

The Journal

Volume 18/1, March 2026

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

Contents

Editorial	4

Contributed Research Articles

An Introduction to Generative Adversarial Nets in R: The RGAN Package	5
gasmodel: An R Package for Generalized Autoregressive Score Models	23
QTLEMM: An R Package for QTL Mapping and Hotspot Detection	39
hdbayes: An R Package for Bayesian Analysis of Generalized Linear Models Using Historical Data.	69
mpitbR: An R Package for Calculating Multidimensional Poverty Indices	94
BayesPPDSurv: An R Package for Bayesian Sample Size Determination and Analysis Using the Power and Normalized Power Prior for Time-To-Event Data	110
Hyper-Poisson Regression Modeling via GAMLSS	127
Estimation of the Log Gaussian Cox Process Model with Point and Aggregated Case Data: The rts2 Package for R.	144
Asymptotic Benchmarking Using the atime package	169
StepReg: A Comprehensive and Intuitive R Package for Stepwise Regression Analysis	188
onewaytests 3: One-Way Tests Independent Groups Designs - R Package and Its Web-Based Tool	206
ggvfields: Vector Field Visualization in R	227
neuralGAM: An R Package for Fitting Generalized Additive Neural Networks	253
generalRSS: Sampling and Inference for Balanced and Unbalanced Ranked Set Sampling	277
GeoThinner: An R Package for Efficient Spatial Thinning of Species Occurrences and Point Data.	299
Kernopt: A Package for Estimating Count Data Distributions with a Discrete Symmet- ric Optimal Kernel.	315
Clustering Binary Data Optimizing the Exclusiveness of Features.	329
Priority Elastic Net: A Hierarchical Framework for Heterogeneous Data Integration	344
AccSamplingDesign: An R Package for Optimizing Acceptance Sampling Plans . . .	368
SurvDNN: Survival Deep Learning Models for Tabular Data.	382

News and Notes

R Foundation News	398

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

As part of our rotating editorial system, I am privileged to serve as Editor-in-Chief of the R Journal for 2026. We thank Rob Hyndman for his service as Editor-in-Chief in 2025 and for his leadership in guiding the journal. We are also pleased to welcome Vincent Arel-Bundock as our new Executive Editor. In addition, we welcome Karissa Whiting and H. Sherry Zhang as new Associate Editors. We extend our gratitude to the Associate Editors whose terms concluded in 2025: Adam Loy, Isabella Gollini, Lucy D'Agostino McGowan, Rasmus Bååth, Susan VanderPlas, and Yanfei Kang.

I am pleased to report that the R Journal continues to grow, both in the number of submissions received and the number of articles published. We are grateful to all authors who have entrusted us with their work, and to the reviewers whose time, expertise, and dedication help maintain the high standards of the journal.

We also thank Mitchell O'Hara-Wild, who served as the inaugural Technical Editor of the journal in 2025. Mitchell has provided invaluable technical support to the R Journal for several years, and we are grateful for his many contributions. We welcome Abhishek Ulayil as Technical Editor for 2026.

In this issue

On behalf of the editorial board, I am pleased to present Volume 18 Issue 1 of the R Journal. This issue features twenty research articles, plus news from the R Foundation.

The twenty articles reflect the steady expansion of statistical methods in R, with a clear push toward making modern approaches more usable in practice. Several of the contributions focus on deep learning and neural network models, bringing tools like GANs and survival deep learning into the R language. Another cluster of work develops Bayesian methods and there is also a strong set of papers on regression and flexible modelling techniques, including variable selection, distributional regression, kernel-based approaches, point processes and time series applications. Finally, a number of packages address applied problems in sampling, survey methods and specialised statistical workflows across fields such as ecology, health, and quantitative genetics.

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

Emi Tanaka

Australian National University

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

An Introduction to Generative Adversarial Nets in R: The RGAN Package

by Marcel Neunhoffer

Abstract This article introduces Generative Adversarial Nets (GAN), a powerful neural net architecture, and presents the RGAN package. RGAN makes it easy to implement GANs in R: It facilitates experimentation with different design choices for GANs, such as value functions, other network architectures, the choice of different noise distributions, the choice of optimization algorithms and update schemes, dropout during data generation and the post-processing of generated GAN samples. Furthermore, advanced users can provide customized functions for each design choice. RGAN is a lightweight package and runs on top of the torch library in R.

1 Introduction

On July 28, 2016, Yann LeCun, Deep Learning pioneer and Chief Artificial Intelligence Scientist at Facebook, was asked “What are some recent and potentially upcoming breakthroughs in deep learning?” in a Public Quora Q&A session.¹ He answered: “The most important one, in my opinion, is adversarial training (also called GAN for Generative Adversarial Networks). [...]”

So what is a Generative Adversarial Network (GAN)? As indicated in the quote above, GANs were first introduced by Goodfellow et al. (2014). They illustrate the idea with the following example: “The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles” (Goodfellow et al., 2014, p. 1).

In short, a GAN is a model that tries to learn an arbitrary joint distribution by comparison. It samples data from a proposal distribution and compares them to samples from the true distribution. Comparing the two samples (samples from the proposal distribution and samples from the true distribution) improves the model for the proposal distribution such that it generates more and more realistic samples. The basic idea of a GAN is surprisingly intuitive. At its core, a GAN is a minimax game with two competing actors: a generator that produces realistic synthetic samples from random noise and a discriminator (or critic)² trying to tell real from synthetic samples.

In GANs, the team of counterfeiters, the generator, is a neural network trained to produce realistic synthetic data examples from random noise. And the police, the discriminator, is a neural network to classify fake and real data. The generator network is trained to fool the discriminator network and uses the feedback of the discriminator to generate increasingly realistic fake data that should eventually be indistinguishable from the real data. At the same time, the discriminator is constantly adapting to the improving generator. Thus, the threshold where the discriminator is fooled increases along with the faking capabilities of the generator. This goes on until (in theory) an equilibrium is reached. At the equilibrium, the discriminator can no longer distinguish between real and fake samples. The generator can achieve this goal by sampling from the underlying distribution of the real data.³

I bring these neural networks to R with the **RGAN** package. The goal of the package is to facilitate experimentation with GANs for an audience primarily working with R. This way, GANs can be included in existing workflows, for example, for data synthesizing.

RGAN is a lightweight package that only relies on **torch** (Falbel and Luraschi, 2022). The **torch** package provides R users with native access to libtorch, the C++ backend that powers PyTorch, enabling tensor computation and deep learning capabilities directly in R without requiring Python or the **reticulate** (Ushey et al., 2022) package. This design choice makes **RGAN** particularly efficient, as **torch** offers automatic differentiation, GPU acceleration when available, and a comprehensive set of neural network modules, all with an R-native interface. By building on **torch**’s foundation, **RGAN** users benefit from fast tensor operations and seamless GPU support while maintaining a familiar R workflow, avoiding the overhead and complexity of cross-language dependencies. For applications, the focus of **RGAN** is on tabular data, yet it is easy to implement GANs for image data. Both use cases are demonstrated in section 4.

¹ As of March 3, 2023, you can find the session here: <https://www.quora.com/q/quorasessionwithyannlecun>.

² More on the difference between a discriminator and a critic can be found in section 3.

³ A GAN is, therefore, a dynamic system where the optimization process is seeking not a minimum (or maximum) but an equilibrium.

In python, software libraries like CTGAN (Xu et al., 2019), TF-GAN, which is part of tensorflow (Abadi et al., 2015), torchgan (Pal and Das, 2021), or mimicry (Lee and Town, 2020) can serve a similar purpose as **RGAN**. Most of these libraries focus on image data.

In R, **ganGenerativeData** (Müller, 2022) makes it possible to synthesize tabular data with a GAN through **reticulate** and **tensorflow**. The focus of **ganGenerativeData** is on producing synthetic data with one specific, pre-configured GAN architecture optimized for tabular data generation. This contrasts with **RGAN**, which emphasizes flexibility and experimentation with GAN architectures.

Specifically, **RGAN** extends the functionality available in R in several key ways:

- **Native R implementation:** **RGAN** runs directly in R via **torch**, eliminating the Python dependency and **reticulate** overhead required by **ganGenerativeData**
- **Architectural flexibility:** While **ganGenerativeData** provides a fixed architecture, **RGAN** allows users to customize network architectures, create custom neural networks, and switch between architectures (e.g., fully connected for tabular data, DCGAN for images)
- **Multiple value functions:** **RGAN** implements multiple GAN variants (original, WGAN, f-WGAN) and supports custom value functions, whereas **ganGenerativeData** uses only the original GAN value function
- **Broader data support:** **RGAN** handles both tabular and image data, while **ganGenerativeData** focuses exclusively on tabular data
- **Research-oriented features:** **RGAN** provides extensive hyperparameter control, post-processing methods (DRS, PGB), and experimental features like dropout during generation

Among the Python libraries mentioned, CTGAN is most similar to **ganGenerativeData** in focusing on tabular data synthesis with specific architectural choices. Libraries like TF-GAN, torchgan, and mimicry offer similar flexibility to **RGAN** in terms of customizable architectures and multiple GAN variants, but are Python-exclusive. **RGAN** essentially brings this Python-level flexibility to R users in a native R environment.

The rest of the paper is organized as follows: In the following section **A Brief Introduction to GANs**, I give a more detailed introduction to GANs. In the section **Designing a GAN**, I introduce the different design decisions a researcher faces when developing a GAN and show how **RGAN** facilitates experimentation with these design choices. The section **Illustrations** offers two working examples with code: The example in **Synthesizing tabular data with default settings** shows how to get started with the synthesization of tabular data quickly. The example in **Synthesizing images with a DCGAN architecture** generates fake images with a customized neural network architecture in **RGAN**. The **Summary and Outlook** section provides an outlook on potential extensions of **RGAN**.

2 A brief introduction to GANs

A GAN model tries to learn an arbitrary joint distribution by comparison. Specifically, the GAN learns the joint distribution of all variables in the training data. For a dataset with variables $(x_1, x_2, \dots, x_p)$, the GAN approximates $p(x_1, x_2, \dots, x_p)$. For conditional GANs, this extends to learning the joint distribution of target variables given predictor variables. The GAN samples data from a proposal distribution and compares them to samples from the true distribution. Comparing the difference between the two samples improves the proposal distribution model, generating more realistic samples.

In GANs, the proposal distribution (fake data) is typically generated by a deep neural network, the generator (G). The comparison of fake and real data is made by a second deep neural network, the discriminator (D). The generator is trained to produce realistic synthetic data examples from random noise. At the same time, the discriminator has the goal of correctly distinguishing fake from real data.⁴

The generator is trained to fool the discriminator network and uses the feedback of the discriminator to generate increasingly realistic fake data that should eventually be indistinguishable from the real data. At the same time, the discriminator is constantly adapting to the improving generating abilities of the generator. Thus, the threshold where the discriminator is fooled increases along with the faking capabilities of the generator. This goes on until (in theory) an equilibrium is reached. In the classical GAN, described by the value function V in equation 1, an optimal discriminator at the equilibrium would be assigning 0.5 probability to both real and fake samples. This is where the discriminator can no longer distinguish between real and fake samples.

