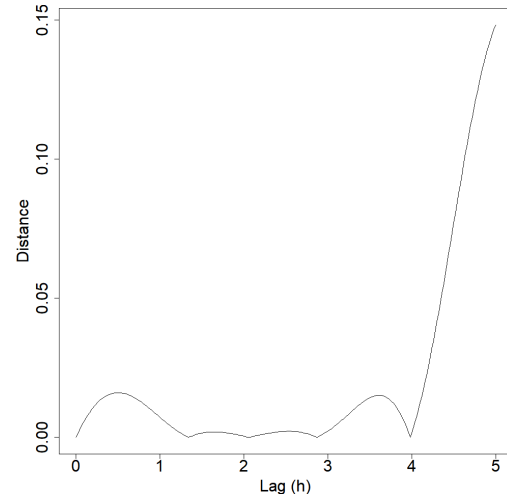


Figure 10: Distances between estimated autocovariance functions.

To compute the moving block and circular bootstrap for estimator (4), one can use the following commands

```
plot(block_bootstrap(X, maxLag, x, l = maxLag), ylim=c(-0.25, 1), cex.axis=2, cex.lab=2)
plot(block_bootstrap(X, maxLag, x, l = maxLag, boot_type = 'circular'),
     ylim=c(-0.25, 1), cex.axis=2, cex.lab=2)
```

Figures 11 and 12 show graphical outputs from the `block_bootstrap` function, where the autocovariance functions are plotted. The solid black line is the estimated autocovariance function, the red dashed line is the average bootstrap autocovariance function, and the grey shaded bootstrap 95% confidence region is calculated pointwise for each lag. Both bootstrap estimates give similar results, with the bootstrap estimate lessening constant summation effects, compared to the original estimate that dips after lag 4.

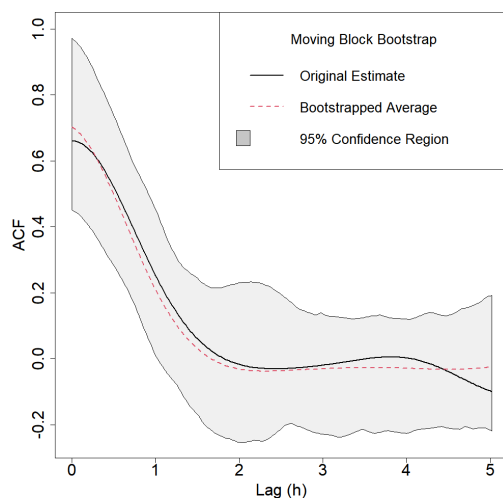


Figure 11: Moving block bootstrap estimates and confidence region.

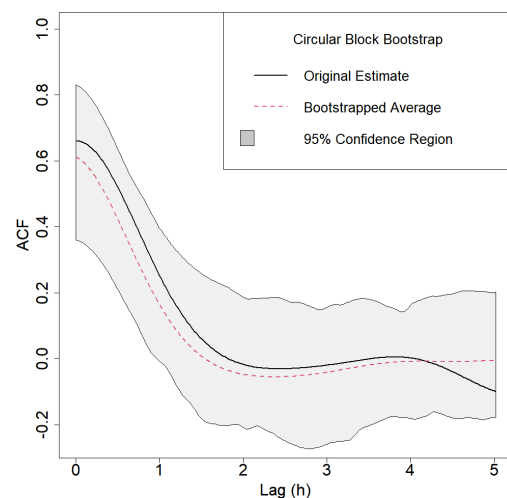


Figure 12: Circular block bootstrap estimates and confidence region.

4.2 Example 2

This example uses the classical yearly sunspot count data from 1700 to 1988, `sunspot.year`, available in the **datasets** R package. This data gives the number of observed yearly sunspots, seen in Figure 13. It has been shown that the number of sunspots exhibits long-range dependent and cyclic behaviours, see Gil-Alana (2009) and Hu et al. (2009), with an approximate 11-year cycle. So, one should expect to see periodicity in the autocovariance estimate at multiples of lag 11. The example demonstrates that estimator (6) has potential issues as its Fourier transform at the corresponding second nonzero frequency takes a negative value. So, only one nonzero frequency can be used to obtain the corrected estimates. Thus, this positive-definite adjustment will not produce any useful result and is not reported. Instead, it is replaced with the non-positive-definite version of estimator (5), denoted by $\hat{C}_H^1(h)$ in the legend of Figure 15. The splines estimator will also be omitted as it does not capture the cyclicity in the autocovariance function.

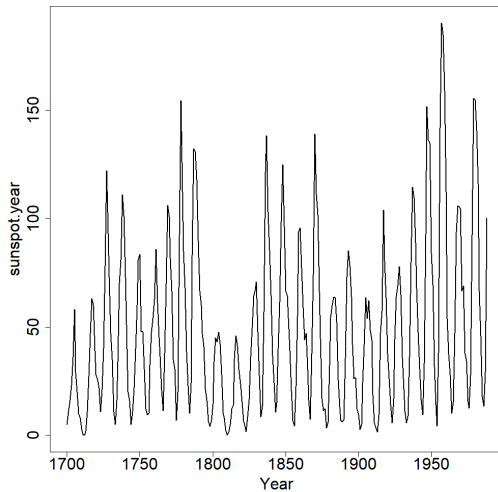


Figure 13: Yearly sunspots count data, from 1700 to 1988.

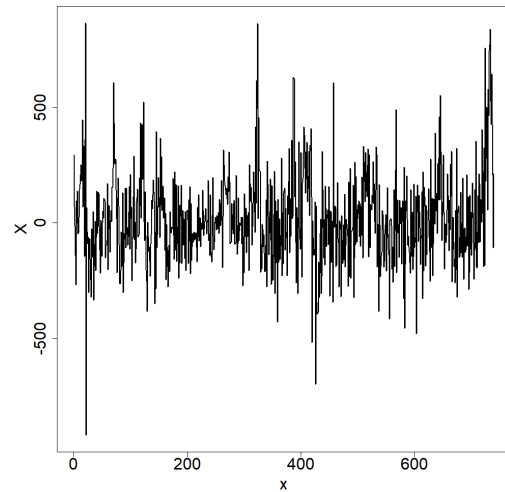


Figure 14: Increments of US unemployment count data.

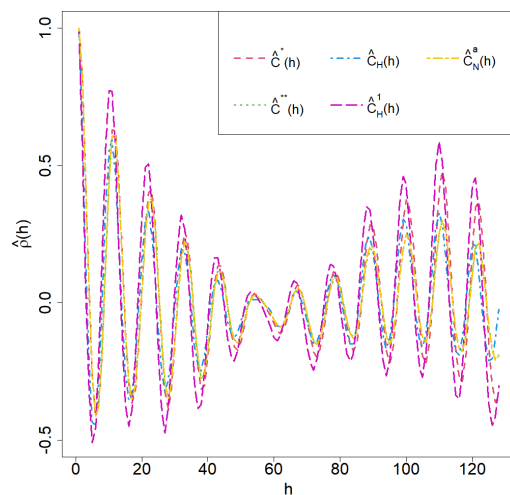


Figure 15: Estimated autocorrelation functions for the sunspots data.

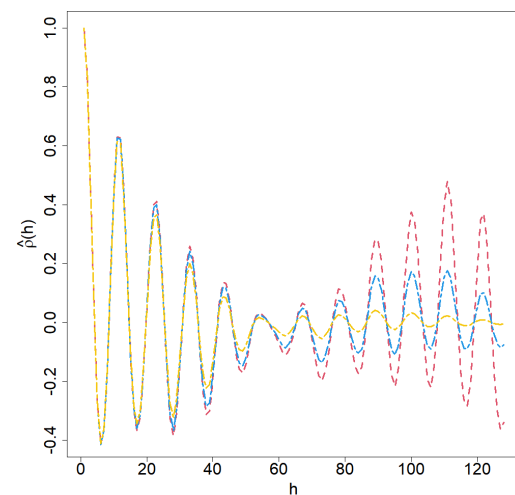


Figure 16: Smoothing of $\hat{C}^*(h)$ autocorrelation estimate.

```
X <- as.vector(sunspot.year)
x <- 1:length(X)
maxLag <- 128

# standard estimators
Cs <- standard_est(X, maxLag = maxLag - 1, pd = FALSE, meanX = mean(X), x = x,
  type = "autocorrelation")
```



```

Css <- standard_est(X, maxLag = maxLag - 1, pd = TRUE, meanX = mean(X), x = x,
  type = "autocorrelation")

# Hall's estimators
hall_1 <- adjusted_est(X, x, x[1:maxLag], b = 0.1, kernel_name = "wave",
  type = "autocorrelation")
hall_2 <- adjusted_est(X, x, x[1:maxLag], b = 0.1, kernel_name = "wave", pd = FALSE,
  type = "autocorrelation")

# tapered
tapered <- tapered_est(X, 0.01, "tukey", maxLag = maxLag - 1, x = x,
  type = "autocorrelation")

par(mar=c(4,5.25,0.25,0.25)+.1)

plot(Cs, lwd=3, lty=2, col=2, type='l', ylim=c(-0.5, 1), xlab=expression(h),
  ylab=expression(hat(rho)*'(h)'), cex.axis=2, cex.lab=2)
lines(Css, lwd=3, lty=3, col=3)
lines(hall_1, lwd=3, lty=4, col=4)
lines(hall_2, lwd=3, lty=5, col=6)
lines(tapered, lwd=3, lty=6, col=7)

legend('topright', c(expression(hat('C')^'*'*(h)'), expression(hat('C')^'*'*(h)'),
  expression(hat('C')[H]*'(h)'), expression(hat('C')[H]^'1'*(h)'),
  expression(hat('C')[N]^'a'*(h)'), col=c(2, 3, 4, 6, 7),
  lty=c(2, 3, 4, 5, 6), lwd=c(rep(3, 5)), y.intersp=1, cex=1.7, ncol=3)

```

As shown in Figure 15, the estimators provide consistent results. However, since no seasonal adjustment or transformation was applied, the estimates may not correspond to the optimal model specification for these data.

We have provided an example of how waves can be reduced and eliminated from the estimates in Figure 16. The red line, the original estimate (computed using estimator (3)), behaves cyclically and has waves of larger amplitude as the estimation lag increases. The blue and yellow lines show the smoothing results by using the wave (with $\theta = 50$) and Gaussian (with $\theta = 4000$) kernels, respectively. Such kernels were chosen to lessen constant summation and other unwanted effects and to bring the estimator to zero. The latter correction may be appropriate when short-range dependence is expected.

4.3 Example 3

We use monthly changes in U.S. unemployment counts data studied by Artiach and Arteche (2011) and sourced from the U.S. Bureau of Labor Statistics <https://www.bls.gov/>. There are 739 months of observations in the time series plotted in Figure 14. We estimated the autocovariance function up to a lag of 144, representing a maximum difference of 12 years between observations. In Artiach and Arteche (2011), the periodogram of this time series has a nonzero singularity indicating cyclic long-memory, see Ayache et al. (2022), where each cycle is roughly 5.5 years. So, we should expect some cyclicity in the estimated autocovariance functions.

The following code produces the estimated autocorrelations plotted in Figure 17.

```

maxLag <- 144
X <- X_bls
x <- 1:length(X)

# standard estimators
Cs <- standard_est(X, maxLag = maxLag - 1, pd = FALSE, x = x)
Css <- standard_est(X, maxLag = maxLag - 1, pd = TRUE, x = x,
  type = "autocorrelation")

# Hall's estimators
hall_1 <- adjusted_est(X, x, x[1:maxLag], b = 0.1, kernel_name = "rational_quadratic",
  type = "autocorrelation")
hall_2 <- truncated_est(X, x, x[1:maxLag], 110, 120, b = 0.1,
  kernel_name = "rational_quadratic", type = "autocorrelation")

# tapered

```

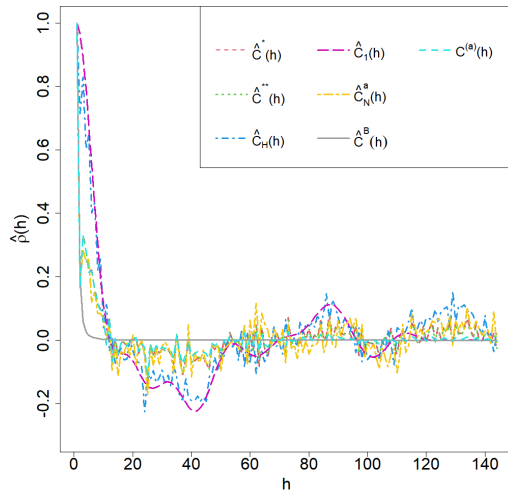


Figure 17: Estimated autocorrelation functions for unemployment increments data.

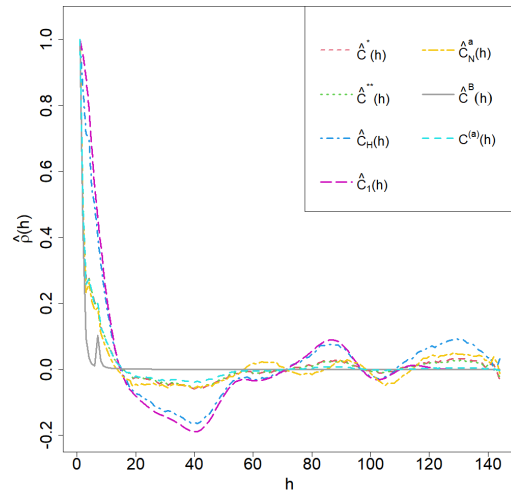


Figure 18: Smoothing of estimates for the unemployment increments data.

```
tapered <- tapered_est(X, 1, "tukey", maxLag = maxLag - 1, x = x,
  type = "autocorrelation")

# splines
splines <- splines_est(X, x, Cs, 3, 2, maxLag = maxLag - 1, type = "autocorrelation")
Cs <- normalise_acf(Cs)

# Correction
corrected <- corrected_est(X, "rational_quadratic", N_T = 5*length(X), maxLag = maxLag - 1,
  x = x, type = "autocorrelation")

colours <- c(2, 3, 4, 6, 7, 8, 13)
ltys <- c(2, 3, 4, 5, 6, 7, 8)
par(mar=c(4,5.25,0.25,0.25)+.1)
plot(Cs, lwd=3, lty=2, col=2, type='l', ylim=c(-0.3, 1), xlab=expression(h),
  ylab=expression(hat(rho)*'(h)'), cex.axis=2, cex.lab=2)
lines(Css, lwd=3, lty=3, col=3)
lines(hall_1, lwd=3, lty=4, col=4)
lines(hall_2, lwd=3, lty=5, col=6)
lines(tapered, lwd=3, lty=6, col=7)
lines(splines, lwd=3, lty=7, col=8)
lines(corrected, lwd=3, lty=8, col=13)

legend('topright', c(expression(hat('C')^'*'*(h)'), expression(hat('C')^'*'*(h)'),
  expression(hat('C')[H]*'(h)'), expression(hat('C')[1]*'(h)'),
  expression(hat('C')[N]^'a'*(h)'), expression(hat('C')^'B'*(h)'),
  expression('C'^'(a)'*(h)'), col=colours, lty=ltys, lwd=c(2, rep(3, 6)),
  y.intersp=1.2, cex=1.8, ncol=3)
```

As can be seen from Figure 17, the estimates have local chaotic fluctuations. To better see the main pattern of the estimates, Figure 18 shows the application of a moving average to each of the estimators in Figure 17. After this smoothing, two estimators, (5) and (6), exhibit cyclicity expected for this data.

4.4 Example 4

This example empirically investigates the computational complexity of the methods by evaluating the runtime and memory usage of the estimators. Autocovariance function estimates were computed for a sequence of simulated time series with Gaussian autocovariance, similar to Example 1. The realisations of the time series were simulated on uniform grids over $[0, 40]$ with lengths $N = 101, 151, \dots, 951, 1001$. This process was repeated 100 times, and the results presented are the medians.

Figure 19 shows the log10-time of the computed estimators. The log scale was used due to the large difference between estimators. For example, estimator (4) was computed in 0.000781015 seconds,

on average, for a time series of length $N = 1001$, whilst estimator (5) was computed in 8.45674 seconds. It is clear that all estimators (3), (4), (7), and (10) perform similarly in terms of time, whilst Hall's estimators (5) and (6) exhibit significant growth in computational time. The splines estimator (8) shows a consistent time usage across all estimates, exhibiting minimal growth as N increases.

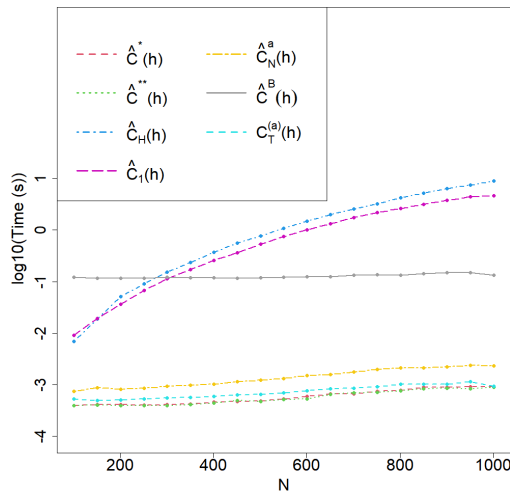


Figure 19: Log-scale computational time (s) for estimates for time series of varying length.

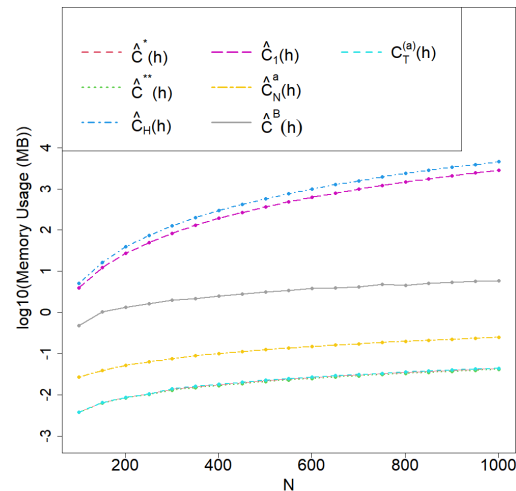


Figure 20: Log-scale memory usage (MB) for estimates for time series of varying length.

Figure 20 shows the log10-memory usage, where the log-scale was chosen for the same reasons as the time case. The estimators (3), (4), and (10) all effectively used the same memory. The estimator (7) is quite close, particularly in terms of growth; however, it is shifted upward. Again, Hall's estimators (5) and (6) exhibit significantly greater memory usage growth than the other estimators. The splines estimator (8) is stable in terms of growth, as in the time case.

It is clear that Hall's estimators, (5) and (6), require significantly more computational time and memory usage compared to other estimators. Thus, these estimators may not be appropriate in the case of Big Data. The other estimators are computed relatively quickly and require significantly less memory, even for large values of N . Further, these limitations should be considered when performing block bootstrap or other statistical procedures involving repeated computation of estimated covariances (e.g., moving window analysis, model diagnostics, Kriging/Gaussian process prediction and interpolation), as the estimator must be called many times. In such cases, using parallelism options, as implemented in the package for bootstrap, may substantially reduce computational time.

5 Summary

The article introduced the package **CovEsts**, which provides several nonparametric autocovariance function estimators available in the literature. The package offers a set of functions that allow users to easily apply the different estimators, as shown in the examples. The autocovariance estimator functions have a high level of flexibility with various options that the user can control, such as a custom kernel, selecting the sampling grid, and controlling the positive-definiteness of estimators, among others. Several functions are provided to generate bootstrap autocovariance confidence regions and to address potential issues in autocovariance estimation. In addition, various metrics are included to compare the performance of the estimators. In the future, the package is planned to be extended to work with similar nonparametric estimators for spatial data.

6 Computation details

R version 4.5 was used to develop, test the package, and produce all outputs in this article.

7 Acknowledgement

This research was partially supported by the Australian Research Council Discovery Projects funding scheme (project DP220101680). Andriy Olenko was also partially supported by La Trobe University's

SCEMS CaRE and Beyond grant. The authors are grateful to Professors N. Cressie, N. Leonenko, and E. Porcu for their feedback on certain approaches to the nonparametric estimation of covariance functions. We also thank the editor, Prof. R. J. Hyndman, and anonymous reviewers for their comments that helped to improve the package and the paper.

References

- J. Arteche. Bootstrapping long memory time series: Application in low frequency estimators. *Econometrics and Statistics*, 29:1–15, 2024. URL <https://doi.org/10.1016/j.ecosta.2021.06.002>. [p227]
- M. Artiach and J. Arteche. Estimation of the frequency in cyclical long-memory series. *Journal of Statistical Computation and Simulation*, 81(11):1627–1639, 2011. URL <https://doi.org/10.1080/00949655.2010.496728>. [p240]
- A. Ayache, M. Fradon, R. Nanayakkara, and A. Olenko. Asymptotic normality of simultaneous estimators of cyclic long-memory processes. *Electronic Journal of Statistics*, 16(1):84 – 115, 2022. URL <https://doi.org/10.1214/21-EJS1953>. [p240]
- A. Bilchouris and A. Olenko. On nonparametric estimation of covariogram. *Austrian Journal of Statistics*, 54(1):112–137, 2025a. URL <https://doi.org/10.17713/ajs.v54i1.1975>. [p224, 225]
- A. Bilchouris and A. Olenko. *CovEsts: Nonparametric Estimators for Covariance Functions*, 2025b. URL <https://CRAN.R-project.org/package=CovEsts>. R package version 1.0.0. [p223]
- O. N. Bjornstad. *ncf: Spatial Covariance Functions*, 2022. URL <https://CRAN.R-project.org/package=ncf>. R package version 1.3-2. [p223]
- G. N. Boshnakov and J. Halliday. *sarima: Simulation and Prediction with Seasonal ARIMA Models*, 2025. URL <https://CRAN.R-project.org/package=sarima>. R package version 0.9.4. [p223]
- P. J. Brockwell and R. A. Davis. *Time Series: Theory and Methods*. Springer New York, 1991. URL <https://doi.org/10.1007/978-1-4419-0320-4>. [p223]
- P. J. Brockwell and R. A. Davis. *Introduction to Time Series and Forecasting*. Springer International Publishing, 2016. URL <https://doi.org/10.1007/978-3-319-29854-2>. [p224]
- D. R. Bull and F. Zhang. *Intelligent Image and Video Compression: Communicating Pictures*. Academic Press, 2nd ed. edition, 2021. URL <https://doi.org/10.1016/C2019-0-00641-3>. [p223]
- K.-S. Chan and B. Ripley. *TSA: Time Series Analysis*, 2022. URL <https://CRAN.R-project.org/package=TSA>. [p223]
- J. Chilès and P. Delfiner. *Geostatistics: Modeling Spatial Uncertainty*. John Wiley & Sons, 2012. URL <https://doi.org/10.1002/9781118136188>. [p224]
- I. Choi, B. Li, and X. Wang. Nonparametric estimation of spatial and space-time covariance function. *Journal of Agricultural, Biological, and Environmental Statistics*, 18(4):611–630, 2013. URL <https://doi.org/10.1007/s13253-013-0152-z>. [p226]
- N. Cressie. Fitting variogram models by weighted least squares. *Journal of the International Association for Mathematical Geology*, 17(5):563–586, 1985. URL <https://doi.org/10.1007/bf01032109>. [p226]
- N. Cressie. *Statistics for Spatial Data*. Wiley, 1993. URL <https://doi.org/10.1002/9781119115151>. [p223, 224]
- J. D. Cryer and K.-S. Chan. *Time Series Analysis*. Springer, 2008. URL <https://doi.org/10.1007/978-0-387-75959-3>. [p223]
- F. Cuevas, E. Porcu, and R. Vallejos. Study of spatial relationships between two sets of variables: A nonparametric approach. *Journal of Nonparametric Statistics*, 25(3):695–714, 2013. URL <https://doi.org/10.1080/10485252.2013.797091>. [p224]
- R. Dahlhaus and H. Künsch. Edge effects and efficient parameter estimation for stationary random fields. *Biometrika*, 74(4):877–882, 1987. URL <https://doi.org/10.1093/biomet/74.4.877>. [p225]
- S. J. Devlin, R. Gnanadesikan, and J. R. Kettenring. Robust estimation and outlier detection with correlation coefficients. *Biometrika*, 62(3):531–545, 1975. URL <https://doi.org/10.1093/biomet/62.3.531>. [p227]

- J. Durbin. The fitting of time-series models. *Review of the International Statistical Institute*, 28(3):233–244, 1960. URL <https://doi.org/10.2307/1401322>. [p224]
- A. Dürre, R. Fried, and T. Liboschik. Robust estimation of (partial) autocorrelation. *WIREs Computational Statistics*, 7(3):205–222, 2015. URL <https://doi.org/10.1002/wics.1351>. [p224]
- L. A. Gil-Alana. Time series modeling of sunspot numbers using long-range cyclical dependence. *Solar Physics*, 257(2):371–381, 2009. URL <https://doi.org/10.1007/s11207-009-9390-1>. [p239]
- B. Gräler, E. Pebesma, and G. Heuvelink. Spatio-temporal interpolation using gstat. *The R Journal*, 8(1): 204–218, 2016. URL <https://doi.org/10.32614/RJ-2016-014>. [p223]
- P. Hall and P. Patil. Properties of nonparametric estimators of autocovariance for stationary random fields. *Probability Theory and Related Fields*, 99(3):399–424, 1994. URL <https://doi.org/10.1007/BF01199899>. [p225]
- P. Hall, N. Fisher, and B. Hoffmann. On the nonparametric estimation of covariance functions. *Annals of Statistics*, 22(4):2115–2134, 1994. URL <https://doi.org/10.1214/aos/1176325774>. [p223, 224, 225]
- P. Hall, J. L. Horowitz, and B.-Y. Jing. On blocking rules for the bootstrap with dependent data. *Biometrika*, 82(3):561–574, 1995. URL <https://doi.org/10.1093/biomet/82.3.561>. [p227]
- H. Hassani. Sum of the sample autocorrelation function. *Random Operators and Stochastic Equations*, 17(2):125–130, 2009. URL <https://doi.org/10.1515/ROSE.2009.008>. [p225]
- H. Hassani, N. Leonenko, and K. Patterson. The sample autocorrelation function and the detection of long-memory processes. *Physica A: Statistical Mechanics and its Applications*, 391(24):6367–6379, 2012. URL <https://doi.org/10.1016/j.physa.2012.07.062>. [p225]
- N. J. Higham. Computing a nearest symmetric positive semidefinite matrix. *Linear Algebra and its Applications*, 103:103–118, 1988. URL [https://doi.org/10.1016/0024-3795\(88\)90223-6](https://doi.org/10.1016/0024-3795(88)90223-6). [p234]
- J. Hu, J. Gao, and X. Wang. Multifractal analysis of sunspot time series: the effects of the 11-year cycle and Fourier truncation. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(02):P02066, 2009. URL <https://doi.org/10.1088/1742-5468/2009/02/p02066>. [p239]
- R. J. Hyndman. Discussion of “high-dimensional autocovariance matrices and optimal linear prediction”. *Electronic Journal of Statistics*, 9(1), 2015. URL <https://doi.org/10.1214/14-EJS953>. [p223]
- R. J. Hyndman and Y. Khandakar. Automatic time series forecasting: The forecast package for R. *Journal of Statistical Software*, 27(3):1–22, 2008. URL <https://doi.org/10.18637/jss.v027.i03>. [p223]
- H. R. Künsch. The jackknife and the bootstrap for general stationary observations. *The Annals of Statistics*, 17(3):1217–1241, 1989. URL <https://doi.org/10.1214/aos/1176347265>. [p227]
- S. Lahiri. On the moving block bootstrap under long range dependence. *Statistics & Probability Letters*, 18(5):405–413, 1993. URL [https://doi.org/10.1016/0167-7152\(93\)90035-h](https://doi.org/10.1016/0167-7152(93)90035-h). [p227]
- S. N. Lahiri. *Resampling Methods for Dependent Data*. Springer, 2003. URL <https://doi.org/10.1007/978-1-4757-3803-2>. [p227]
- R. Y. Liu and K. Singh. Moving blocks jackknife and bootstrap capture weak dependence. In R. LePage and L. Billard, eds, *Exploring the Limits of Bootstrap*, pages 225–248, 1992. URL <https://www.wiley.com/en-us/Exploring+the+Limits+of+Bootstrap-p-9780471536314>. [p227]
- D. Massart, B. Vandeginste, S. Deming, Y. Michotte, and L. Kaufman. Chapter 14 - correlation methods. In *Chemometrics: A Textbook*, volume 2 of *Data Handling in Science and Technology*, pages 191–213. Elsevier, 2003. URL [https://doi.org/10.1016/S0922-3487\(08\)70227-2](https://doi.org/10.1016/S0922-3487(08)70227-2). [p223]
- T. L. McMurphy and D. N. Politis. Banded and tapered estimates for autocovariance matrices and the linear process bootstrap. *Journal of Time Series Analysis*, 31(6):471–482, 2010. URL <https://doi.org/10.1111/j.1467-9892.2010.00679.x>. [p223]
- D. N. Politis and J. P. Romano. A circular block-resampling procedure for stationary data. In R. LePage and L. Billard, eds, *Exploring the Limits of Bootstrap*, pages 263–270, 1992. URL <https://www.wiley.com/en-us/Exploring+the+Limits+of+Bootstrap-p-9780471536314>. [p227]
- D. N. Politis and H. White. Automatic block-length selection for the dependent bootstrap. *Econometric Reviews*, 23(1):53–70, 2004. URL <https://doi.org/10.1081/ETC-120028836>. [p227]

- P. J. Rousseeuw and G. Molenberghs. Transformation of non positive semidefinite correlation matrices. *Communications in Statistics - Theory and Methods*, 22(4):965–984, 1993. URL <https://doi.org/10.1080/03610928308831068>. [p227]
- R. H. Shumway and D. S. Stoffer. *Time Series Analysis and Its Applications: With R Examples*. Springer, 2025. URL <https://doi.org/10.1007/978-3-031-70584-7>. [p223]
- D. Stoffer and N. Poison. *astsa: Applied Statistical Time Series Analysis*, 2024. URL <https://CRAN.R-project.org/package=astsa>. [p223]
- A. Yaglom. *Correlation Theory of Stationary and Related Random Functions*, volume I: Basic Results. Springer, 1987. URL <https://doi.org/10.1007/978-1-4612-4628-2>. [p224, 226, 233, 235]

Adam Bilchouris
Department of Mathematical and Physical Sciences
La Trobe University
Australia
ORCID: 0009-0002-4649-0247
a.bilchouris@latrobe.edu.au

Andriy Olenko
Department of Mathematical and Physical Sciences
La Trobe University
Australia
ORCID: 0000-0002-0917-7000
a.olenko@latrobe.edu.au

deseats: An R Package for Data-Driven Trend and Seasonality Estimation in Time Series

by Dominik Schulz

Abstract The R package *deseats* is introduced that allows for the application of a data-driven locally-weighted-regression algorithm for estimating the trend and the seasonality in univariate, equidistant time series with short-memory errors. A corresponding data-driven semiparametric model with autoregressive moving-average errors and point as well as interval forecasting approaches are established and the main functions of the *deseats* package are described. *deseats* is applied under consideration of real-world time series and the seasonal component identification is compared to that of STL, TRAMO-SEATS and X13-ARIMA. Finally, the quality of the bandwidth selection algorithm and the consistency of the automated component estimators are highlighted through a simulation study, while the quality of seasonality estimation is compared to that of well-established and widely-used methods, including aforementioned methods and others, in a second simulation study. The new algorithm captures the error-autocorrelation well and often produces seasonality estimates with mean squared error smaller than or at least similar to other methods, if the number of observations is sufficiently large. This finding also holds in simulated scenarios, where the underlying model assumption of the determinism of trend and seasonal components is violated.

1 Introduction

The decomposition of time series is an important element in the analysis of time series, as it allows a series to be split into various components. Such a distinction between different components is also often desired, for example in economics, because it permits trend-obscuring seasonal effects to be identified and removed to generate a seasonally adjusted series, which achieves a simplified interpretability of observed time series “without a significant loss of information” (Bell and Hillmer, 1984). Commonly, a univariate time series is split into a trend, a seasonal and a remainder term. To achieve a suitable decomposition regarding this basic model, various methods have been established (see also Chapter 1.5 in Brockwell and Davis, 2016, for an overview of basic approaches) such as moving averages. Hillmer and Tiao (1982) introduced a seasonal adjustment approach based on ARIMA models, whereas Pierce (1978) studied models with a mixture of deterministic and stochastic seasonality. Furthermore, the federal statistical offices of many industrial nations have introduced their own procedures for decomposing macroeconomic time series throughout the last decades such as the *Berlin Procedure 4.1* (BV4.1, Cieplik, 2006; Speth, 2004) in Germany, for which the foundations were laid out by Heiler (1970), the method *Time Series Regression with ARIMA Noise, Missing Values and Outliers with Signal Extraction in ARIMA Time Series* (TRAMO-SEATS, Gómez and Maravall Herrero, 1996) by the bank of Spain, the X12-ARIMA procedure (Findley et al., 1998), or lately the extension to X13-ARIMA, included in the X-13ARIMA-SEATS program (U.S. Census Bureau, 2025), in the USA. Note that the abbreviation X13 will be used throughout to refer to the X13-ARIMA method. Overall guidelines in the European Union to ensure comparability of seasonal adjustments were published by European Commission and Eurostat (2015) and European Commission and Eurostat (2024). A detailed overview of the history of seasonal adjustment and selected methods is provided in Mazzi et al. (2018).