GANs turn a typical unsupervised learning task (learning a joint density) into a supervised learning problem (learning to distinguish between fake and real data), using the observation that it is easier to sample from p than to learn the distribution explicitly.

⁴It wants to discriminate between fake and real data, hence the name. However, novel architectures don't necessarily use classifiers to compare fake and real data. In these GANs, the second network is usually named a critic network.

Formally, this two-player minimax game can be written as:

$$\min_G \max_D V(D, G) = \min_G \max_D \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))] \quad (1)$$

where $p_{data}(x)$ is the joint distribution of the real data, x is a sample from $p_{data}(x)$. The generator network $G(z)$ takes as an input z from $p_z(z)$, where z is a random sample from a probability distribution $p_z(z)$. This sample is also called a noise sample. Usually, GANs are set up to either sample z from uniform or Gaussian distributions.⁵ Passing the noise sample z through G generates a sample of synthetic data, which is then fed into the discriminator D . The discriminator takes as input a set of labeled data, either real examples from $p_{data}(x)$ or generated examples from $G(z)$, and is trained to distinguish between real data and fake data. In the original GAN setup (Goodfellow et al., 2014), this is a standard binary classification problem.⁶

D is trained to maximize the probability of assigning the correct label to training examples and samples from $G(z)$. G is trained to minimize $\log(1 - D(G(z)))$. Thus, the goal of the discriminator is to maximize function V , whereas the generator's goal is to minimize it. In practice, this is achieved by iteratively updating the two networks, holding the weights of the other network constant.

The equilibrium point for a GAN is achieved when G produces samples that come from the true underlying data distribution $p_{data}(x)$ and D is uncertain about the origin of the samples.

3 Designing a GAN

So far, many different GAN architectures have been proposed. Many of these GAN models are named, e.g., WGAN (Arjovsky et al., 2017), WGAN-GP (Gulrajani et al., 2017), KL-WGAN (Song and Ermon, 2020), DCGAN (Radford et al., 2016), CGAN (Mirza and Osindero, 2014), CTGAN (Xu et al., 2019), StyleGAN (Karras et al., 2019), CycleGAN (Zhu et al., 2017) or DiscoGAN (Kim et al., 2017) to name a few. Typically, these new GAN models address some areas for improvement in prior models, where most of the development focuses on generating more and more realistic images.

In this section, I first introduce the basic functionality of the RGAN package. Then, I briefly overview some of these design decisions researchers face when developing a GAN for their application. Some of these design decisions are similar to those when designing neural network models (or other machine learning models).⁷

Other decisions are unique to the setup of GAN models.⁸ Generally, the interaction of all these design choices still needs to be better understood and is an active research area. Table 1 gives an overview of the most important hyperparameters and design choices supported by RGAN.

⁵Research suggests that other prior distributions might be preferable depending on the application. Huster et al. (2021) show that a Pareto distribution performs better than a normal distribution for skewed and heavy-tailed real data.

⁶Thus, the output layer of the discriminator uses a sigmoid function as the activation function and the standard binary cross-entropy loss can be used.

⁷For example, data pre-processing, the choice of the network architecture(s), or the choice of an optimizer with its related hyperparameters.

⁸For example, the choice of the noise distribution $p(z)$ or the GAN value function.

Table 1: Overview of Hyperparameters and Design choices and their default values in **RGAN**.

Hyperparameter	Default in RGAN	Available Options	Use Cases & Advantages	Practical Tuning Implications
Value Function	"original"	• "original": Binary cross-entropy • "wasserstein": WGAN • "f-wgan": KL-WGAN • Custom functions	• Original: Simple classification, intuitive • Wasserstein: Addresses vanishing gradients • F-WGAN: Better empirical results • Custom: Research experimentation	Affects gradient stability; Wasserstein often more stable but requires weight clipping; original may suffer from mode collapse
Network Architecture	Fully connected (2 layers, 128 neurons each)	• Fully connected (customizable) • DCGAN (for images) • Custom torch::nn_module objects	• FC: Tabular data, quick prototyping • DCGAN: Image synthesis • Custom: Domain-specific requirements	Deeper networks = more capacity but slower training; width affects representation power; architecture should match data type
Dimension of Noise	2	Any positive integer	• Small: fewer latent dimensions • Large: more latent dimensions	Affects generator's ability to learn target distribution; should match intrinsic dimension of data
Noise Distribution	"normal"	• "normal": Standard Gaussian • "uniform": [-1, 1] • Custom functions (e.g., Pareto)	• Normal: General purpose • Uniform: DCGAN standard • Pareto: Heavy-tailed real data	Affects generator's ability to match target distribution; Pareto better for skewed data
Optimizer	Adam (base_lr=0.0001) ttur_factor = 4	• Any torch optimizer • Separate G/D optimizers • TTUR (different learning rates for G and D)	• Adam: Good default, adaptive • SGD: More control, simpler • TTUR: Stabilizes training	Higher lr = faster but unstable; TTUR factor >1 slows discriminator, preventing overfitting
Dropout	0.5 (training) FALSE (generation)	• Rate: 0.0-1.0 • eval_dropout: TRUE/FALSE	• Training: Regularization • Generation with dropout increases diversity • No dropout: More consistent outputs	Higher dropout = more regularization but may underfit; eval dropout trades consistency for diversity
Post-processing	None	• DRS (Discriminator Rejection) • PGB (Post-GAN Boosting) • None	• DRS: Quality filtering • PGB: Ensemble improvement • None: Fastest, direct output	DRS reduces sample size but improves quality; PGB requires multiple generators

Hyperparameter	Default in RGAN	Available Options	Use Cases & Advantages	Practical Tuning Implications
Batch Size	50	Any positive integer	• Small (10-50): More updates, noisy gradients • Large (128+): Stable, fewer updates	Larger = more stable gradients but slower convergence; memory constraints on GPU
Epochs	150	Any positive integer	• Few (10-50): Quick experiments • Many (150+): Better convergence	More epochs $\neq$ always better (can overfit discriminator); monitor convergence

In the following, I briefly describe the design choices to be made, how **RGAN** implements default options, and how **RGAN** can be used to customize all these design choices to facilitate experimentation.

3.1 The general RGAN workflow

Before discussing the various design choices and default values available when implementing a GAN and how to implement them, I first introduce the main functions and a typical workflow with the **RGAN** package.

The `gan_trainer()` function The core function in **RGAN** is `gan_trainer()`, which orchestrates the entire GAN training process. This function accepts the following key arguments:

```
gan_trainer(
  data,
  noise_dim = 2,
  noise_distribution = "normal",
  value_function = "original",
  data_type = "tabular",
  generator = NULL,
  generator_optimizer = NULL,
  discriminator = NULL,
  discriminator_optimizer = NULL,
  base_lr = 1e-04,
  ttur_factor = 4,
  weight_clipper = NULL,
  batch_size = 50,
  epochs = 150,
  plot_progress = FALSE,
  plot_interval = "epoch",
  eval_dropout = FALSE,
  synthetic_examples = 500,
  plot_dimensions = c(1, 2),
  device = "cpu"
)
```

A typical **RGAN workflow** Assuming tabular data that is available as a `matrix` or `data.frame` object with the name `data` in the R Session, a standard **RGAN** workflow consists of three main steps:

1. Data Preparation: Transform and standardize (i.e., z-standardize numerical columns and one-hot encode categorical columns) input data. The **RGAN** package provides the `data_transformer` class for easy transformation and back-transformation.

```
transformer <- data_transformer$new()
transformer$fit(data)
transformed_data <- transformer$transform(data)
```

2. GAN Training: Train the GAN with desired hyperparameters by calling the `gan_trainer()` function (here all hyperparameters and choices are set to default values).

```
trained_gan <- gan_trainer(transformed_data)
```

3. Synthetic Data Generation: Generate and transform synthetic samples

```
synthetic_data <- sample_synthetic_data(trained_gan,  
                                         transformer)
```

3.2 Choosing a value function

Description of the design choices. An important design decision is the choice of the value function V . In the original GAN by Goodfellow et al. (2014) and described by the value function in equation 1, the value function was motivated by turning the unsupervised learning task into a binary classification problem. After all, the original GAN value function is just binary-cross entropy loss.⁹

While this is an intuitive formulation, if the discriminator is very good at distinguishing between real and fake data, it is tough to learn (i.e., update the network weights) due to vanishing gradients. This means that the gradient values are close to 0 and cannot provide meaningful information to update the weights during backpropagation.

To address this weakness, Arjovsky et al. (2017) proposed an alternative value function based on the Wasserstein distance. They use the observation that the original GAN value function essentially minimizes the Jensen-Shannon divergence between encoded fake and real data and note that using the Wasserstein distance instead has some desirable properties. This leads to the following value function:

$$V(D, G)_{\|D\|_L \leq 1} = \mathbb{E}_{x \sim p_{\text{data}}(x)} [D(x)] + \mathbb{E}_{z \sim p_z(z)} [(1 - D(G(z)))] \quad (2)$$

This is the value function for the so-called Wasserstein GAN or short WGAN as defined in (Arjovsky et al., 2017). Note that for the Wasserstein value function, the discriminator must be continuous with a Lipschitz constant ≤ 1 , indicated by the constraint $\|D\|_L \leq 1$ in equation 2. The Lipschitz constraint means that the discriminator function cannot change too rapidly; formally, there exists a constant $L \leq 1$ such that $|D(x_1) - D(x_2)| \leq L \cdot |x_1 - x_2|$ for all inputs x_1, x_2 . This constraint is typically enforced by clipping the discriminator's weights to a pre-specified range, e.g., $[-0.01, 0.01]$.

Work by Song and Ermon (2020) shows that f-divergences and Wasserstein GANs can be bridged. They propose a novel GAN value function that leads to better empirical results on many tasks.

A more general formulation for the value function is therefore given by

$$V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)} [f_1(D(x))] + \mathbb{E}_{z \sim p_z(z)} [f_2(1 - D(G(z)))] \quad (3)$$

Where in the original GAN $f_1(a) = f_2(a) = \log(a)$, in the Wasserstein GAN $f_1(a) = f_2(a) = a$ and for the GAN value function proposed by Song and Ermon (2020) $f_1(a) = a$ and $f_2(a) = wa$, where w is a weight for the discriminator scores on fake examples, calculated as $w = \exp(a) / \mathbb{E}[\exp(a)]$ (using the exponential function $\exp(\cdot)$).

Implementation of default design choices in RGAN. In RGAN, these three value functions are readily implemented (original, wasserstein and f-wgan) and can be chosen as an option, e.g., `gan_trainer(..., value_function = "wasserstein")`, the default value function is original.