In this paper, we introduce an improved version of the algorithm called *DeSeaTS* (deseasonalize time series, Feng, 2013), whose original idea was proposed by Feng (1999), for decomposing a univariate time series that is, like BV4.1, based on the assumption that the trend can be approximated locally by polynomials and that the seasonal component can be locally approximated by trigonometric polynomials. Nonetheless, instead of using fixed filters regardless of the time series under consideration as in BV4.1, we propose a data-driven iterative plug-in (IPI, Gasser et al., 1991) algorithm to estimate a globally optimal bandwidth that is afterwards used in the locally weighted regressions. In comparison to the original version of *DeSeaTS* by Feng (2013) and to its closely related robust alternative by Heiler and Feng (2004), the updated version of the algorithm presented here does not rely on the strong assumption of independent and identically distributed (iid) errors but instead allows for short-memory errors, while it still maintains favorable characteristics of the original algorithm, such as the convergence rate of the bandwidth estimator. Alongside the algorithm, we also introduce a new package for the statistical programming language R (R Core Team, 2026) called *deseats* (Feng and Schulz, 2026, version 1.1.2), which is readily available for download on CRAN (Comprehensive R Archive Network), that allows its users to implement the approaches discussed hereinafter.

Current extensions for R let users implement a wide range of decomposition approaches. The package `neverhpfiler` (Shea, 2025) can be used to obtain stationary residuals from a time series with trend and seasonality following the idea by Hamilton (2018). Its drawback is that the built-in time series filter requires the manual setting of hyperparameters, for which Hamilton (2018) proposes default settings for various scenarios. By combining `x13binary` (Eddelbuettel and Sax, 2025), `seasonal` (Sax and Eddelbuettel, 2018) and `seasonalview` (Sax, 2014), an R interface to the X-13ARIMA-SEATS program by the US Census Bureau is provided. Similar functionality is also given through the packages `x12` (Kowarik et al., 2014) and `RJDemetra` (la Tente et al., 2024). Classical decomposition is achieved with `decompose()` of the package `stats` (R Core Team, 2026) and regression-based decomposition is implemented in `stR` (Dokumentov and Hyndman, 2025). Furthermore, `stlplus` (Hafen, 2016) provides access to enhanced seasonal-trend decomposition using LOESS (STL, Cleveland et al., 1990) and both TBATS (trigonometric seasonality, Box-Cox transformation, ARMA errors, trend and seasonal components, De Livera et al., 2011) and an automated STL approach are incorporated in the package `forecast` (Hyndman and Khandakar, 2008) via the functions `tbats()` and `mstl()`, respectively.

The paper is structured as follows. In Section 2, theoretical information on the DeSeaTS algorithm, the underlying model and corresponding forecasting techniques is given, whereas an overview of the deseats package is provided in Section 3. Applications of the deseats package to real-world data are given in Section 4. In the same section, two simulation studies are conducted: one on the DeSeaTS bandwidth selection algorithm and another to compare the seasonal component estimation quality of DeSeaTS and other methods under various settings. Section 5 concludes.

2 Theoretical foundations of DeSeaTS

In the first subsection, the basic time series model considered in DeSeaTS, the idea of locally weighted regression and the asymptotically optimal bandwidth in this context are presented. The second subsection gives details on how the important variance factor accounting for the short-range dependency structure in the errors is estimated. In the third subsection, the proposed IPI-algorithm is explained. Furthermore, the basic forecasting approach for our seasonal semiparametric model is shown in the fourth subsection.

2.1 Locally weighted regression

Following Feng (2013), denote by $\{Y_t\}, t = 1, \dots, n$, a time series with discrete and equidistant time index, whose realization is $\{y_t\}, t = 1, \dots, n$. Consider an additive component model for time series with a nonparametric trend and seasonality

$$Y_t = g(x_t) + s(x_t) + \zeta_t, \quad (1)$$

where g is a smooth and slowly changing nonparametric trend function and where s is a slowly changing seasonal component with seasonal period p_s . $x_t = t/n$ is the time rescaled to the interval $[0, 1]$ and ζ_t is a stationary rest with $E(\zeta_t) = 0$, short-range dependence and spectral density at frequency zero $c_f = f(0) = (2\pi)^{-1} \sum_{\tau=-\infty}^{\infty} \gamma(\tau) < \infty$, where $\gamma(\tau)$ are the autocovariances in ζ_t at lag $\tau \in \mathbb{Z}$. For the purpose of estimating c_f using the methodology described in the next subsection, the summability of cumulants of ζ_t up to order 8, among other assumptions, is required (Bühlmann, 1996), which is fulfilled if ζ_t are governed by a stationary autoregressive moving-average (ARMA) process with $E(\zeta_t^{8+\delta}) < \infty$ with $\delta > 0$ (Feng et al., 2020). In the context of this model, both g and s are deterministic components that affect the time-mean in Y_t so that $E(Y_t) = m(x_t) = g(x_t) + s(x_t)$ is the deterministic trend-seasonal component, whereas ζ_t is stochastic.

In the literature, other model assumptions on the components are sometimes imposed. For example, a seasonal integrated ARMA (SARIMA) assumes both a stochastic trend and stochastic seasonality, which are hence also the assumptions in the related TRAMO-SEATS program. Different assumptions lead, however, to inherently different implications for contrasting methods and have been discussed for a long period (Pierce, 1978; DeJong et al., 1992). If an observed time series violates model assumptions clearly, robustness of a method is not always guaranteed, and thus decomposition quality of a particular decomposition method may strongly depend on the observed series at hand.

Model (1) is an extension of the additive component model with a nonparametric trend but without seasonality proposed by Feng et al. (2020) and Feng et al. (2022). Moreover, it is equivalent to the base model proposed in the procedure BV4.1 formerly used by the German Federal Statistical Office with one major distinction: as described by Speth (2004), the variables of the error process $\{\zeta_t\}$ in the BV4.1 base model are assumed to be iid.

The basis for the DeSeaTS algorithm is locally weighted regression (Cleveland, 1979). Local

polynomial regression as a special case is discussed in detail by [Fan and Gijbels \(1996\)](#). Following their ideas, the following assumptions in alignment with [Feng \(2013\)](#) have to be imposed, where h is the bandwidth and $k = r + 1, r \in \{1, 3, 5, \dots\}$:

- (A1) $h \rightarrow 0$ and $nh \rightarrow 0$ for $n \rightarrow \infty$;
- (A2) g is at least k -times continuously differentiable;
- (A3) s is exactly periodic with period p_s .

The assumption (A3) is relevant theoretically so that the (asymptotic) bias in $\hat{s}$ as an estimator of s does not need to be considered ([Feng, 2013](#)). While (A3) is restrictive and, as will be shown in Section 4, unrealistic, because seasonal patterns usually seem to change over time in observed time series, it is shown by the simulations in Section 4 that the algorithm provides suitable results even in instances where (A3) is being violated. The implications that follow Assumptions (A1) and (A2), on the other hand, are well-known assumptions in nonparametric regression in the context of the estimation of only g .

Based on (A1) to (A3), the following locally weighted regression approach can be considered for the trend and the seasonality in time series as for example explained by [Heiler and Feng \(2000\)](#). g can be approximated locally in the proximity of a rescaled time point $x_0 \in \{x_1, x_2, \dots, x_n\}$ by a polynomial of order r :

$$g(x_t) \approx \tilde{g}(x_t) = \sum_{j=0}^r \beta_j(x_0)(x_t - x_0)^j, \quad (2)$$

where $\beta_j(x_0), j = 0, \dots, r$, are real-valued coefficients in the approximation of $g(x_t)$ in the proximity of x_0 . Moreover, s can be represented locally by a trigonometric polynomial

$$s(x_t) \approx \tilde{s}(x_t) = \sum_{i=1}^q \{ \alpha_{1,i}(x_0) \cos[\lambda_i n(x_t - x_0)] + \alpha_{2,i}(x_0) \sin[\lambda_i n(x_t - x_0)] \}, \quad (3)$$

where $q = [p_s/2]$ with $[\cdot]$ denoting the integer part and where $\lambda_1 = 2\pi/p_s$ is the seasonal frequency and $\lambda_i = i\lambda_1$ for $i = 2, \dots, q$. Note that for even p_s , $\alpha_{2,q}(x_0) \sin[\lambda_q n(x_t - x_0)]$ has to be omitted throughout ([Feng, 2013](#)). Furthermore, $\alpha_{1,i}(x_0)$ and $\alpha_{2,i}(x_0), i = 1, \dots, q$, are real-valued coefficients when approximating $s(x_t)$ around x_0 .

Consequently, this leads to

$$Q(x_0) = \sum_{t=1}^n \{Y_t - \tilde{g}(x_t) - \tilde{s}(x_t)\}^2 W\left(\frac{x_t - x_0}{h}\right) \quad (4)$$

as the locally weighted sum of squares that should be minimized, where W is a weighting function, in more detail a density, with compact support $[-1, 1]$ and where h is the bandwidth. For the purpose of this paper, we only consider weighting functions of the type

$$W_\mu(u) = C_\mu (1 - u^2)^\mu \mathbb{I}_{[-1,1]}(u) \quad (5)$$

with C_μ being a constant that can be omitted, as it cancels out when solving (4), and with $\mathbb{I}_{[-1,1]}(u)$ being equal to one for $u \in [-1, 1]$ and zero otherwise. Furthermore, $\mu = 0, 1, \dots$ is a constant that affects the smoothness of W_μ . Nonetheless, for simplicity, in the proposed algorithm only $\mu = 0, 1, 2, 3$ will be considered.

Following [Feng \(2013\)](#), define

$$\mathbf{X}_1 = \begin{pmatrix} 1 & x_1 - x_0 & \dots & (x_1 - x_0)^r \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n - x_0 & \dots & (x_n - x_0)^r \end{pmatrix} \quad (6)$$

and

$$\mathbf{X}_2 = \begin{pmatrix} \cos[\lambda_1(1 - t_0)] & \sin[\lambda_1(1 - t_0)] & \dots & \cos[\lambda_q(1 - t_0)] & \{\sin[\lambda_q(1 - t_0)]\} \\ \vdots & \vdots & \ddots & \vdots & \left\{ \begin{smallmatrix} \vdots \\ \vdots \end{smallmatrix} \right\} \\ \cos[\lambda_1(n - t_0)] & \sin[\lambda_1(n - t_0)] & \dots & \cos[\lambda_q(n - t_0)] & \{\sin[\lambda_q(n - t_0)]\} \end{pmatrix}, \quad (7)$$

where entries marked by $\{\cdot\}$ are omitted for even p_s . Then $\mathbf{X} = (\mathbf{X}_1 \vdots \mathbf{X}_2)$. Furthermore, let $\mathbf{y} = (y_1, y_2, \dots, y_n)'$ be a vector containing the observations of $\{Y_t\}$ and define $w_t(x_0) = W((x_t - x_0)/h)$ and then $\mathbf{w} = (w_1(x_0), w_2(x_0), \dots, w_n(x_0))'$, so that $\mathbf{W} = \text{diag}(\mathbf{w})$ ([Feng, 2013](#)). Moreover, let $\mathbf{i}_{g,j}$ be a

$(r+1) \times 1$ vector with zeros everywhere except for the j -th entry, which equals one, and define $\mathbf{i}_s$ to be an $(p_s - 1) \times 1$ vector with zeros in the even entries and ones in the odd entries. The solution to (4) can now be expressed as

$$\hat{g}(x_0) = (\mathbf{i}'_{g,1}, \mathbf{0}') (\mathbf{X}' \mathbf{W} \mathbf{X})^{-1} \mathbf{X}' \mathbf{W} \mathbf{y} \quad (8)$$

$$\hat{s}(x_0) = (\mathbf{0}', \mathbf{i}'_s) (\mathbf{X}' \mathbf{W} \mathbf{X})^{-1} \mathbf{X}' \mathbf{W} \mathbf{y} \quad \text{and} \quad (9)$$

$$\hat{m}(x_0) = (\mathbf{i}'_{g,1}, \mathbf{i}'_s) (\mathbf{X}' \mathbf{W} \mathbf{X})^{-1} \mathbf{X}' \mathbf{W} \mathbf{y}, \quad (10)$$

where $\mathbf{0}$ is a column vector with only zeros of adequate size. The solutions following (8), (9) and (10) are consequently $\hat{g}(x_0) = \hat{\beta}_0(x_0)$, $\hat{s}(x_0) = \sum_{i=1}^q \hat{\alpha}_{1,i}(x_0)$ and $\hat{m}(x_0) = \hat{\beta}_0(x_0) + \sum_{i=1}^q \hat{\alpha}_{1,i}(x_0)$, where $\hat{\beta}_0(x_0), \hat{\alpha}_{1,1}(x_0), \dots, \hat{\alpha}_{1,q}(x_0)$ are the estimates of the corresponding coefficients in (4). In order to estimate $g^{(k)}(x_0)$, the k -th derivative of g at the rescaled time point x_0 , $\mathbf{i}_{g,1}$ has to be replaced by $\mathbf{i}_{g,k+1}$ in (8), leading to $\hat{g}^{(k)}(x_0) = k! \hat{\beta}_k(x_0)$. Note that the estimation of $\hat{g}^{(k)}(x_0)$ requires further assumptions similar to (A1) and (A2) (see (A1') and (A2') in Feng, 2013).

k , as defined previously, is the order of the asymptotically equivalent kernel $K_{r,\mu}$ for estimating g . As shown by Feng (2013), the asymptotic mean integrated squared error (AMISE) under iid errors ξ_t can be expressed as a sum of two fractions, i.e. of a (squared) bias and a variance component. It is assumed, in analogy to the bandwidth selection for local polynomial regression (Beran and Feng, 2002), that the AMISE under autocorrelated errors is the same as that in Feng (2013) with the adjustment that σ^2 therein in the variance part needs to be replaced by $2\pi c_f$, so that the AMISE under short-memory errors is then given by

$$\text{AMISE}(h) = h^{2k} \frac{I[g^{(k)}] b_{(k)}^2}{(k!)^2} + \frac{2\pi c_f (d_b - c_b) \{R(K_{r,\mu}) + (p_s - 1)R(W_\mu)\}}{nh} \quad (11)$$

with $I[g^{(k)}] = \int_{c_b}^{d_b} [g^{(k)}(x)]^2 dx$, $b_{(k)} = \int_{-1}^1 u^k K_{r,\mu}(u) du$ and $R(K) = \int_{-1}^1 K^2(u) du$. $0 \leq c_b < d_b \leq 1$ are two constants introduced to reduce the effect of boundary points on the bandwidth. Proof of formula (11) is not given here in detail; however, a rough sketch of the proof can be stated as follows: the (squared) asymptotic bias (at an interior point) of $\hat{m}(x_0)$ and the asymptotic variance of $\hat{g}(x_0)$ (at an interior point) are unaffected by the autocorrelation and are hence the same as for iid data as in Feng (2013). On the other hand, the asymptotic variance of $\hat{g}(x_0)$ (at an interior point) is that as shown in the proof of Theorem 1 in Beran and Feng (2002) under $\delta = 0$, whereas the asymptotic variance of $\hat{s}(x_t)$ (at an interior point) can be obtained following the same proof under consideration of the asymptotically equivalent kernel $K_s(u) = Q_s(t)W_\mu(u)$ instead for estimating $\hat{s}(x_0)$ (see Equation (4.21) in Feng, 1999), with $Q_s(t) = s - 1$ for $(t \bmod p_s) = 0$ and $Q_s(t) = -1$ for $(t \bmod p_s) \neq 0$ (both for $t \in T_0 := \{t_0 - b, t_0 - b + 1, \dots, t_0 + b - 1, t_0 + b\}$ with b being the equivalent positive integer-valued bandwidth for the relative bandwidth h and $t_0 = nx_0$). Thus, $(nh)\text{var}[\hat{s}(x_0)] = V_n(h)[1 + o(1)]$ with $V_n(h) = (nh)2\pi c_f \sum_{i=t_0-b}^{t_0+b} w_{s,i}^2 + o(1)$, where $w_{s,i}$ are the weights defined through $(\mathbf{0}', \mathbf{i}'_s) (\mathbf{X}' \mathbf{W} \mathbf{X})^{-1} \mathbf{X}' \mathbf{W}$ in (9). Following the proof of Corollary 4.6 in Feng (1999) adjusted for our setting with autocorrelated data, so that the constant that is multiplied with the sum of the squared weights is no longer only σ^2 but $2\pi c_f$ as shown for $V_n(h)$ here, we conclude $\text{var}[\hat{s}(x_0)] = (nh)^{-1} 2\pi c_f (p_s - 1) R(W_\mu) \{1 + O[(nh)^{-1}]\}$. Since also the third result of Corollary 4.6 in Feng (1999) holds, i.e. that $\hat{g}(x_0)$ and $\hat{s}(x_0)$ are asymptotically uncorrelated, as that part of the proof does not change under short-memory errors, we thus have

$$\text{var}[\hat{m}(x_0)] = (nh)^{-1} 2\pi c_f \{R(K_{r,\mu}) + (p_s - 1)R(W_\mu)\} \{1 + O[(nh)^{-1}]\} \quad (12)$$

and formula (11) follows. Calculating the first derivative of the AMISE (11) with respect to h and then solving for h yields the asymptotically optimal bandwidth

$$h_A = n^{-\frac{1}{2k+1}} \cdot \left(\frac{(k!)^2 2\pi c_f (d_b - c_b) \{R(K_{r,\mu}) + (p_s - 1)R(W_\mu)\}}{I[g^{(k)}] b_{(k)}^2} \right)^{\frac{1}{2k+1}}. \quad (13)$$

This result still holds (for interior points), if assumption (A3) is relaxed in such a way that the changes in the seasonal component follow exactly a simple linear function, because then the local constant seasonality estimator implied by our approach would still be unbiased, as changes in the seasonal component to the left and to the right of the estimation time point x_0 would cancel each other out exactly. However, we presume that h_A is at least a rough approximation to the true asymptotically optimal bandwidth under even stronger violations of (A3), since we assume that deviations in s from

$\tilde{s}$ may approximately cancel each other out locally.

Ultimately, since no model assumptions are imposed on $\tilde{\zeta}_t$ in the bandwidth selection algorithm proposed within this paper, our overall model is completed by assuming a simple ARMA model

$$\phi(B)\tilde{\zeta}_t = \theta(B)\varepsilon_t, \quad (14)$$

where $\phi(B) = 1 - \sum_{i=1}^l \phi_i B^i$ and $\theta(B) = 1 + \sum_{j=1}^v \theta_j B^j$, with B as the backshift operator, are the characteristic AR and MA polynomials without common roots, for the errors after the trend and seasonality adjustment. ϕ_i , $i = 1, \dots, l$, and θ_j , $j = 1, \dots, v$, are real-valued coefficients and ε_t are iid random variables with $E(\varepsilon_t) = 0$ and $\text{var}(\varepsilon_t) = \sigma_\varepsilon^2 < \infty$. (1) together with (14) will be called an S-Semi-ARMA (seasonal semiparametric ARMA) model.

2.2 The methodology for estimating c_f

c_f is the spectral density at frequency zero of the process governing the short-memory errors $\tilde{\zeta}_t$ in (1). This implies that $2\pi c_f$ in the asymptotically optimal bandwidth (13) is the sum of autocovariances of the process driving $\tilde{\zeta}_t$, which is the main improvement over the asymptotically optimal bandwidth in Feng (2013). Therein, the factor $\sigma_\zeta^2 = \text{var}(\tilde{\zeta}_t)$ is considered instead of $2\pi c_f$ due to iid $\tilde{\zeta}_t$. Note that for iid $\tilde{\zeta}_t$, our suggestion reduces correctly to the special case with $2\pi c_f = \sigma_\zeta^2$. Analogously to Feng et al. (2020), an estimate of c_f can be obtained in a nonparametric way from trend- and seasonality-adjusted observations $\tilde{\zeta}_t$. Therefore, we copy the adjusted Bühlmann algorithm (Bühlmann, 1996) proposed by Feng et al. (2020). In contrast to Feng et al. (2020), we do not consider a bandwidth correction factor, because we find that such a correction factor, adjusted for the scenario with trend and seasonality, sometimes leads to overestimation of the bandwidth in the bandwidth selection algorithm proposed here.

The algorithm can be summarized briefly as follows, given an initial window width $L_0 = \lfloor n/2 \rfloor$ with $\lfloor \cdot \rfloor$ denoting the integer part. We begin with the global step of the procedure at $j = 0$.

- i) Increase j by one. Follow Bühlmann (1996) to obtain estimates of $\int (f^2(\lambda)) d\lambda$ and $\int f^{(1)}(\lambda) d\lambda$, which is the first generalized derivative of $f(\lambda)$, using an adjusted window width $L_j^* = \lfloor L_{j-1}/n^{2/21} \rfloor$.
- ii) L_j is then obtained from Equation (5) in Bühlmann (1996).
- iii) Repeat i) and ii) until either convergence or a maximum number of iterations is reached. The final L_j is denoted by L_G as the selected global window width.

Afterwards, a local window width is obtained for the frequency of interest $\lambda = 0$.

- iv) The local formula in Equation (5) in Bühlmann (1996) is considered for $\lambda = 0$, where $\int f^{(1)}(\lambda) d\lambda$ is estimated beforehand using $L^* = \lfloor L_G/n^{2/21} \rfloor$. The result of this local step is the final estimated window width L .

Ultimately, as illustrated by Feng et al. (2020), we consider the lag-window estimator

$$\hat{c}_{f,L} = (2\pi)^{-1} \sum_{l=-L}^L w_l \hat{\gamma}(l), \quad (15)$$

where L is exactly the window width selected through the described algorithm. Furthermore, $\hat{\gamma}(l)$ are the sample autocovariances of $\tilde{\zeta}_t$ and w_l are Bartlett-window weights following the window width L with the lag $l = 0$ at the center.

For further, more detailed information on this particular part of the algorithm, we refer the reader to Bühlmann (1996), Feng et al. (2020) and Feng et al. (2022).

2.3 The proposed IPI-algorithm

In (13), fixed estimates for all iterations of the IPI-algorithm can be obtained for all quantities with the exception of c_f and $I \left[m^{(k)} \right]$. Therefore, the proposed algorithm reads almost identically as proposed by Feng et al. (2020). Consider a variable $j = 0, 1, \dots, J$ that indicates the current iteration of the algorithm, where J is an arbitrary maximum number of iterations, for example $J = 20$. Moreover, denote by $\hat{c}_f$ and $I \left[\hat{g}^{(k)} \right]$ estimates of c_f and $I \left[g^{(k)} \right]$, respectively.

- i) For iteration $j = 0$, set an arbitrary starting bandwidth h_0 , for example $h_0 = 0.1$.

ii) Increase j by one. Obtain $\hat{c}_f$ and $I[\hat{g}^{(k)}]$ and compute

$$\hat{h}_j = n^{-\frac{1}{2k+1}} \cdot \left(\frac{(k!)^2 2\pi\hat{c}_f(d_b - c_b) \{R(K_{r,\mu}) + (p_s - 1)R(W_\mu)\}}{2k I[\hat{g}^{(k)}] b_{(k)}^2} \right)^{\frac{1}{2k+1}}.$$

iii) Conduct ii) either until convergence or until a total of J iterations is reached. The estimated optimal bandwidth is then the bandwidth reached through convergence or $\hat{h}_J$.

In analogy to [Feng et al. \(2020\)](#) and their IPI-algorithm for identifying nonparametric trends, we also propose to estimate c_f from residuals $\tilde{\xi}_{j,t} = y_t - \hat{m}_j(x_t)$ in each iteration as described in the previous subsection, where $\hat{m}_j(x_t)$ is the mean function estimated via locally weighted regression in iteration j of the algorithm. Moreover, analogously to [Feng et al. \(2020\)](#), we propose to estimate $g^{(k)}$ needed for $I[\hat{g}^{(k)}]$ in each iteration by means of locally weighted regression using a bandwidth $\hat{h}_{j-1}^\nu$, where ν is a suitable constant to inflate the bandwidth, for example $\nu = 5/7$ (optimal factor) or $\nu = 5/9$ (naive factor) for $r = 1$ and $\nu = 9/11$ (optimal factor) or $\nu = 9/13$ (naive factor) for $r = 3$, so that either $I[\hat{g}^{(k)}]$ or $\hat{g}^{(k)}$ reach their optimal rates of convergence, because under regularity conditions an asymptotically MSE-optimal bandwidth for estimating $I[\hat{g}^{(k)}]$ should be $h_{d,A} = O(n^{-1/(2k+3)})$, whereas the optimal bandwidth for estimating $\hat{g}^{(k)}$ directly should be $h_{d,A} = O(n^{-1/(2k+5)})$, if for simplicity $\hat{g}^{(k)}$ is obtained using a polynomial order $r_d = k + 1$ together with kernel-order $k_d = k + 2$. This result is the same as in [Feng \(2013\)](#), because the optimal order of $h_{d,A}$, which is dominated by the squared bias, is unaffected by the presence of autocorrelated short-memory errors.

In addition, the estimator $\hat{c}_f$ as considered here has rate of convergence $O_p(n^{-1/3})$ ([Bühlmann, 1996; Feng et al., 2020](#)). Since $I[\hat{g}^{(k)}]$ has the same asymptotic bias and variance as for independent errors (see Equations (18) and (19) in [Feng, 2013](#)), we conclude that the error in $\hat{c}_f$ is asymptotically negligible compared to that of $\hat{I}$, so that the bandwidth estimator $\hat{h}$ following the proposed IPI-algorithm has the convergence rate

$$\frac{\hat{h} - h_A}{h_A} = \begin{cases} O_p(n^{-2/(2k+3)}), & \text{for optimal inflation factors,} \\ O_p(n^{-2/(2k+5)}), & \text{for naive inflation factors.} \end{cases} \quad (16)$$

These are the same common rates as those under iid errors ([Feng, 2013](#)) and in the context of automated local polynomial regression ([Francisco-Fernández et al., 2004; Feng et al., 2020](#)).

2.4 Forecasting technique for S-Semi-ARMA models

Consider an S-Semi-ARMA model (1) with (14). After the decomposition using locally weighted regression with a bandwidth selected by the bandwidth selection algorithm described in the previous subsection, we have trend estimates $\hat{g}(x_t)$, seasonality estimates $\hat{s}(x_t)$ and trend- as well as seasonally-adjusted values $\tilde{\xi}_t = y_t - \hat{g}(x_t) - \hat{s}(x_t) = y_t - \hat{m}(x_t)$, $t = 1, \dots, n$. Following the post-decomposition assumption (14), an ARMA model can be fitted to the series $\{\tilde{\xi}_t\}$ in a separate step using quasi-maximum-likelihood estimation (QMLE), for example using the function `arima()` of the `stats` package, where the optimal orders could be identified through common criteria like the Bayesian information criterion (BIC, [Schwarz, 1978](#)) or the Akaike information criterion (AIC, [Akaike, 1974](#)). As the first simulation study in Section 4 will show, $\hat{m}(x_t)$ as an estimator of $m(x_t)$ can usually be considered to be consistent, thus implying $\tilde{\xi}_t \xrightarrow{P} \xi_t$ and consequently leading to the QMLE based on $\{\tilde{\xi}_t\}$ being consistent with respect to the true set of ARMA parameters $(\phi_1, \dots, \phi_r, \theta_1, \dots, \theta_v)'$ of $\{\xi_t\}$. The result of the automated decomposition in the first step together with the ARMA fitting in the second step is a fitted S-Semi-ARMA model that can be used for forecasting. Denote by $\kappa \in \mathbb{N}$ the forecasting horizon. As for example illustrated by [Beran and Ocker \(1999\)](#), forecasts based on $\hat{g}$ received via locally weighted regression are often obtained by extrapolating $\hat{g}$ linearly. In context of a fitted trend, whose values $\hat{g}(x_{n-1})$ and $\hat{g}(x_n)$ are known, a simple sample-based linear extrapolation would be:

$$\hat{g}(x_{n+\kappa}) = \hat{g}(x_n) + \kappa \cdot [\hat{g}(x_n) - \hat{g}(x_{n-1})]. \quad (17)$$

For $r = 1$ and given that $\hat{g}$ behaves locally approximately like a linear function in the proximity of x_n , $\hat{g}(x_{n+\kappa})$ as presented in (17) provides a reasonable forecast, as the point $\hat{g}(x_n)$ together with the increment $\hat{g}(x_n) - \hat{g}(x_{n-1})$ define fully how the locally linear $\hat{g}$ will evolve in the future for small κ . For $r = 3$, however, the increment in (17) may provide a too simplistic forecast in some cases, if $\hat{g}$ possesses too much curvature at the right boundary. Further study on this is left for elsewhere.

Analogously to the extrapolation of $\hat{g}$, given the assumption (A3), we extrapolate $\hat{s}$ constantly under consideration of $\delta = \lceil \kappa / p_s \rceil$, where $\lceil \cdot \rceil$ denotes once again the integer part:

$$\hat{s}(x_{n+\kappa}) = \hat{s}(x_{n+\kappa-p_s(\delta+1)}). \quad (18)$$

The effect of $\hat{g}$ and $\hat{s}$ is assumed to be negligible on the forecasts for $\{\tilde{\zeta}_t\}$ based on $\{\tilde{\zeta}_t\}$. Thus, common approaches, for example the method by Box et al. (2008), can be used to obtain forecasts for $\{\tilde{\zeta}_t\}$ using the ARMA model fitted to $\{\tilde{\zeta}_t\}$. After all forecasts have been obtained, the total point forecast is

$$\hat{y}_{n+\kappa} = \hat{g}(x_{n+\kappa}) + \hat{s}(x_{n+\kappa}) + \hat{\zeta}_{n+\kappa}. \quad (19)$$

Furthermore, in analogy to Schulz et al. (2026), the effect of the uncertainty in $\hat{g}_{n+\kappa} + \hat{s}_{n+\kappa}$ is omitted, which allows the forecasting intervals to be derived solely from the ARMA part of the S-Semi-ARMA model. Similar assumptions are made for example in Chapter 10.3 in Hyndman and Athanasopoulos (2021) in the context of regression models with ARMA errors. Let $\hat{a}_j, j = 0, 1, 2, \dots$, be the coefficients of the MA(∞) representation of the ARMA part of a fitted S-Semi-ARMA model and denote the estimated innovation variance by $\hat{\sigma}_\varepsilon^2$. The variance of the prediction error at time point $n + \kappa$ is then estimated approximately through $\widehat{\text{var}}(Y_{n+\kappa}) \equiv \widehat{\text{var}}(Y_{n+\kappa} | Y_n, \dots, Y_1) = \hat{\sigma}_\varepsilon^2 \sum_{j=0}^{\kappa-1} \hat{a}_j^2$, which can be considered to construct suitable forecasting intervals under normality of the innovation process $\{\varepsilon_t\}$. For easier notation, denote $\hat{m}(x_{n+\kappa}) = \hat{g}(x_{n+\kappa}) + \hat{s}(x_{n+\kappa})$. At confidence level α , commonly $\alpha = 0.95, 0.99$, define $\alpha^* = (1 - \alpha)/2$ and let Φ_x be the x -quantile of a standard normal distribution. Then under normality of $\{\varepsilon_t\}$ and by ignoring the uncertainty stemming from $\hat{m}(x_{n+\kappa})$, the forecasting interval for $Y_{n+\kappa}$ at level $100\alpha\%$ is

$$\left[\hat{m}(x_{n+\kappa}) - \Phi_{1-\alpha^*} \sqrt{\widehat{\text{var}}(Y_{n+\kappa})}, \hat{m}(x_{n+\kappa}) + \Phi_{1-\alpha^*} \sqrt{\widehat{\text{var}}(Y_{n+\kappa})} \right]. \quad (20)$$

Alternatively, for independent innovations, a simple bootstrap can be employed for a large enough number of observations, where $\hat{m}_{n+\kappa}$ is kept throughout and is added to bootstrapped future paths according to the fitted ARMA part. Assume that $\{\hat{\varepsilon}_t\}, t = 1, \dots, n$, are the ARMA residuals from the QMLE applied to $\{\tilde{\zeta}_t\}$. For each $j \in \{1, \dots, J^*\}$ with large enough J^* , for example $J^* = 1000$, and given a forecasting horizon κ , draw a random sample $\{\tilde{\varepsilon}_{j,t}\}, t = n+1, \dots, n+\kappa$, (with replacement) from $\{\hat{\varepsilon}_t\}, t = 1, \dots, n$. Let $\tilde{\zeta}_{j,t} = \tilde{\zeta}_t$ and $\hat{\varepsilon}_{j,t} = \hat{\varepsilon}_t$ for $t = 1, \dots, n$. Then, for each $j \in \{1, \dots, J^*\}$, compute

$$\tilde{\zeta}_{j,n+\kappa} = \sum_{i=1}^l \hat{\phi}_i \tilde{\zeta}_{j,n+\kappa-i} + \sum_{l=1}^v \hat{\theta}_l \hat{\varepsilon}_{j,n+\kappa-l} + \hat{\varepsilon}_{j,n+\kappa} \quad (21)$$

recursively starting at $c = 1$ and ending at $c = \kappa$, where $\hat{\phi}_1, \dots, \hat{\phi}_l$ and $\hat{\theta}_1, \dots, \hat{\theta}_v$ are the estimates of $\phi_1, \dots, \phi_l$ and $\theta_1, \dots, \theta_v$, respectively, and where unavailable future $\hat{\varepsilon}_{j,t}$ for $t = n+1, \dots, n+\kappa$ are replaced by the drawn $\tilde{\varepsilon}_{j,t}$ for the corresponding j . Considering for each j only $\tilde{\zeta}_{j,n+\kappa}$, this results in the collection of values $\{\tilde{\zeta}_{j,n+\kappa}\}, j = 1, \dots, J^*$. Define $Q_{x,n+\kappa}$ to be the sample x -quantile of the collection $\{\tilde{\zeta}_{j,n+\kappa}\}$. Then the bootstrapped forecasting interval for $Y_{n+\kappa}$, ignoring uncertainty from $\hat{m}(x_{n+\kappa})$, at level $100\alpha\%$ are given through

$$[\hat{m}(x_{n+\kappa}) + Q_{\alpha^*,n+\kappa}, \hat{m}(x_{n+\kappa}) + Q_{1-\alpha^*,n+\kappa}]. \quad (22)$$

Note that these bootstrapped forecasting intervals additionally also omit uncertainty in the estimated ARMA coefficients. To increase the computation speed, the collections of future bootstrap ARMA observations $\{\tilde{\zeta}_{j,n+\kappa}\}$ are obtained in parallel in `deseats` using the `future` package (Bengtsson, 2021) to ensure the same results under the same initial seed regardless of the number of cores being used in the parallel computation.