Further customization with RGAN. Users can also provide custom value functions to work with the RGAN `gan_trainer()` function. The only requirements are that the value function takes as inputs the discriminator scores of real and fake data, works on and with `torch` data types, and outputs a named list with the entries `d_loss` and `g_loss`. For example, this is what the original GAN value function looks like in code:

```
GAN_value_fct <- function(real_scores, fake_scores) {  
  # Compute log(1 - fake_scores) once and reuse  
  log_one_minus_fake <- torch::torch_log(1 - fake_scores)
```

⁹Note that this is the same as the negative log-likelihood of the Bernoulli distribution.


```

d_loss <-
  torch::torch_log(real_scores) + log_one_minus_fake

d_loss <- -d_loss$mean()

g_loss <- log_one_minus_fake$mean()

return(list(d_loss = d_loss,
            g_loss = g_loss))
}

```

When a custom value function is passed to `gan_trainer()`, it is called at each training iteration within the training loop. The training loop follows this flow: (1) sample a minibatch of real data; (2) generate fake data by passing noise through the generator; (3) compute discriminator scores for both real and fake data (`real_scores` and `fake_scores`); (4) call the value function with these scores to obtain `d_loss` and `g_loss`; (5) backpropagate `d_loss` to update discriminator weights; (6) backpropagate `g_loss` to update generator weights. This design allows users to experiment with arbitrary training objectives while **RGAN** handles the data loading, network updates, and training orchestration.

3.3 Choosing and customizing network architectures

Description of the design choices. Parts of the network architecture, specifically the discriminator (or critic) network architecture, depend on the chosen value function. For the original GAN value function, the discriminator acts as a binary classifier outputting a probability that a sample is real. Therefore, the output layer must return values between 0 and 1, achieved by using the sigmoid activation function $\sigma(x) = 1/(1 + \exp(-x))$.¹⁰ In contrast, for WGAN the discriminator (called a “critic” in this context) does not output a probability but rather a score measuring how “real” a sample appears. Since this score represents a distance measure rather than a probability, the output does not need to be restricted to $[0, 1]$. Therefore, a linear output layer (no activation function) is used.¹¹

In general, the discriminator must have enough capacity, that is, the network’s ability to represent complex functions, which is determined by factors such as the number of parameters, depth (number of layers), and width (neurons per layer). What constitutes “enough capacity” depends on the application. For image synthesization, convolutional neural network architectures achieve the best results. For time series, recurrent neural network architectures are sensible choices. For tabular data, simpler, fully connected network architectures are usually sufficient.

Implementation of default design choices in **RGAN.** In **RGAN**, some commonly used architectures are already implemented, e.g., DCGAN¹² (Radford et al., 2016) for image data and a GAN with fully connected neural networks for tabular data. If no networks are specified in a call to `gan_trainer` it defaults to fully connected networks for both the generator and discriminator, with two hidden layers, 128 neurons in each layer, and a dropout rate of 0.5.

Further customization with **RGAN.** The included fully connected network architecture can easily be customized. For example, if users want to customize the number of layers and neurons per layer, they can create custom neural networks with a call to

```

generator <- Generator(noise_dim = 2,
                      data_dim = 2,
                      hidden_units = list(256, 128, 64))

and

discriminator <- Discriminator(data_dim = 2,
                              hidden_units = list(256, 128, 64),
                              sigmoid = TRUE)

```

¹⁰This is one of the reasons for the vanishing gradients.

¹¹For WGAN to work, however, restrictions on the discriminator weights must be imposed, either by weight clipping or a gradient penalty (e.g., in the WGAN-GP implementation (Gulrajani et al., 2017)).

¹²Short for Deep Convolutional GAN.

respectively. These networks can then be fed to `gan_trainer(..., generator = generator, discriminator = discriminator)`. Where `data_dim` needs to match the number of columns in the input dataset and `noise_dim` needs to match the dimensions of the noise distribution (more in section 3.3). The list passed to `hidden_units` specifies both the depth of the neural network and the width of each layer. For example, `list(256, 128, 64)` means that the network will have three hidden layers (determined by the length of the list), where the first hidden layer will contain 256 neurons, the second hidden layer 128, and the third hidden layer 64. For the discriminator network, users can also specify whether the output layer should have a sigmoid activation function or be a linear output layer. This choice depends on the value function chosen for a specific GAN (see section 3.1).

If users want to synthesize images and use the DCGAN architecture included in **RGAN**, they can achieve this by setting up the DCGAN networks by calling `generator <- DCGAN_Generator()` and `discriminator <- DCGAN_Discriminator()`.¹³ Again, these networks can be included in the call to `gan_trainer(..., generator = generator, discriminator = discriminator)`. I describe a complete working example with the DCGAN architecture and loading images from a user-specified folder in section 4.

Furthermore, users can provide custom network architectures for the discriminator and generator networks by passing a suitable `torch::nn_module` object to the `gan_trainer` function. For the generator, that means the input to the `torch::nn_module` needs to match the dimensions of the chosen noise distribution, and the output needs to line up with the dimensions of the real data. The input must match the dimensions of the real data for the discriminator, and the output needs to be just a 1-dimensional vector with the discriminator scores.

The number of hidden layers, the depth and width of these layers, and their activation functions are hyperparameters that should be optimized for a particular application. In **RGAN**, it is easy to experiment with these hyperparameters and tailor them for a specific application.

3.4 Choosing the noise distribution

Description of the design choices. In most GAN applications, the noise distribution for the generator $p(z)$ is typically a standard normal distribution or a uniform distribution on the interval $(-1, 1)$.

However, [Huster et al. \(2021\)](#) show that the choice of noise distribution influences the behavior of GAN training. For example, learning skewed distributions¹⁴ with standard noise distributions is hard. Other noise distributions, such as sampling from a Pareto distribution, could have preferable properties for different applications.

Implementation of default design choices in **RGAN.** This is an active area of research, and **RGAN** makes it easy to implement custom prior distributions. Users can easily choose between a normal distribution and a uniform distribution when specifying the options of `gan_trainer`.

Further customization with **RGAN.** On top of these default options, users can also provide their functions. For example, this is how sampling from the normal distribution is implemented:

```
normal_noise <- torch::torch_randn
```

and this function would be used as follows `gan_trainer(..., noise_distribution = normal_noise)` for training a GAN.

3.5 Choosing optimization algorithms

Description of the design choices. The optimization of GANs is notoriously hard. Research on the optimization of GANs shows (see, e.g., [Heusel et al., 2017](#); [Schaefer and Anandkumar, 2019](#)) that gradient ascent descent (i.e., iteratively training the generator and discriminator with the same optimizer) might not be the best option.

In particular, [Heusel et al. \(2017\)](#) have shown that using two different learning rates for the optimization of the discriminator and generator (they call this scheme “two time-scale update rule”) improves GAN training. Besides the learning rate, the concrete choice of an optimizer can also influence GAN training. The batch size, how many examples are sampled from the training data per iteration, and potentially further hyperparameters of standard optimizers (typically momentum hyperparameters) influence how quickly a GAN can learn.

¹³If no options are specified, this will initialize the default DCGAN architecture with a `noise_dim = 100` and a linear output layer for the discriminator.

¹⁴A typical social science example for such skewed distributions could be the distribution of income.

Implementation of default design choices in *RGAN*. In *RGAN*, it is easy to put the optimizers for the generator and discriminator on two separate time scales by setting the `ttur_factor` in `gan_trainer()`. The `ttur_factor` is a multiplier for the provided `base_lr`, so the generator will be updated with the `base_lr` and the discriminator with `ttur_factor*base_lr`. By default, *RGAN* uses the Adam optimizer (Kingma and Ba, 2014) as implemented in `torch::optim_adam`, and the default minibatch size is set to 50 examples per iteration.

Further customization with *RGAN*. *RGAN* facilitates experimentation with different optimizers and hyperparameter settings. Users can directly pass any *torch* optimizer with any self-chosen hyperparameter settings to `gan_trainer`, e.g.,

```
gan_trainer(...,
            generator_optimizer = torch::optim_sgd(g_net$parameters, lr = 0.1))
```

3.6 Choosing dropout during training and data generation

Description of the design choices. In neural network architectures, so-called dropout layers (Srivastava et al., 2014) can be used to regularize network training and prevent overfitting. In each training iteration, a dropout layer randomly sets a pre-specified percentage of all neurons to 0. A helpful side effect of such dropout layers in classical neural networks is that dropout can also be used to estimate the model uncertainty of neural networks (Gal and Ghahramani, 2016).

Anecdotal evidence shows that dropout layers might also help train GANs.¹⁵ Furthermore, Isola et al. (2016) argue that dropout in the generator during data generation provides stochasticity for the synthetic data that promotes the diversity of the generated data.

Yet, a systematic evaluation of the effect of dropout in GANs on the produced synthetic data, especially on the statistical utility of tabular synthetic data, still needs to be included and warrants more research.

Implementation of default design choices in *RGAN*. By making it easy to experiment with dropout layers in discriminator and generator networks, *RGAN* can facilitate such research. When working with the default networks in `gan_trainer`, the dropout rate is set to 0.5 in both the discriminator and generator networks. By default, dropout is disabled during synthetic data generation but can be enabled by setting `gan_trainer(..., eval_dropout = TRUE)`.

Further customization with *RGAN*. By providing custom networks, the dropout rate during training can be adjusted. In section 4 I provide a simple example of an experiment of synthetic data generated with and without dropout during data generation.

3.7 Post-Processing GAN samples

Description of the design choices. Since convergence to equilibrium cannot be guaranteed, the samples produced after the last update need not be the best. However, several methods have been proposed to boost the samples' fidelity by sampling from multiple generators.¹⁶ These post-processing methods leverage two empirical observations. First, the learned discriminator can assess the quality of generated examples (Azadi et al., 2019). And second, while an arbitrary generator (for example, the last generator) can produce an inadequate representation of the underlying data distribution, a mixture distribution of samples from multiple generators can improve the quality of generated examples (see, e.g., Beaulieu-Jones et al., 2019; Neunhoffer et al., 2021). Methods include Discriminator Rejection Sampling (DRS) (Azadi et al., 2019), Metropolis-Hastings GAN (MH) (Turner et al., 2019), or Post-GAN Boosting (PGB) (Neunhoffer et al., 2021).

¹⁵For example, this <https://github.com/soumith/ganhacks> list of tips for GAN training by some of the authors of influential GAN papers (Arjovsky et al., 2017; Radford et al., 2016; Denton et al., 2016; Zhao et al., 2016) mentions dropout in the generator as a means to stabilize GAN training.

¹⁶There are many ways to sample from multiple generators. E.g., each update step during training produces a slightly different generator, so the sequence of generators can be thought of as a distribution of generators. Or when using dropout in the generator during the generation of synthetic examples and repeating this step multiple times, this will also produce a distribution of generators. Furthermore, methods that train multiple generators in parallel also exist (e.g., Arora et al., 2017; Hoang et al., 2018).

Implementation of default design choices in `RGAN`. DRS and PGB are currently implemented as part of `RGAN`.

4 Illustrations

In this section, I provide two illustrations of how `RGAN` can be used. The first example shows how to quickly train a first GAN on tabular data with the default settings of `RGAN`, including a simple experiment of what dropout during synthetic data generation does. The second example shows that `RGAN` can also synthesize image data and serves as a tutorial on providing custom neural network architectures and optimizers to `gan_trainer`.

Note on training times: For the tabular data example with 1,000 samples and default settings (150 epochs, batch size 50), training completes in approximately 1–2 minutes on a modern CPU. The image synthesis example using the CelebA dataset trains for approximately 17 minutes per epoch on a GPU (e.g., GeForce RTX 2070), or approximately 7 hours per epoch on CPU.¹⁷ Training times will vary depending on hardware, dataset size, network architecture, and hyperparameter choices.

4.1 Synthesizing tabular data with default settings

As a simple illustration to get started with `RGAN`, I show how to generate synthetic copies of a simple tabular data set. First, I create some toy data¹⁸ and transform it (for numeric data, this means standardizing the data) to facilitate training.

```
data <- sample_toydata()
transformer <- data_transformer$new()
transformer$fit(data)
transformed_data <- transformer$transform(data)
```

The real data for this example is displayed in panel (A) of figure 1. x on the x-axis is drawn from a standard normal distribution, and y on the y-axis is $x^2 + \mathcal{N}(0, 0.3)$. Panel (B) of figure 1 shows the data after transformation. The `data_transformer` applies z-standardization to numeric columns, subtracting the mean and dividing by the standard deviation. Since x is already standard normal, its scale changes minimally; y undergoes more noticeable rescaling but retains a similar visual range because the original variance was relatively small ($\approx 0.3^2$ around the quadratic mean).

Now the `gan_trainer()` can train a GAN to generate synthetic data from `transformed_data`.¹⁹ By default, `gan_trainer` shows a simple progress bar to monitor training and estimate how long training will take.²⁰

```
trained_gan <- gan_trainer(transformed_data)
```

This will set up the necessary generator, discriminator, and respective optimizers. The default value function is set to the original GAN value function described in equation 1. Then, the `gan_trainer` trains the GAN for 150 epochs using mini-batches of 50 real examples ($m = 50$) at each update step.

With the trained networks (which will be collected as part of the output of class `trained_RGAN` in the object `trained_gan`), it is then easy to sample synthetic data from the last generator using the function `sample_synthetic_data`. Note that to transform the synthetic data back to the original scale of the data, I pass the transformer to the `sample_synthetic_data` function. Both can be passed to the function `GAN_update_plot` to see the data alongside the synthetic data. This produces the plot in panel (A) of figure 2.

```
synthetic_data <- sample_synthetic_data(trained_gan, transformer)
```

```
GAN_update_plot(
```

¹⁷These timings are approximate and intended as rough guidance. Actual performance depends on system configuration.

¹⁸The default in `sample_toydata` is $N = 1000$.

¹⁹Similarly, data could be used directly. A first experiment could be to assess the convergence of the GAN when using the original data instead of the transformed data.

²⁰The `gan_trainer` also provides the option to observe the intermediary output of the generator during training. If users set the option `plot_progress = TRUE`, a plot of real and synthetic data will be produced after each epoch. For cases where an epoch might take too long, or it is sufficient to look at the intermediary output in a longer interval, users can input the number of steps after which a new plot should be produced to the option `plot_interval`.

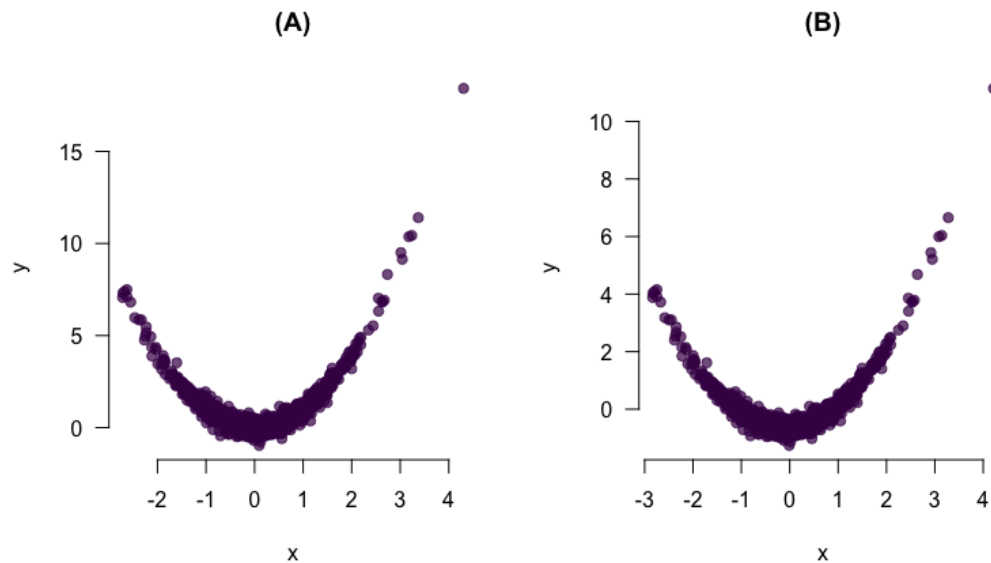


Figure 1: The real data before training the GAN. Panel (A) shows the data on the original scale and panel (B) shows the data after applying the data transformer.

```
data = data,
synth_data = synthetic_data,
main = "(A)"
)
```

Looking at the generated examples in panel (A) of figure 2 shows that while the GAN seems to approximate the functional form of the underlying data well, the produced examples have a smaller variance than the original data.

As a simple experiment, I also produce synthetic data from the same generator but with dropout during data generation. I do this by setting the `eval_dropout` setting of `trained_gan` to `TRUE`. This produces the plot in panel (B) of figure 2.

```
trained_gan$settings$eval_dropout <- TRUE

synthetic_data <- sample_synthetic_data(trained_gan, transformer)

GAN_update_plot(
  data = data,
  synth_data = synthetic_data,
  main = "(B)"
)
```

Now the generated synthetic data in yellow looks more like the underlying real data in purple. How good the data is could be assessed using several utility metrics for synthetic data (see, e.g., [Snoke et al., 2016](#)). Note that these utility metrics are not included in **RGAN** itself. Common approaches include: (1) comparing summary statistics (means, variances, correlations) between real and synthetic data; (2) the propensity mean squared error (pMSE), which measures how well a classifier can distinguish real from synthetic data (lower values indicate better utility); and (3) downstream task performance, where models trained on synthetic data are evaluated on real test data. In R, packages such as **synthpop** ([Nowok et al., 2016](#)) provide functions for computing utility metrics. Each approach has trade-offs: summary statistics are easy to interpret but may miss complex dependencies; pMSE provides a single aggregate measure but requires training additional models; downstream evaluation is task-specific and computationally intensive but most directly measures practical utility.

4.2 The effect of different design choices/hyperparameters on GAN training

Training GANs requires careful tuning of hyperparameters, as the interplay between generator and discriminator optimization is inherently unstable. Table 1 summarizes the key hyperparameters in

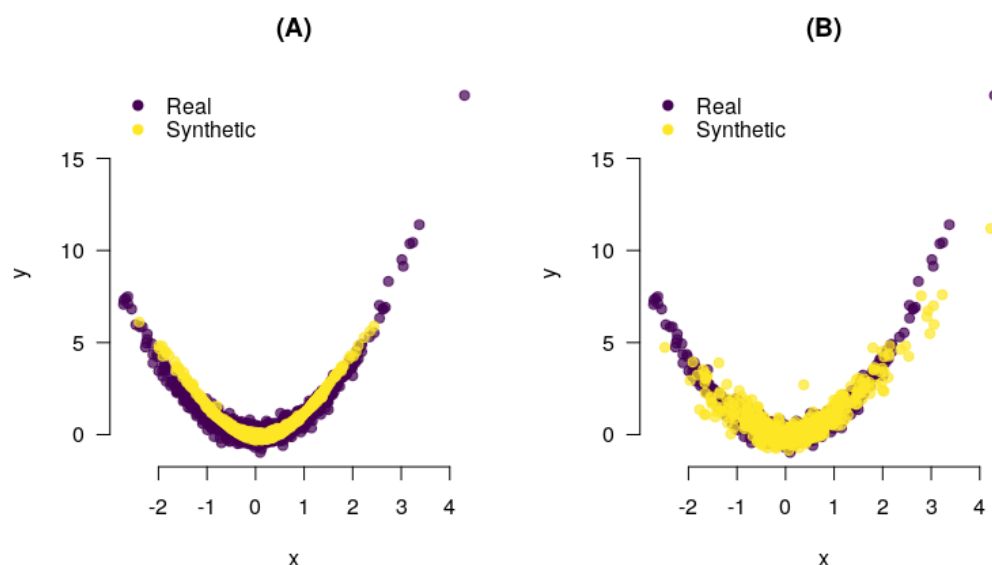


Figure 2: The synthetic data after training the GAN. Panel (A) shows the generated data without dropout during generation on the original scale. Panel (B) shows the generated data with dropout during data generation.

RGAN, but understanding their practical implications helps guide experimentation:

Learning rate: The learning rate controls how much network weights change at each update. A learning rate that is too high can cause training instability or oscillation, while one that is too low leads to slow convergence. GANs often benefit from separate learning rates for the generator and discriminator. In **RGAN**, the `ttur_factor` multiplies the base learning rate for the discriminator, implementing the “two time-scale update rule” (Heusel et al., 2017). A `ttur_factor` > 1 (default is 4) slows down the discriminator relative to the generator, which can prevent the discriminator from becoming too strong too quickly, a common cause of vanishing gradients for the generator.

Batch size: Larger batch sizes provide more stable gradient estimates but fewer weight updates per epoch. Smaller batch sizes (e.g., 10–50) produce noisier gradients that can help escape local minima but may lead to unstable training. For tabular data, the default batch size of 50 works well for most applications. For image data, larger batch sizes (64–128) are common, though memory constraints on GPUs may limit this choice.

Network architecture: Deeper and wider networks have more capacity to represent complex distributions, but also require more data and are prone to overfitting. For tabular data with a modest number of features, two hidden layers with 128 neurons each (the **RGAN** default) is often sufficient. More complex data may benefit from additional layers or wider layers. The `hidden_units` parameter in `Generator()` and `Discriminator()` allows easy experimentation: `list(256, 128, 64)` creates three layers with decreasing width, which can help the network learn hierarchical representations.

Value function: The choice of value function affects gradient dynamics. The original GAN can suffer from vanishing gradients when the discriminator is too strong. The Wasserstein value function provides more informative gradients throughout training but requires weight clipping or gradient penalties. The f-WGAN value function often provides a good balance. Users should experiment with different value functions for their specific application.