Note that both approaches for constructing forecasting intervals following an S-Semi-ARMA are implemented in the `predict` method of the `deseats` package. If instead an S-Semi-ARMA is fitted to log-data and suitable mean point forecasts on the retransformed scale are needed, the option `expo = TRUE` in the `predict` method can be employed. Bounds of forecasting intervals are then simply exponentiated. By default, the option `adjust.bias = TRUE` is implemented to adjust for biases in the retransformed point forecasts. For `method = "norm"`, the original-scale point forecasts are then computed as $\exp(\hat{y}_{n+\kappa}) \exp[\widehat{\text{var}}(Y_{n+\kappa})/2]$ following the relationship between a normal distribution (for the log-data forecasts) and a log-normal distribution (for the original-scale forecasts). For `method = "boot"`, we obtain $(J^*)^{-1} \sum_{j=1}^{J^*} \exp[\hat{m}_{n+\kappa} + \tilde{\zeta}_{j,n+\kappa}]$ for a given κ as the bootstrapped original-scale point forecast. For `adjust.bias = FALSE`, the log-data point forecasts are simply exponentiated.

3 Implementation in the deseats package

The deseats package in R comes with various functions for decomposing time series and various methods for plotting and forecasting. In particular, the package is designed to make the previously described automated semiparametric estimation and forecasting approaches available to a wide audience without presuming a deep understanding of the theory behind the methods. A common workflow using the package is planned to make use of a few select functions in a particular order and, due to the automated nature of the underlying DeSeaTS algorithm, without the need for numerous and complex adjustments in the function calls for most observed time series. Therefore here, the main functions relevant for a decomposition according to the DeSeaTS algorithm are briefly discussed. For information on the functions for decomposing time series using ordinary least squares, moving averages or even the BV4.1 base model, among other approaches, we refer the reader to the official documentation of the deseats package. To decompose a time series using automated locally weighted regression, the function `set_options` has to be called in order to define the general settings for the data-driven decomposition. It contains the following arguments.

- `order_poly` is the order of local polynomials for the trend. The default is a local cubic trend with `order_poly = 3`.
- `season` is the seasonal period of the observed time series. Using the default `season = NA_real_` leads to the seasonal period being detected automatically from the input time series of class `"ts"`.
- `kernel_fun` specifies the second-order kernel function to use as the weighting function. Available are the uniform (`"uniform"`), the epanechnikov (`"epanechnikov"`), the bisquare (`"bisquare"`), and the triweight (`"triweight"`) kernel. The default is epanechnikov.
- `bwidth` is the (relative) bandwidth to consider for the locally weighted regression. The default `bwidth = NA_real_` triggers the automatic bandwidth selection when calling `deseats()`.
- `boundary_method` clarifies which bandwidth method to use at boundary points. When setting `boundary_method` to `"extend"`, the same number of observations will be used for smoothing throughout even at boundary points, where the window of observations with non-zero weight is extended to the interior at boundary points. For `boundary_method = "shorten"`, the default, the number of observations considered stays the same at interior points but steadily decreases at boundary points when approaching the first or last observation time point.

An S4 class object is returned by this function that contains the stated information required for locally weighted regression.

After the general settings have been defined using `set_options`, the main function `deseats` can be called for identifying the trend and the seasonality in a univariate time series via locally weighted regression, which makes use of the following arguments.

- `y` is a time series object, preferably of class `"ts"`.
- `smoothing_options` should receive the output of `set_options`.
- `bwidth_start` controls the starting bandwidth for the algorithm. For local linear trends, the default is `bwidth_start = 0.1`, whereas it is `bwidth_start = 0.2` for local cubic trends.
- `inflation_rate` controls the inflation rate for the bandwidth selection algorithm. By default, for local linear trends the optimal (`"optimal"`) rate is used, while the naive one (`"naive"`) is considered for local cubic trends, which is fine for most applications (see also Section 4).
- `correction_factor` defines whether or not to use a bandwidth correction factor similar to the one considered in [Feng et al. \(2020\)](#) for estimating c_f . The default setting is `FALSE`, because we found a frequent overestimation of the bandwidth in the setting with seasonality in simulations.
- `autocor` is an argument that can be used to switch between different assumptions for the error process. By default, `autocor = TRUE` assumes autocorrelated errors, whereas setting it to `FALSE` runs the bandwidth selection under consideration of iid errors.
- `drop` tells the function what part of the observations to omit at each boundary in the bandwidth selection process. The default for local linear trends is `0.05`, while it is `0.1` for local cubic trends.
- `error_model` can be used to either set no further model assumptions on the errors (`error_model = "free"`), which is the default, or to already assume ARMA errors in the bandwidth selection (`error_model = "ARMA"`). This is only relevant when `autocor = TRUE`.
- `nar_lim` and `arma_mean` are only relevant for `error_model = "ARMA"`, because they allow to set further options on the ARMA selection and specification.
- `inf_criterion` controls the information criterion for the ARMA model selection for `error_model = "ARMA"`. Available are `"bic"` (the default) and `"aic"`, which correspond to the BIC and the AIC.

To improve the computation speed of the data-driven algorithm, it is fully implemented in the programming language C++ for a speed increase using the packages [Rcpp](#) ([Eddelbuettel and François,](#)

2011) and **RcppArmadillo** (Eddelbuettel and Sanderson, 2014). An extension of `deseats()` is provided by `s_semiarma()`, which allows one to apply `deseats()` to a time series object and then immediately afterwards fit an ARMA model (14) to the residuals. To specify the ARMA orders ι and v , the novel argument `arma_options` can be used. Alternatively, the aforementioned arguments `nar_lim`, `nma_lim`, `inf_criterion` and additionally `arma_mean` can be considered in `s_semiarma()` to adjust the automated ARMA selection process for the residuals. Both `deseats()` and `s_semiarma()` return S4 class objects with predefined `print()` and `plot()` methods that allow users to visualize the results easily from a vast selection of default options. For objects returned by `s_semiarma()`, a `predict()` method is built in to make multi-step point and interval forecasts as explained in the last subsection of Section 2.

The package makes use of an S4-class system throughout to benefit from the stricter and more robust control over objects and method dispatch compared to other object systems. As a consequence, all decomposition functions in the `deseats` package return consistently structured output objects and dispatch methods can be reused easily internally, thus keeping the structure of the package simple to maintain. Furthermore, the S4-object validation in `deseats` helps users to catch errors early in setting up the automated decomposition via `set_options()`. Furthermore, the package's main functions are built to work well with objects of class "ts", so that information such as the seasonal period can be read directly from the input object. In addition, this feature also simplifies the formatting of the x-axis in plots created through the package's plotting methods.

The `deseats` package also provides an animation function based on the **animation** (Xie, 2013) package that can be applied to objects returned by the function `deseats()` to visualize decomposition results, locally fitted values and the realized weights applied to $\{y_1, \dots, y_n\}$ implied by (10) at each estimation time point. Moreover, the package also provides an interactive **shiny** (Chang et al., 2025) app for manual decomposition of time series that can be called with the function `run_decomposition()`. Therefore, the package can also provide support in the teaching of locally weighted regression for decomposition of seasonal time series.

A disadvantage of the DeSeaTS algorithm and therefore of the `deseats` package is the inherent nature of locally weighted regression: at each observation time point, a separate weighted regression has to be conducted. Given the bandwidth selection algorithm presented in this article, regressions have to be run at each time point multiple times, until the bandwidth is found. Therefore, for common seasonal time series of multiple hundreds of observations, the `deseats()` function finishes within a few seconds. For extremely long time series with ten thousands of observations, the DeSeaTS algorithm may, however, take notably longer, because each single additional observation leads to multiple additional weighted regressions being conducted internally. Moreover, the bandwidth selection algorithm uses the asymptotically optimal bandwidth (13); therefore, reliable bandwidth estimates and consequently decomposition results via DeSeaTS cannot be expected for a small number of observations. In particular, for settings with a large number of local regressors, for example for $r = 3$ and $p_s = 12$, the algorithm can easily run into near-singular regressor matrices, if n is too small. From our experience, users should hence consider the DeSeaTS algorithm for time series with $n \geq 10p_s$ as the bare minimum, preferably with even larger n for the asymptotic approximation to hold.

4 Application

In the following, the DeSeaTS algorithm is at first applied via the `deseats` package to decompose seasonal real-world data, to conduct seasonal adjustments and to produce suitable point and interval forecasts of these series. Since STL (Cleveland et al., 1990) is a popular alternative for seasonal adjustment that also works for any positive integer-valued seasonal period, we employ `mstl()` of the forecast package (Hyndman and Khandakar, 2008) as a benchmark. Benchmark point forecasts are generated by the corresponding `forecast()` method for the output of `mstl()` with the option `method = "arima"`, so that the total point forecasts are generated as the point forecasts from an automatically selected ARIMA model for the seasonally adjusted series plus the estimated seasonal component by STL from the last year or week, respectively. Possible retransformations are applied afterwards and bias adjustments of retransformed point forecasts are implemented. Thereinafter, two simulation studies follow. The first study is done to show that the DeSeaTS bandwidth selection algorithm works as intended. The second one compares the quality of seasonal component estimation among various popular methods, such as DeSeaTS, STL, X13 and TRAMO-SEATS, under different setups.

4.1 Application to real-world data

The `deseats` package is applied to three different observed time series in order to conduct a seasonal adjustment. The series are the quarterly real log-GDP of the US (GDP in billions of USD) from the

first quarter of 1947 to the fourth quarter in 2019 (LGDP) with $p_s = 4$, the monthly number of US persons that are not part of the US labor force (in millions of persons) from January 1990 to December 2019 (NLF) with $p_s = 12$, and the daily log-transformed confirmed COVID-19 cases in Germany (in thousands of cases) from Tuesday of calendar week 22 of year 2021 to Tuesday of week 48 of year 2021 (LCO) with $p_s = 7$. The sources are the online databank of the Federal Reserve Bank of St. Louis (FRED) for the series LGDP and NLF, and the COVID-19 Data Hub for LCO. Calendar effects, which are still present in the series, are ignored for simplicity, because the effects are assumed to be negligible for the selected series. The examples are selected from different countries, research areas and are with various seasonal periods. They are displayed via grey solid lines in Figure 1. The series without log-transformation for LGDP and LCO are shown in Figure 2. All three series are, on their original scale, available in the deseats package and already formatted as time series objects of class "ts".

For the purpose of comparison, automated STL via `mstl()` of the forecast package is employed in addition to the DeSeaTS algorithm, because STL is applicable also when p_s is different from 4 and 12. For DeSeaTS, the default settings are considered, so that local cubic trends are used throughout, with the exception of the LGDP series, where a local linear trend will be considered, as it leads to a slightly improved seasonal adjustment in this case. We find that a local cubic trend is often favorable due to one particular reason: the algorithm with this setting is found to usually return better decomposition results for combinations of smaller sample sizes with complex trends and is thus often the safer choice. Nonetheless, please note that the DeSeaTS algorithm usually also works well for $r = 1$, especially for large n , as will be shown in the simulation studies in Section 4. Application code will only be given for DeSeaTS; codes regarding STL as well as the code for the addition of STL's results to the figures below is moved to the accompanying R script, so that the focus within this article stays on the application of the deseats package. Requirement for running the codes is a one-time installation of the deseats package via `install.packages("deseats")`.

```
library(deseats)      # Attach the deseats-package
LGDP <- log(GDP)      # Get data
NLF <- NOLABORFORCE
LCO <- log(COVID)
sm_options3 <- set_options()          # Local cubic trend
sm_options1 <- set_options(order_poly = 1) # Local linear trend
LGDP_deseats <- deseats(LGDP, sm_options1) # Run DeSeaTS
NLF_deseats <- deseats(NLF, sm_options3)
LCO_deseats <- deseats(LCO, sm_options3)
```

The previous code shows the individual-level application of `deseats()` to the example data. At first, we obtain the time series objects from the deseats package and conduct log-transformations on the objects GDP and COVID, so that the log-transformed series have a roughly constant variation around their trends (see also Figure 1). Now LGDP, NLF and LCO are each assumed to follow (at least approximately) an additive component model as specified in Equation (1). Note that the objects GDP, NOLABORFORCE and COVID in the deseats package are already formatted as time series objects of class "ts" with correctly specified observation frequency, so that the frequency settings align with the corresponding p_s for each series. This simplifies the application of the deseats package notably, as its functions automatically read p_s from the input "ts" object. After setting up the input objects, we specify the smoothing options for the bandwidth selection algorithm with `set_options()`. The default uses a local cubic trend, which can be adjusted to a local linear trend with the setting `order_poly = 1`. By default, an epanechnikov kernel ($\mu = 1$ in the weighting function (5)) is implemented (`kernel_fun = "epanechnikov"`), but can be set to a few other options as well. Afterwards, we call the function `deseats()` and pass the input time series as well as the objects `sm_options1` and `sm_options3` with the smoothing settings to the function. `deseats()` now automatically estimates the bandwidth for locally weighted regression from the input data and immediately applies the selected bandwidth to decompose the input series. Key features, such as the estimated bandwidth, the estimated trend, the estimated seasonal component, the residuals and the seasonally adjusted series, can be accessed from the output of `deseats()` by applying the methods `bwidth()`, `trend()`, `season()`, `residuals()` and `deseasonalize()`, respectively.

A decomposition plot and other selectable plots of the estimation objects can be created using a designated plot-method. In the following, a figure of the estimated trend and seasonal components following DeSeaTS for the three example series is generated. An analogous autoplot-method is provided by the deseats package for the framework of `ggplot2` (Wickham, 2016). The option `which = 5`, selects a plot that shows the input data together with the estimated trend and the estimated seasonal component. All available plotting options can be seen in the package manual or in the R console by calling the plot-method without setting the argument `which`. As a further aspect, note that by default the estimated seasonal component is centered around zero in the plot; however, using the argument

`s_around`, the estimated seasonal component can be shifted so that all elements in the plot are clearly visible.

```
plot(LGDP_deseats, which = 5, s_around = 3.9, main = "(a) LGDP", xlab = "Year")
plot(NLF_deseats, which = 5, s_around = 55, main = "(b) NLF", xlab = "Year")
plot(LCO_deseats, which = 5, s_around = -2.2, main = "(c) LCO",
     xlab = "Calendar week of year 2021")
```

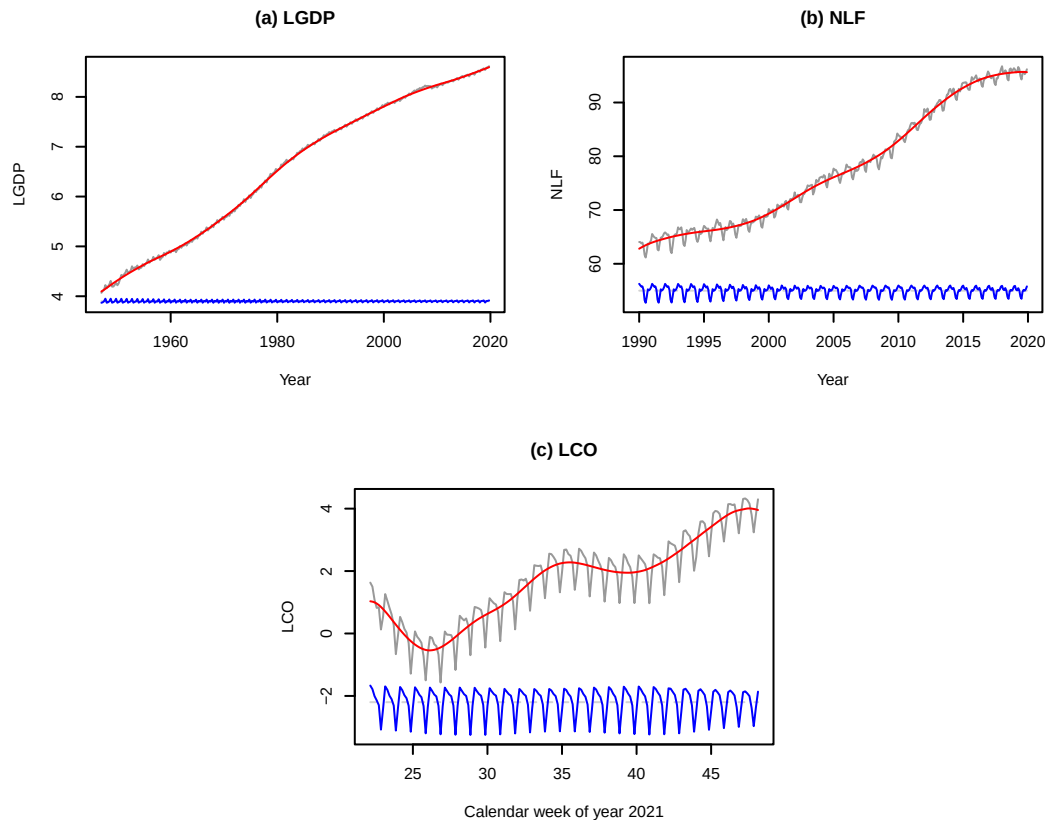


Figure 1: The estimated trend and seasonal components for the LGDP, NLF and LCO data following DeSeaTS. The estimated seasonal components are shifted for better visibility.

The plots containing the observed series, the estimated trends and the estimated seasonal components are displayed in Figure 1. At least for the given examples, deseats seems to provide trend estimates that smoothly follow along the levels of the series, and slowly changing seasonality estimates. For the LGDP series, the trend smooths over the small drop at around year 2008. Note, however, that for the three series LGDP, NLF and LCO do not show any extreme jumps or structural breaks, they do not have highly complex trends and the seasonal pattern does not change very quickly over time; those are, nonetheless, characteristics often observed in real data and deseats may struggle to provide good decomposition results in such cases. However, robustifying the DeSeaTS procedure for more complex scenarios is left for future research.

The seasonally adjusted series are obtained in the following using the aforementioned method `deseasonalize()`. In case of the LGDP and LCO data, exponentiating leads back to the original scale of the time series objects GDP and COVID within deseats.

```
GDP_sa <- exp(deseasonalize(LGDP_deseats))
NLF_sa <- deseasonalize(NLF_deseats)
COVID_sa <- exp(deseasonalize(LCO_deseats))
ts.plot(GDP, GDP_sa, col = c("grey60", "blue"), main = "(a) Real GDP of the US",
        xlab = "Year", ylab = "Billions of USD")
ts.plot(NLF, NLF_sa, col = c("grey60", "blue"), ylab = "Millions of persons",
        main = "(b) US persons not in the labor force", xlab = "Year")
```



```
ts.plot(COVID, COVID_sa, col = c("grey60", "blue"),
        xlab = "Calendar week of year 2021", ylab = "Thousands of cases",
        main = "(c) Newly confirmed COVID-19 Cases in Germany")
```

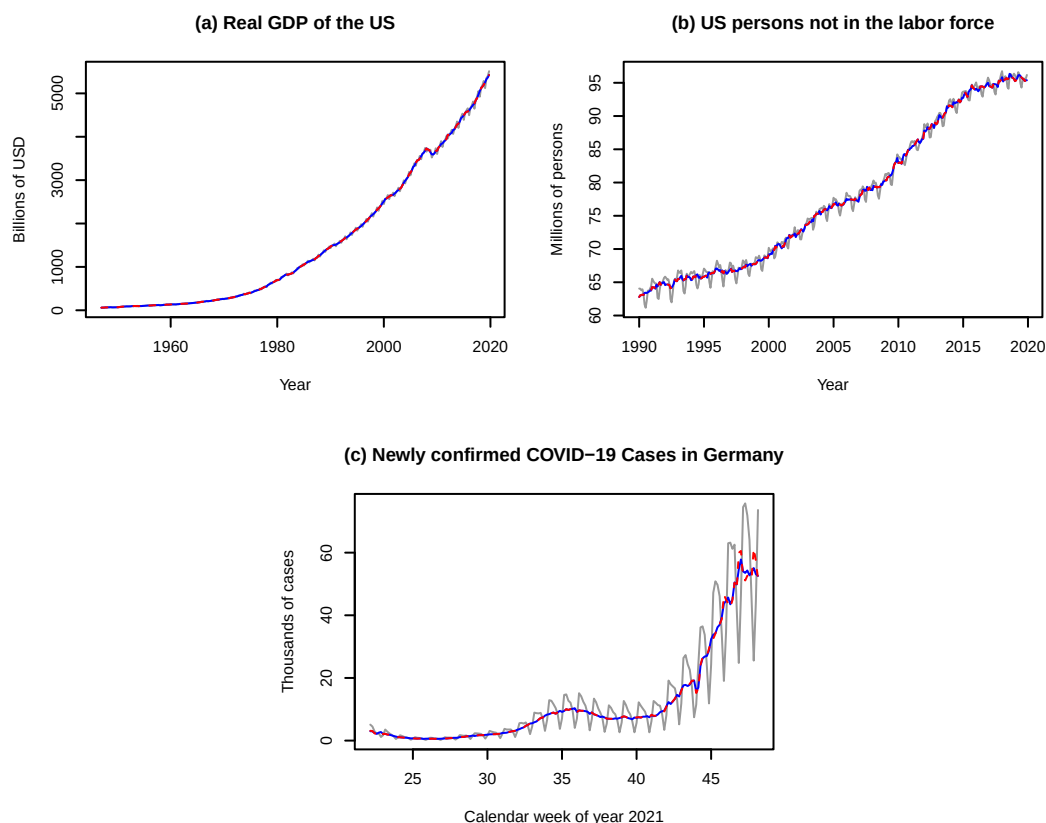


Figure 2: The original example series (grey and solid) together with the seasonally adjusted series following DeSeaTS (blue and solid). The seasonally adjusted series following automated STL decompositions (red and dashed) are displayed as a benchmark.

The original-scale series (solid and grey) together with the seasonally adjusted series by DeSeaTS (solid and blue) are shown in Figure 2. As a benchmark, the seasonally adjusted series in accordance with an automated application of STL via `mstl()` of the forecast package (dashed and red) are displayed as well. As a first impression, the differences between the two seasonally adjusted series seem to be marginal in all three cases. Near the end of the COVID series, however, the series obtained by STL includes a few clear peaks that have been reduced in the adjusted series following DeSeaTS. Interested readers may also want to compare these results to the seasonally adjusted series obtained by X13 and TRAMO-SEATS, which are the international standard procedures for seasonal adjustment in the European Union. The adjusted series are displayed in Figure 7 in the appendix and were obtained by applying the functions `x13()` and `tramoseats()` (with default settings) of the RJDemetra package to the objects GDP and NOLABORFORCE directly. An application of those functions to the COVID data is not possible, as they are designed for quarterly and monthly series.

Entire S-Semi-ARMA models are fitted by applying `s_semiarima()` to the series of interest. Therein, `deseats()` is applied initially for decomposing the data; afterwards, an ARMA model is automatically selected for the resulting residual series in accordance with the BIC. Alternatively, implementing `inf_criterion = "aic"` uses the AIC for the ARMA model identification instead. In the following, we furthermore restrict the AR and MA orders to be selected from the values 0 to 2 via `nar_lim <- c(0, 2)` and `nma_lim <- c(0, 2)`, so that the selected ARMA models for the residuals do not become too complex.

```
nar_lim <- nma_lim <- c(0, 2)
model_LGDP <- s_semiarima(LGDP, sm_options1, nar_lim = nar_lim, nma_lim = nma_lim)
model_NLF <- s_semiarima(NLF, sm_options3, nar_lim = nar_lim, nma_lim = nma_lim)
model_LCO <- s_semiarima(LCO, sm_options3, nar_lim = nar_lim, nma_lim = nma_lim)
```


Table 1: Selected bandwidths for the three series LGDP, NLF and LCO, estimated ARMA parameters and corresponding standard errors for DeSeaTS residuals and p -values for QS tests and Friedman tests applied to S-Semi-ARMA residuals.

Series	Bandwidth	$\hat{\phi}_1$	$se(\hat{\phi}_1)$	$\hat{\phi}_2$	$se(\hat{\phi}_2)$	$\hat{\theta}_1$	$se(\hat{\theta}_1)$	p -value (QS)	p -value (Friedman)
LGDP	0.0806	1.5442	0.0935	-0.6545	0.0825	-0.4083	0.1121	0.3140	0.9623
NLF	0.2113	0.6885	0.0380	—	—	—	—	1.0000	0.9992
LCO	0.1256	—	—	—	—	0.5485	0.0542	1.0000	0.9958

@par_model gives access to the fitted ARMA models for the residual series. In Table 1, the estimation information is summarized for the three series LGDP, NLF and LCO. Therein, the three selected bandwidths by DeSeaTS are stated, which range from 0.0806 for LGDP to 0.2113 for NLF. Furthermore, the estimated ARMA parameters and corresponding standard errors are displayed as selected in the fitted S-Semi-ARMA models. An ARMA(2, 1) model was fitted to the DeSeaTS residuals for LGDP with a strongly positive lag-1 AR coefficient estimate of 1.5442 and negative lag-2 AR and lag-1 MA coefficient estimates -0.6545 and -0.4083. An AR(1) model was picked by BIC for the DeSeaTS residuals of NLF with a strongly positive coefficient estimate 0.6885. Conversely, an MA(1) model was selected for the LCO residuals with strongly positive MA coefficient estimate 0.5485. All estimated coefficients are statistically significant at least at the 5% level of significance.

In addition, QS tests, i.e. a special form of the Ljung-Box test at seasonal lags (Ljung and Box, 1978), and Friedman rank tests (Friedman, 1937, 1939) were applied to the S-Semi-ARMA residuals fitted to the three series. For the QS tests, we check for the statistical significance of autocorrelations at the first and second seasonal lags; for the Friedman tests, we do not consider residuals in incomplete years / weeks. The p -values obtained are also stated in Table 1. According to the Friedman tests, we clearly cannot reject the null hypothesis of no stable seasonality in any of the cases. Simultaneously, the null hypothesis of no seasonality cannot be rejected in all three cases following the QS tests either, as the smallest obtained p -value is 0.3140 in the context of the LGDP series. Since the test decisions align, we conclude that the removal of the seasonality from the three series was successful.

4.2 Forecasting examples

Following the fitted S-Semi-ARMA models, point and interval forecasts can be created as discussed in the last subsection of Section 2. For simplicity, we use method = "boot" in the calls to the predict() method to compute the interval forecasts from a bootstrap. For reproducibility, a seed needs to be set via set.seed(). Since extrapolating the trend and seasonal component estimates into the future is highly unstable if the forecasting horizon is too large, we produce forecasts for one cycle of the seasonal pattern. To exponentiate the forecasts back to the scale of the original GDP and COVID objects in deseats, we implement expo = TRUE in these two cases. By default (adjust.bias = TRUE), the point forecasts are adjusted for bias upon retransformation to the original scale.

```
set.seed(1)
fc_GDP_DS <- predict(model_LGDP, n.ahead = 4, method = "boot", expo = TRUE)
fc_NLF_DS <- predict(model_NLF, n.ahead = 12, method = "boot")
fc_COVID_DS <- predict(model_LCO, n.ahead = 7, method = "boot", expo = TRUE)
```

A designated plot() method allows one to produce quick figures of the forecasts. The corresponding forecasting plots for the three examples are displayed in Figure 3.

```
plot(fc_GDP_DS, main = "(a) Real GDP of the US", xlab = "Year",
     ylab = "Billions of USD", xlim = c(2010, 2020.75), ylim = c(3500, 6000))
plot(fc_NLF_DS, main = "(b) US persons not in the labor force", xlab = "Year",
     ylab = "Millions of persons", xlim = c(2010, 2020.917), ylim = c(80, 97.2))
plot(fc_COVID_DS, main = "(c) Newly confirmed COVID-19 Cases in Germany",
     xlab = "Calendar week of year 2021", ylab = "Thousands of cases",
     xlim = c(45, 49.143), ylim = c(10, 85))
```

The point forecasts (solid and blue) extend the series logically into the future, and the 95%- and 99%-forecasting intervals give a first impression of the uncertainty related to each point forecast. As a benchmark, the point forecasts (red and dashed) from an automated ARIMA model with added

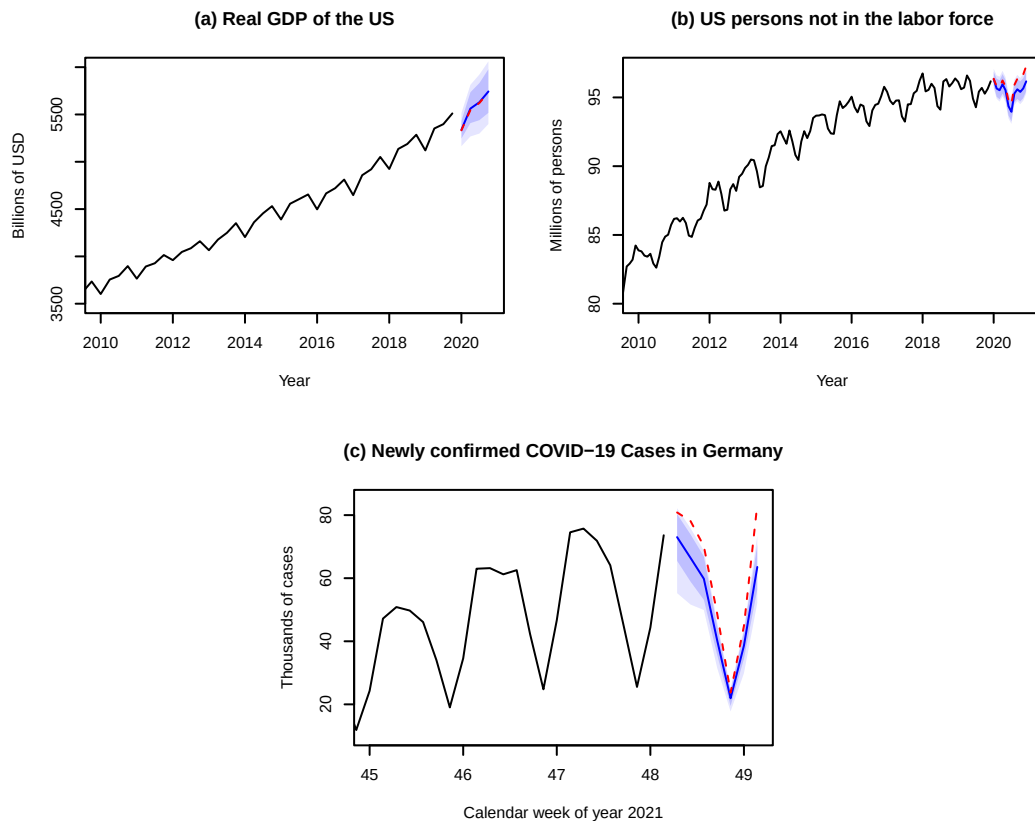


Figure 3: S-Semi-ARMA point and 95% / 99% interval forecasts (blue) together with automated ARIMA-based STL point forecasts (red) as a benchmark.

seasonality estimated via STL by `mst1()` of the forecast package are displayed as well; the extra code can be obtained from the accompanying material. For the GDP series (Figure 3(a)), the point forecasts are close to each other. In fact, the point forecasts of the benchmark method lie within the 95%-forecasting intervals by the S-Semi-ARMA models in both cases. For the labor force series (Figure 3(b)), the point forecasts align roughly initially, but after a few months they diverge, leading to the benchmark forecasts leaving the forecasting intervals by S-Semi-ARMA after 12 months. For the COVID-19 series (Figure 3(c)), the point forecasts do not align for most of the future time points. The benchmark forecasts are almost always larger than the upper bound of the 99%-forecasting interval of the S-Semi-ARMA, with the exception of the first and fourth future day. The reason for the difference in point forecasts in the last two cases is that the S-Semi-ARMA forecast, even with linear extrapolation of the trend, also considers the curvature of the fitted trend at the end points to some degree, which is usually more pronounced if a local cubic trend is considered. A comparison of quantified forecasting quality between various methods is, however, not within the scope of this article.

4.3 Simulation studies

Two different simulation studies are conducted hereinafter. The first study is done to analyze whether the proposed extended DeSeaTS algorithm estimates the smoothing bandwidth suitably both under the assumptions stated in this article and under a slowly changing seasonal component, i.e. under a mild relaxation of assumption (A3) in the first subsection of Section 2. The same simulation study also researches the estimation quality of the trend, the seasonal component and of the trend-seasonal component by DeSeaTS. The second simulation study compares the quality of seasonal component estimation of DeSeaTS in direct comparison to other established seasonal adjustment approaches under various settings.

Bandwidth study

The first simulation study is conducted to verify that the DeSeaTS algorithm works well in practice for series with trends, seasonal components and errors of varying degrees of complexity. We consider the

trend functions

$$\begin{aligned} g_1(x_t) &= 5x_t + 1, \\ g_2(x_t) &= 2.5 \tanh[5(x_t - 0.4)] + 3.5 \quad \text{and} \\ g_3(x_t) &= 3.2x_t + 0.92 \sin[3.2\pi(x_t - 0.2)] + 1.85, \end{aligned}$$

i.e. $g_1(x_t)$ is a simple linear trend, $g_2(x_t)$ represents a possible simple trend for macroeconomic growth and $g_3(x_t)$ is a more complex trend not necessarily representative of the progression of macroeconomic time series. Using these definitions, all trends range roughly in between 1 and 6 for $x_t \in [0, 1]$. In addition, two error processes are considered: $\{e_{1,t}\}$ is an AR(1)-process with AR-coefficient $\phi_{1,1} = 0.75$ and innovation variance $\sigma_{e,1}^2 = 0.083^2$ and $\{e_{2,t}\}$ follows an AR(2)-process with $\phi_{1,2} = 1.05$, $\phi_{2,2} = -0.3$ and $\sigma_{e,2}^2 = 0.06^2$. Throughout, innovations are independently normally distributed with mean zero and the corresponding innovation variance. The seasonal components considered here are $s_{1,4}(x_t)$ and $s_{2,4}(x_t)$ for quarterly data as well as $s_{1,12}(x_t)$ and $s_{2,12}(x_t)$ for monthly data. We define

$$\begin{aligned} s_{1,4}(x_t) &= \begin{cases} -0.39, & \text{for } [(t-1) \bmod 4] = 0, \\ 0.01, & \text{for } [(t-1) \bmod 4] = 1, \\ -0.04, & \text{for } [(t-1) \bmod 4] = 2, \\ 0.42, & \text{for } [(t-1) \bmod 4] = 3, \end{cases} \quad \text{and} \\ s_{1,12}(x_t) &= \begin{cases} -0.39, & \text{for } [(t-1) \bmod 12] = 0, \\ 0.05, & \text{for } [(t-1) \bmod 12] = 1, \\ -0.01, & \text{for } [(t-1) \bmod 12] = 2, \\ -0.16, & \text{for } [(t-1) \bmod 12] = 3, \\ -0.31, & \text{for } [(t-1) \bmod 12] = 4, \\ -0.24, & \text{for } [(t-1) \bmod 12] = 5, \\ -0.25, & \text{for } [(t-1) \bmod 12] = 6, \\ -0.10, & \text{for } [(t-1) \bmod 12] = 7, \\ 0.50, & \text{for } [(t-1) \bmod 12] = 8, \\ 0.62, & \text{for } [(t-1) \bmod 12] = 9, \\ 0.34, & \text{for } [(t-1) \bmod 12] = 10, \\ -0.05, & \text{for } [(t-1) \bmod 12] = 11. \end{cases} \end{aligned}$$

$s_{1,4}(x_t)$ and $s_{1,12}(x_t)$ are hence exactly periodic. In addition, we have $s_{2,4}(x_t) = f_{sc}(x_t)s_{1,4}(x_t)$ and $s_{2,12}(x_t) = f_{sc}(x_t)s_{1,12}(x_t)$, where

$$f_{sc}(x_t) = 0.83[1 + 0.1 \sin(50\pi x_t/9) + 0.2 \cos(3\pi x_t)],$$

so that $s_{2,4}(x_t)$ and $s_{2,12}(x_t)$ have slowly changing amplitudes over time.

Furthermore, for the simulation, we check for $n_1 = 15p_s$, $n_2 = 30p_s$ and $n_3 = 60p_s$, $p_s = 4$ for quarterly data and $p_s = 12$ for monthly data, as numbers of observations, so that we are simulating data for 15, 30 and 60 years, respectively. Since DeSeaTS considers an asymptotically optimal bandwidth (13) internally, the smallest number of observations considered, n_1 , is not set to an overly small value. In the following, the asymptotically optimal bandwidth is calculated in each of the setting combinations, however, whenever $s_{2,4}$ or $s_{2,12}$ are used, the values can only be interpreted as rough approximations, as Equation (13) holds only under exact periodicity or constantly linearly changing seasonality. Moreover, for $g_1(x_t)$ we have $h_A \rightarrow \infty$. Thus, in these situations, we consider $h_A = 0.49$, since this is the maximum value for the bandwidth estimate the algorithm implemented in deseats can return. Given the trends g_1 , g_2 and g_3 , the seasonal components $s_{1,4}$, $s_{2,4}$, $s_{1,12}$ and $s_{2,12}$, the error processes $\{e_{1,t}\}$ and $\{e_{2,t}\}$, and the numbers of observations n_1 , n_2 and n_3 , we check 72 constellations. For each of them we produce 1000 simulated series. The DeSeaTS bandwidth selection algorithm and automated decomposition is used under $r = 1$ and $r = 3$ with both optimal and naive inflation rates. By default, $c_b = 0.05$, $d_b = 0.95$ and a starting bandwidth of 0.1 are employed whenever $r = 1$, and $c_b = 0.1$, $d_b = 0.9$ are considered with the starting bandwidth 0.2 for $r = 3$. Furthermore, an epanechnikov kernel ($\mu = 1$) is considered throughout. We run the DeSeaTS algorithm to estimate the bandwidth for each simulated series and obtain the sample mean, the sample standard deviation and the sample mean squared error (MSE) of the bandwidth estimates for each setting. Furthermore, the mean MSE is obtained for the trend estimates ($\hat{g}$), the seasonality estimates ($\hat{s}$) and for the trend estimates together with the seasonality estimates ($\hat{m} = \hat{g} + \hat{s}$).

In Figures 4 and 5, examples of simulated series under these settings are illustrated together with the true trends and seasonal components and their estimates. For simplicity, the illustrated simulated series were obtained with n_1 and the same realization of $\{e_{1,t}\}$. The estimation results were obtained using deseats considering $r = 3$ with `inflation_rate = "naive"` and `drop = 0.1`, i.e. the default settings for $r = 3$ within deseats. It is apparent that Subfigures (b), (d) and (f) in Figures 4 and 5 show a slow change in the seasonal component s_2 over time, resulting in slowly changing phases of smaller

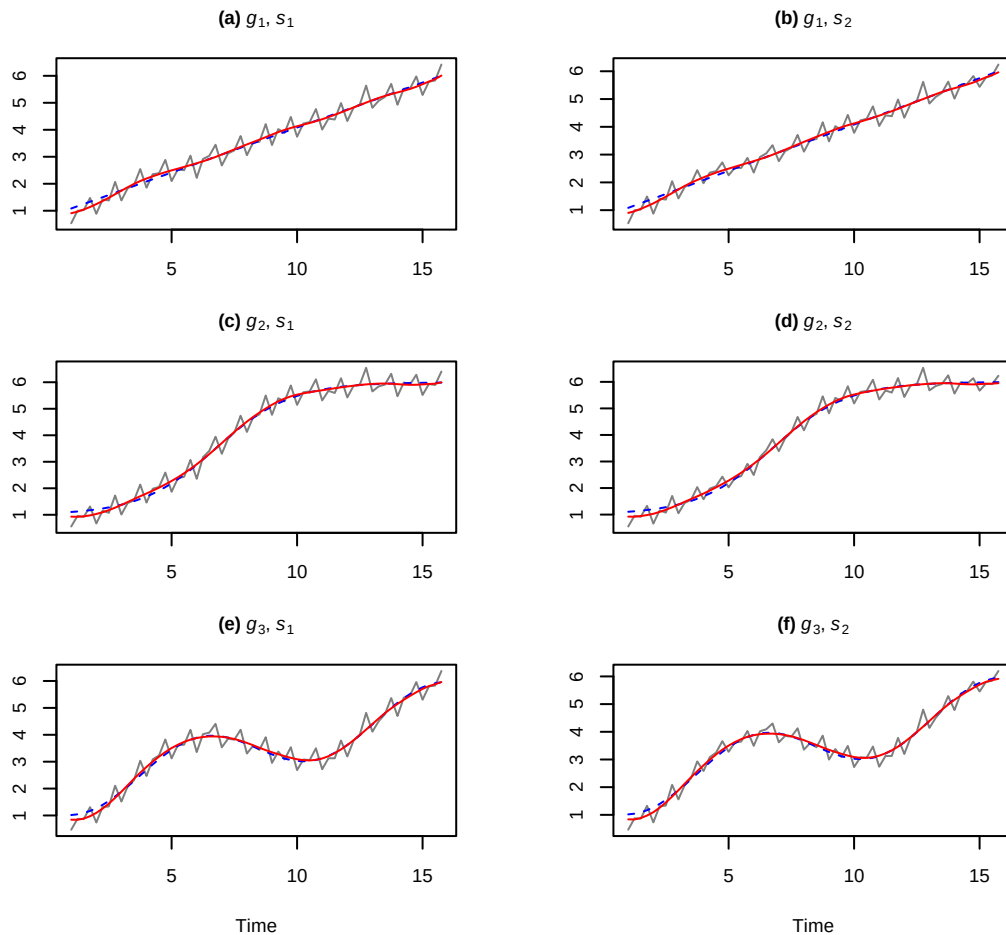


Figure 4: Examples of quarterly simulated series (grey and solid) together with the true underlying trends (blue and dashed) for n_1 and the estimated trends (red and solid) using the DeSeaTS framework with $r = 3$. All examples are with number of observations n_1 and the same simulated errors from $\{e_{1,t}\}$.

and larger seasonal variation. Conversely, the seasonal component s_1 implemented in Subfigures (a), (c) and (e) remains exactly periodic throughout. Furthermore, the increase in complexity of the trends from g_1 over g_2 to g_3 is visible in Figure 4, because of the rise in the number of (local) minima and maxima and turning points. Regarding the estimated trends displayed as solid red lines in Figure 4, we can observe that, despite the low number of observations, the true trends, which are shown as blue dashed lines, are captured well in all scenarios, including the Subfigures 4(a) and 4(b), where the simple linear trend g_1 was considered. Similarly, the estimated seasonal components illustrated as solid red lines in Figure 5 are close to the true seasonal components, shown as grey solid lines, throughout. Even the slow change in the seasonal component s_2 is captured to some degree in the estimates in Subfigures (b), (d) and (f). However, it is observable at the very beginning and the very end of the time period that the seasonality estimates by DeSeaTS cannot seem to follow the changes of the true component as quickly, as the true component shows its quickest changes during those periods.

Figures 8 to 15 in the appendix show boxplots of the estimated bandwidths for the various simulation settings. Analogously, Tables 2 to 5 in the appendix show the sample mean of the estimated bandwidths $\text{Mean}(\hat{h})$, the sample standard deviations of the estimated bandwidths $\text{SD}(\hat{h})$, the sample mean squared error $\text{MSE}(\hat{h})$ and the average sample MSEs over all time points and iterations for the estimated trend ($\text{MSE}(\hat{g})$), for the estimated seasonal component ($\text{MSE}(\hat{s})$) and for the estimated trend-seasonal component ($\text{MSE}(\hat{m})$). MSEs and $\text{SD}(\hat{h})$ values in the tables were multiplied by 1000. The given asymptotic bandwidths h_A are only valid under the exactly periodic component s_1 ; they are only rough approximations under s_2 .

We can conclude that DeSeaTS under $r = 3$ can sometimes struggle to select the optimal bandwidth under a very simplistic simple linear trend g_1 , as the bandwidth is clearly underestimated. However, the MSE of the bandwidth estimator still reduces slightly for an increase in the number of observations

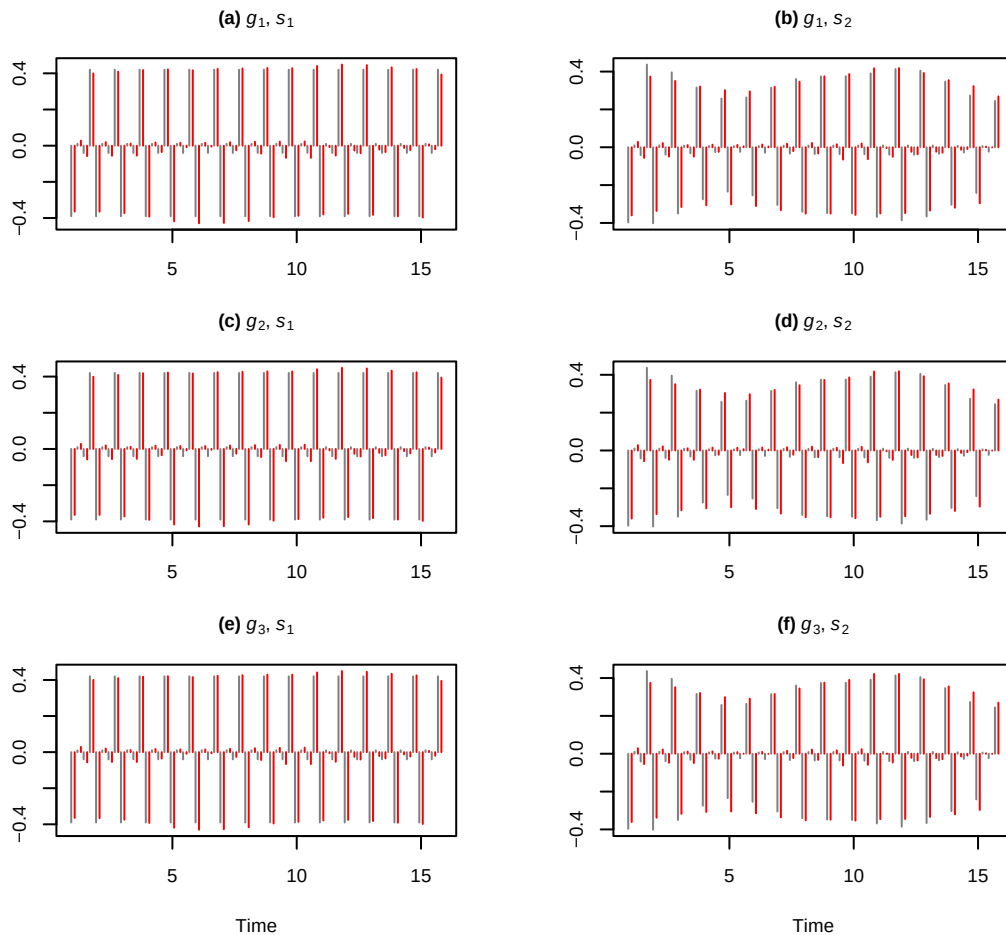


Figure 5: True underlying seasonal components (grey and solid) of the quarterly simulated examples and the estimated seasonal components (red and solid) using the DeSeaTS framework with $r = 3$. On the time axis, the true values are slightly shifted to the left and the estimates slightly to the right for better comparability. All examples are with number of observations n_1 and the same simulated errors from $\{e_{1,t}\}$.

and the average MSEs of the estimated components still clearly decrease. Furthermore, considering more complex trends g_2 and g_3 , the bandwidth selection seems to be most stable under an optimal inflation rate for $r = 1$ and under a naive inflation rate for $r = 3$. In some instances, $r = 1$ with naive inflation rate also works well for large enough n , but usually comes with a larger $\text{SD}(\hat{h})$. Consistency of the bandwidth selection is apparent from the decrease in $\text{MSE}(\hat{h})$ for an increase in the number of observations. $\text{SD}(\hat{h})$ can sometimes increase when increasing the number of observations from n_1 to n_2 ; we presume that this is due to the relatively small number of observations n_1 and the hence more unstable behavior of the bandwidth selector, as asymptotics may not yet be a good enough approximation under n_1 . In addition, in some few instances, for example $r = 3$ with g_3, s_1 and $\{e_{1,t}\}$ in the quarterly setting, one may yet need a larger number of observations, so that the bandwidth estimator may become even closer to the true h_A . This is because quarterly data contains fewer observations to estimate the sum of autocovariances $2\pi c_f$ from than monthly data (for the same amount of years). Likewise, $\hat{h}$ is expected to converge more slowly to h_A for $r = 3$ than for $r = 1$ (see Equation (16)). Converging behavior in this context is observable nonetheless. Importantly, the bandwidth selector also seems to converge to the (approximated) true h_A for s_2 in all cases with an increase in n .

Due to the decrease in $\text{MSE}(\hat{g})$, $\text{MSE}(\hat{s})$ and $\text{MSE}(\hat{m})$ by approximately constant rates, we have a strong indication that DeSeaTS leads to consistent estimators of the components. This implies also consistency of the error estimator $\hat{\xi}_t = y_t - \hat{m}(x_t)$ and consequently of ARMA parameters obtained from $\hat{\xi}_t$ instead of the non-observable ξ_t .

Study on seasonality estimation

We consider the following additive base model:

$$Y_{ijk,t} = \alpha G_{i,t} + \beta S_{j,t} + \gamma E_{k,t}$$

with $G_{i,t}$, $i = 1, 2$, as a (deterministic or stochastic) trend, $S_{j,t}$, $j = 1, 2, 3$, as a (deterministic or stochastic) seasonal component, and $E_{k,t}$, $k = 1, 2, 3$ as a stochastic error. $\alpha, \beta, \gamma > 0$ are real constants. $G_{1,t}$ is the same as $g_3(x_t)$ from the first simulation study but normalized to have (sample) mean zero and (sample) variance one and $G_{2,t}$ is obtained from a random walk with iid $N(0.05, 1)$ innovations and again normalized. $S_{1,t}$ is $s_1(x_t)$ from the first simulation study but normalized to have (sample) mean zero and (sample) variance one, $S_{2,t}$ is the normalized analogue of $s_2(x_t)$ from the first simulation study, and $S_{3,t}$ is obtained from $S_{3,t} = -\sum_{l=1}^{p_s-1} S_{3,t-l} + e_t$ and then normalized, where $e_t \stackrel{\text{iid}}{\sim} N(0, 0.025)$.

Furthermore, we have $E_{1,t} \stackrel{\text{iid}}{\sim} N(0, 1)$, $E_{2,t}$ being governed by an AR(1) process with AR-parameter 0.75 and innovation variance $1 - 0.75^2$, and $E_{3,t}$ follows an AR(2) process with parameters 1.05 and -0.3 as well as the innovation variance $0.7(1.3^2 - 1.05^2)/(1.3)$. All three error variants are hence with (theoretical) mean zero and (theoretical) variance one. Note that $G_{1,t}$ is deterministic, while $G_{2,t}$ is stochastic. $S_{1,t}$ and $S_{2,t}$ are both deterministic, while $S_{1,t}$ is exactly periodic and $S_{2,t}$ is changing slowly over time. In contrast, $S_{3,t}$ is stochastic. Example realizations of the stochastic components $G_{2,t}$ and $S_{3,t}$ and their quickly changing behavior can be observed in Figure 6.

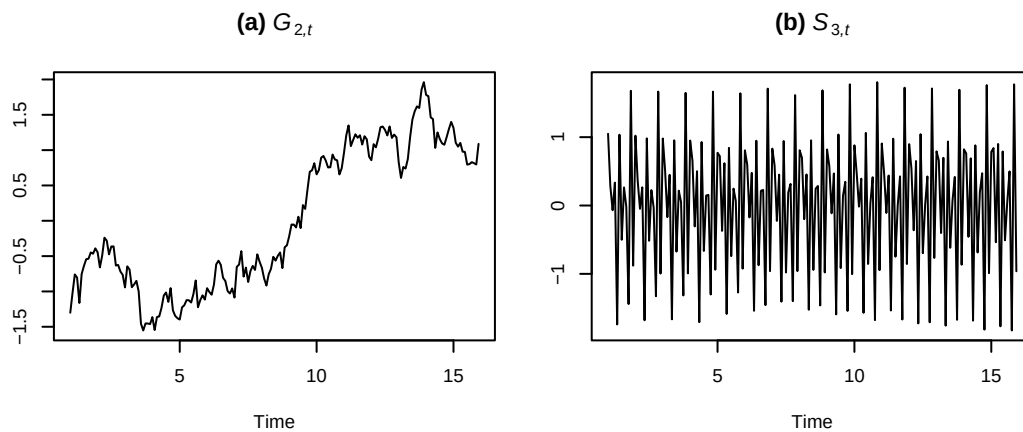


Figure 6: A realization of the stochastic trend $G_{2,t}$ and of the stochastic seasonal component $S_{3,t}$ for 15 years.

Since DeSeaTS uses an asymptotically valid bandwidth formula internally, we consider for the comparison $n_1 = 15p_s$, $n_2 = 30p_s$ and $n_3 = 60p_s$ observations; numbers of observations smaller than n_1 are not considered here and should be studied elsewhere. For simplicity, we only consider $p_s = 4, 12$ as the two most frequent seasonal periods in macroeconomic time series. In addition, we keep $\alpha = 1$ and $\beta = 0.4$ constant throughout and consider $\gamma = 0.1, 0.2$. Given two trends, three seasonal components, three error variations, three numbers of observations, two seasonal periods and two settings of γ , we arrive at 216 different setting combinations. For each of them, we simulate 1000 series and apply different methods to them.

In the following, the abbreviations DS, X13, TS, STL, TBATS and MA are used for DeSeaTS, X13-ARIMA, TRAMO-SEATS, STL, TBATS and the `decompose()` function, respectively. X13 and TS will be implemented using the functions `x13()` and `tramoseats()` of the `RJDemetra` package, where, aside from turning off calendar adjustments, default settings will be employed. STL will be applied by means of the automated function `mst1()` of the `forecast` package, and TBATS will be implemented following the function `tbats()` within the same package. DeSeaTS will be tested with different algorithm settings, so that the abbreviations $DS_{1,\text{opt}}$, $DS_{1,\text{nai}}$, $DS_{3,\text{opt}}$ and $DS_{3,\text{nai}}$ indicate the order of local polynomial r as well as the inflation factor (optimal or naive). We consider the final seasonality estimates as returned by those functions.

Tables 6 through 17 in the appendix contain the simulated numerical results following the different specifications. They contain the average sample MSE (Mean) of the seasonality estimates over all time points and 1000 simulated series as well as the sample standard deviation (SD) of the average MSEs obtained for each of the 1000 series (separately for each estimation method). Both quantities are multiplied by 1000. From the tables we can conclude the following: for S_1 , MA always produces the smallest average MSE in our simulation. This is not unexpected, as MA assumes an exactly periodic component and estimates the seasonality using global averaging of detrended values of the same time

unit. In some instances, it can also produce the smallest average MSE among the tested methods for n_1 under more complex settings. The worst performing method on average for quarterly series is TBATS, which produces in the given simulation notably larger average MSE values than the other approaches throughout without a structural decrease in average MSE. Moreover, TBATS is the only method that, in rare cases, failed to run the decomposition successfully; its stated average MSE and standard deviation values hence rely sometimes on fewer simulated series. Its performance is, however, clearly better for monthly series under S_1 and S_2 . For S_3 , its estimation quality decreases again.

X13 is never the best-performing method under any of the settings in our simulation study. Under n_1 , it usually performs better than DS, but yet still worse than STL or TS. Under n_1 , STL sometimes performs best for G_1 together with S_3 . Aside from the cases where MA and STL have the smallest average MSE, the minimum-MSE methods are always TS or DS. DS appears to usually exceed other methods for larger numbers of observations under S_2 for all three error processes regardless of whether a deterministic trend (G_1) or a stochastic trend (G_2) is considered. While this holds true for n_2 and n_3 in most cases, there are rare exceptions, where TS works slightly better. For quarterly data with S_2 and larger numbers of observations n_2 and n_3 , DS with local cubic trend is on average preferable under iid errors, and DS with local linear trend under autocorrelated errors, where the differences in estimation quality between the algorithm with optimal and naive inflation rate are usually minimal. For monthly data with S_2 , DS is more sensitive to the magnitude of errors given different trend settings. For $\gamma = 0.1$, DS with local linear trend works better for G_1 , if the errors are autocorrelated, whereas DS with local cubic trend results in smaller average MSE for G_2 . Under $\gamma = 0.2$ and S_2 , DS with local cubic trend is on average the best method for larger numbers of observations regardless of the trend setting.

If the stochastic seasonal component S_3 is considered, TS is often the most competitive method. The disadvantage of DS for this setting is expected, as the method assumes a deterministic seasonal component. Hence, random short-term changes in the seasonality cannot be identified properly by DS. Nonetheless, as can be seen in Tables 14 and 17, DS with local cubic trend can sometimes outperform TS even under S_3 for a large number of observations. In all other cases with S_3 , the best DS subroutine is usually only roughly 0.2 to 0.4 points in MSE worse than TS for large n . Furthermore, in these instances, we can observe a stronger decrease in the average MSE by DS than by TS. While left for future research, it seems not unreasonable to assume that DS may outperform TS even for S_3 , if the number of observations was even larger than n_3 .

5 Concluding remarks

A data-driven algorithm for decomposing univariate time series into a trend component, a seasonal component and a remainder component using locally weighted regression is proposed that extends already established algorithms for the case when the remainder is not iid. Moreover, the algorithm is fully nonparametric, as no further model assumptions are imposed on the remainder term. Therefore, a seasonal semiparametric extension of the ARMA model is introduced that can be fitted in two simple consecutive steps and used for producing point and interval forecasts. All methods described are made applicable within the R package *deSeaTS*. The decomposition performance of the proposed method on real-world data is validated against STL as a benchmark, whereas the seasonal adjustments by X13-ARIMA and TRAMO-SEATS are also shown as a reference whenever possible. In case of the three selected examples series, the US-GDP, the non-laborforce US-series, and the German COVID cases, all of which have different seasonal periods, *DeSeaTS* leads to suitable seasonal adjustments following two common seasonality tests.

A simulation study confirmed the consistency of the *DeSeaTS* bandwidth selector and of the correspondingly estimated trend, seasonal and trend-seasonal components, even if the assumption of exact periodicity is violated. Ultimately, a second simulation study compared the quality of seasonality estimation of various popular methods. In our setup, *DeSeaTS* usually works best in identifying the seasonal component under a slowly changing but still deterministic seasonal component, regardless of deterministic or stochastic trends and regardless of iid or autocorrelated errors, if the number of observations is sufficiently large. In our simulation, this was usually fulfilled for quarterly and monthly series of 30 or 60 years. Under an exactly periodic seasonal component, approaches with global seasonality estimation are preferable. Such a behavior is, however, not observable in most real-world data. In addition, the study showed that TRAMO-SEATS usually works best under a stochastic seasonal component. While *DeSeaTS* produced competitive results for large numbers of observations in such instances, it outperformed TRAMO-SEATS only on few occasions. In spite of this, the rate of decrease in average MSE of the *DeSeaTS* seasonality estimator, compared to that of TRAMO-SEATS, suggests that the *DeSeaTS* seasonality estimator could exceed that of TRAMO-SEATS in quality also under stochastic seasonality for even larger numbers of observations than those considered here. Thus, while *DeSeaTS* works poorly for small numbers of observations (by design), it is a viable and flexible choice for decomposition of seasonal time series and seasonal adjustment.

However, the introduced algorithm and the current version of the deseats package can clearly be improved. First, the current algorithm requires the seasonal component to be exactly periodic at least in theory. While we find that the algorithm also works reasonably well under violations of this assumption, like a slowly changing deterministic seasonal component, an improved algorithm, for which this assumption could be relaxed, is desirable. Moreover, the development of a procedure in which two separate bandwidths, one for the trend and one for the seasonal component, are estimated in a data-driven way, could be a promising endeavor. An additional feature could be the consideration of long-range dependence in the data. In the current state of the algorithm only short-memory errors are assumed, which could lead to misspecifications in some cases. As a further point, DeSeaTS may be non-robust with respect to outliers and structural breaks, which was not studied in detail within this article. A further disadvantage is that DeSeaTS in its current version does not account for trading-day effects. Ultimately, while we have shown the consistency of the trend and seasonality estimates following DeSeaTS in the given simulation studies, which implies that the residuals must approach the true errors, an additional simulation study on the parametric part of the seasonal Semi-ARMA is left for future research.

Computational details

The numerical and visual results in this paper were obtained using R 4.5.1 with the deseats 1.1.2 package, the RJDemetra 0.2.8 package and the forecast 8.23.0 package. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/>.

Acknowledgments: We thank the Federal Statistical Office of Germany for allowing us to implement the estimation method of the BV4.1 base model into the deseats package. Moreover, we are grateful to CRAN for their help during the publication of the R package deseats. We would also like to thank Prof. Dr. Yuanhua Feng and Ms. Shujie Li for helpful discussions, and two anonymous reviewers for thoughtful feedback to help improve the empirical section of this article.

Appendix

5.1 X13 and TRAMO-SEATS real example figures

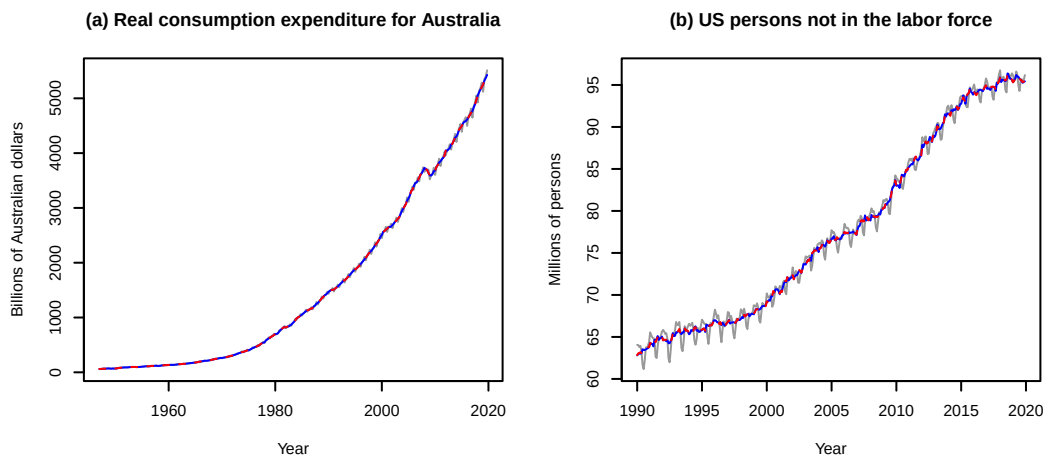


Figure 7: Seasonally adjusted series for the US-GDP series and of the series about US persons not in the labor force following the automated X13 (blue and solid) and TRAMO-SEATS (red and dashed) subroutines of RJDemetra+. X13 used a 5-terms Henderson trend filter and a 3x5 seasonal filter for the series in (a), whereas a 13-terms Henderson trend filter and a 3x5 seasonal filter were considered for the series in (b).

5.2 Tables and figures on bandwidth simulation study

On the following pages, the Tables 2 through 5 and the Figures 8 through 15 with the results for the first simulation study described in Section 4 are included. Brief interpretations of the results are given within Section 4. The tables include the asymptotically optimal bandwidths h_A for each setting, the sample mean over the estimated bandwidths $\text{Mean}(\hat{h})$, the sample standard deviation over the estimated bandwidths $\text{SD}(\hat{h})$ (multiplied by 1000), the sample MSE over the estimated bandwidths $\text{MSE}(\hat{h})$ (multiplied by 1000), and the average sample MSE over all series and time points for the estimated trend, the estimated seasonal component, and the estimated trend-seasonal component, denoted by $\text{MSE}(\hat{\xi})$, $\text{MSE}(\hat{s})$ and $\text{MSE}(\hat{m})$, respectively, and multiplied by 1000. The figures show the distributions of the estimated bandwidths.

Table 2: DeSeaTS simulation results under consideration of quarterly data and error process $\{e_{1,t}\}$. $SD(\hat{h})$, $MSE(\hat{h})$, $MSE(\hat{g})$, $MSE(\hat{s})$ and $MSE(\hat{m})$ were multiplied by 1000.