4.3 Synthesizing images with a DCGAN architecture

To showcase the **RGAN** customization options, this illustration generates synthetic images using a DCGAN (Radford et al., 2016) architecture. The training data for this example is the CelebA dataset (Liu et al., 2015)²¹ that contains 202,599 images²² of celebrities’ faces.²³

²¹The images can be obtained from the authors’ website: <https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html>. Last accessed: February 03, 2026.

²²With a batch size of 128, as in the DCGAN paper, this means the GAN is updated for 1,582 steps per epoch.

²³Training this model on a machine with a GPU (here a GeForce RTX 2070) takes about 17 minutes per epoch. On a notebook without a GPU, one epoch takes about 7 hours.

First, I use `torchvision` to make images in a folder on my hard drive available for training the networks with `torch`. Furthermore, the DCGAN architecture expects input images to be of size 64×64 pixels. To ensure that the loaded images have the expected format, I apply some standard transformations to the raw images.

```
dataset <- torchvision::image_folder_dataset(root = "path/to/celeba",
  transform = function(x) {
    x = torchvision::transform_to_tensor(x)
    x = torchvision::transform_resize(x, size = c(64, 64))
    x = torchvision::transform_center_crop(x, c(64, 64))
    x = torchvision::transform_normalize(x, c(0.5, 0.5, 0.5), c(0.5, 0.5, 0.5))
    return(x)
  })
```

Next, I load the DCGAN_Generator and DCGAN_Discriminator, and make them available on the GPU by passing them to the "cuda" device.²⁴ The noise_dim (the number of input features for the generator network) is set to 100.

```
device <- "cuda"

g_net <- DCGAN_Generator(dropout_rate = 0, noise_dim = 100)$to(device = device)

d_net <- DCGAN_Discriminator(dropout_rate = 0, sigmoid = FALSE)$to(device = device)
```

Next, the optimizers for both networks can be set up. I use the same optimizers and hyperparameters as [Radford et al. \(2016\)](#) in the original DCGAN paper.

```
g_optim <- torch::optim_adam(g_net$parameters, lr = 0.0002, betas = c(0.5, 0.999))

d_optim <- torch::optim_adam(d_net$parameters, lr = 0.0002, betas = c(0.5, 0.999))
```

Now, these objects and a couple of additional options can be passed to `gan_trainer`. The DCGAN architecture expects the noise samples to be in a three-dimensional tensor (`noise_dim = c(100, 1, 1)`), even though the second and third dimensions are only length 1). Another important option is to set `data_type = "image"` to ensure the `gan_trainer` produces the output in the correct format.

```
trained_gan <- gan_trainer(
  data = dataset,
  noise_dim = c(100, 1, 1),
  noise_distribution = "uniform",
  data_type = "image",
  value_function = "wasserstein",
  generator = g_net,
  generator_optimizer = g_optim,
  discriminator = d_net,
  discriminator_optimizer = d_optim,
  plot_progress = TRUE,
  plot_interval = 10,
  batch_size = 128,
  synthetic_examples = 16,
  device = device,
  eval_dropout = FALSE,
  epochs = 1
)
```

I set `plot_progress = TRUE` and `plot_interval = 10` to observe the training progress. This means 16 images²⁵ will be shown after every tenth update step.

In figure 3, I show 16 examples of generated faces after just one epoch of training the DCGAN with the settings as in the code above. I do not expect to generate state-of-the-art fake images after training

²⁴If no GPU is available, the device should be set to "cpu". If you use Apple Silicon you can also set the device to "mps" which would use the Apple GPU.

²⁵16 is an arbitrary number, but it is easy to show these images in a 4×4 grid.



Figure 3: Generated faces after one epoch of training the DCGAN with the parameters specified in the main text.

just for one epoch with a relatively small and simple DCGAN architecture. However, even after a few update steps with the settings described above, it should be possible to make out the outlines of faces in the generated images.

With **RGAN**, more elaborate network architectures could be provided, and training could be optimized.²⁶

5 Summary and outlook

With **RGAN**, I introduce a package to facilitate training and experimentation with GANs in R. The goal is to open up this powerful neural net framework to users that primarily work in R, especially to facilitate experimentation with synthetic data from GANs from a statistical perspective.

I provide two illustrations of how **RGAN** can be used. The first illustration shows the basic

²⁶Yet, achieving state-of-the-art performance can be prohibitively expensive. For example, the developers of StyleGAN2 (Karras et al., 2020), a GAN that produces high definition real looking images, transparently states the computing resources that went into the project: “The entire project, including all exploration, consumed 132 MWh of electricity, of which 0.68 MWh went into training the final FFHQ model. In total, we used about 51 single-GPU years of computation (Volta class GPU)” (8). A large amount of computing time, especially from the perspective of a statistician or social scientist. At the time of writing, the price for such a GPU, e.g., an Nvidia Tesla V100 GPU, is around \$9000, i.e., to get all the experiments done in one year, you would need at least \$459000 to buy the GPUs. Using cloud computing, e.g., through Amazon Web Services (AWS), you can access a computer with eight such GPUs for \$12.24 per hour. To run all the experiments that (Karras et al., 2020) did, you would need 55845 hours of such a server, totaling $\approx$ \$685000.

functionality of **RGAN** and how to synthesize tabular data quickly. The second illustration shows that training GANs with **RGAN** is easy to customize and synthesize images with a DCGAN architecture.

GANs have broad applications beyond the examples shown here. In statistics and data science, GANs can address class imbalance problems by generating synthetic samples of underrepresented classes, improving classifier performance on imbalanced datasets. In medical and health research, GANs enable sharing of realistic synthetic patient data for collaborative research while preserving patient privacy. This is particularly valuable in cancer research and clinical trials where real data is sensitive and scarce. The post-processing methods in **RGAN** (DRS and PGB) can help ensure that synthetic samples maintain high fidelity to real data characteristics.

Future iterations of **RGAN** will include different network architectures, for example, a CTGAN (Xu et al., 2019) inspired architecture to make the generation of mixed tabular data (i.e., data with real-valued and categorical columns) easier. Recurrent neural network (RNN) architectures could extend **RGAN** to time series data, where GANs can learn temporal dependencies. Conditional GANs (Mirza and Osindero, 2014) would allow generation of data conditioned on specific attributes, enabling targeted data augmentation. Another development area will focus on making the **opacus** (Yousefpour et al., 2021) python library available in **RGAN** to train GANs with formal differential privacy guarantees, providing mathematical bounds on the privacy risk of synthetic data.

The latest stable version of **RGAN** is available from CRAN; development versions can be found on <https://github.com/mneunhoe/RGAN>.²⁷

Computational details

The results in this paper were obtained using R 4.5.2 with the **RGAN** 0.1.1 package, the **torch** 0.16.3 package, and the **torchvision** 0.8.0 (Falbel, 2022) package to load images into **torch**. I used the **viridis** 0.6.5 (Garnier et al., 2021) package for the color palettes in the visualizations. R itself and all packages used are available from the Comprehensive Archive Network (CRAN) at <https://CRAN.R-project.org/>.

Acknowledgments

I am grateful to Thomas Gschwend, Christian Arnold, Pia Kürzdörfer, Guido Ropers, Oliver Rittmann, and Oke Bahnsen for feedback on earlier versions of this manuscript.

References

- M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <https://www.tensorflow.org/>. Software available from tensorflow.org. [p6]
- M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 06–11 Aug 2017. URL <https://proceedings.mlr.press/v70/arjovsky17a.html>. [p7, 10, 13]
- S. Arora, R. Ge, Y. Liang, T. Ma, and Y. Zhang. Generalization and equilibrium in generative adversarial nets (GANs), 2017. URL <https://arxiv.org/abs/1703.00573>. [p13]
- S. Azadi, C. Olsson, T. Darrell, I. Goodfellow, and A. Odena. Discriminator rejection sampling. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=S1GkToR5tm>. [p13]

²⁷The development version (0.2.0) on GitHub includes several additional features: differentially private GAN training via `dp_gan_trainer()` using OpenDP (The OpenDP Project, 2025) for formal privacy guarantees; CTGAN-style mode-specific normalization in the `data_transformer` for handling multi-modal distributions; enhanced post-GAN boosting with model checkpointing; and built-in evaluation metrics for assessing synthetic data quality including statistical divergence measures and privacy risk scores.

- B. K. Beaulieu-Jones, Z. S. Wu, C. Williams, R. Lee, S. P. Bhavnani, J. B. Byrd, and C. S. Greene. Privacy-preserving generative deep neural networks support clinical data sharing. *Circulation: Cardiovascular Quality and Outcomes*, 12(7):e005122, 2019. doi: 10.1161/CIRCOUTCOMES.118.005122. [p13]
- E. Denton, S. Gross, and R. Fergus. Semi-supervised learning with context-conditional generative adversarial networks, 2016. URL <https://arxiv.org/abs/1611.06430>. [p13]
- D. Falbel. *torchvision: Models, Datasets and Transformations for Images*, 2022. URL <https://CRAN.R-project.org/package=torchvision>. R package version 0.4.1. [p19]
- D. Falbel and J. Luraschi. *torch: Tensors and Neural Networks with 'GPU' Acceleration*, 2022. URL <https://CRAN.R-project.org/package=torch>. R package version 0.7.2. [p5]
- Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR. URL <https://proceedings.mlr.press/v48/gal16.html>. [p13]
- S. Garnier, N. Ross, R. Rudis, A. onio Pedro Camargo, A. onio Pedro, M. Sciaini, and C. Scherer. *viridis - Colorblind-Friendly Color Maps for R*, 2021. URL <https://sjmgarnier.github.io/viridis/>. R package version 0.6.2. [p19]
- I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. *Advances in Neural Information Processing Systems* 27, pages 2672–2680, 2014. ISSN 10495258. doi: 10.1017/CBO9781139058452. URL <http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>. [p5, 7, 10]
- I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of Wasserstein GANs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL <https://proceedings.neurips.cc/paper/2017/file/892c3b1c6dccc52936e27cbd0ff683d6-Paper.pdf>. [p7, 11]
- M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL <https://proceedings.neurips.cc/paper/2017/file/8a1d694707eb0fef65871369074926d-Paper.pdf>. [p12, 16]
- Q. Hoang, T. D. Nguyen, T. Le, and D. Phung. MGAN: Training generative adversarial nets with multiple generators. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=rkmu5b0a->. [p13]
- T. Huster, J. Cohen, Z. Lin, K. Chan, C. Kamhoua, N. O. Leslie, C.-Y. J. Chiang, and V. Sekar. Pareto GAN: Extending the representational power of GANs to heavy-tailed distributions. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4523–4532. PMLR, 18–24 Jul 2021. URL <https://proceedings.mlr.press/v139/huster21a.html>. [p7, 12]
- P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks, 2016. URL <https://arxiv.org/abs/1611.07004>. [p13]
- T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. URL https://openaccess.thecvf.com/content_CVPR_2019/papers/Karras_A_Style-Based_Generator_Architecture_for_Generative_Adversarial_Networks_CVPR_2019_paper.pdf. [p7]
- T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. Analyzing and improving the image quality of StyleGAN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. URL https://openaccess.thecvf.com/content_CVPR_2020/papers/Karras_Analyzing_and_Improving_the_Image_Quality_of_StyleGAN_CVPR_2020_paper.pdf. [p18]
- T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim. Learning to discover cross-domain relations with generative adversarial networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1857–1865. PMLR, 06–11 Aug 2017. URL <https://proceedings.mlr.press/v70/kim17a.html>. [p7]