<i>r</i>	Inflation	Quantity	<i>s</i> ₁									<i>s</i> ₂								
			<i>g</i> ₁			<i>g</i> ₂			<i>g</i> ₃			<i>g</i> ₁			<i>g</i> ₂			<i>g</i> ₃		
			<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃
1	optimal	<i>h</i> _{<i>A</i>}	0.49	0.49	0.49	0.16	0.14	0.12	0.12	0.11	0.09	0.49	0.49	0.49	0.16	0.14	0.12	0.12	0.11	0.09
		Mean($\hat{h}$)	0.15	0.16	0.17	0.11	0.11	0.10	0.10	0.08	0.08	0.15	0.16	0.17	0.11	0.11	0.10	0.10	0.08	0.08
		SD($\hat{h}$)	41.73	47.60	48.95	7.69	11.08	9.10	4.01	7.94	6.39	42.15	47.40	49.90	7.90	11.08	9.12	4.08	7.94	6.38
		MSE($\hat{h}$)	115.66	110.12	103.64	3.42	1.57	0.60	0.52	0.56	0.28	117.61	111.27	103.80	3.40	1.56	0.60	0.55	0.57	0.28
		MSE($\hat{g}$)	6.73	4.03	2.25	9.41	6.11	3.93	13.38	8.11	5.23	6.79	4.06	2.25	9.44	6.11	3.93	13.38	8.09	5.23
		MSE($\hat{s}$)	0.86	0.41	0.19	1.14	0.56	0.29	1.21	0.68	0.37	1.12	0.66	0.47	1.22	0.62	0.33	1.25	0.71	0.38
		MSE($\hat{m}$)	7.55	4.41	2.43	10.18	6.65	4.21	14.10	8.78	5.59	7.87	4.69	2.72	10.27	6.72	4.26	14.20	8.80	5.61
	naive	Mean($\hat{h}$)	0.19	0.20	0.21	0.12	0.12	0.11	0.11	0.10	0.09	0.18	0.20	0.21	0.12	0.12	0.11	0.11	0.10	0.09
		SD($\hat{h}$)	50.33	55.66	57.00	18.27	14.81	10.59	4.49	10.16	7.77	51.99	55.87	57.56	18.51	14.76	10.56	4.41	10.16	7.78
		MSE($\hat{h}$)	94.51	86.51	79.72	2.37	0.73	0.23	0.26	0.20	0.11	96.29	86.70	79.62	2.29	0.71	0.22	0.27	0.21	0.11
		MSE($\hat{g}$)	6.21	3.58	1.98	10.08	6.55	4.21	13.94	9.14	5.72	6.29	3.59	1.98	10.14	6.56	4.22	13.89	9.11	5.72
		MSE($\hat{s}$)	0.72	0.34	0.16	1.13	0.51	0.26	1.27	0.61	0.34	1.13	0.77	0.63	1.24	0.60	0.33	1.31	0.65	0.36
		MSE($\hat{m}$)	6.88	3.90	2.14	10.72	7.01	4.46	14.19	9.82	6.05	7.36	4.34	2.61	10.87	7.11	4.54	14.28	9.85	6.08
3	optimal	<i>h</i> _{<i>A</i>}	0.49	0.49	0.49	0.30	0.27	0.25	0.26	0.24	0.22	0.49	0.49	0.49	0.30	0.27	0.25	0.26	0.24	0.22
		Mean($\hat{h}$)	0.16	0.16	0.17	0.16	0.16	0.16	0.15	0.15	0.16	0.15	0.16	0.17	0.15	0.16	0.16	0.15	0.15	0.16
		SD($\hat{h}$)	25.48	23.64	25.00	24.41	22.15	22.87	20.11	18.23	17.18	26.06	23.62	24.87	24.74	22.14	22.65	20.87	18.20	17.17
		MSE($\hat{h}$)	111.63	108.83	105.75	20.16	13.58	8.58	11.43	6.98	4.05	113.06	109.27	106.06	20.87	13.75	8.65	11.73	7.08	4.08
		MSE($\hat{g}$)	9.92	6.67	4.06	9.98	6.71	4.10	10.11	6.84	4.21	10.00	6.69	4.07	10.06	6.73	4.10	10.19	6.86	4.22
		MSE($\hat{s}$)	0.81	0.39	0.19	0.81	0.39	0.19	0.84	0.40	0.20	1.11	0.63	0.42	1.11	0.62	0.42	1.10	0.61	0.39
		MSE($\hat{m}$)	10.61	7.03	4.24	10.67	7.07	4.28	10.83	7.21	4.40	10.91	7.28	4.48	10.97	7.32	4.51	11.11	7.43	4.60
	naive	Mean($\hat{h}$)	0.18	0.18	0.19	0.18	0.18	0.19	0.17	0.17	0.18	0.18	0.18	0.19	0.17	0.18	0.19	0.17	0.17	0.17
		SD($\hat{h}$)	27.34	27.93	30.17	25.83	25.01	26.55	20.50	19.16	17.85	30.10	27.97	29.99	27.07	25.38	26.32	21.75	19.50	17.82
		MSE($\hat{h}$)	98.07	94.71	90.27	14.94	9.25	5.09	8.04	4.48	2.21	99.32	95.28	90.51	15.62	9.40	5.15	8.40	4.54	2.23
		MSE($\hat{g}$)	9.50	6.27	3.73	9.58	6.34	3.81	9.80	6.59	4.00	9.59	6.30	3.74	9.67	6.36	3.81	9.87	6.60	4.00
		MSE($\hat{s}$)	0.72	0.35	0.17	0.73	0.35	0.17	0.75	0.36	0.18	1.11	0.69	0.52	1.10	0.68	0.50	1.09	0.65	0.45
		MSE($\hat{m}$)	10.09	6.59	3.89	10.18	6.66	3.97	10.41	6.92	4.17	10.52	6.94	4.24	10.59	7.00	4.30	10.76	7.21	4.44

Table 3: DeSeaTS simulation results under consideration of quarterly data and error process $\{e_{2,t}\}$. SD $(\hat{h})$, MSE $(\hat{h})$, MSE $(\hat{g})$, MSE $(\hat{s})$ and MSE $(\hat{m})$ were multiplied by 1000.

<i>r</i>	Inflation	Quantity	<i>s</i> ₁									<i>s</i> ₂								
			<i>g</i> ₁			<i>g</i> ₂			<i>g</i> ₃			<i>g</i> ₁			<i>g</i> ₂			<i>g</i> ₃		
			<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃
1	optimal	<i>h</i> _A	0.49	0.49	0.49	0.14	0.13	0.11	0.11	0.09	0.08	0.49	0.49	0.49	0.14	0.13	0.11	0.11	0.09	0.08
		Mean $(\hat{h})$	0.15	0.17	0.18	0.11	0.10	0.10	0.10	0.08	0.07	0.15	0.17	0.18	0.11	0.10	0.10	0.10	0.08	0.07
		SD $(\hat{h})$	42.70	49.28	55.85	7.99	10.24	7.39	3.91	7.40	5.21	41.25	49.49	55.46	8.29	10.20	7.38	4.00	7.24	5.20
		MSE $(\hat{h})$	114.47	105.01	98.99	1.47	0.55	0.20	0.07	0.15	0.08	116.78	105.16	99.34	1.46	0.54	0.20	0.08	0.15	0.09
		MSE $(\hat{g})$	4.16	2.25	1.22	6.59	4.14	2.58	10.52	5.88	3.45	4.22	2.24	1.22	6.60	4.14	2.58	10.50	5.85	3.44
		MSE $(\hat{s})$	0.43	0.19	0.09	0.61	0.28	0.14	0.67	0.34	0.18	0.68	0.48	0.40	0.67	0.34	0.18	0.71	0.37	0.20
		MSE $(\hat{m})$	4.56	2.42	1.30	6.83	4.43	2.72	10.68	6.22	3.63	4.86	2.70	1.62	6.90	4.49	2.76	10.76	6.23	3.64
	naive	Mean $(\hat{h})$	0.19	0.22	0.23	0.12	0.12	0.11	0.11	0.09	0.08	0.19	0.22	0.23	0.12	0.12	0.11	0.11	0.09	0.08
		SD $(\hat{h})$	53.74	62.07	63.45	16.09	12.60	8.21	4.53	9.15	6.38	54.69	61.30	64.09	16.81	12.53	8.18	4.48	9.12	6.38
		MSE $(\hat{h})$	89.99	78.42	72.25	0.92	0.21	0.07	0.02	0.09	0.04	92.06	79.05	72.21	0.88	0.20	0.07	0.02	0.09	0.04
		MSE $(\hat{g})$	3.77	1.96	1.05	7.22	4.60	2.83	11.11	6.92	3.84	3.82	1.97	1.05	7.30	4.63	2.83	11.09	6.88	3.83
		MSE $(\hat{s})$	0.35	0.15	0.07	0.63	0.25	0.13	0.74	0.31	0.17	0.79	0.65	0.61	0.73	0.34	0.18	0.77	0.35	0.19
		MSE $(\hat{m})$	4.07	2.11	1.12	7.31	4.82	2.95	10.78	7.31	4.01	4.54	2.61	1.66	7.46	4.93	3.01	10.88	7.33	4.02
3	optimal	<i>h</i> _A	0.49	0.49	0.49	0.28	0.25	0.24	0.24	0.22	0.20	0.49	0.49	0.49	0.28	0.25	0.24	0.24	0.22	0.20
		Mean $(\hat{h})$	0.15	0.16	0.17	0.15	0.16	0.17	0.14	0.16	0.16	0.15	0.16	0.17	0.15	0.16	0.17	0.14	0.15	0.16
		SD $(\hat{h})$	25.67	24.47	26.15	24.60	22.82	22.80	20.95	18.13	16.15	26.74	24.63	25.96	25.15	22.75	22.60	21.45	18.23	16.10
		MSE $(\hat{h})$	116.70	107.66	101.92	16.65	9.28	5.02	9.01	4.50	2.11	117.70	107.87	102.27	17.14	9.37	5.09	9.34	4.54	2.13
		MSE $(\hat{g})$	6.93	4.11	2.28	6.96	4.15	2.32	7.06	4.26	2.41	7.02	4.11	2.28	7.05	4.16	2.33	7.13	4.26	2.42
		MSE $(\hat{s})$	0.44	0.19	0.09	0.44	0.19	0.09	0.45	0.19	0.09	0.69	0.43	0.34	0.68	0.42	0.33	0.67	0.40	0.30
		MSE $(\hat{m})$	7.24	4.27	2.36	7.27	4.31	2.40	7.38	4.42	2.50	7.53	4.51	2.62	7.56	4.55	2.65	7.64	4.63	2.70
	naive	Mean $(\hat{h})$	0.17	0.19	0.20	0.17	0.18	0.19	0.16	0.17	0.18	0.17	0.19	0.20	0.17	0.18	0.19	0.16	0.17	0.18
		SD $(\hat{h})$	29.98	29.72	31.65	28.13	26.47	25.10	22.55	19.06	15.73	31.32	30.12	31.79	28.56	26.26	24.75	23.05	19.03	15.70
		MSE $(\hat{h})$	101.23	92.08	86.20	11.50	5.61	2.56	5.78	2.48	0.98	102.24	92.33	86.45	11.91	5.68	2.61	6.06	2.53	1.00
		MSE $(\hat{g})$	6.50	3.79	2.09	6.58	3.87	2.18	6.73	4.06	2.32	6.57	3.80	2.09	6.63	3.88	2.18	6.79	4.06	2.32
		MSE $(\hat{s})$	0.38	0.17	0.08	0.38	0.17	0.08	0.39	0.18	0.08	0.74	0.53	0.46	0.73	0.51	0.42	0.70	0.46	0.35
		MSE $(\hat{m})$	6.78	3.93	2.16	6.86	4.01	2.25	7.03	4.20	2.40	7.15	4.29	2.54	7.21	4.36	2.59	7.34	4.49	2.67

Table 4: DeSeaTS simulation results under consideration of monthly data and error process $\{e_{1,t}\}$. $SD(\hat{h})$, $MSE(\hat{h})$, $MSE(\hat{g})$, $MSE(\hat{s})$ and $MSE(\hat{m})$ were multiplied by 1000.

<i>r</i>	Inflation	Quantity	<i>s</i> ₁									<i>s</i> ₂								
			<i>g</i> ₁			<i>g</i> ₂			<i>g</i> ₃			<i>g</i> ₁			<i>g</i> ₂			<i>g</i> ₃		
			<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃
1	optimal	<i>h</i> _{<i>A</i>}	0.49	0.49	0.49	0.16	0.14	0.12	0.12	0.11	0.09	0.49	0.49	0.49	0.16	0.14	0.12	0.12	0.11	0.09
		Mean($\hat{h}$)	0.23	0.26	0.27	0.16	0.15	0.13	0.12	0.11	0.10	0.23	0.26	0.27	0.16	0.15	0.13	0.12	0.11	0.10
		$SD(\hat{h})$	69.33	72.99	74.11	27.58	12.66	8.57	12.78	9.47	5.79	68.82	73.93	73.54	27.54	12.63	8.55	12.47	9.49	5.79
		$MSE(\hat{h})$	70.44	59.17	55.43	0.76	0.17	0.08	0.16	0.12	0.05	72.25	59.35	55.74	0.76	0.17	0.08	0.16	0.12	0.05
		$MSE(\hat{g})$	2.35	1.23	0.60	9.22	5.26	3.05	11.16	7.50	4.28	2.37	1.24	0.60	9.22	5.28	3.05	11.03	7.43	4.27
		$MSE(\hat{s})$	1.62	0.74	0.36	2.20	1.15	0.65	3.05	1.49	0.84	2.30	1.53	1.18	2.49	1.33	0.75	3.11	1.55	0.88
		$MSE(\hat{m})$	3.91	1.97	0.96	11.09	6.39	3.69	12.57	9.02	5.11	4.61	2.76	1.79	11.34	6.58	3.80	12.60	9.03	5.14
	naive	Mean($\hat{h}$)	0.28	0.30	0.30	0.18	0.16	0.14	0.15	0.13	0.11	0.28	0.30	0.30	0.18	0.16	0.14	0.15	0.13	0.11
		$SD(\hat{h})$	63.27	62.83	63.42	23.50	13.15	9.31	29.23	10.33	6.89	64.71	63.57	63.07	22.78	13.00	9.26	29.14	10.30	6.87
		$MSE(\hat{h})$	47.49	41.56	38.62	0.79	0.44	0.30	1.51	0.59	0.23	48.19	41.95	38.75	0.81	0.45	0.30	1.47	0.57	0.22
		$MSE(\hat{g})$	2.11	1.13	0.54	11.05	6.52	3.82	19.63	10.56	5.45	2.12	1.14	0.54	11.23	6.57	3.84	19.38	10.50	5.43
		$MSE(\hat{s})$	1.35	0.64	0.31	1.99	1.07	0.60	2.81	1.33	0.78	2.32	1.66	1.36	2.35	1.29	0.74	2.99	1.43	0.83
		$MSE(\hat{m})$	3.42	1.77	0.86	12.71	7.56	4.42	20.09	11.69	6.21	4.39	2.78	1.90	13.23	7.84	4.57	20.06	11.77	6.25
3	optimal	<i>h</i> _{<i>A</i>}	0.49	0.49	0.49	0.29	0.27	0.25	0.25	0.23	0.21	0.49	0.49	0.49	0.29	0.27	0.25	0.25	0.23	0.21
		Mean($\hat{h}$)	0.19	0.20	0.20	0.18	0.19	0.20	0.18	0.19	0.18	0.19	0.20	0.20	0.18	0.19	0.20	0.18	0.18	0.18
		$SD(\hat{h})$	29.30	30.12	32.09	27.48	26.56	25.51	22.74	18.80	16.13	30.33	31.67	32.14	27.96	26.98	25.52	22.64	18.86	16.03
		$MSE(\hat{h})$	93.01	86.27	82.68	11.97	6.24	3.24	5.66	2.57	1.19	93.30	86.50	82.67	12.18	6.37	3.26	5.77	2.62	1.20
		$MSE(\hat{g})$	4.78	2.60	1.36	4.85	2.68	1.44	5.02	2.85	1.58	4.82	2.61	1.36	4.89	2.69	1.45	5.06	2.85	1.58
		$MSE(\hat{s})$	1.90	0.89	0.43	1.91	0.90	0.44	1.96	0.94	0.47	2.33	1.33	0.90	2.33	1.32	0.87	2.33	1.30	0.81
		$MSE(\hat{m})$	6.49	3.46	1.78	6.58	3.55	1.88	6.79	3.76	2.04	6.92	3.90	2.25	7.00	3.97	2.31	7.16	4.11	2.38
	naive	Mean($\hat{h}$)	0.22	0.23	0.24	0.21	0.22	0.22	0.20	0.20	0.20	0.22	0.23	0.24	0.21	0.22	0.22	0.20	0.20	0.20
		$SD(\hat{h})$	34.31	40.66	40.77	30.55	29.63	25.90	22.37	18.09	14.39	37.07	40.61	40.53	32.01	29.18	25.76	22.64	17.93	14.44
		$MSE(\hat{h})$	76.47	67.81	65.19	7.11	2.97	1.31	3.07	1.07	0.41	76.64	68.28	65.35	7.33	3.07	1.34	3.16	1.11	0.42
		$MSE(\hat{g})$	4.43	2.38	1.24	4.58	2.58	1.41	4.92	2.85	1.61	4.48	2.39	1.24	4.64	2.58	1.41	4.96	2.84	1.60
		$MSE(\hat{s})$	1.68	0.78	0.38	1.70	0.80	0.40	1.78	0.86	0.43	2.28	1.43	1.05	2.28	1.39	0.97	2.28	1.33	0.87
		$MSE(\hat{m})$	5.94	3.14	1.61	6.11	3.36	1.81	6.52	3.68	2.03	6.55	3.79	2.28	6.70	3.94	2.38	7.00	4.14	2.46

Table 5: DeSeaTS simulation results under consideration of monthly data and error process $\{e_{2,t}\}$. $SD(\hat{h})$, $MSE(\hat{h})$, $MSE(\hat{g})$, $MSE(\hat{s})$ and $MSE(\hat{m})$ were multiplied by 1000.

<i>r</i>	Inflation	Quantity	<i>s</i> ₁									<i>s</i> ₂								
			<i>g</i> ₁			<i>g</i> ₂			<i>g</i> ₃			<i>g</i> ₁			<i>g</i> ₂			<i>g</i> ₃		
			<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃	<i>n</i> ₁	<i>n</i> ₂	<i>n</i> ₃
1	optimal	<i>h</i> _{<i>A</i>}	0.49	0.49	0.49	0.14	0.13	0.11	0.11	0.09	0.08	0.49	0.49	0.49	0.14	0.13	0.11	0.11	0.09	0.08
		Mean($\hat{h}$)	0.25	0.27	0.28	0.15	0.13	0.11	0.12	0.10	0.09	0.25	0.27	0.28	0.15	0.13	0.11	0.12	0.10	0.09
		$SD(\hat{h})$	75.93	79.07	83.89	25.93	10.24	6.19	7.35	8.04	4.58	77.26	79.39	83.13	25.78	10.27	6.19	7.17	7.98	4.57
		$MSE(\hat{h})$	63.73	55.23	50.89	0.77	0.18	0.06	0.15	0.16	0.05	64.19	55.67	51.22	0.77	0.18	0.06	0.13	0.15	0.05
		$MSE(\hat{g})$	1.29	0.64	0.31	6.70	3.53	1.89	8.85	5.28	2.78	1.30	0.64	0.32	6.76	3.55	1.89	8.76	5.21	2.77
		$MSE(\hat{s})$	1.34	0.63	0.30	2.02	1.08	0.62	2.65	1.39	0.80	2.11	1.47	1.20	2.25	1.20	0.70	2.70	1.43	0.83
		$MSE(\hat{m})$	2.60	1.26	0.61	8.43	4.57	2.50	10.07	6.57	3.58	3.36	2.10	1.52	8.71	4.72	2.58	10.14	6.55	3.60
	naive	Mean($\hat{h}$)	0.30	0.31	0.31	0.17	0.15	0.12	0.14	0.12	0.09	0.30	0.31	0.31	0.17	0.15	0.12	0.14	0.12	0.09
		$SD(\hat{h})$	65.52	66.37	68.36	20.85	10.95	7.01	23.56	7.95	4.62	67.77	66.20	68.43	20.08	10.82	7.00	23.27	7.98	4.63
		$MSE(\hat{h})$	42.28	37.18	35.57	1.18	0.63	0.28	1.39	0.71	0.20	42.24	37.63	35.80	1.23	0.66	0.28	1.34	0.70	0.20
		$MSE(\hat{g})$	1.16	0.58	0.29	8.49	4.63	2.40	14.36	7.75	3.54	1.17	0.58	0.29	8.64	4.68	2.41	14.15	7.72	3.54
		$MSE(\hat{s})$	1.13	0.54	0.27	1.79	0.98	0.57	2.59	1.22	0.74	2.18	1.62	1.37	2.10	1.16	0.67	2.71	1.29	0.78
		$MSE(\hat{m})$	2.26	1.12	0.55	10.03	5.61	2.97	14.44	8.98	4.28	3.30	2.20	1.66	10.48	5.84	3.08	14.41	9.04	4.31
3	optimal	<i>h</i> _{<i>A</i>}	0.49	0.49	0.49	0.27	0.25	0.23	0.23	0.22	0.20	0.49	0.49	0.49	0.27	0.25	0.23	0.23	0.22	0.20
		Mean($\hat{h}$)	0.19	0.20	0.21	0.19	0.20	0.20	0.18	0.18	0.18	0.19	0.20	0.21	0.19	0.20	0.20	0.18	0.18	0.18
		$SD(\hat{h})$	29.98	32.71	32.09	27.31	26.45	22.67	21.25	16.45	12.75	31.18	32.76	31.31	28.44	26.35	22.46	21.97	16.61	12.72
		$MSE(\hat{h})$	90.91	83.06	79.46	7.63	3.44	1.51	3.34	1.26	0.51	91.42	83.50	79.80	7.82	3.52	1.54	3.40	1.29	0.52
		$MSE(\hat{g})$	2.78	1.42	0.71	2.86	1.50	0.79	3.05	1.65	0.89	2.81	1.42	0.71	2.89	1.50	0.79	3.07	1.65	0.89
		$MSE(\hat{s})$	1.61	0.76	0.37	1.63	0.77	0.38	1.69	0.81	0.41	2.06	1.23	0.87	2.06	1.21	0.82	2.06	1.17	0.74
		$MSE(\hat{m})$	4.25	2.15	1.07	4.33	2.25	1.17	4.58	2.44	1.30	4.68	2.63	1.57	4.75	2.68	1.60	4.93	2.79	1.63
	naive	Mean($\hat{h}$)	0.22	0.24	0.24	0.22	0.22	0.22	0.20	0.20	0.19	0.22	0.24	0.24	0.21	0.22	0.22	0.20	0.20	0.19
		$SD(\hat{h})$	37.00	41.31	41.00	29.67	26.20	21.19	20.17	15.24	11.17	37.85	42.41	40.36	30.35	25.92	21.02	20.64	15.37	11.17
		$MSE(\hat{h})$	73.22	65.00	61.77	3.88	1.35	0.52	1.50	0.44	0.15	73.69	65.49	62.10	4.09	1.42	0.53	1.59	0.46	0.16
		$MSE(\hat{g})$	2.54	1.29	0.64	2.70	1.46	0.80	3.02	1.69	0.93	2.57	1.29	0.65	2.73	1.46	0.80	3.05	1.69	0.93
		$MSE(\hat{s})$	1.42	0.67	0.32	1.44	0.69	0.34	1.53	0.75	0.39	2.04	1.35	1.03	2.03	1.28	0.92	2.02	1.20	0.79
		$MSE(\hat{m})$	3.83	1.94	0.96	4.01	2.13	1.14	4.40	2.43	1.32	4.44	2.62	1.68	4.58	2.72	1.72	4.86	2.87	1.72

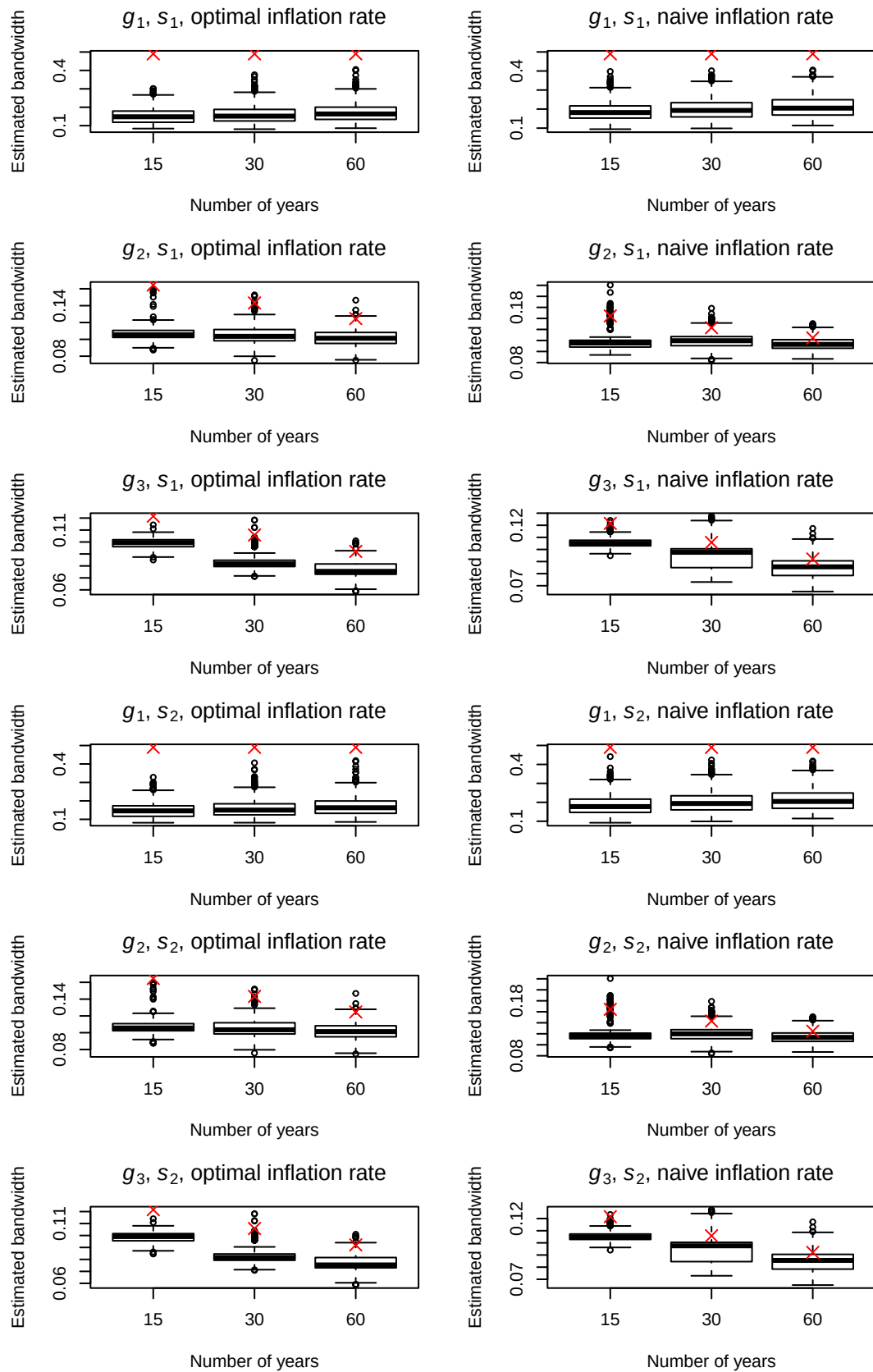


Figure 8: Boxplots of estimated bandwidths using $r = 1$ for simulated quarterly data with $\{e_{1,t}\}$. Red crosses mark the true h_A .

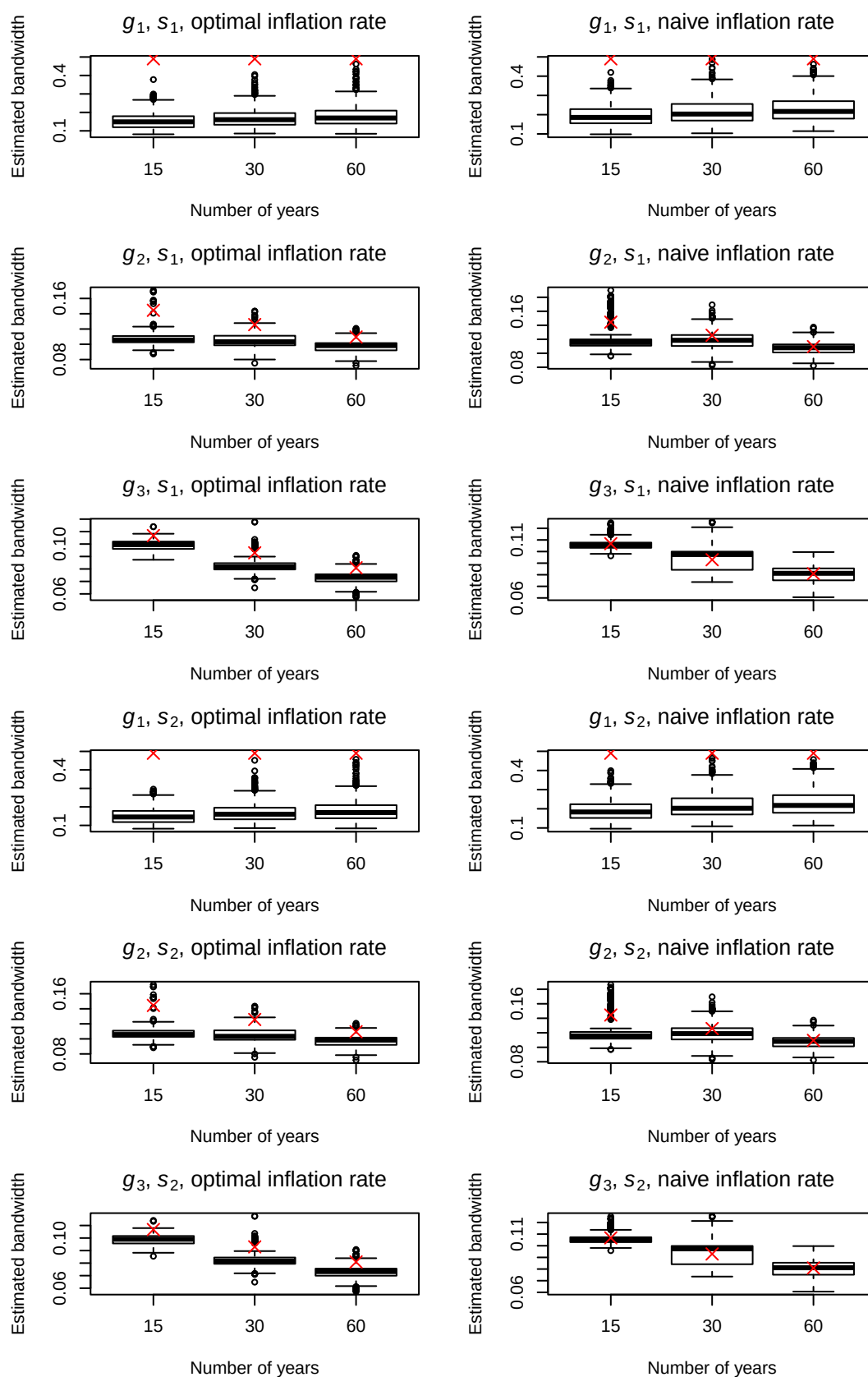


Figure 9: Boxplots of estimated bandwidths using $r = 1$ for simulated quarterly data with $\{e_{2,t}\}$. Red crosses mark the true h_A .

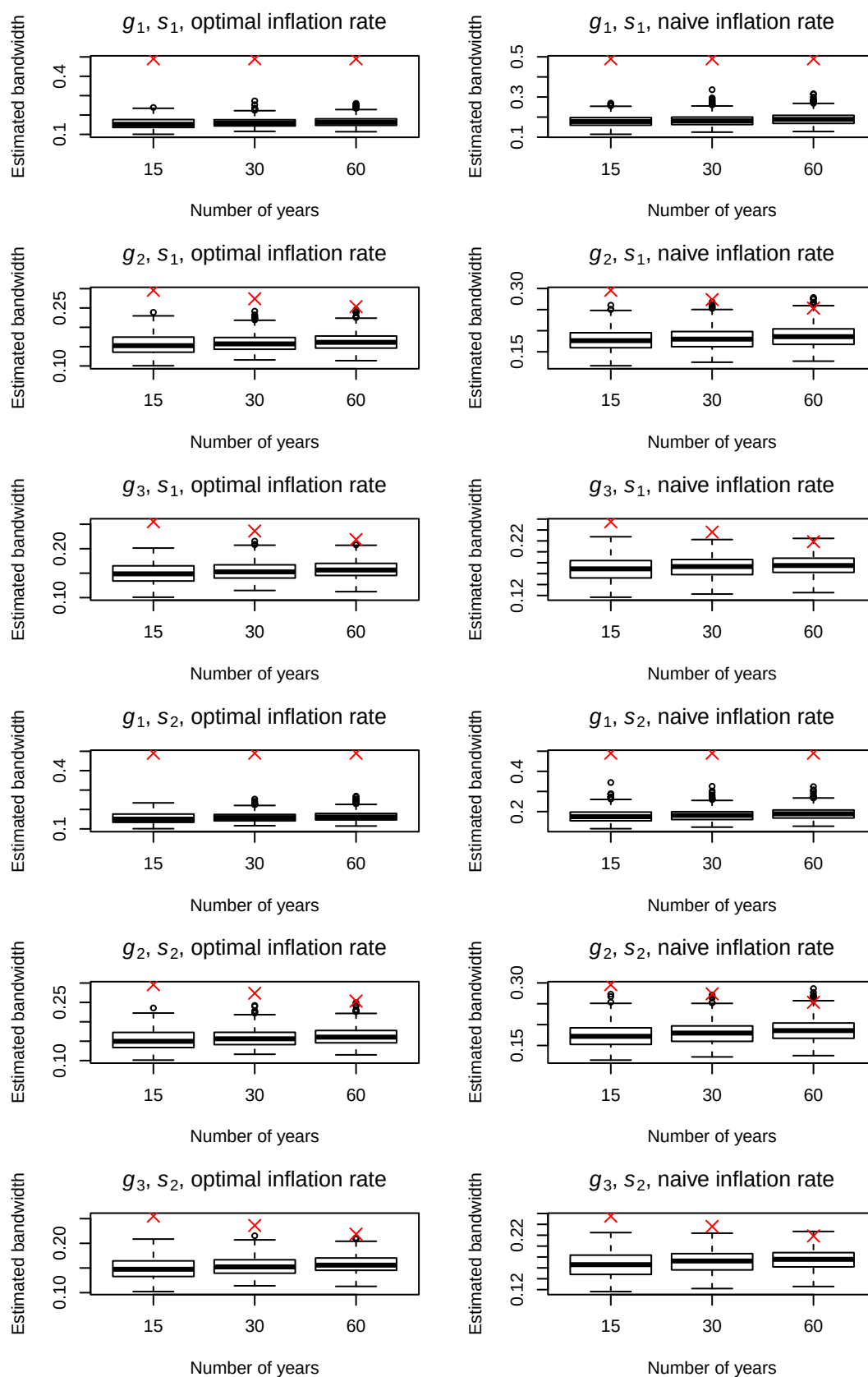


Figure 10: Boxplots of estimated bandwidths using $r = 3$ for simulated quarterly data with $\{e_{1,t}\}$. Red crosses mark the true h_A .