- D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014. URL <https://arxiv.org/abs/1412.6980>. [p13]
- K. S. Lee and C. Town. Mimicry: Towards the reproducibility of GAN research, 2020. URL <https://arxiv.org/abs/2005.02494>. [p6]
- Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), ICCV '15*, pages 3730–3738, USA, 2015. IEEE Computer Society. ISBN 9781467383912. doi: 10.1109/ICCV.2015.425. URL <https://doi.org/10.1109/ICCV.2015.425>. [p16]
- M. Mirza and S. Osindero. Conditional generative adversarial nets, 2014. URL <https://arxiv.org/abs/1411.1784>. [p7, 19]
- W. Müller. *ganGenerativeData*, 2022. URL <https://cran.r-project.org/package=ganGenerativeData>. R package version 1.3.3. [p6]
- M. Neunhoffer, S. Wu, and C. Dwork. Private post-{gan} boosting. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=6isfR3JCbi>. [p13]
- B. Nowok, G. M. Raab, and C. Dibben. synthpop: Bespoke creation of synthetic data in R. *Journal of Statistical Software*, 74(11):1–29, 2016. doi: 10.18637/jss.v074.i11. URL <https://www.jstatsoft.org/article/view/v074i11>. [p15]
- A. Pal and A. Das. TorchGAN: A flexible framework for GAN training and evaluation. *Journal of Open Source Software*, 6(66):2606, 2021. doi: 10.21105/joss.02606. URL <https://doi.org/10.21105/joss.02606>. [p6]
- A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL <http://arxiv.org/abs/1511.06434>. [p7, 11, 13, 16, 17]
- F. Schaefer and A. Anandkumar. Competitive gradient descent. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL <https://proceedings.neurips.cc/paper/2019/file/56c51a39a7c77d8084838cc920585bd0-Paper.pdf>. [p12]
- J. Snok, G. Raab, B. Nowok, C. Dibben, and A. Slavkovic. General and specific utility measures for synthetic data, 2016. URL <https://arxiv.org/abs/1604.06651>. [p15]
- J. Song and S. Ermon. Bridging the gap between f-GANs and Wasserstein GANs. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9078–9087. PMLR, 13–18 Jul 2020. URL <https://proceedings.mlr.press/v119/song20a.html>. [p7, 10]
- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL <http://jmlr.org/papers/v15/srivastava14a.html>. [p13]
- The OpenDP Project. *opendp: R Bindings for the OpenDP Library*, 2025. URL <https://opendp.org/>. R package version 0.14.1. [p19]
- R. Turner, J. Hung, E. Frank, Y. Saatchi, and J. Yosinski. Metropolis-Hastings generative adversarial networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6345–6353. PMLR, 09–15 Jun 2019. URL <https://proceedings.mlr.press/v97/turner19a.html>. [p13]
- K. Ushey, J. Allaire, and Y. Tang. *reticulate: Interface to 'Python'*, 2022. URL <https://CRAN.R-project.org/package=reticulate>. R package version 1.24. [p5]
- L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni. Modeling tabular data using conditional GAN. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL <https://proceedings.neurips.cc/paper/2019/file/254ed7d2de3b23ab10936522dd547b78-Paper.pdf>. [p6, 7, 19]

- A. Yousefpour, I. Shilov, A. Sablayrolles, D. Testuggine, K. Prasad, M. Malek, J. Nguyen, S. Ghosh, A. Bharadwaj, J. Zhao, G. Cormode, and I. Mironov. Opacus: User-friendly differential privacy library in PyTorch. *arXiv preprint arXiv:2109.12298*, 2021. URL <https://arxiv.org/abs/2109.12298>. [p19]
- J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network, 2016. URL <https://arxiv.org/abs/1609.03126>. [p13]
- J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. URL https://openaccess.thecvf.com/content_ICCV_2017/papers/Zhu_Unpaired_Image-To-Image_Translation_ICCV_2017_paper.pdf. [p7]

Marcel Neunhoffer
Ludwig-Maximilians-Universität München
Social Data Science and AI Lab
Department of Statistics
Munich, Germany

Institute for Employment Research
Statistical Methods Research Department
Nuremberg, Germany
<https://www.marcel-neunhoffer.com>
ORCID: 0000-0002-9137-5785
marcel@marcel-neunhoffer.com

gasmodel: An R Package for Generalized Autoregressive Score Models

by Vladimír Holý

Abstract Generalized autoregressive score (GAS) models are a class of observation-driven time series models that employ the score to dynamically update time-varying parameters of the underlying probability distribution. GAS models have been extensively studied and numerous variants have been proposed in the literature to accommodate diverse data types and probability distributions. This paper introduces the `gasmodel` package, which has been designed to facilitate the estimation, forecasting, and simulation of a wide range of GAS models. The package provides a rich selection of distributions, offers flexible options for specifying dynamics, and allows users to incorporate exogenous variables. Model estimation utilizes the maximum likelihood method.

1 Introduction

The generalized autoregressive score (GAS) models, introduced by [Creal et al. \(2013\)](#) and [Harvey \(2013\)](#), have emerged as a valuable and contemporary framework for time series modeling. These models, also referred to as dynamic conditional score (DCS) models or score-driven models, offer flexibility by accommodating various underlying probability distributions and time-varying parameters. GAS models are observation-driven, effectively capturing the dynamic behavior of time-varying parameters through the autoregressive term and the score, i.e., the gradient of the log-likelihood function. The GAS framework enables the formulation of a wide range of dynamic models suitable for many commonly used univariate and multivariate data types.

There are several packages and code available in R that handle GAS models. One notable package is `GAS` developed by [Ardia et al. \(2018\)](#) and [Ardia et al. \(2019\)](#), which provides functionality for both univariate and multivariate GAS models. The current version of the `GAS` package, 0.3.4, supports 16 distributions. However, the model specification in the `GAS` package is somewhat limited, only allowing for basic dynamics without the inclusion of exogenous variables. Additionally, this package lacks distributions for certain more specialized data types, such as circular, compositional, and ranking data. The package thus supports only a limited selection of GAS models found in the literature. For a more detailed comparison of the `gasmodel` and `GAS` packages, see the `comparison_gas` vignette from the `gasmodel` package. Another relevant R package is `betategarch` by [Sucarrat \(2013\)](#), which deals specifically with the Beta-Skew-t-EGARCH model, a GAS model for time-varying volatility based on the Student's t-distribution. In Python, the `PyFlux` library by [Taylor \(2018\)](#) deals with time series analysis and features various GAS models including the Beta-Skew-t-EGARCH model, standard GAS models, GAS random walk models, GAS pairwise comparison models, and GAS regression models. In Julia, the `ScoreDrivenModels.jl` package by [Bodin et al. \(2020\)](#) provides a framework for standard GAS models. The Time Series Lab program by [Lit et al. \(2021\)](#) is a stand-alone GUI application designed to model and forecast time series, including standard GAS models, GAS pairwise comparison models, and GAS regression models. Additional R, Python, MATLAB, and Ox code for some specific GAS models, often associated with individual research papers, can be found on the www.gasmodel.com website.

In this paper, we present the `gasmodel` package, which is designed to provide comprehensive functionality that encompasses a wide range of GAS models documented in the existing literature. It offers versatile model specification and core features available for the entire spectrum of implemented distributions. The current version of the `gasmodel` package, 0.6.2, offers a selection of 36 distributions, catering to various univariate and multivariate data types such as binary, categorical, ranking, count, integer, circular, interval, compositional, duration, and real data types. A comprehensive list of these distributions is provided in Table 1. For details on each distribution, see the `distributions` vignette from the `gasmodel` package. Model specification within the package allows for flexible customization, enabling users to incorporate different parametrizations, exogenous variables, joint and separate modeling of exogenous variables and dynamics, higher score and autoregressive orders, custom and unconditional initial values of time-varying parameters, fixed and bounded values of coefficients, and missing values. Model estimation is performed by the maximum likelihood method. Standard errors of coefficients are estimated using the empirical Hessian matrix. Furthermore, the package offers a range of functionalities including forecasting, simulation, bootstrapping, and assessment of parameter uncertainty. Comprehensive documentation is provided with the package, offering details on each distribution and its corresponding parametrizations.

The `gasmodel` package is accessible on CRAN at cran.r-project.org/package=gasmodel. Addition-

Table 1: List of available distributions and their parametrizations. First parametrization is the default.

Label	Distribution	Dim.	Data Type	Parametrizations
alaplace	Asymmetric Laplace	Uni.	Real	meanscale
bernoulli	Bernoulli	Uni.	Binary	prob
beta	Beta	Uni.	Interval	conc, meansize, meanvar
bisa	Birnbaum-Saunders	Uni.	Duration	scale
burr	Burr	Uni.	Duration	scale
cat	Categorical	Multi.	Categorical	worth
dirichlet	Dirichlet	Multi.	Compositional	conc
dpois	Double Poisson	Uni.	Count	mean
explog	Exponential-Logarithmic	Uni.	Duration	rate
exp	Exponential	Uni.	Duration	scale, rate
fisk	Fisk	Uni.	Duration	scale
gamma	Gamma	Uni.	Duration	scale, rate
ged	Generalized Error	Uni.	Real	meanscale
gengamma	Generalized Gamma	Uni.	Duration	scale, rate
geom	Geometric	Uni.	Count	mean, prob
kuma	Kumaraswamy	Uni.	Interval	conc
laplace	Laplace	Uni.	Real	meanscale
logistic	Logistic	Uni.	Real	meanscale
logitnorm	Logit-Normal	Uni.	Interval	logitmeanvar
lognorm	Log-Normal	Uni.	Duration	logmeanvar
lomax	Lomax	Uni.	Duration	scale
mvnorm	Multivariate Normal	Multi.	Real	meanvar
mvt	Multivariate Student's t	Multi.	Real	meanvar
negbin	Negative Binomial	Uni.	Count	nb2, prob
norm	Normal	Uni.	Real	meanvar
pluce	Plackett-Luce	Multi.	Ranking	worth
pois	Poisson	Uni.	Count	mean
rayleigh	Rayleigh	Uni.	Duration	scale
skellam	Skellam	Uni.	Integer	meanvar, diff, meandisp
t	Student's t	Uni.	Real	meanvar
vonmises	von Mises	Uni.	Circular	meanconc
weibull	Weibull	Uni.	Duration	scale, rate
zigeom	Zero-Inflated Geometric	Uni.	Count	mean
zinegbin	Zero-Inflated Negative Binomial	Uni.	Count	nb2
zipois	Zero-Inflated Poisson	Uni.	Count	mean
ziskellam	Zero-Inflated Skellam	Uni.	Integer	meanvar, diff, meandisp

ally, users can find the development version of the package on GitHub at github.com/vladimirholyy/gasmodel, providing them with the opportunity to report any bugs or issues they encounter.