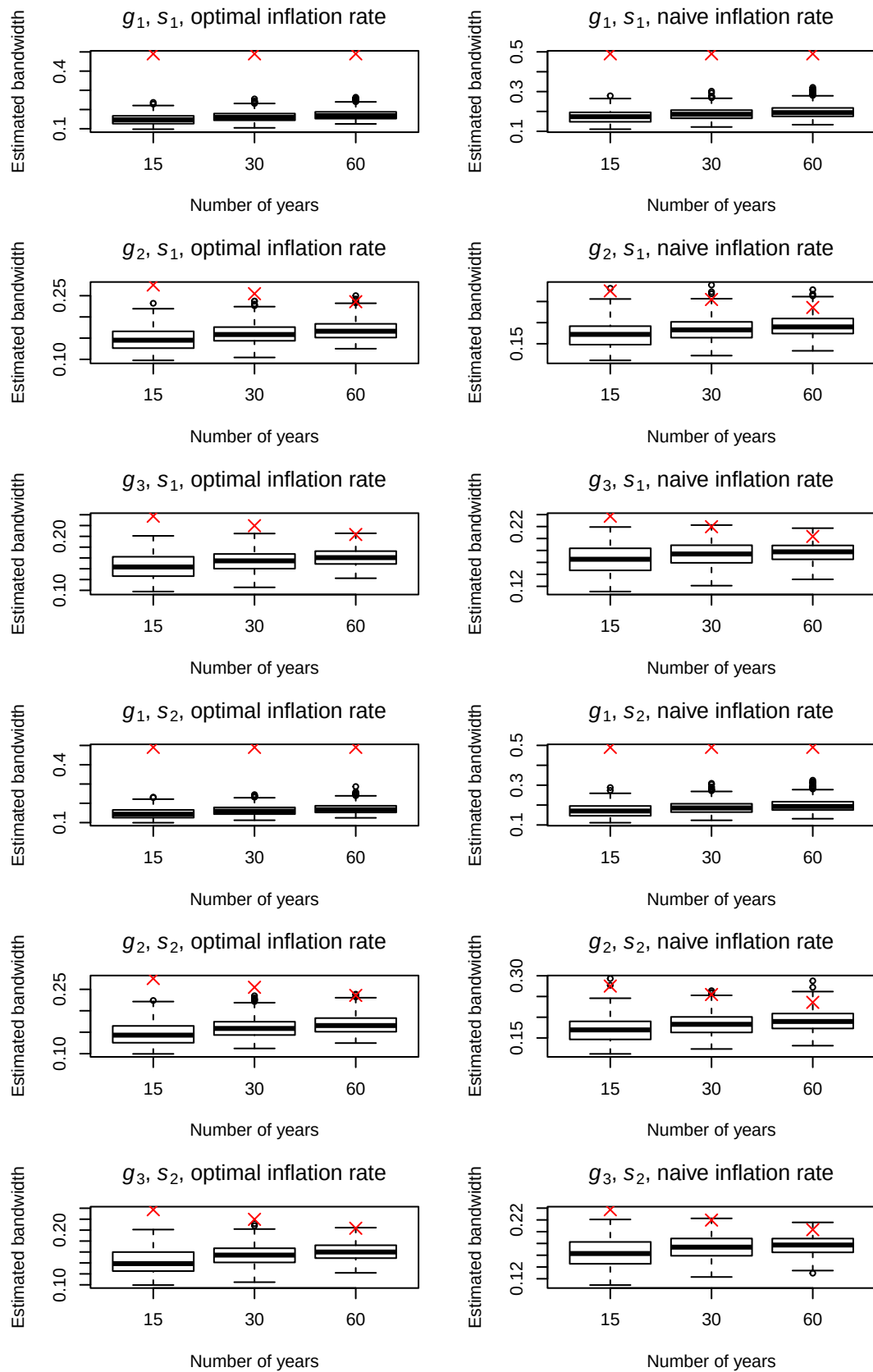


Figure 11: Boxplots of estimated bandwidths using $r = 3$ for simulated quarterly data with $\{e_{2,t}\}$. Red crosses mark the true h_A .

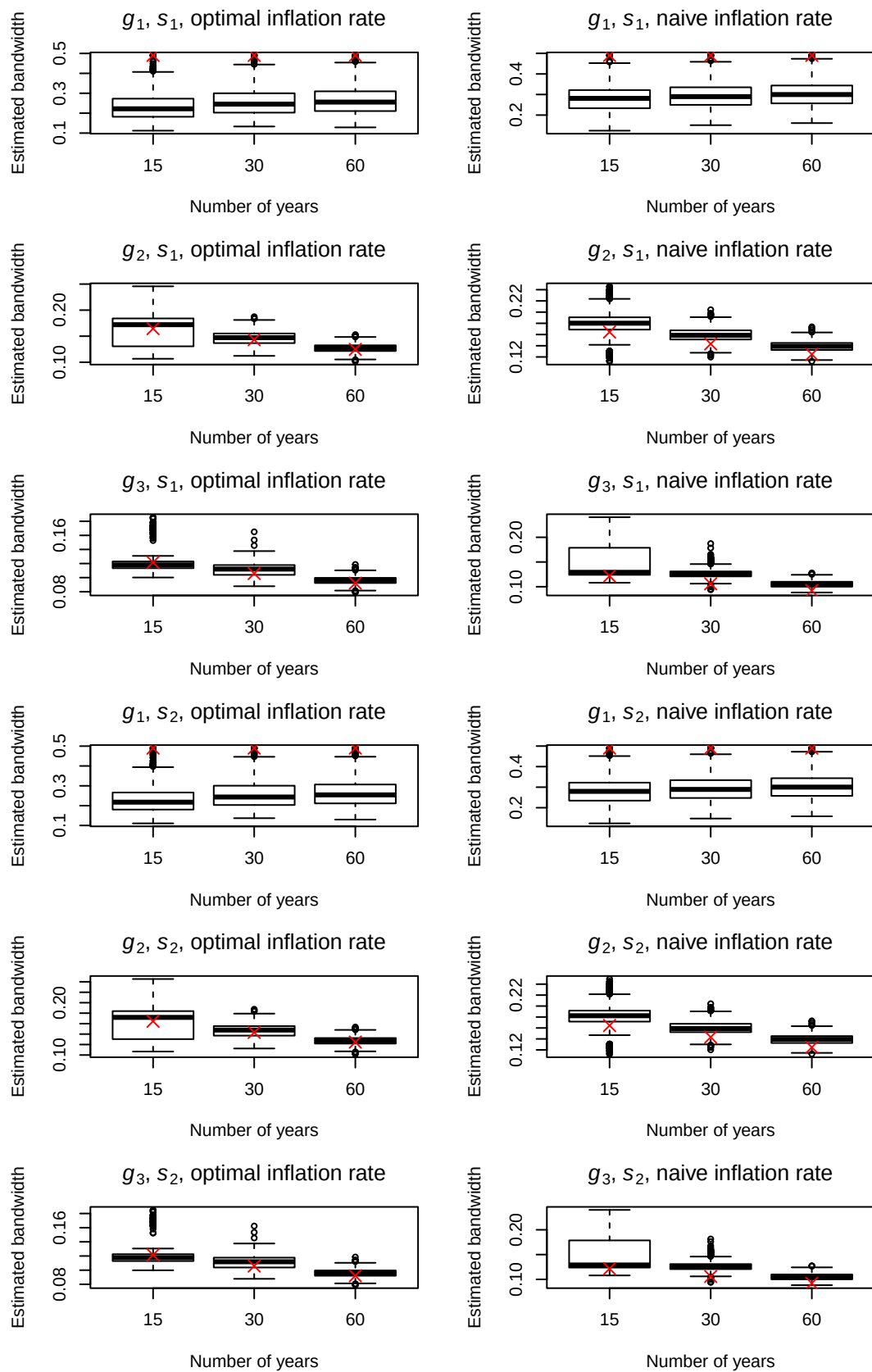


Figure 12: Boxplots of estimated bandwidths using $r = 1$ for simulated monthly data with $\{e_{1,t}\}$. Red crosses mark the true h_A .

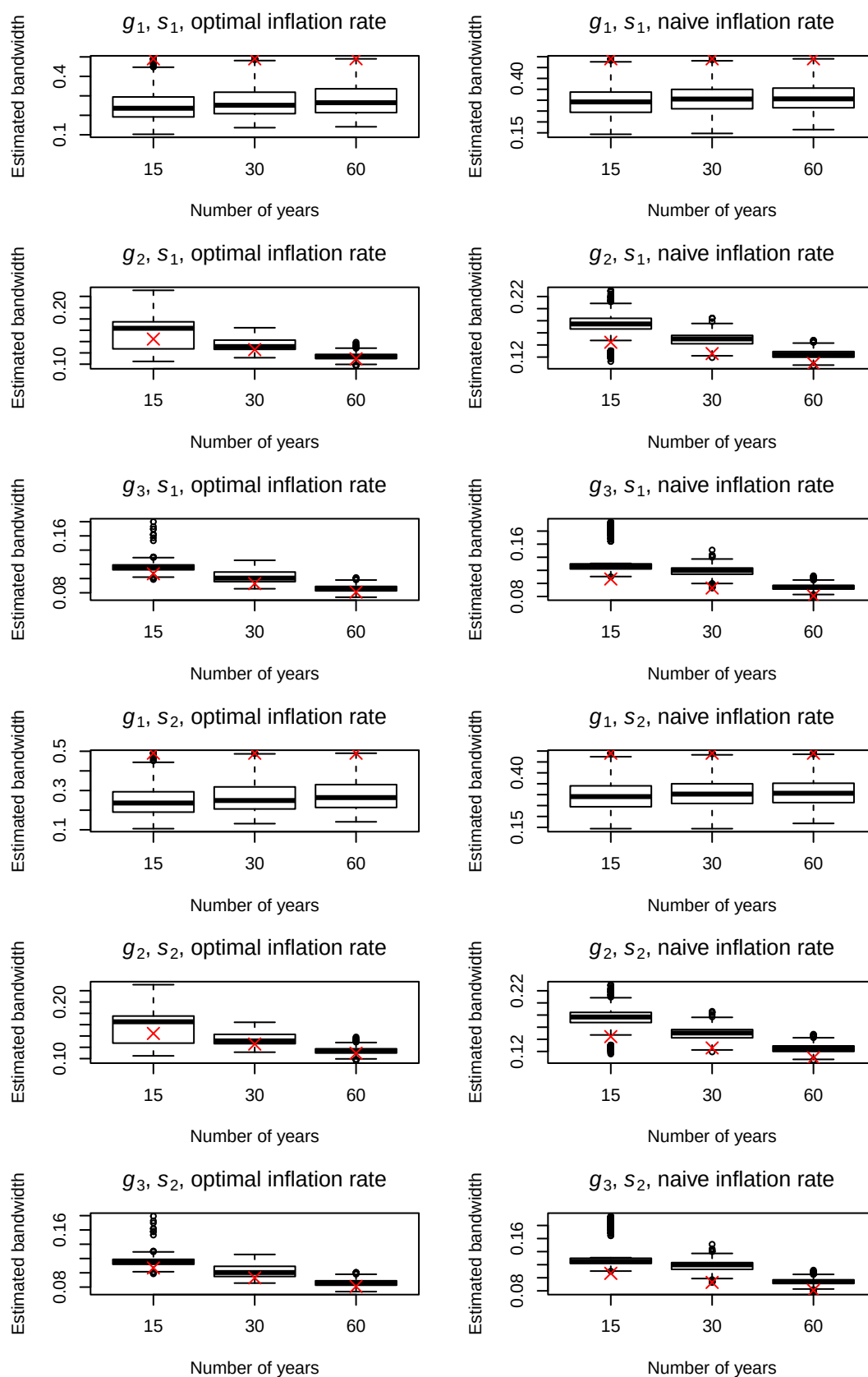


Figure 13: Boxplots of estimated bandwidths using $r = 1$ for simulated monthly data with $\{e_{2,t}\}$. Red crosses mark the true h_A .

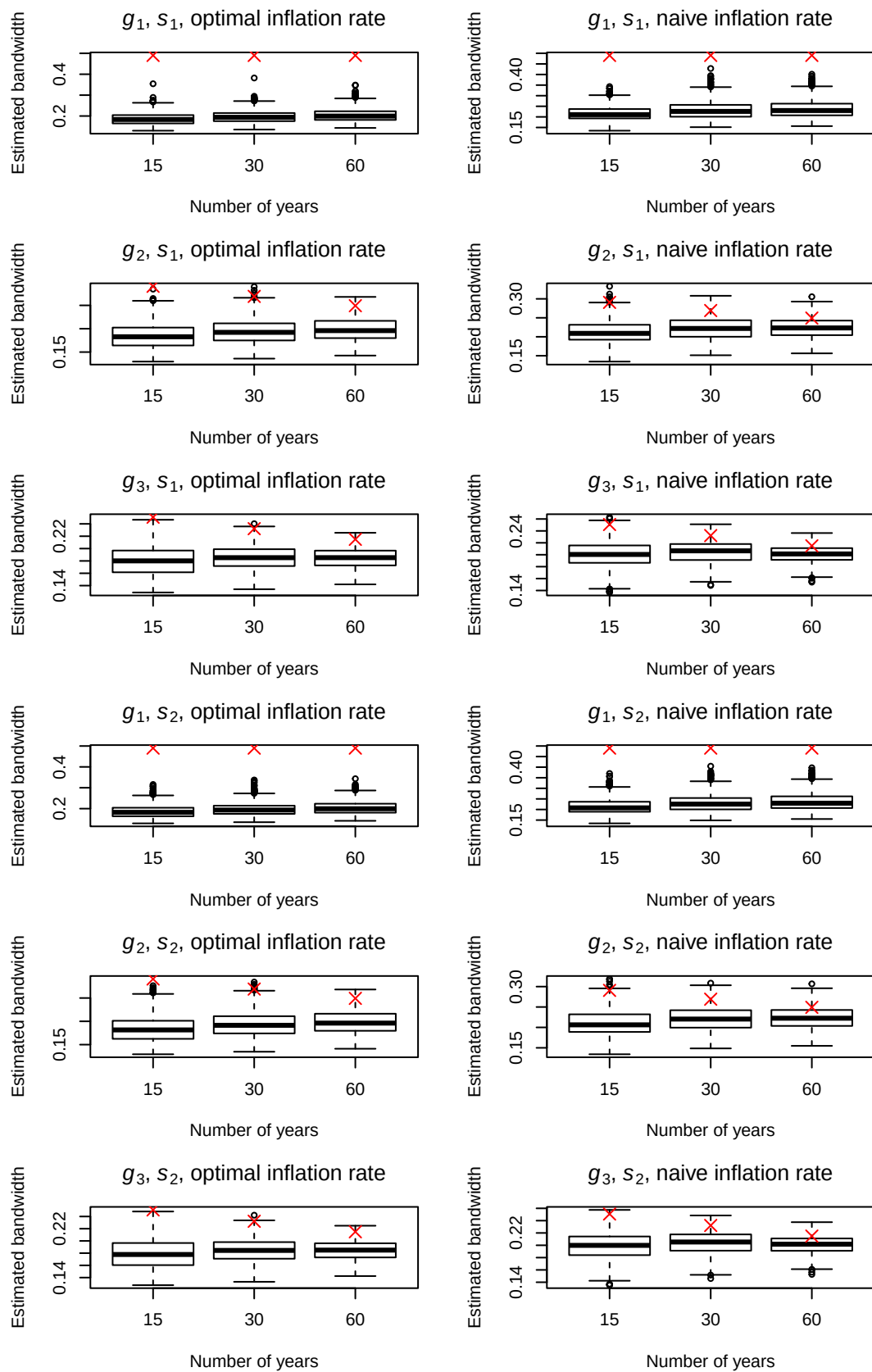


Figure 14: Boxplots of estimated bandwidths using $r = 3$ for simulated monthly data with $\{e_{1,t}\}$. Red crosses mark the true h_A .

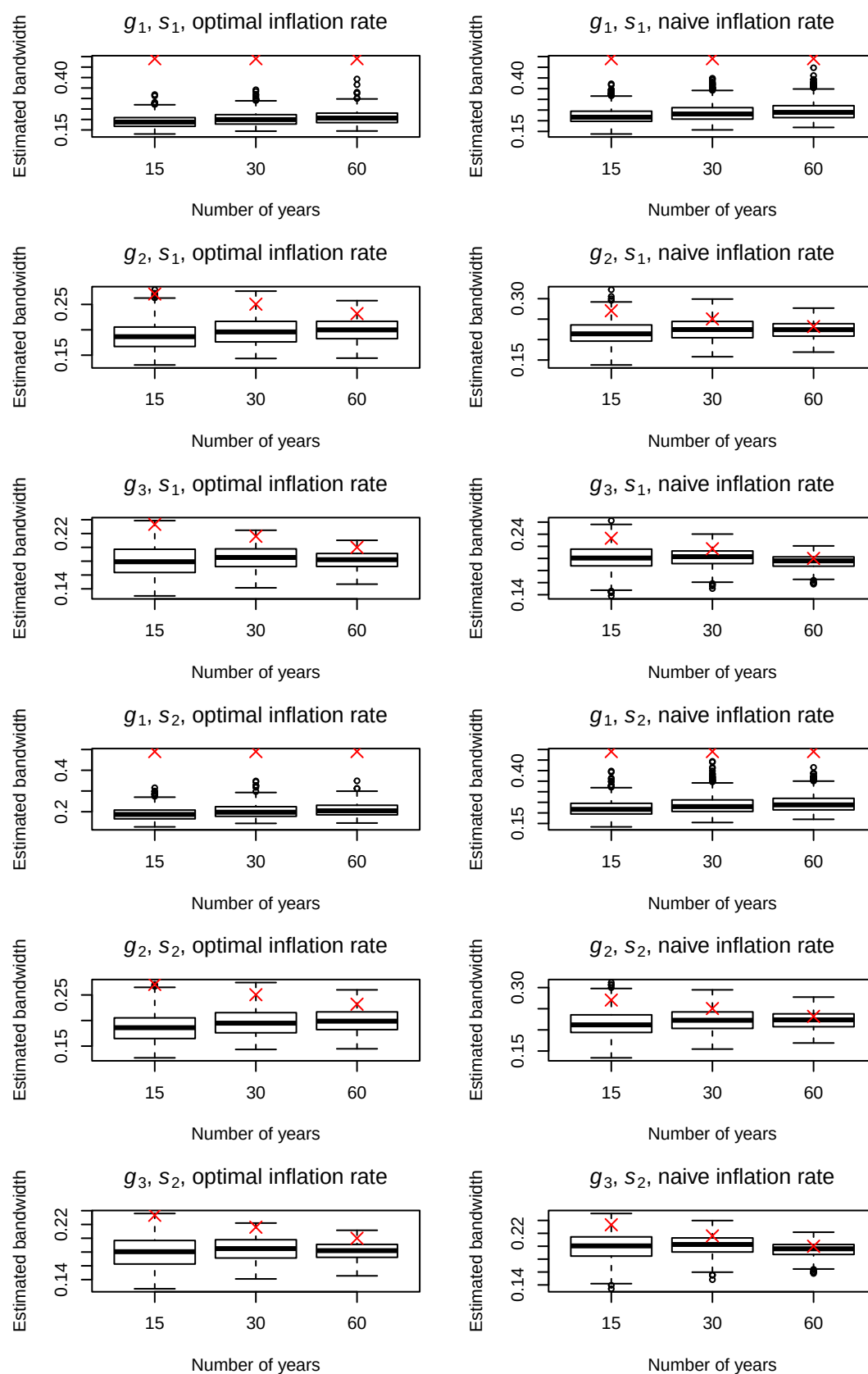


Figure 15: Boxplots of estimated bandwidths using $r = 3$ for simulated monthly data with $\{e_{2,t}\}$. Red crosses mark the true h_A .

5.3 Tables on comparative simulation study

On the following pages, the Tables 6 through 17 include the numerical results of the second simulation study described in Section 4. Brief interpretations of the results are given in Section 4. The tables contain the average mean squared errors (Mean) of the estimated seasonality over all series and time points in accordance with various methods. The sample standard deviations of the series-wise average mean squared errors are also given. For a better overview, the abbreviations **DS** (DeSeaTS), **X13** (X13-ARIMA), **TS** (TRAMO-SEATS), **STL** (STL), **TBATS** (TBATS) and **MA** (decompose() function in R) are used. Furthermore, the index given for **DS** describes the considered order of polynomial r and the inflation rate (either **opt** or **nai** for optimal or naive).

Table 6: Results of the seasonality estimation simulation study for quarterly data with error $\{E_{1,t}\}$ and $\gamma = 0.1$. The simulated mean MSE (Mean) and the MSE standard deviation (SD), both multiplied by 1000, are displayed for various methods. The smallest MSE values in each column are highlighted.

Method	Quantity	S_1						S_2						S_3					
		G_1			G_2			G_1			G_2			G_1			G_2		
		n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3
DS _{1,opt}	Mean	2.92	1.95	1.21	19.54	5.81	1.75	3.08	2.01	1.23	20.11	6.20	2.08	3.11	2.26	1.75	20.04	6.13	2.85
	SD	0.87	0.51	0.28	15.54	4.62	1.10	0.90	0.52	0.29	15.52	4.58	1.08	1.02	0.87	0.88	16.08	4.41	2.19
DS _{1,nai}	Mean	2.89	1.85	1.14	17.22	5.10	1.51	3.04	1.91	1.17	18.08	5.73	2.06	3.08	2.18	1.72	17.53	5.55	2.82
	SD	0.86	0.50	0.28	13.86	4.04	0.99	0.90	0.50	0.28	13.79	3.99	1.00	1.01	0.93	0.93	14.22	3.94	2.30
DS _{3,opt}	Mean	2.11	1.00	0.51	17.28	4.98	1.49	2.90	1.65	1.06	18.17	5.53	1.96	2.49	1.74	1.93	18.05	5.54	2.82
	SD	0.82	0.37	0.18	14.55	3.93	0.93	0.89	0.46	0.25	14.68	3.92	0.96	1.20	1.31	2.14	15.46	4.04	2.43
DS _{3,nai}	Mean	1.88	0.93	0.48	15.45	4.57	1.33	2.86	1.75	1.15	16.61	5.34	2.03	2.31	1.74	2.00	16.09	5.18	2.84
	SD	0.75	0.36	0.18	13.23	3.74	0.85	0.85	0.47	0.26	13.37	3.71	0.91	1.18	1.42	2.25	13.53	3.78	2.59
X13	Mean	1.34	1.20	1.10	9.97	5.29	3.03	3.94	1.78	1.29	12.97	6.13	3.21	2.18	1.93	1.82	11.38	6.00	3.69
	SD	0.71	0.49	0.37	9.08	4.11	1.84	0.90	0.54	0.32	9.21	4.16	1.82	1.59	1.24	0.91	10.48	4.33	2.08
TS	Mean	1.75	0.79	0.41	6.05	1.57	0.54	4.11	2.09	1.13	10.94	5.37	2.28	2.81	1.78	1.43	8.95	3.72	2.40
	SD	1.66	0.65	0.34	13.71	2.58	0.74	1.34	0.68	0.32	14.50	3.12	1.25	5.16	0.96	0.73	13.57	3.20	1.71
STL	Mean	0.96	1.00	0.98	7.42	4.50	2.65	4.76	2.17	1.08	11.20	5.66	2.73	2.10	1.80	1.68	8.77	5.18	3.29
	SD	0.53	0.37	0.26	6.91	3.58	1.52	0.60	0.50	0.27	6.96	3.64	1.53	2.23	1.47	1.05	8.49	3.72	2.00
TBATS	Mean	88.90	88.99	89.27	90.28	89.68	89.36	90.88	90.99	91.26	92.25	91.62	91.15	68.45	70.43	73.34	63.19	68.99	73.27
	SD	1.71	0.32	0.26	4.46	3.18	2.71	1.40	0.27	0.15	4.30	2.42	2.25	49.30	48.41	47.70	44.70	47.97	48.01
MA	Mean	0.56	0.27	0.12	4.11	1.16	0.34	5.02	4.73	4.58	8.55	5.62	4.79	3.79	6.27	11.21	7.68	6.97	11.50
	SD	0.46	0.21	0.10	5.10	1.41	0.34	0.48	0.22	0.10	5.09	1.41	0.33	5.75	8.48	16.37	9.05	9.14	16.18

Table 7: Results of the seasonality estimation simulation study for quarterly data with error $\{E_{2,t}\}$ and $\gamma = 0.1$. The simulated mean MSE (Mean) and the MSE standard deviation (SD), both multiplied by 1000, are displayed for various methods. The smallest MSE values in each column are highlighted.

Method	Quantity	S_1						S_2						S_3					
		G_1			G_2			G_1			G_2			G_1			G_2		
		n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3
DS _{1,opt}	Mean	0.77	0.43	0.23	17.81	4.52	1.24	0.92	0.51	0.28	18.38	4.92	1.60	1.00	0.81	0.87	18.16	5.08	2.26
	SD	0.24	0.12	0.06	16.90	3.85	1.12	0.27	0.13	0.07	16.86	3.81	1.17	0.56	0.58	1.02	16.22	4.14	1.86
DS _{1,nai}	Mean	0.80	0.39	0.21	15.16	3.96	1.08	0.95	0.50	0.28	16.03	4.61	1.66	1.05	0.82	0.92	15.50	4.62	2.32
	SD	0.25	0.12	0.06	13.88	3.43	0.98	0.28	0.13	0.07	13.96	3.40	1.09	0.62	0.66	1.12	13.57	3.72	2.19
DS _{3,opt}	Mean	0.53	0.25	0.12	15.50	3.93	1.07	1.18	0.81	0.65	16.24	4.49	1.57	0.92	0.97	1.41	16.22	4.57	2.25
	SD	0.21	0.10	0.05	13.76	3.37	1.02	0.34	0.23	0.19	13.69	3.36	1.05	0.86	1.11	1.86	14.91	3.67	1.98
DS _{3,nai}	Mean	0.48	0.23	0.11	13.82	3.56	0.97	1.33	0.99	0.84	14.81	4.35	1.69	0.91	1.03	1.54	14.41	4.28	2.32
	SD	0.19	0.09	0.04	12.43	3.11	0.95	0.36	0.27	0.22	12.38	3.08	0.99	0.93	1.26	2.05	13.14	3.48	2.21
X13	Mean	0.30	0.27	0.26	8.90	4.11	2.14	2.66	0.62	0.36	11.92	4.86	2.30	1.02	0.85	0.77	9.77	4.84	2.66
	SD	0.16	0.11	0.07	9.49	3.58	1.91	0.47	0.16	0.08	9.48	3.67	1.95	1.58	0.78	0.65	9.30	3.86	1.83
TS	Mean	0.24	0.17	0.07	5.53	1.20	0.42	1.09	0.68	0.41	10.24	4.56	1.75	0.86	0.67	0.63	8.47	3.21	1.86
	SD	0.26	0.14	0.08	10.61	1.83	0.93	0.36	0.19	0.14	10.85	3.05	1.49	0.43	0.32	0.32	15.32	3.34	1.35
STL	Mean	0.24	0.23	0.23	6.66	3.55	1.88	4.03	1.42	0.32	10.47	4.71	1.98	1.41	1.05	0.87	7.82	4.29	2.46
	SD	0.14	0.09	0.06	6.95	3.07	1.65	0.19	0.18	0.08	7.02	3.12	1.66	3.01	1.25	0.99	7.47	3.48	1.72
TBATS	Mean	88.73	88.88	89.26	90.09	89.39	89.42	90.78	90.69	90.86	92.18	91.39	91.24	71.01	71.05	71.41	64.20	67.54	71.66
	SD	1.64	0.91	2.29	4.19	2.38	2.23	0.67	0.92	0.73	4.33	1.86	2.50	49.12	48.60	49.66	45.31	48.12	48.20
MA	Mean	0.13	0.06	0.03	3.81	0.93	0.25	4.59	4.52	4.48	8.27	5.39	4.70	3.51	5.82	10.44	7.21	6.99	10.36
	SD	0.11	0.05	0.03	4.96	1.22	0.34	0.13	0.05	0.03	5.01	1.23	0.34	7.08	7.88	15.00	8.77	9.31	13.58

Table 8: Results of the seasonality estimation simulation study for quarterly data with error $\{E_{3,t}\}$ and $\gamma = 0.1$. The simulated mean MSE (Mean) and the MSE standard deviation (SD), both multiplied by 1000, are displayed for various methods. The smallest MSE values in each column are highlighted.

Method	Quantity	S_1						S_2						S_3					
		G_1			G_2			G_1			G_2			G_1			G_2		
		n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3
DS _{1,opt}	Mean	0.57	0.29	0.15	17.43	4.54	1.22	0.71	0.37	0.20	18.00	4.98	1.55	0.80	0.69	0.84	18.96	5.20	2.20
	SD	0.19	0.09	0.04	15.61	4.15	1.09	0.21	0.10	0.05	15.55	4.17	1.09	0.57	0.70	1.10	16.12	4.46	1.81
DS _{1,nai}	Mean	0.63	0.26	0.14	14.60	3.90	1.05	0.78	0.39	0.21	15.52	4.60	1.62	0.86	0.72	0.90	16.05	4.75	2.24
	SD	0.21	0.08	0.04	13.06	3.62	0.95	0.25	0.10	0.05	13.03	3.63	1.00	0.63	0.76	1.19	13.85	4.17	1.93
DS _{3,opt}	Mean	0.39	0.17	0.08	15.19	3.98	1.02	0.99	0.75	0.66	15.93	4.55	1.51	0.76	0.91	1.47	16.63	4.70	2.22
	SD	0.16	0.07	0.03	13.49	3.66	0.95	0.31	0.23	0.19	13.28	3.69	0.95	0.95	1.20	2.09	13.91	3.99	1.91
DS _{3,nai}	Mean	0.34	0.15	0.07	13.64	3.61	0.92	1.19	0.95	0.85	14.65	4.41	1.63	0.76	0.98	1.62	15.03	4.40	2.32
	SD	0.14	0.06	0.03	12.23	3.37	0.87	0.33	0.26	0.22	12.14	3.41	0.89	1.00	1.34	2.35	12.92	3.76	2.19
X13	Mean	0.19	0.18	0.16	8.91	4.18	2.05	2.40	0.48	0.23	11.93	4.93	2.19	0.85	0.73	0.67	10.46	4.96	2.49
	SD	0.11	0.07	0.05	9.04	3.99	1.85	0.46	0.12	0.05	9.03	4.10	1.86	1.40	0.83	0.71	9.71	4.24	1.78
TS	Mean	0.14	0.09	0.03	5.75	1.26	0.36	0.79	0.53	0.28	10.25	4.52	1.67	0.68	0.56	0.52	8.80	3.23	1.84
	SD	0.17	0.08	0.05	12.53	2.05	0.81	0.29	0.18	0.09	12.63	3.02	1.47	0.57	0.34	0.32	15.44	3.15	1.64
STL	Mean	0.16	0.15	0.15	6.83	3.58	1.80	3.95	1.33	0.24	10.64	4.76	1.89	1.26	0.99	0.84	8.13	4.36	2.33
	SD	0.10	0.06	0.04	7.01	3.43	1.57	0.14	0.14	0.05	7.03	3.50	1.57	2.09	1.31	1.07	7.44	3.77	1.69
TBATS	Mean	88.74	88.91	89.28	90.26	89.62	89.15	90.68	90.68	90.83	92.19	91.44	91.23	68.61	69.49	70.18	64.00	69.34	72.56
	SD	1.32	1.03	1.24	4.99	2.88	2.29	0.80	0.65	0.84	4.16	1.83	1.42	48.27	47.79	48.88	44.76	47.31	47.79
MA	Mean	0.08	0.04	0.02	3.72	0.98	0.25	4.54	4.50	4.47	8.17	5.44	4.70	3.34	6.04	10.52	7.24	7.14	11.49
	SD	0.07	0.03	0.02	4.54	1.52	0.33	0.09	0.04	0.02	4.54	1.53	0.33	5.73	9.25	15.20	8.58	11.49	15.41

Table 9: Results of the seasonality estimation simulation study for quarterly data with error $\{E_{1,t}\}$ and $\gamma = 0.2$. The simulated mean MSE (Mean) and the MSE standard deviation (SD), both multiplied by 1000, are displayed for various methods. The smallest MSE values in each column are highlighted.

Method	Quantity	S_1						S_2						S_3					
		G_1			G_2			G_1			G_2			G_1			G_2		
		n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3
DS _{1,opt}	Mean	11.38	6.51	3.70	26.21	9.44	3.69	11.60	6.64	3.76	26.86	9.84	4.03	11.44	6.82	4.42	26.75	9.68	4.84
	SD	3.41	1.91	1.01	16.70	5.46	1.62	3.46	1.93	1.01	16.75	5.38	1.55	3.50	2.09	1.50	17.57	5.20	2.49
DS _{1,nai}	Mean	10.78	6.00	3.48	22.96	8.26	3.19	11.06	6.16	3.55	23.85	8.90	3.73	10.90	6.36	4.25	23.27	8.66	4.56
	SD	3.48	1.82	0.97	14.91	4.79	1.49	3.48	1.83	0.98	14.91	4.68	1.42	3.53	2.04	1.53	15.24	4.67	2.55
DS _{3,opt}	Mean	8.30	3.82	1.93	23.21	8.05	3.12	9.27	4.63	2.63	24.15	8.64	3.61	8.56	4.55	3.46	24.15	8.61	4.50
	SD	3.30	1.47	0.73	15.71	4.48	1.35	3.36	1.54	0.77	15.78	4.49	1.36	3.52	2.03	2.45	16.85	4.73	2.62
DS _{3,nai}	Mean	7.25	3.47	1.77	20.90	7.38	2.80	8.50	4.54	2.69	22.00	8.18	3.50	7.63	4.31	3.46	21.48	7.97	4.37
	SD	2.97	1.41	0.69	14.30	4.34	1.24	3.08	1.50	0.76	14.32	4.30	1.28	3.17	2.07	2.63	14.75	4.44	2.77
X13	Mean	5.36	4.67	4.28	13.91	8.85	6.42	8.22	5.65	4.76	17.01	9.81	6.75	6.26	5.51	5.24	15.29	9.57	7.19
	SD	2.87	1.99	1.51	9.82	4.85	2.55	2.90	1.93	1.37	9.94	4.85	2.43	3.20	2.30	1.79	11.60	5.20	2.66
TS	Mean	4.58	2.09	1.32	8.05	2.55	1.07	9.13	5.20	3.11	13.15	6.87	4.21	6.30	4.01	3.31	11.64	5.36	3.73
	SD	4.95	1.88	1.20	13.23	2.85	1.04	4.99	1.96	1.23	15.16	3.79	1.48	5.36	2.49	1.84	14.96	4.88	2.35
STL	Mean	3.83	3.97	3.94	10.17	7.40	5.62	7.62	5.14	4.03	13.96	8.56	5.70	4.96	4.69	4.61	11.53	8.06	6.25
	SD	2.12	1.49	1.03	7.65	4.11	1.94	2.20	1.63	1.04	7.69	4.20	1.95	3.09	2.08	1.47	9.45	4.33	2.38
TBATS	Mean	89.38	89.46	89.51	90.84	89.92	89.12	91.79	91.46	91.47	92.85	91.80	91.32	67.26	68.99	73.11	63.24	68.61	72.24
	SD	3.80	1.03	0.58	5.48	3.84	4.57	6.75	0.77	0.42	5.07	3.55	4.40	48.17	47.39	47.67	44.35	48.58	47.62
MA	Mean	2.24	1.06	0.50	5.65	1.90	0.72	6.71	5.52	4.95	10.09	6.36	5.17	5.38	7.03	11.59	9.26	7.73	11.88
	SD	1.85	0.85	0.41	5.75	1.83	0.61	1.88	0.85	0.41	5.74	1.83	0.61	5.98	8.52	16.38	9.64	9.23	16.17

Table 10: Results of the seasonality estimation simulation study for quarterly data with error $\{E_{2,t}\}$ and $\gamma = 0.2$. The simulated mean MSE (Mean) and the MSE standard deviation (SD), both multiplied by 1000, are displayed for various methods. The smallest MSE values in each column are highlighted.

Method	Quantity	S_1						S_2						S_3					
		G_1			G_2			G_1			G_2			G_1			G_2		
		n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3
DS _{1,opt}	Mean	2.93	1.44	0.73	19.47	5.35	1.68	3.11	1.61	0.86	19.99	5.77	2.06	3.15	1.91	1.58	19.95	5.92	2.71
	SD	0.92	0.46	0.23	17.12	3.98	1.22	0.95	0.47	0.24	17.03	3.92	1.24	1.09	0.87	1.38	16.48	4.32	1.96
DS _{1,nai}	Mean	2.79	1.28	0.66	16.48	4.68	1.45	3.10	1.55	0.85	17.30	5.35	2.08	3.05	1.83	1.61	17.05	5.34	2.71
	SD	0.96	0.45	0.22	14.05	3.55	1.07	0.95	0.45	0.24	13.91	3.47	1.14	1.15	0.97	1.56	13.90	3.88	2.28
DS _{3,opt}	Mean	2.07	0.99	0.48	16.97	4.67	1.46	2.85	1.62	1.08	17.71	5.25	1.97	2.45	1.72	1.82	17.78	5.32	2.64
	SD	0.84	0.38	0.19	13.83	3.46	1.09	0.90	0.45	0.31	13.78	3.43	1.12	1.19	1.19	1.96	15.05	3.80	2.03
DS _{3,nai}	Mean	1.85	0.90	0.43	15.16	4.23	1.31	2.86	1.76	1.29	16.19	5.04	2.06	2.28	1.72	1.96	15.90	4.95	2.67
	SD	0.75	0.36	0.18	12.55	3.20	1.01	0.86	0.48	0.36	12.51	3.16	1.05	1.22	1.37	2.21	13.44	3.61	2.25
X13	Mean	1.22	1.08	1.04	9.79	4.92	2.93	3.85	1.68	1.15	12.83	5.70	3.11	2.02	1.78	1.64	10.69	5.68	3.45
	SD	0.65	0.42	0.29	9.58	3.71	2.02	0.83	0.49	0.30	9.59	3.72	2.05	1.73	0.97	0.82	9.53	4.01	1.88
TS	Mean	0.81	0.34	0.18	6.20	1.46	0.55	3.64	1.83	1.06	10.81	5.18	2.28	1.87	1.50	1.36	8.63	3.66	2.29
	SD	0.77	0.31	0.19	12.32	2.14	1.07	1.22	0.62	0.36	13.62	2.70	1.36	1.18	0.83	0.73	12.78	3.07	1.50
STL	Mean	0.93	0.92	0.92	7.30	4.24	2.58	4.72	2.11	1.01	11.11	5.40	2.68	2.07	1.74	1.57	8.54	4.97	3.13
	SD	0.54	0.35	0.24	7.05	3.15	1.73	0.59	0.47	0.26	7.13	3.20	1.74	3.07	1.29	1.03	7.66	3.60	1.77
TBATS	Mean	89.11	88.91	88.99	90.17	89.49	89.28	91.00	90.93	91.02	92.28	91.44	91.34	70.72	71.19	73.58	64.84	67.13	71.55
	SD	2.17	1.28	4.28	4.69	2.61	2.40	1.10	0.67	3.44	4.21	2.09	2.08	48.92	48.02	47.49	45.47	47.79	48.91
MA	Mean	0.52	0.24	0.12	4.18	1.10	0.34	4.98	4.70	4.58	8.64	5.56	4.79	3.88	6.01	10.53	7.58	7.16	10.45
	SD	0.45	0.19	0.10	5.12	1.30	0.40	0.47	0.20	0.10	5.17	1.32	0.41	7.08	7.88	15.00	8.82	9.31	13.58

Table 11: Results of the seasonality estimation simulation study for quarterly data with error $\{E_{3,t}\}$ and $\gamma = 0.2$. The simulated mean MSE (Mean) and the MSE standard deviation (SD), both multiplied by 1000, are displayed for various methods. The smallest MSE values in each column are highlighted.

Method	Quantity	S_1						S_2						S_3					
		G_1			G_2			G_1			G_2			G_1			G_2		
		n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3
DS _{1,opt}	Mean	2.13	0.94	0.48	18.48	5.05	1.50	2.31	1.13	0.60	18.94	5.51	1.86	2.35	1.44	1.39	19.94	5.73	2.52
	SD	0.73	0.33	0.15	15.71	4.21	1.15	0.72	0.34	0.17	15.34	4.19	1.12	0.90	0.92	1.44	16.29	4.52	1.86
DS _{1,nai}	Mean	1.97	0.84	0.43	15.50	4.36	1.29	2.34	1.14	0.61	16.41	5.10	1.89	2.23	1.42	1.45	16.84	5.22	2.52
	SD	0.76	0.31	0.15	13.53	3.72	1.00	0.71	0.34	0.17	13.38	3.69	1.02	1.21	1.03	1.57	14.05	4.25	1.99
DS _{3,opt}	Mean	1.55	0.66	0.31	16.40	4.45	1.27	2.21	1.30	0.97	17.07	5.04	1.78	1.87	1.41	1.76	17.68	5.19	2.49
	SD	0.65	0.27	0.12	13.80	3.69	0.98	0.66	0.37	0.28	13.49	3.72	0.98	1.13	1.27	2.21	14.02	4.04	1.96
DS _{3,nai}	Mean	1.33	0.59	0.28	14.64	4.04	1.14	2.27	1.51	1.20	15.68	4.86	1.89	1.73	1.45	1.92	15.92	4.85	2.56
	SD	0.58	0.25	0.12	12.59	3.43	0.91	0.65	0.41	0.33	12.60	3.45	0.91	1.11	1.45	2.54	12.96	3.80	2.24
X13	Mean	0.78	0.70	0.65	9.47	4.67	2.54	3.30	1.16	0.81	12.46	5.47	2.68	1.51	1.34	1.24	11.03	5.53	3.01
	SD	0.43	0.28	0.18	9.14	4.04	1.89	0.63	0.33	0.21	9.11	4.17	1.89	1.37	0.92	0.77	9.89	4.41	1.83
TS	Mean	0.49	0.21	0.11	6.28	1.45	0.46	2.78	1.52	0.86	10.41	4.96	2.07	1.47	1.23	1.10	9.55	3.55	2.15
	SD	0.54	0.25	0.11	12.76	2.69	0.98	1.12	0.75	0.40	11.89	2.89	1.60	0.90	0.80	0.65	17.57	3.46	1.75
STL	Mean	0.61	0.61	0.60	7.22	4.01	2.26	4.40	1.79	0.68	11.04	5.19	2.34	1.68	1.43	1.29	8.55	4.82	2.78
	SD	0.37	0.25	0.17	7.14	3.46	1.61	0.42	0.35	0.18	7.17	3.54	1.61	2.10	1.34	1.08	7.58	3.83	1.74
TBATS	Mean	89.00	88.86	89.16	90.31	89.65	89.30	90.88	90.99	91.18	92.24	91.61	91.23	68.34	69.72	73.21	63.57	69.38	73.21
	SD	1.17	0.94	1.38	4.61	2.40	1.99	1.14	2.02	1.29	4.44	1.65	1.68	48.02	47.41	48.04	44.29	47.20	48.64
MA	Mean	0.33	0.15	0.08	3.94	1.09	0.31	4.78	4.61	4.53	8.39	5.55	4.76	3.57	6.16	10.58	7.46	7.26	11.55
	SD	0.30	0.13	0.07	4.68	1.54	0.37	0.31	0.13	0.07	4.69	1.55	0.37	5.72	9.25	15.20	8.65	11.50	15.41

Table 12: Results of the seasonality estimation simulation study for monthly data with error $\{E_{1,t}\}$ and $\gamma = 0.1$. The simulated mean MSE (Mean) and the MSE standard deviation (SD), both multiplied by 1000, are displayed for various methods. The smallest MSE values in each column are highlighted.

Method	Quantity	S_1						S_2						S_3					
		G_1			G_2			G_1			G_2			G_1			G_2		
		n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3
DS _{1,opt}	Mean	3.60	2.62	1.52	15.90	4.23	1.21	3.72	2.67	1.54	17.04	5.16	1.97	3.67	2.78	1.78	15.31	4.73	2.02
	SD	0.57	0.45	0.18	13.82	3.08	0.70	0.57	0.46	0.18	13.66	3.05	0.81	0.56	0.45	0.23	11.52	3.24	0.83
DS _{1,nai}	Mean	3.53	2.34	1.47	13.77	3.76	1.09	3.67	2.39	1.49	15.31	5.04	2.11	3.61	2.52	1.74	13.77	4.34	2.01
	SD	0.55	0.36	0.18	11.55	2.80	0.63	0.56	0.36	0.18	11.50	2.80	0.82	0.55	0.37	0.24	10.47	2.89	0.83
DS _{3,opt}	Mean	2.30	1.16	0.61	16.65	4.31	1.19	3.11	1.88	1.22	17.50	5.09	1.88	2.49	1.57	1.35	16.38	4.80	2.00
	SD	0.46	0.24	0.12	14.24	3.15	0.68	0.50	0.29	0.16	14.12	3.14	0.73	0.47	0.34	0.43	12.30	3.39	0.83
DS _{3,nai}	Mean	2.17	1.10	0.58	15.11	3.93	1.08	3.15	1.96	1.29	16.25	4.96	2.03	2.38	1.54	1.36	14.70	4.46	1.98
	SD	0.45	0.23	0.11	12.93	2.88	0.62	0.48	0.28	0.16	12.78	2.88	0.71	0.46	0.34	0.44	11.06	3.11	0.81
X13	Mean	1.72	1.56	1.38	11.04	5.43	2.92	4.63	2.28	1.75	14.09	6.23	3.16	2.26	2.10	1.97	11.34	6.01	3.44
	SD	0.54	0.41	0.35	9.32	3.83	1.58	0.59	0.37	0.22	9.35	3.77	1.53	0.57	0.41	0.34	8.47	4.05	1.58
TS	Mean	1.08	0.48	0.30	4.71	1.23	0.46	4.25	2.05	1.31	9.14	5.11	2.04	1.84	1.45	1.27	6.00	2.92	1.80
	SD	0.96	0.55	0.29	5.24	1.13	0.70	0.97	0.48	0.39	5.25	1.59	1.01	0.78	0.49	0.36	4.88	1.69	0.72
STL	Mean	1.18	1.21	1.21	8.17	4.37	2.40	4.95	2.32	1.29	11.96	5.50	2.48	1.73	1.63	1.57	8.35	4.86	2.82
	SD	0.34	0.23	0.16	7.43	3.04	1.29	0.37	0.31	0.17	7.40	3.10	1.30	0.48	0.35	0.26	6.48	3.32	1.35
TBATS	Mean	1.37	1.01	0.83	5.44	1.88	0.99	5.83	5.45	5.26	9.96	6.34	5.43	26.53	28.89	30.61	30.85	30.69	32.50
	SD	0.32	0.16	0.07	5.30	1.11	0.26	0.33	0.16	0.07	5.30	1.09	0.26	26.60	27.17	27.06	28.52	29.32	32.04
MA	Mean	0.67	0.32	0.16	4.56	1.14	0.31	5.24	4.82	4.62	9.15	5.63	4.78	2.31	3.61	6.19	6.08	4.45	6.43
	SD	0.29	0.14	0.07	5.03	1.02	0.25	0.31	0.14	0.07	5.05	1.03	0.25	1.15	2.23	3.74	4.57	2.45	3.79

Table 13: Results of the seasonality estimation simulation study for monthly data with error $\{E_{2,t}\}$ and $\gamma = 0.1$. The simulated mean MSE (Mean) and the MSE standard deviation (SD), both multiplied by 1000, are displayed for various methods. The smallest MSE values in each column are highlighted.

Method	Quantity	S_1						S_2						S_3					
		G_1			G_2			G_1			G_2			G_1			G_2		
		n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3
DS _{1,opt}	Mean	1.93	0.94	0.54	14.62	3.63	0.97	2.11	1.10	0.62	15.85	4.60	1.72	2.03	1.19	0.97	13.77	3.98	1.76
	SD	0.48	0.24	0.11	13.60	3.18	0.73	0.47	0.24	0.12	13.50	3.19	0.84	0.46	0.27	0.29	11.48	2.89	0.84
DS _{1,nai}	Mean	1.78	0.84	0.49	12.79	3.30	0.87	2.20	1.08	0.61	14.39	4.55	1.90	1.89	1.12	0.97	12.28	3.70	1.76
	SD	0.56	0.21	0.11	11.40	2.80	0.67	0.47	0.24	0.12	11.44	2.84	0.84	0.52	0.27	0.31	10.18	2.64	0.83
DS _{3,opt}	Mean	1.24	0.59	0.30	15.74	3.73	0.96	2.09	1.42	1.09	16.65	4.52	1.68	1.43	1.02	1.15	14.91	4.04	1.74
	SD	0.39	0.18	0.08	13.92	3.14	0.74	0.46	0.30	0.23	13.91	3.15	0.78	0.38	0.30	0.55	12.27	3.02	0.85
DS _{3,nai}	Mean	1.13	0.54	0.27	14.15	3.39	0.87	2.25	1.63	1.28	15.34	4.46	1.85	1.34	1.02	1.20	13.36	3.79	1.74
	SD	0.36	0.17	0.08	12.79	2.91	0.68	0.48	0.32	0.23	12.72	2.93	0.76	0.37	0.32	0.59	11.06	2.76	0.82
X13	Mean	0.84	0.77	0.75	10.37	4.59	2.25	3.69	1.33	0.85	13.33	5.28	2.36	1.26	1.16	1.11	9.91	4.88	2.60
	SD	0.30	0.21	0.14	9.78	3.84	1.62	0.39	0.27	0.15	9.76	3.86	1.61	0.39	0.29	0.25	7.98	3.55	1.57
TS	Mean	0.44	0.23	0.15	4.52	1.12	0.35	2.40	1.22	0.72	8.98	4.57	1.62	1.21	1.01	0.94	5.58	2.74	1.65
	SD	0.38	0.27	0.29	5.29	1.35	0.39	0.86	0.32	0.24	5.43	2.42	0.97	0.44	0.32	0.27	4.46	1.81	0.82
STL	Mean	0.63	0.64	0.64	7.71	3.79	1.94	4.41	1.75	0.72	11.51	4.92	2.01	1.15	1.05	1.00	7.52	4.09	2.26
	SD	0.27	0.18	0.12	7.80	3.15	1.38	0.30	0.27	0.13	7.79	3.19	1.37	0.42	0.30	0.26	6.27	2.98	1.36
TBATS	Mean	1.03	0.82	0.75	5.22	1.69	0.93	5.56	5.31	5.21	9.73	6.19	5.38	27.37	29.00	31.00	31.07	30.24	32.23
	SD	0.26	0.12	0.13	5.59	1.17	0.31	0.26	0.13	0.07	5.53	1.20	0.29	27.84	27.35	37.39	28.25	29.96	29.38
MA	Mean	0.36	0.17	0.08	4.39	1.02	0.25	4.93	4.67	4.55	8.97	5.52	4.72	1.98	3.35	6.41	5.61	4.23	6.38
	SD	0.22	0.11	0.05	5.31	1.12	0.27	0.24	0.11	0.05	5.34	1.13	0.27	1.09	2.08	4.38	4.30	2.37	4.11

Table 14: Results of the seasonality estimation simulation study for monthly data with error $\{E_{3,t}\}$ and $\gamma = 0.1$. The simulated mean MSE (Mean) and the MSE standard deviation (SD), both multiplied by 1000, are displayed for various methods. The smallest MSE values in each column are highlighted.

Method	Quantity	S_1						S_2						S_3					
		G_1			G_2			G_1			G_2			G_1			G_2		
		n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3
DS _{1,opt}	Mean	2.26	1.14	0.67	14.39	3.89	1.04	2.44	1.28	0.74	15.53	4.82	1.81	2.38	1.39	1.06	14.51	4.27	1.83
	SD	0.62	0.32	0.16	12.62	3.40	0.75	0.62	0.33	0.18	12.53	3.33	0.82	0.61	0.36	0.27	13.01	3.10	0.96
DS _{1,nai}	Mean	2.06	1.02	0.62	12.78	3.50	0.93	2.48	1.24	0.72	14.31	4.74	1.99	2.20	1.31	1.05	12.90	3.92	1.82
	SD	0.68	0.30	0.16	11.12	3.09	0.68	0.61	0.32	0.17	11.12	3.05	0.82	0.69	0.34	0.28	11.51	2.74	0.92
DS _{3,opt}	Mean	1.47	0.70	0.35	15.45	3.95	1.00	2.35	1.57	1.17	16.32	4.73	1.71	1.68	1.15	1.18	15.30	4.29	1.80
	SD	0.51	0.25	0.12	13.09	3.44	0.70	0.58	0.35	0.24	13.12	3.39	0.75	0.53	0.36	0.48	13.19	3.05	0.92
DS _{3,nai}	Mean	1.33	0.65	0.33	14.01	3.57	0.91	2.49	1.76	1.34	15.14	4.62	1.89	1.55	1.13	1.23	13.91	3.99	1.80
	SD	0.48	0.24	0.11	12.04	3.13	0.65	0.59	0.36	0.24	12.05	3.08	0.74	0.50	0.37	0.51	12.07	2.77	0.91
X13	Mean	0.97	0.89	0.86	9.99	4.78	2.32	3.77	1.45	0.98	12.87	5.50	2.44	1.41	1.29	1.20	10.39	5.16	2.71
	SD	0.39	0.28	0.18	9.18	4.00	1.51	0.50	0.34	0.21	9.17	4.02	1.53	0.48	0.35	0.26	9.00	3.54	1.67
TS	Mean	0.51	0.34	0.86	4.34	1.19	0.45	2.77	1.63	1.70	8.82	4.62	1.74	1.56	1.53	1.39	5.68	2.92	1.86
	SD	0.48	0.51	1.44	5.07	1.49	0.68	0.92	0.79	1.89	5.10	2.32	1.02	0.60	0.83	0.91	4.45	1.81	0.89
STL	Mean	0.76	0.76	0.77	7.48	3.99	2.03	4.53	1.87	0.85	11.24	5.11	2.11	1.29	1.19	1.11	7.75	4.33	2.39
	SD	0.36	0.26	0.17	7.23	3.42	1.29	0.40	0.34	0.19	7.23	3.43	1.29	0.50	0.36	0.26	6.79	2.99	1.48
TBATS	Mean	1.12	0.86	0.76	4.92	1.75	0.95	5.64	5.36	5.22	9.47	6.23	5.38	26.42	28.71	32.20	30.09	31.05	43.35
	SD	0.35	0.18	0.12	5.01	1.26	0.37	0.33	0.16	0.08	4.98	1.26	0.28	27.46	27.56	28.76	26.95	28.51	285.12
MA	Mean	0.41	0.21	0.10	4.18	1.05	0.25	4.99	4.70	4.57	8.76	5.55	4.72	2.15	3.46	6.19	5.68	4.29	6.47
	SD	0.30	0.15	0.08	4.77	1.19	0.25	0.32	0.16	0.08	4.80	1.19	0.26	1.24	2.07	3.61	4.44	2.61	3.95

Table 15: Results of the seasonality estimation simulation study for monthly data with error $\{E_{1,t}\}$ and $\gamma = 0.2$. The simulated mean MSE (Mean) and the MSE standard deviation (SD), both multiplied by 1000, are displayed for various methods. The smallest MSE values in each column are highlighted.

Method	Quantity	S_1						S_2						S_3					
		G_1			G_2			G_1			G_2			G_1			G_2		
		n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3
DS _{1,opt}	Mean	14.08	8.30	4.65	22.47	7.66	3.05	14.24	8.40	4.70	23.64	8.56	3.79	14.08	8.47	5.00	21.82	8.20	3.86
	SD	2.22	1.24	0.63	14.78	3.48	1.01	2.21	1.25	0.63	14.57	3.36	0.95	2.16	1.25	0.69	12.40	3.62	1.02
DS _{1,nai}	Mean	13.63	7.74	4.35	19.59	6.80	2.75	13.80	7.86	4.41	21.14	8.07	3.75	13.67	7.96	4.73	19.59	7.44	3.65
	SD	2.37	1.28	0.61	12.25	3.16	0.91	2.32	1.28	0.62	12.18	3.04	0.89	2.26	1.27	0.67	11.23	3.26	0.99
DS _{3,opt}	Mean	8.66	4.29	2.19	23.55	7.78	2.99	9.71	5.23	3.04	24.41	8.56	3.66	8.82	4.72	3.03	23.39	8.28	3.79
	SD	1.94	0.95	0.45	14.83	3.46	0.90	1.88	0.96	0.49	14.74	3.43	0.90	1.89	0.96	0.67	13.15	3.70	0.97
DS _{3,nai}	Mean	7.87	3.96	2.06	21.38	7.09	2.71	9.25	5.15	3.08	22.57	8.12	3.65	8.07	4.45	2.96	20.95	7.65	3.61
	SD	1.83	0.90	0.43	13.54	3.17	0.84	1.77	0.92	0.48	13.49	3.13	0.86	1.79	0.91	0.68	11.82	3.46	0.95
X13	Mean	6.83	5.84	5.51	16.32	10.11	7.11	10.09	7.38	6.35	19.57	11.30	7.83	7.45	6.62	6.28	16.57	10.78	7.84
	SD	2.17	1.69	1.40	10.06	4.43	2.36	1.98	1.34	1.24	9.93	4.11	2.10	2.04	1.63	1.39	9.17	4.50	2.20
TS	Mean	3.46	1.98	1.08	6.83	2.33	1.11	7.68	5.93	3.37	11.19	6.57	4.42	4.62	3.49	2.85	8.11	4.35	3.06
	SD	2.78	2.45	1.57	5.63	1.67	1.08	2.43	2.59	1.55	5.69	1.67	1.25	2.71	1.92	1.33	5.17	2.00	1.09
STL	Mean	4.71	4.82	4.86	11.71	7.95	6.03	8.48	5.93	4.94	15.52	9.08	6.11	5.24	5.23	5.21	11.81	8.46	6.46
	SD	1.35	0.94	0.64	7.95	3.27	1.49	1.38	1.02	0.65	7.93	3.34	1.50	1.34	0.97	0.70	6.96	3.61	1.49
TBATS	Mean	3.64	2.11	1.37	7.81	2.96	1.54	8.14	6.54	5.78	12.28	7.43	5.97	28.35	29.64	31.02	32.47	31.04	31.90
	SD	1.35	0.64	0.29	5.90	1.36	0.43	1.39	0.66	0.30	5.89	1.37	0.44	26.42	27.05	26.75	27.97	28.22	28.60
MA	Mean	2.66	1.27	0.62	6.57	2.09	0.78	7.24	5.77	5.09	11.16	6.58	5.24	4.29	4.57	6.66	8.09	5.40	6.90
	SD	1.14	0.56	0.26	5.52	1.26	0.39	1.17	0.56	0.26	5.53	1.26	0.39	1.53	2.30	3.75	5.04	2.57	3.81

Table 16: Results of the seasonality estimation simulation study for monthly data with error $\{E_{2,t}\}$ and $\gamma = 0.2$. The simulated mean MSE (Mean) and the MSE standard deviation (SD), both multiplied by 1000, are displayed for various methods. The smallest MSE values in each column are highlighted.

Method	Quantity	S_1						S_2						S_3					
		G_1			G_2			G_1			G_2			G_1			G_2		
		n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3
DS _{1,opt}	Mean	5.91	3.03	1.68	17.85	5.40	1.90	6.56	3.40	1.91	19.17	6.41	2.68	5.91	3.33	2.25	17.08	5.77	2.69
	SD	2.01	0.85	0.41	13.80	3.54	0.88	1.82	0.85	0.42	13.90	3.46	0.90	1.89	0.85	0.53	11.97	3.21	0.96
DS _{1,nai}	Mean	4.81	2.64	1.51	15.72	4.91	1.70	5.91	3.25	1.85	17.32	6.20	2.79	4.91	2.99	2.15	15.23	5.31	2.60
	SD	1.73	0.79	0.39	11.80	3.10	0.82	1.56	0.80	0.41	11.67	3.07	0.87	1.60	0.79	0.55	10.72	2.91	0.92
DS _{3,opt}	Mean	4.87	2.31	1.13	19.39	5.53	1.88	5.82	3.25	2.08	20.31	6.36	2.63	5.00	2.75	2.03	18.65	5.87	2.66
	SD	1.55	0.74	0.33	14.31	3.40	0.89	1.52	0.76	0.44	14.25	3.37	0.88	1.47	0.75	0.68	12.73	3.31	0.96
DS _{3,nai}	Mean	4.35	2.06	1.02	17.42	5.03	1.70	5.66	3.38	2.30	18.65	6.14	2.72	4.51	2.56	2.03	16.69	5.45	2.58
	SD	1.44	0.69	0.32	13.23	3.16	0.81	1.44	0.76	0.45	13.14	3.15	0.84	1.37	0.72	0.72	11.47	3.04	0.92
X13	Mean	3.34	3.09	3.01	12.89	6.96	4.50	6.25	3.78	3.15	15.90	7.68	4.67	3.84	3.55	3.41	12.49	7.27	4.88
	SD	1.22	0.84	0.55	10.26	4.13	1.76	1.24	0.87	0.53	10.24	4.11	1.73	1.21	0.82	0.56	8.44	3.82	1.69
TS	Mean	1.53	0.77	0.45	5.67	1.73	0.75	6.06	4.30	2.06	10.14	5.98	3.08	2.74	2.29	2.02	6.90	3.69	2.54
	SD	1.15	0.73	0.59	5.85	1.82	0.94	1.37	1.40	0.83	5.98	2.26	1.36	1.27	1.07	0.62	5.36	2.21	1.12
STL	Mean	2.51	2.54	2.56	9.53	5.73	3.87	6.28	3.65	2.64	13.33	6.85	3.94	2.97	2.95	2.91	9.37	6.01	4.18
	SD	1.07	0.73	0.48	8.12	3.39	1.50	1.10	0.83	0.49	8.09	3.44	1.50	1.05	0.76	0.53	6.68	3.23	1.48
TBATS	Mean	2.08	1.34	1.00	6.30	2.25	1.16	6.62	5.82	5.45	10.85	6.74	5.62	28.25	30.26	30.38	32.33	30.28	32.15
	SD	0.94	0.46	0.22	6.06	1.40	0.40	0.94	0.45	0.21	6.04	1.42	0.38	28.02	28.08	26.29	28.68	28.19	27.80
MA	Mean	1.43	0.67	0.33	5.41	1.54	0.49	6.00	5.17	4.80	9.99	6.04	4.96	3.01	3.86	6.65	6.65	4.76	6.62
	SD	0.89	0.42	0.20	5.68	1.37	0.37	0.91	0.43	0.20	5.72	1.38	0.37	1.33	2.13	4.38	4.72	2.49	4.12

Table 17: Results of the seasonality estimation simulation study for monthly data with error $\{E_{3,t}\}$ and $\gamma = 0.2$. The simulated mean MSE (Mean) and the MSE standard deviation (SD), both multiplied by 1000, are displayed for various methods. The smallest MSE values in each column are highlighted.