The rest of the paper is as follows. In Section 2, we outline the key characteristics of GAS models. In Section 3, we present an overview of the **gasmodel** package. In Section 4, we present a case study demonstrating the practical application of the package. We conclude the paper in Section 5.

2 Generalized autoregressive score models

2.1 Background

The concept of utilizing the score as a driving mechanism for dynamics in time series was independently developed at both Vrije Universiteit Amsterdam and the University of Cambridge. At Vrije Universiteit Amsterdam, researchers established a comprehensive general methodology that encompasses various models driven by the score, known as the generalized autoregressive score (GAS) models. The initial findings were presented in the working paper [Creal et al. \(2008\)](#), which was subsequently published as [Creal et al. \(2013\)](#). At the University of Cambridge, the initial focus was on a specific model that employed the Student's t-distribution with dynamic volatility, named Beta-t(E)GARCH. This approach was introduced in a working paper [Harvey and Chakravarty \(2008\)](#). The book by [Harvey \(2013\)](#) explores a variety of dynamic location and scale models driven by the score, referring to them as dynamic conditional score (DCS) models. Both [Creal et al. \(2013\)](#) and [Harvey \(2013\)](#) are widely recognized as seminal contributions to the literature on GAS models. More

recently, in order to reconcile different terminologies used in the literature, the term “score-driven models” has also emerged as a synonymous label.

The Scopus database reports 799 articles containing the phrase “generalized autoregressive score” or “dynamic conditional score”, as of August 17, 2025. The website www.gasmodel.com lists 438 articles, working papers, and books on GAS models, as of August 11, 2025.

2.2 Basic notation

The goal is to model time series $y_t, t = 1, \dots, T$, which can be univariate or multivariate, continuous or discrete. Let f_t denote the vector of time-varying parameters and g the vector of static parameters. Let $p(y_t|f_t, g)$ denote the density function in the case of a continuous variable, or the probability mass function in the case of a discrete variable.

Constructing a model involves two main components: selecting an appropriate distribution and specifying the dynamics of its time-varying parameters.

2.3 Score as the key ingredient

In GAS models, the key ingredient driving the dynamics of the parameter vector f_t is the score, i.e., the gradient of the log-likelihood function,

$$\nabla(y_t, f_t) = \frac{\partial \ln p(y_t|f_t, g)}{\partial f_t}. \quad (1)$$

The score has zero expected value and its variance is known as the Fisher information,

$$\mathcal{I}(f_t) = E \left[\left(\frac{\partial \ln p(y_t|f_t, g)}{\partial f_t} \right)^2 \middle| f_t, g \right]. \quad (2)$$

The score quantifies the discrepancy between the fitted distribution, determined by f_t , and a particular observation y_t . As such, it can be employed as a correction term following the realization of an observation. When the score is positive, it suggests that the parameter of interest should be increased to better accommodate the observed data. Conversely, when the score is negative, decreasing the parameter would help in aligning the distribution with the observation. When the score is zero, it indicates that the current parameter value represents the optimal fit for the specific observation at hand.

An advantage of the score is that it takes into account the shape of the distribution. To illustrate this point, [Creal et al. \(2013\)](#) consider two GARCH models: one based on the normal distribution and another based on the Student's t-distribution. Now, imagine an extreme observation occurs. Due to its heavier tails, the Student's t-distribution assigns a higher probability to such extreme observations compared to the normal distribution. Crucially, this distinction is also mirrored in the score. Specifically, when assuming the normal distribution, the score for the extreme observation will have a significantly higher absolute value compared to when assuming the Student's t-distribution. The dynamics can thus reflect the shape of the distribution.

The simple difference between expectation and realization, commonly used as a correction term in various time series models, may not always be effective for distributions with specific support. [Harvey et al. \(2024\)](#) highlight this limitation in the context of circular time series. To illustrate this, let us suppose the expected value of an angle is 0.01 radians, but the actual observation turns out to be 6.27 radians. Although the numerical difference between these values is substantial, their corresponding angles are very similar as 0 radians and 2π radians represent the exact same angle. This discrepancy highlights the inadequacy of using a simple difference metric. On the other hand, the score respects the circular nature of the data. For instance, when working with the von Mises distribution characterized by a time-varying location parameter μ_t and a static concentration parameter ν , the score for μ_t is equal to $\nu \sin(y_t - \mu_t)$. By employing the sine function, the score accounts for the circularity of the data and ensures that the angular differences are appropriately considered during the analysis.

2.4 Dynamics of time varying parameters

In GAS models, time-varying parameters f_t follow the recursion

$$f_t = \omega + \sum_{j=1}^P \alpha_j S(f_{t-j}) \nabla(y_{t-j}, f_{t-j}) + \sum_{k=1}^Q \varphi_k f_{t-k}, \quad (3)$$

where ω is the intercept, α_j are the score parameters, φ_k are the autoregressive parameters, and $S(f_t)$ is a scaling function for the score. In the case of a single time-varying parameter, all these quantities are scalar. In the case of multiple time-varying parameters, ω and $\nabla(y_{t-j}, f_{t-j})$ are vectors, α_j and φ_k are diagonal matrices, and $S(f_t)$ is a square matrix. In the majority of empirical studies, it is common practice to set the score order P and the autoregressive order Q to 1. Furthermore, one of three scaling functions is typically chosen: the unit function, the inverse of the Fisher information, or the square root of the inverse of the Fisher information. When the latter is used, the scaled score has unit variance. However, the choice of the scaling function is not always a straightforward task and is closely tied to the underlying distribution.

The dynamics of the model can be expanded to incorporate exogenous variables as

$$f_t = \omega + \sum_{i=1}^M \beta_i x_{ti} + \sum_{j=1}^P \alpha_j S(f_{t-j}) \nabla(y_{t-j}, f_{t-j}) + \sum_{k=1}^Q \varphi_k f_{t-k}, \quad (4)$$

where β_i are the regression parameters associated with the exogenous variables x_{ti} . Alternatively, a different model can be obtained by defining the recursion in the fashion of regression models with dynamic errors as

$$f_t = \omega + \sum_{i=1}^M \beta_i x_{ti} + e_t, \quad e_t = \sum_{j=1}^P \alpha_j S(f_{t-j}) \nabla(y_{t-j}, f_{t-j}) + \sum_{k=1}^Q \varphi_k e_{t-k}. \quad (5)$$

The key distinction between the two models lies in the impact of exogenous variables on f_t . Specifically, in the former model formulation, exogenous variables influence all future parameters through both the autoregressive term and the score term. In the latter model formulation, they affect future parameters only through the score term. When no exogenous variables are included, the two specifications are equivalent, although with differently parameterized intercepts. When numerically optimizing parameter values, the latter model exhibits faster convergence, thanks to the dissociation of ω from φ_k . In the context of ARIMA models, the differences between the two model specifications are discussed by Hyndman (2010). In the context of a GAS model for ranking data, Holý (2025) advocates the use of a regression model with dynamic errors in an application to ice hockey results.

Other model specifications can be obtained by imposing various restrictions on ω , β_i , α_j , or φ_k . In addition, it is possible to have different orders P and Q for individual parameters when multiple parameters are time-varying. Furthermore, the set of exogenous variables can also vary for different parameters.

The recursive nature of f_t necessitates the initialization of the first few elements $f_1, \dots, f_{\max\{P, Q\}}$. A sensible approach is to set them to the long-term value,

$$\bar{f} = \begin{cases} \left(1 - \sum_{k=1}^Q \varphi_k\right)^{-1} \left(\omega + \sum_{i=1}^M \beta_i \frac{1}{N} \sum_{t=1}^N x_{ti}\right) & \text{in model (4),} \\ \omega + \sum_{i=1}^M \beta_i \frac{1}{N} \sum_{t=1}^N x_{ti} & \text{in model (5).} \end{cases} \quad (6)$$

2.5 Maximum likelihood estimation

GAS models can be straightforwardly estimated by the maximum likelihood method. Let $\theta = (\omega, \beta_1, \dots, \beta_M, \alpha_1, \dots, \alpha_P, \varphi_1, \dots, \varphi_Q, g)'$ denote the vector of all parameters to be estimated. The estimate $\hat{\theta}$ is then obtained by maximizing the full log-likelihood as

$$\hat{\theta} = \arg \max_{\theta} \sum_{t=1}^T \ln p(y_t | f_t, g). \quad (7)$$

Alternatively, the conditional log-likelihood can be maximized, which excludes the initial $\max\{P, Q\}$ terms. The maximization of the log-likelihood function can be accomplished using various general-purpose algorithms designed for solving nonlinear optimization problems.

The standard errors of the estimated parameters can be obtained using the standard maximum likelihood asymptotics. Under appropriate regularity conditions, the maximum likelihood estimator $\hat{\theta}$ is consistent and asymptotically normal. Specifically, it satisfies:

$$\sqrt{T}(\hat{\theta} - \theta_0) \xrightarrow{d} N(0, -E[H(\theta_0)]^{-1}), \quad (8)$$

where θ_0 represents the true parameter values and $H(\theta)$ denotes the Hessian of the log-likelihood, defined as

$$H(\theta) = \frac{\partial^2 \ln p(y_t | f_t, g)}{\partial \theta \partial \theta'}. \quad (9)$$

In finite samples, the expected value of the Hessian can be approximated by the empirical Hessian of the log-likelihood evaluated at the estimated parameter values $\hat{\theta}$. This empirical Hessian provides an estimate of the curvature of the log-likelihood function and serves as a practical substitute for the true expected value of the Hessian when finite-sample inference is required.

The conditions for the consistency and asymptotic normality of the estimator depend on the specific distributional assumptions and dynamics of the model and need to be verified on a case-by-case basis. Each distribution may have its own specific characteristics and requirements for maximum likelihood estimation. For the general asymptotic theory regarding GAS models and maximum likelihood estimation, see Blasques et al. (2014), Blasques et al. (2018), and Blasques et al. (2022).

2.6 Theoretical and empirical properties

The use of the score for updating time-varying parameters is optimal in an information theoretic sense. For an investigation of the optimality properties of GAS models, see Blasques et al. (2015) and Blasques et al. (2021).