Method	Quantity	S_1						S_2						S_3					
		G_1			G_2			G_1			G_2			G_1			G_2		
		n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3	n_1	n_2	n_3
DS _{1,opt}	Mean	6.89	3.68	2.11	18.39	5.97	2.15	7.54	4.02	2.30	19.56	6.94	2.94	7.09	4.00	2.61	18.39	6.35	2.93
	SD	2.39	1.17	0.59	13.53	3.90	1.01	2.31	1.17	0.61	13.35	3.78	0.99	2.55	1.18	0.65	13.71	3.57	1.19
DS _{1,nai}	Mean	5.74	3.24	1.91	16.32	5.37	1.92	6.79	3.78	2.20	17.87	6.65	3.02	5.81	3.60	2.48	16.38	5.78	2.82
	SD	2.19	1.11	0.55	11.94	3.59	0.94	2.07	1.11	0.59	11.82	3.47	0.96	2.19	1.10	0.65	12.13	3.14	1.12
DS _{3,opt}	Mean	5.72	2.72	1.34	19.87	6.06	2.09	6.72	3.72	2.35	20.81	6.88	2.84	5.94	3.17	2.22	19.55	6.42	2.89
	SD	2.06	1.02	0.47	13.84	3.88	0.94	2.08	1.03	0.56	13.90	3.81	0.96	2.09	1.05	0.68	13.80	3.47	1.17
DS _{3,nai}	Mean	5.09	2.45	1.22	17.95	5.48	1.90	6.45	3.80	2.53	19.11	6.58	2.91	5.29	2.96	2.20	17.72	5.92	2.80
	SD	1.91	0.96	0.45	12.76	3.56	0.89	1.94	1.00	0.55	12.73	3.49	0.93	1.91	1.00	0.70	12.55	3.16	1.14
X13	Mean	3.90	3.57	3.44	12.88	7.43	4.87	6.74	4.24	3.57	15.80	8.15	5.02	4.41	4.05	3.82	13.25	7.84	5.31
	SD	1.56	1.10	0.72	9.75	4.43	1.74	1.58	1.17	0.72	9.75	4.42	1.75	1.62	1.13	0.75	9.33	3.91	1.92
TS	Mean	1.85	1.57	5.09	5.65	2.11	1.59	6.81	5.04	5.18	10.38	6.50	3.73	3.62	4.41	3.67	7.28	4.29	3.39
	SD	1.57	3.15	8.60	5.39	2.87	3.87	2.42	3.89	7.71	6.58	3.54	3.44	1.91	4.71	4.30	5.22	2.82	2.55
STL	Mean	3.02	3.05	3.06	9.72	6.21	4.30	6.79	4.15	3.15	13.48	7.34	4.39	3.49	3.48	3.39	9.94	6.57	4.66
	SD	1.44	1.03	0.70	7.86	3.84	1.53	1.48	1.12	0.72	7.87	3.86	1.54	1.46	1.06	0.73	7.17	3.33	1.76
TBATS	Mean	2.30	1.48	1.06	6.20	2.33	1.21	6.85	5.99	5.52	10.74	6.83	5.68	27.65	29.46	32.44	31.10	31.39	32.62
	SD	1.21	0.62	0.30	5.58	1.58	0.44	1.25	0.63	0.31	5.50	1.60	0.45	27.55	27.84	28.32	26.77	28.30	28.64
MA	Mean	1.64	0.82	0.40	5.47	1.63	0.53	6.22	5.32	4.86	10.05	6.13	5.00	3.39	4.04	6.49	6.92	4.88	6.75
	SD	1.19	0.61	0.30	5.39	1.54	0.42	1.22	0.62	0.30	5.42	1.54	0.42	1.66	2.14	3.63	4.90	2.73	3.98

References

- H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974. URL <https://doi.org/10.1109/TAC.1974.1100705>. [p251]
- W. R. Bell and S. C. Hillmer. Issues involved with the seasonal adjustment of economic time series. *Journal of Business & Economic Statistics*, 2(4):291–320, 1984. URL <https://doi.org/10.1080/07350015.1984.10509398>. [p246]
- H. Bengtsson. A unifying framework for parallel and distributed processing in R using futures. *The R Journal*, 13(2):273–291, 2021. URL <https://doi.org/10.32614/RJ-2021-048>. [p252]
- J. Beran and Y. Feng. Local polynomial fitting with long-memory, short-memory and antipersistent errors. *Annals of the Institute of Statistical Mathematics*, 54(2):291–311, 2002. URL <https://doi.org/10.1023/A:1022469818068>. [p249]
- J. Beran and D. Ocker. SEMIFAR forecasts, with applications to foreign exchange rates. *Journal of Statistical Planning and Inference*, 80(1):137–153, 1999. URL [https://doi.org/10.1016/S0378-3758\(98\)00247-X](https://doi.org/10.1016/S0378-3758(98)00247-X). [p251]
- G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time series analysis: Forecasting and control*. John Wiley & Sons, Hoboken, NJ, 4th edition, 2008. ISBN 978-0-470-27284-8. [p252]
- P. J. Brockwell and R. A. Davis. *Introduction to time series analysis and forecasting*. Springer, Cham, Switzerland, 3rd edition, 2016. URL <https://doi.org/10.1007/978-3-319-29854-2>. [p246]
- P. Bühlmann. Locally adaptive lag-window spectral estimation. *Journal of Time Series Analysis*, 17(3):247–270, 1996. URL <https://doi.org/10.1111/j.1467-9892.1996.tb00275.x>. [p247, 250, 251]
- W. Chang, J. Cheng, J. Allaire, C. Sievert, B. Schloerke, Y. Xie, J. Allen, J. McPherson, A. Dipert, and B. Borges. *shiny: Web application framework for R*, 2025. URL <https://doi.org/10.32614/CRAN.package.shiny>. R package version 1.11.1. [p254]
- U. Cieplik. Bv4.1 – Methodology and user-friendly software for decomposing economic time series. In *Conference on seasonality, seasonal adjustment and their implications for short-term analysis and forecasting*, Luxembourg, 2006. Office for Official Publications of the European Communities. URL <https://ec.europa.eu/eurostat/documents/3888793/5841609/KS-DT-06-012-EN.PDF/8d4340d2-de7e-45f8-b1e9-c95a0ca1ee4b>. [p246]
- R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning. STL: A seasonal-trend decomposition procedure based on Loess. *Journal of Official Statistics*, 6(1):3–73, 1990. URL <http://www.nniem.ru/file/news/2016/stl-statistical-model.pdf>. [p247, 254]
- W. S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74(368):829–836, 1979. URL <https://doi.org/10.1080/01621459.1979.10481038>. [p247]
- A. M. De Livera, R. J. Hyndman, and R. D. Snyder. Forecasting time series with complex seasonal patterns using exponential smoothing. *Journal of the American Statistical Association*, 106(496):1513–1527, 2011. URL <https://doi.org/10.1198/jasa.2011.tm09771>. [p247]
- D. DeJong, J. Nankervis, N. Savin, and C. Whiteman. Integration versus trend stationarity in time series. *Econometrica*, 60(2):423–433, 1992. URL <https://doi.org/10.2307/2951602>. [p247]
- A. Dokumentov and R. J. Hyndman. *stR: Seasonal trend decomposition using regression*, 2025. URL <https://doi.org/10.32614/CRAN.package.stR>. R package version 0.7.1. [p247]
- D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL <https://doi.org/10.18637/jss.v040.i08>. [p253]
- D. Eddelbuettel and C. Sanderson. RcppArmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, 71:1054–1063, March 2014. URL <https://doi.org/10.1016/j.csda.2013.02.005>. [p254]
- D. Eddelbuettel and C. Sax. *x13binary: Provide the 'x13ashtml' seasonal adjustment binary*, 2025. URL <https://doi.org/10.32614/CRAN.package.x13binary>. R package version 1.1.61.1. [p247]
- European Commission and Eurostat. *ESS guidelines on seasonal adjustment – 2015 edition*. Publications Office of the European Union, 2015. URL <https://doi.org/10.2785/731879>. [p246]

- European Commission and Eurostat. *ESS guidelines on seasonal adjustment – 2024 edition*. Publications Office of the European Union, 2024. URL <https://doi.org/10.2785/705102>. [p246]
- J. Fan and I. Gijbels. *Local polynomial modelling and its applications: Monographs on statistics and applied probability* 66. Routledge, New York, 1st edition, 1996. URL <https://doi.org/10.1201/9780203748725>. [p248]
- Y. Feng. *Kernel- and locally weighted regression – With application to time series decomposition*. Verlag für Wissenschaft und Forschung, Berlin, Germany, 1st edition, 1999. [p246, 249]
- Y. Feng. An iterative plug-in algorithm for decomposing seasonal time series using the Berlin method. *Journal of Applied Statistics*, 40(2):266–281, 2013. URL <https://doi.org/10.1080/02664763.2012.740626>. [p246, 247, 248, 249, 250, 251]
- Y. Feng and D. Schulz. *deseats: Data-driven locally weighted regression for trend and seasonality in TS*, 2026. URL <https://doi.org/10.32614/CRAN.package.deseats>. R package version 1.1.2. [p246]
- Y. Feng, T. Gries, and M. Fritz. Data-driven local polynomial for the trend and its derivatives in economic time series. *Journal of Nonparametric Statistics*, 32(2):510–533, 2020. URL <https://doi.org/10.1080/10485252.2020.1759598>. [p247, 250, 251, 253]
- Y. Feng, T. Gries, S. Letmathe, and D. Schulz. The smoots package in R for semiparametric modeling of trend stationary time series. *The R Journal*, 14(1):182–195, 2022. URL <https://doi.org/10.32614/rj-2022-017>. [p247, 250]
- D. F. Findley, B. C. Monsell, W. R. Bell, M. C. Otto, and B.-C. Chen. New capabilities and methods of the X-12-ARIMA seasonal-adjustment program. *Journal of Business & Economic Statistics*, 16(2):127–152, 1998. URL <https://doi.org/10.1080/07350015.1998.10524743>. [p246]
- M. Francisco-Fernández, J. Opsomer, and J. M. Vilar-Fernández. A plug-in bandwidth selector for local polynomial regression estimator with correlated errors. *Journal of Nonparametric Statistics*, 16(1–2):127–151, 2004. URL <https://doi.org/10.1080/10485250310001622848>. [p251]
- M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, Dec. 1937. URL <https://doi.org/10.2307/2279372>. [p258]
- M. Friedman. A correction: The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 34(205):109, Mar. 1939. URL <https://doi.org/10.2307/2279169>. [p258]
- T. Gasser, A. Kneip, and W. Köhler. A flexible and fast method for automatic smoothing. *Journal of the American Statistical Association*, 86(415):643–652, 1991. URL <https://doi.org/10.1080/01621459.1991.10475090>. [p246]
- V. Gómez and A. Maravall Herrero. *Programs TRAMO and SEATS: Instructions for the user (beta version: September 1996)*. Banco de España. Servicio de Estudios, 1996. URL <https://www.bde.es/f/webbde/SES/Secciones/Publicaciones/PublicacionesSeriadas/DocumentosTrabajo/96/Fich/dt9628e.pdf>. [p246]
- R. Hafen. *stlplus: Enhanced seasonal decomposition of time series by Loess*, 2016. URL <https://doi.org/10.32614/CRAN.package.stlplus>. R package version 0.5.1. [p247]
- J. D. Hamilton. Why you should never use the Hodrick-Prescott filter. *Review of Economics and Statistics*, 100(5):831–843, 2018. URL https://doi.org/10.1162/rest_a_00706. [p247]
- S. Heiler. Theoretische Grundlagen des “Berliner Verfahrens”. In: Wolfgang Wetzel: *Neuere Entwicklungen auf dem Gebiet der Zeitreihenanalyse, Sonderheft*, 1:67–93, 1970. [p246]
- S. Heiler and Y. Feng. Data-driven decomposition of seasonal time series. *Journal of Statistical Planning and Inference*, 91(2):351–363, 2000. URL [https://doi.org/10.1016/S0378-3758\(00\)00187-7](https://doi.org/10.1016/S0378-3758(00)00187-7). [p248]
- S. Heiler and Y. Feng. A robust data-driven version of the Berlin method. *CoFE Discussion Paper, No. 00/15*. University of Konstanz, Center of Finance and Econometrics (CoFE), 2004. URL <https://www.econstor.eu/bitstream/10419/85180/1/dp00-15.pdf>. [p246]
- S. C. Hillmer and G. C. Tiao. An ARIMA-model-based approach to seasonal adjustment. *Journal of the American Statistical Association*, 77(377):63–70, 1982. URL <https://doi.org/10.1080/01621459.1982.10477767>. [p246]

- R. J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and practice*. OTexts, Melbourne, Australia, 3rd edition, 2021. URL <https://otexts.com/fpp3/>. [p252]
- R. J. Hyndman and Y. Khandakar. Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software*, 27(3):1–22, 2008. URL <https://doi.org/10.18637/jss.v027.i03>. [p247, 254]
- A. Kowarik, A. Meraner, M. Templ, and D. Schopfhauser. Seasonal adjustment with the R packages x12 and x12GUI. *Journal of Statistical Software*, 62:1–21, 2014. URL <https://doi.org/10.18637/jss.v062.i02>. [p247]
- A. Q. la Tente, A. Michalek, J. Palate, and R. Baeyens. *RJDemetra: Interface to 'JDemetra+' seasonal adjustment Software*, 2024. URL <https://doi.org/10.32614/CRAN.package.RJDemetra>. R package version 0.2.8. [p247]
- G. M. Ljung and G. E. P. Box. On a measure of a lack of fit in time series models. *Biometrika*, 65(2): 297–303, 1978. URL <https://doi.org/10.1093/biomet/65.2.297>. [p258]
- G. L. Mazzi, D. Ladiray, and D. A. Rieser. *Handbook on seasonal adjustment*. Publications Office of the European Union, Luxembourg, Luxembourg, 2018. URL <https://ec.europa.eu/eurostat/documents/3859598/8939616/KS-GQ-18-001-EN-N.pdf/7c4d120a-4b8a-441b-aefd-6afe81a7cf59?t=1533194231000>. [p246]
- D. A. Pierce. Seasonal adjustment when both deterministic and stochastic seasonality are present. In A. Zellner, editor, *Seasonal analysis of economic time series*, pages 242–280. National Bureau of Economic Research, Cambridge, MA, 1978. URL <http://www.nber.org/chapters/c4327>. [p246, 247]
- R Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2026. URL <https://www.R-project.org/>. [p246, 247]
- C. Sax. *seasonalview: Graphical user interface for seasonal adjustment*, 2014. URL <https://doi.org/10.32614/CRAN.package.seasonalview>. R package version 1.0.0. [p247]
- C. Sax and D. Eddelbuettel. Seasonal adjustment by X-13ARIMA-SEATS in R. *Journal of Statistical Software*, 87(11):1–17, 2018. URL <https://doi.org/10.18637/jss.v087.i11>. [p247]
- D. Schulz, Y. Feng, T. Gries, M. Fritz, and S. Letmathe. Diagnosing the trend and bootstrapping the forecasting intervals using a semiparametric ARMA. Working paper 168 at the Center for International Economics (CIE), Paderborn University, 2026. URL <https://groups.uni-paderborn.de/wp-wiwi/RePEc/pdf/ciepap/WP168.pdf>. [p252]
- G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. URL <https://www.jstor.org/stable/2958889>. [p251]
- J. M. Shea. *neverhpfilter: An alternative to the Hodrick-Prescott filter*, 2025. URL <https://doi.org/10.32614/CRAN.package.neverhpfilter>. R package version 0.5-0. [p247]
- H.-T. Speth. *Komponentenzerlegung und Saisonbereinigung ökonomischer Zeitreihen mit dem Verfahren BV4.1*. German Federal Statistical Office, Wiesbaden, Germany, 2004. URL https://www.destatis.de/DE/Methoden/Saisonbereinigung/BV41-methodenbericht-Heft3_2004.pdf?__blob=publicationFile. [p246, 247]
- U.S. Census Bureau. *X-13ARIMA-SEATS reference manual*. Washington, D.C., USA, 2025. URL <https://www2.census.gov/software/x-13arima-seats/x13as/unix-linux/documentation/docx13as.pdf>. [p246]
- H. Wickham. *ggplot2: Elegant graphics for data analysis*. Springer, Cham, Switzerland, 2016. URL <https://doi.org/10.1007/978-3-319-24277-4>. [p255]
- Y. Xie. animation: An R package for creating animations and demonstrating statistical methods. *Journal of Statistical Software*, 53(1):1–27, 2013. URL <https://doi.org/10.18637/jss.v053.i01>. [p254]

Dominik Schulz
Paderborn University
Department Economics
Faculty of Business Administration and Economics
Warburger Straße 100, 33098 Paderborn, Germany
dominik.schulz@uni-paderborn.de

Bioconductor Notes, July 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

```
#> Warning: package 'BiocPkgTools' was built under R version 4.5.2
```

Bioconductor [3.23](#), released in April 2026, is now available. It is compatible with R 4.6 and consists of 2418 software packages, 436 experiment data packages, 928 up-to-date annotation packages, 28 workflows, and 8 books. [Books](#) are built regularly from source, ensuring full reproducibility; an example is the community-developed [Orchestrating Single-Cell Analysis with Bioconductor](#).

3 Core Team and Infrastructure Updates

3.1 New Bioconductor Package Submission Process

Bioconductor is moving towards using R-universe for its daily build system. As we move in this direction it was also necessary to update the submission process for Bioconductor packages. While the daily builders are still transitioning, the new submission process location is now live. The new system utilizes GitHub Actions to trigger review milestones and R-Universe as the build/check backend. The new system provides a smoother experience; it is more automated and avoids administrative steps that have historically bottlenecked the review process. See [New Package Submission Process](#) blog post for relevant links and documentation.

3.2 New packages

NEWS summaries for three contributed packages chosen at random from the 94 new software contributions are:

- **fourSynergy**: An ensemble algorithm for improved 4C-seq interaction calling.
- **DOTSeq**: A framework for differential ORF translation analysis using matched Riboseq and RNA-seq data.
- **singIST**: Toolkits for singIST analysis using pseudobulked Seurat objects from disease models and human data.

See the NEWS section in the [release announcement](#) for a complete account of changes throughout the ecosystem.

3.3 Bioconductor Maintainer Validation System

Bioconductor policies include being an active and reachable maintainer. Maintainer emails in the DESCRIPTION of packages often go stale as maintainers change positions. There is also a necessity to have maintainers opt into Bioconductor policies and procedures as they change over time. We have created an application that uses Amazon Simple Email Service (SES) to send periodic emails to maintainers to check if the endpoint is reachable and to send a verification opt-in of current Bioconductor policies and procedures and code of conduct once a year. We expect this system to improve over time. A summary of the current implementation is available on the [Bioconductor blog post](#)

3.4 Looking Ahead

Bioconductor is in the process of transitioning to utilize R-universe for daily builds of Bioconductor packages. Updates will be posted on Bioconductor blogs and mailing lists as we progress forward.

Bioconductor is also transitioning from utilizing `git.bioconductor.org` privately hosted git server to GitHub for canonical Bioconductor package codebase. There will be instructions to developers with necessary updates as we progress forward. End users should not be affected in any way.

4 Community and Impact

4.1 Community Team Updates

The Bioconductor Community Team continues to expand developer and community engagement activities.

In April 2026, Developer Engagement Lead Nicholas Cooley published a [blog post](#) introducing the new Developer Engagement Lead role and highlighting current initiatives, including the Developer Forum, the Champions Programme, developer hackathons, and improving Bioconductor documentation and discoverability.

The monthly [Developer Forum](#) brings together package developers and community members to discuss technical topics, share expertise, and identify areas for improvement across the Bioconductor ecosystem. Developer hackathons complement these activities by providing opportunities for collaborative problem solving, mentoring, and community building. A developer hackathon was held alongside EuroBioC2026 in Turku, Finland, on 1–2 June 2026, with a further hackathon planned as a post-conference event at BioC2026 in Seattle on 13–14 August 2026.

The Bioconductor Africa seminar series, launched in April 2026 and led by Laurah Nyasita Ondari, provides regular online seminars for researchers across Africa. Recordings and upcoming events are available through the [Bioconductor Training website](#).

4.2 Regional Community Activities

In addition, the BioAsia working group recently launched a bi-monthly seminar series highlighting the research being undertaken throughout the Asia-Pacific, and aiming to bring together members of the Bioconductor community within the region.

The Bioconductor LATAM seminar series hosted six seminars featuring researchers from across the Bioconductor community. The talks covered genomics, transcriptomics, proteomics, and software development, showcasing the breadth of research and development activities in the region and helping to strengthen connections among Bioconductor users, researchers, and developers throughout Latin America.

Details of upcoming seminars are available on the Bioconductor [events page](#).

4.3 Outreachy Internships

The impact of Outreachy participation continues beyond the internship period. Former Outreachy intern Chioma Oselu remains actively involved in the Bioconductor community through BugSigDB. In April 2026, Chioma Oselu and Svetlana Ugarcina Perovic delivered a [BugSigDB workshop](#) at the International Biocuration Conference. Chioma was also awarded a travel fellowship and received first prize for her [conference presentation](#).

Several current and former Outreachy interns and applicants continue to contribute to BugSigDB through three active community curation teams, demonstrating the lasting engagement fostered through the programme.

5 Conferences and Workshops

5.1 Upcoming

- **BioC2026:** The [North American Bioconductor Conference](#) will be held 10 to 12 August 2026 in Seattle. Registration is open until 27 July. This year there will be two post-conference events 13 to 14 August, a Bioconductor Carpentry single-cell workshop and a hackathon. Registration for those is [here](#).
- **BioCAsia 2026:** BioCAsia 2026 will be held as part of the ABACBS conference in Melbourne, 19 to 20 November. For more information, visit the [conference website](#).
- **EuroBioC2027:** The 2027 European Bioconductor Conference will take place in Basel, Switzerland 8-10 September 2027.

5.2 Recap

- **EuroBioC2026:** The European Bioconductor Conference was held in June 2026 in Turku, Finland. A blog post recap of the conference can be found in the [Bioconductor blog](#). Recordings of talks are available in a [YouTube playlist](#).

6 Boards and Working Groups Updates

Two new working groups have recently been established:

- [Early Career Council](#) to strengthen support and engagement for early-career community members.
- [FAIR Data Sharing](#) to coordinate community activities around FAIR data practices and interoperability.

The annual call for new members of the Technical Advisory Board (TAB) and Community Advisory Board (CAB) is also [open](#).

7 Using Bioconductor

Start using Bioconductor by installing the most recent version of R and evaluating the commands

```
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install()
```

Install additional packages and dependencies, e.g., [SingleCellExperiment](#), with

```
BiocManager::install("SingleCellExperiment")
```

[Docker](#) images provides a very effective on-ramp for power users to rapidly obtain access to standardized and scalable computing environments. Key resources include:

- [bioconductor.org](#) to install, learn, use, and develop Bioconductor packages.
- A list of [available software](#) linking to pages describing each package.
- A question-and-answer style [user support site](#) and developer-oriented [mailing list](#).
- A community Zulip workspace ([sign up](#)) for extended technical discussion.
- The [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

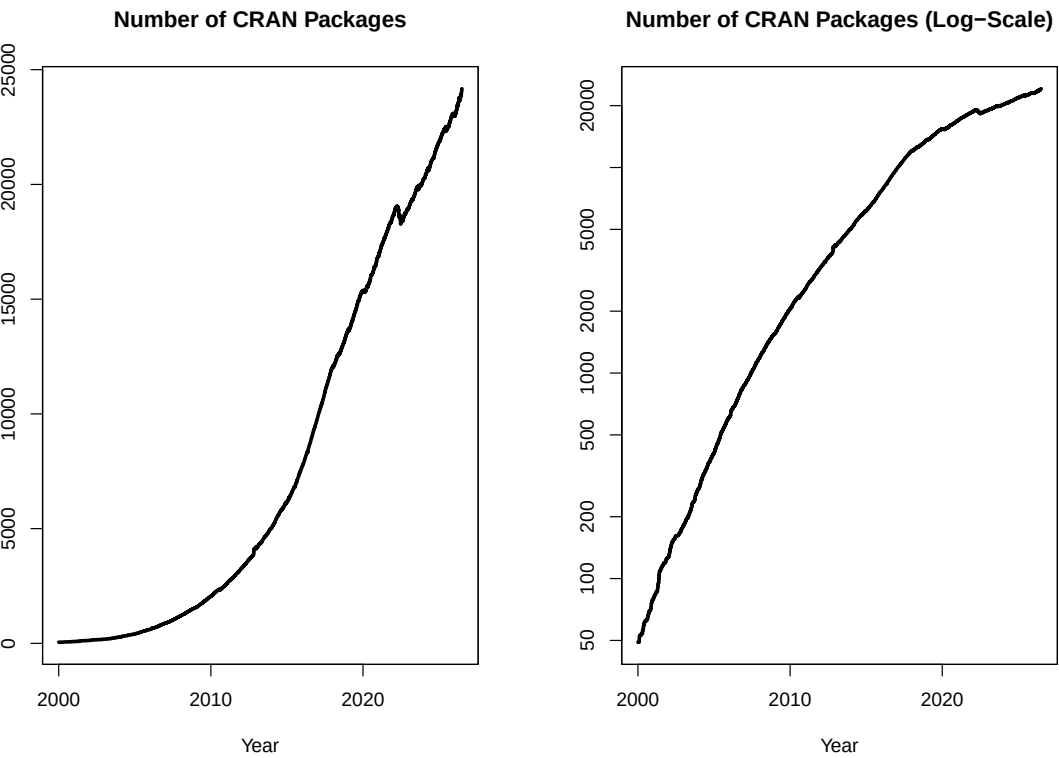
Changes on CRAN

2026-01-01 to 2026-06-30

by Kurt Hornik, Uwe Ligges, and Achim Zeileis

1 CRAN growth

In the past 6 months, 1992 new packages were added to the CRAN package repository. 462 packages were unarchived, 1273 were archived and 1 had to be removed. The following shows the growth of the number of active packages in the CRAN package repository:



On 2026-06-30, the number of active packages was around 24105.

2 CRAN package submissions

From January 2026 to June 2026 CRAN received 18293 package submissions. For these, 30170 actions took place of which 21673 (72%) were auto processed actions and 8497 (28%) manual actions.

Minus some special cases, a summary of the auto-processed and manually triggered actions follows:

	archive	inspect	newbies	pending	pretest	publish	recheck	waiting
auto	5153	1553	5275	312	0	5738	1983	798
manual	3270	3	57	49	145	4051	642	256

These include the final decisions for the submissions which were

	archive	publish
auto	4836 (27.1%)	5208 (29.2%)
manual	3244 (18.2%)	4558 (25.5%)

where we only count those as *auto* processed whose publication or rejection happened automatically in all steps.

3 CRAN mirror security

Currently, there are 88 official CRAN mirrors, 72 of which provide both secure downloads via 'https' and use secure mirroring from the CRAN master (via rsync through ssh tunnels). Since the R 3.4.0 release, `chooseCRANmirror()` offers these mirrors in preference to the others which are not fully secured (yet).

4 CRAN Task View Initiative

Currently, there are 49 task views (see <https://CRAN.R-project.org/web/views/>), with median and mean numbers of CRAN packages covered 108 and 123, respectively. Overall, these task views cover 4996 CRAN packages, which is about 21% of all active CRAN packages.

Kurt Hornik
WU Wirtschaftsuniversität Wien
Austria
ORCID: [0000-0003-4198-9911](https://orcid.org/0000-0003-4198-9911)
Kurt.Hornik@R-project.org

Uwe Ligges
TU Dortmund
Germany
ORCID: [0000-0001-5875-6167](https://orcid.org/0000-0001-5875-6167)
Uwe.Ligges@R-project.org

Achim Zeileis
Universität Innsbruck
Austria
ORCID: [0000-0003-0918-3766](https://orcid.org/0000-0003-0918-3766)
Achim.Zeileis@R-project.org

News from the Forwards Taskforce

by Heather Turner

Abstract [Forwards](#) is an R Foundation taskforce working to widen the participation of under-represented groups in the R project and in related activities, such as the *useR!* conference. This report rounds up activities of the taskforce during the first half of 2026.

1 Accessibility

gwynn gebeyhu is now chair of the American Statistical Association Committee of Statistics and Disability ([ASA CSD](#)). This committee serves the disability community within the ASA, but also has a broader remit to provide guidance to statisticians on best practices: improving accessibility to statistical information, empowering ethical research on disability and supporting accessible statistical education.

2 Community engagement

Ella Kaye continued work under her 2025 Software Sustainability Institute Fellowship, nurturing [rainbowR](#) and making it sustainable for the long term. [rainbowR](#) is a community that supports, promotes and connects LGBTQ+ people who code in R, and spreads awareness of LGBTQ+ issues through data-driven activism. They now have an Asia-Pacific (APAC) region meetup, as well as one that covers the Europe-Middle East-Africa (EMEA) and Americas (AMER) regions. [rainbowR](#) became an ‘unincorporated association’ and formalised its governance. At their first Annual General Meeting (January 2026), [rainbowR](#) established their constitution and elected a leadership committee.

3 R Contribution/on-ramps

Heather Turner led the organisation of three R Dev Days under the umbrella of the [R Contribution Working Group](#). These events, where participants work collaboratively on contributions related to base R, were all run as satellites to R conferences: [Rencontres R 2026](#) (Nantes, France), [Cascadia R Conf 2026](#) (Portland, USA) and [useR! 2026](#) (Warsaw, Poland). All events were run in hybrid format: it proved difficult to engage many new contributors attending remotely, but this format was effective for bringing in additional expertise from the R Core Team and experienced contributors.

Ella Kaye presented [C for R users: Contributing to base R](#) to the Data Science Learning Community Advanced R Book Club and [YOU can contribute to base R!](#) to The Data Science Lab.

4 Conferences

Ella Kaye chaired the organising committee for the first [rainbowR conference](#), which took place online, 25-26 February 2026. The conference comprised 4 workshops, 2 keynotes, 6 regular talks, 7 lightning talks, 2 socials and active Discord server for participant chat. 175 people attended at least one event and there was a surge in sign-ups to join [rainbowR](#). This event was largely possible due to Ella’s 2025 Software Sustainability Institute Fellowship; the community leaders are evaluating the feasibility of running the event again in 2027.

Several members of Forwards were involved in the organization of [useR! 2026](#). Dillon Sparks and Kevin O’Brien were on the Organizing Committee, helping with the [useR! 2026](#) survey and social media respectively. Ella Kaye was on the Diversity, Equity, Inclusion and Accessibility (DEIA) Committee; among other things this committee managed the Diversity Scholarship program, which Jonathan Godfrey was a reviewer for. Ella also chaired the Code of Conduct committee, helping to ensure the conference was a safe space for all.

Dillon Sparks and Lois Adler-Johnson prepared a [useR! 2025 Report](#) summarizing the results of the survey from last year’s [useR!](#) conference, giving insight into participant demographics, engagement and satisfaction. Data from the survey is also used in the [useR! infoboard](#) collating data over multiple years, which Jesica Formoso is helping to update.

Kevin O’Brien has been helping to promote the [GhanaR 2026 conference](#), which took place 9-10 July 2026.

5 Teaching

After a strong uptake for our online R Package Development Workshops in 2025, we decided to run them again in June-July 2026. As before, we organised two cohorts to cater for people in different parts of the world, with the first cohort led by Ella Kaye and Pao Corrales, and the second led by Joyce Robbins, Heather Turner, and Jesica Formoso. Jesi joined Forwards after the 2025 workshops and we were happy to include her on the teaching team this year.

We extended the five sessions from 1.5 to 2 hours, so we were less rushed, which seemed to work better. Numbers were a little lower than the previous year, most likely due to the dates only be advertised 1-2 weeks before the first session, however more than 30 learners attended the sessions.

Video recordings were only shared with registered participants, but the (lightly updated) slides are available at <https://forwards.github.io/package-dev/> under the CC-BY-NC-SA 4.0 license.

6 Membership changes

Six members stepped down from the taskforce at the start of the year: Mine Cetinkaya-Rundel, Michael Chirico, Daniela Cialfi, Zane Dax, Saranjeet Kaur Bhogal, and Maria Prokofieva. We thank them for their contributions to the work of Forwards.

Heather Turner

University of Birmingham

Birmingham, United Kingdom

<https://www.heatherturner.net/>

ORCID: 0000-0002-1256-3375

heather.turner@r-project.org

R Foundation News

by *Torsten Hothorn*

1 Donations and members

Membership fees and donations received between 2026-04-11 and 2026-07-17.

2 Donations

Spyridon Fortis (United States); Kristhian Leandro Peña Cerón (Germany)

3 Supporting institutions

Alfred Mueller Analytic Services, München (Germany); NIFU Nordic Institute for Studies in Innovation, Research and Education, Oslo (Norway); The University of Auckland, Statistics Department, Auckland (New Zealand); University of Iowa, Iowa City (United States)

4 Supporting members

Vedo Alagic (Austria); Kristoffer Winther Balling (Denmark); Joaquín Baquer-Miravete (Spain); Ashanka Beligaswatte (Australia); Daniel Bermúdez (Spain); Andreas Büttner (Germany); Robert Carnell (United States); Giuseppe Corbelli (Italy); Rafael Costa (Brazil); Alistair Cullum (United States); Ajit de Silva (United States); Elliott Deal (United States); Dubravko Dolic (Germany); Serban Dragne (United Kingdom); Philip Eigen (United States); Guenter Faes (Germany); Emily Galvin (United States); Jörg große Schlarmann (Germany); Susan Gruber (United States); Chris Hanretty (United Kingdom); James Harris (United States); Stefan Hartmann (Germany); Gerold Hepp (Austria); Knut Helge Jensen (Norway); Christian Kampichler (Netherlands); Katharina Keszy (Germany); Sebastian Krantz (Germany); Teemu Daniel Laajala (Finland); Thierry Lecerf (Switzerland); Ivan Marino (Italy); Oliver Marugg (Switzerland); harvey minnigh (Puerto Rico); Yaseer Suhaimi Mohd (Malaysia); Jens Oehlschlägel (Germany); Jaesung James Park (South Korea); Victor PERIER (France); Bill Pikounis (United States); Charles Raffaele (United States); Romit Samanta (United Kingdom); Dominic Schuhmacher (Germany); Robert Skeřil (Czechia); Murray Sondergard (Canada); Berthold Stegemann (Germany); Tim Taylor (United Kingdom); Fred Viole (United States)

Torsten Hothorn

Universität Zürich

Switzerland

ORCID: [0000-0001-8301-0471](https://orcid.org/0000-0001-8301-0471)

Torsten.Hothorn@R-project.org