Generally, the GAS models perform quite well when compared to alternatives, including parameter-driven models. For a comparison of the GAS models to alternative models, see Koopman et al. (2016) and Blazsek and Licht (2020).

2.7 Notable models

The GAS class includes many well-known econometric models, such as the generalized autoregressive conditional heteroskedasticity (GARCH) model of Bollerslev (1986) based on the normal distribution, the autoregressive conditional duration (ACD) model of Engle and Russell (1998) based on the exponential distribution, and the count model of Davis et al. (2003) based on the Poisson distribution.

More recently, a variety of novel score-driven models has been proposed, such as the Beta-t-(E)GARCH model of Harvey and Chakravarty (2008), a multivariate Student's t volatility model of Creal et al. (2011), a Dirichlet model of Calvori et al. (2013), the GRAS copula model of De Lira Salvatierra and Patton (2015), the realized Wishart-GARCH model of Hansen et al. (2016), a bimodal Birnbaum-Saunders model of Fonseca and Cribari-Neto (2018), a Skellam model of Koopman et al. (2018), a circular model of Harvey et al. (2024), a Bradley-Terry model of Gorgi et al. (2019), a bivariate Poisson model of Koopman and Lit (2019), a censoring model of Harvey and Ito (2020), a double Poisson mixture model of Holý and Tomanová (2022), a ranking model of Holý and Zouhar (2022), a Tobit model of Harvey and Liao (2023), and a zero-inflated negative binomial model of Blasques et al. (2024).

For an overview of various GAS models, see Artemova et al. (2022) and Harvey (2022).

3 Features of the package

3.1 Model specification and estimation

The heart of the `gasmodel` package is the `gas()` function, which serves as a powerful tool for estimating both univariate and multivariate GAS models. This function offers extensive flexibility with its wide range of arguments:

```
gas(y, x = NULL, distr, param = NULL, scaling = "unit", regress = "joint",
    p = 1L, q = 1L, par_static = NULL, par_link = NULL, par_init = NULL,
    lik_skip = 0L, coef_fix_value = NULL, coef_fix_other = NULL,
    coef_fix_special = NULL, coef_bound_lower = NULL, coef_bound_upper = NULL,
    coef_start = NULL, optim_function = wrapper_optim_nloptr, optim_arguments =
    list(opts = list(algorithm = "NLOPT_LN_NELDERMEAD", xtol_rel = 0, maxeval =
    1e+06)), hessian_function = wrapper_hessian_stats, hessian_arguments = list(),
    print_progress = FALSE)
```

However, at its core, it only requires two essential inputs: a time series `y` and a distribution `distr`. All other arguments come with default values, ensuring that the function can be readily used even with minimal specifications.

A time series `y` can be represented as either a vector of length T or a $T \times 1$ matrix in the case of univariate series. In the multivariate case, it should be a $T \times N$ matrix, where N denotes the dimension of the series.

Additionally, there is an option to include exogenous variables x . When incorporating a single variable that is common for all time-varying parameters, a numeric vector of length T can be provided. For multiple variables that are common for all time-varying parameters, a $T \times M$ numeric matrix can be used. In cases where there are individual variables for each time-varying parameter, a list of numeric vectors or matrices following the aforementioned formats can be utilized. To control whether the variables are included in the dynamics equation together, as in (4), the argument `regress` can be set to the value "joint". Alternatively, if separate equations for dynamics and regression are preferred, as in (5), `regress` can be set to the value "sep". The default behavior is `regress = "joint"`, which is the model users might expect when incorporating exogenous variables. However, in general, we recommend using `regress = "sep"` to improve the numerical performance of the optimizer and to provide more straightforward interpretability of the model.

The selection of the distribution in the `gas()` function is determined by the `distr` argument. Some distributions have multiple parametrizations available, which can be specified using the `param` argument. It is important to note that certain parameters may have restrictions imposed on them, and these restrictions should be considered in the model dynamics. However, it may not always be possible to satisfy these restrictions, or it may require additional constraints on the coefficients controlling the dynamics. To handle parameter restrictions, it is generally recommended to use a link function that transforms the parameters into unrestricted real numbers. By default, the logarithmic function is applied to time-varying parameters in the interval $(0, \infty)$, while the logistic function is used for time-varying parameters in the interval $(0, 1)$. The static parameters are unaffected. This behavior can be modified by the `par_link` argument, which takes the form of a logical vector. The TRUE values indicate that the logarithmic/logistic link is applied to the corresponding parameters. The list of available distributions and their parametrizations can be obtained using the `distr()` function. Alternatively, Table 1 provides the relevant information.

The determination of time-varying and static parameters is guided by the `par_static` argument, which takes the form of a logical vector. The TRUE values indicate static parameters. By default, the first parameter of the distribution is considered time-varying, while the remaining parameters are treated as static. The score order P and the autoregressive order Q are selected by the `p` and `q` arguments respectively. These arguments can take either a single non-negative integer or a vector of non-negative integers when different orders are required for different parameters.

The choice of scaling function for the score is determined by the `scaling` argument. The supported scaling options include the unit scaling (`scaling = "unit"`), the scaling based on the inverse of the Fisher information matrix (`scaling = "fisher_inv"`), and the scaling based on the inverse square root of the Fisher information matrix (`scaling = "fisher_inv_sqrt"`). The latter two scalings utilize the Fisher information for the time-varying parameters exclusively. If the preference is to use the full Fisher information matrix, which includes both time-varying and static parameters, the `"full_fisher_inv"` or `"full_fisher_inv_sqrt"` scaling options can be selected. For the individual Fisher information associated with each parameter, the `"diag_fisher_inv"` and `"diag_fisher_inv_sqrt"` scaling options are available. It should be noted that when the parametrization is orthogonal (see `distr()`), there are no differences among these scaling variants.

The first $\max\{P, Q\}$ initial values of the time-varying parameters are by default set to their long-term values (6). It is also possible to assign specific values to the initial parameters using the `par_init` argument. During the maximization of the log-likelihood, the initial values can be included, resulting in the computation of the full likelihood, which is the default option. Alternatively, the initial values can be omitted, leading to the computation of the conditional likelihood by specifying `lik_skip = NULL`. To exclude a specified number of first few values from the likelihood calculation, a non-negative integer can be provided to `lik_skip`.

Restrictions on estimated coefficients can be enforced using several arguments. The `coef_fix_value` argument allows coefficients to be fixed at specific values, using a numeric vector where NA values indicate coefficients that are not fixed. To set coefficients as linear combinations of other coefficients, the `coef_fix_other` argument can be used. It requires a square matrix with multiples of the estimated coefficients, which are added to the fixed coefficients. A coefficient given by row is fixed on a coefficient given by column. By this logic, all rows corresponding to the estimated coefficients should contain only NA values. All columns corresponding to the fixed coefficients should also contain only NA values. For convenience, common coefficient structures can be specified by name using the `coef_fix_special` argument. Examples include `panel_structure`, `zero_sum_intercept`, and `random_walk`. To impose lower and upper bounds on coefficients, the `coef_bound_lower` and `coef_bound_upper` arguments can be utilized, respectively.

The `coef_start` argument allows for the specification of the starting values of coefficients used in the optimization process. If no values are provided, the starting values are automatically selected from a small grid of values. To define the optimization function, the `optim_function` argument is used. The function should be formatted according to the required specifications. Two wrapper functions are avail-

able for convenience: `wrapper_optim_stats()`, which utilizes the `optim()` function from the `stats` package, and `wrapper_optim_nloptr()`, which utilizes the `nloptr()` function from the `nloptr` package (Ypma and Johnson, 2020). Additional arguments can be passed to the optimization function as a list using the `optim_arguments` argument. Similarly, the Hessian matrix can be computed using the function specified in the `hessian_function` argument. Three wrappers are available: `wrapper_hessian_stats` for the `optimHess()` function from the `stats` package, `wrapper_hessian_pracma` for the `hessian()` function from the `pracma` package (Borchers, 2023), and `wrapper_hessian_numderiv` for the `hessian()` function from the `numDeriv` package (Gilbert and Varadhan, 2022). Additional arguments for the Hessian function can be passed as a list using the `hessian_arguments` argument. If desired, a detailed computation report can be continuously printed by setting the `print_progress` argument to `TRUE`.

The function returns a list of S3 class `gas`. This list consists of five components: `data`, `model`, `control`, `solution`, and `fit`, each of which is also a list. The `data` component contains the supplied time series and exogenous variables. The `model` component contains the specification of the model structure and size. The `control` component contains the settings that control the optimization and Hessian computation. The `solution` component contains the raw results of the optimization and Hessian computation. Lastly, and most importantly, the `fit` component contains comprehensive estimation results. When an object of the `gas` class is printed, it provides a concise summary similar to the `summary.lm()` function from the `stats` package. Various generic functions can be applied to `gas` objects, including `summary()`, `plot()`, `coef()`, `vcov()`, `fitted()`, `residuals()`, `logLik()`, `AIC()`, `BIC()`, and `confint()`.

3.2 Forecasting

Forecasting of GAS models is performed using the `gas_forecast()` function. This function offers two forecasting methods. The `mean_path` method filters the time-varying parameters based on zero score and then generates the mean of the time series. The `simulated_paths` method repeatedly simulates time series, simultaneously filters time-varying parameters, and then estimates mean, standard deviation, and quantiles. See Blasques et al. (2016b) for more details on this method.

To use the `gas_forecast()` function, an estimated GAS model is required. Typically, the output of the `gas()` function (a `gas` object) can be supplied via the `gas_object` argument:

```
gas_forecast(gas_object, method = "mean_path", t_ahead = 1L, x_ahead = NULL,
  rep_ahead = 1000L, quant = c(0.025, 0.975))
```

Alternatively, multiple arguments including the data, model specification, and estimated coefficients can be manually specified:

```
gas_forecast(method = "mean_path", t_ahead = 1L, x_ahead = NULL,
  rep_ahead = 1000L, quant = c(0.025, 0.975), y, x = NULL, distr, param = NULL,
  scaling = "unit", regress = "joint", p = 1L, q = 1L, par_static = NULL,
  par_link = NULL, par_init = NULL, coef_est = NULL)
```

The forecasting method is determined by the `method` argument. The number of observations to forecast can be specified using the `t_ahead` argument. If exogenous variables are utilized, their values must be provided for the forecasted period using the `x_ahead` argument. For the `simulated_paths` method, the number of simulations can be controlled using the `rep_ahead` argument, and the desired quantiles can be specified using the `quant` argument.

The function returns a list of S3 class `gas_forecast` with three components: `data`, `model`, `forecast`. The `data` component contains the supplied time series and exogenous variables. The `model` component contains the specification of the model structure and size. The `forecast` component contains the mean of the forecasted observations, along with standard deviations and quantiles if the `simulated_paths` method is used. Available generic functions are `summary()` and `plot()`.

3.3 Simulation

Basic simulation of GAS models is handled by the `gas_simulate()` function.

The `gas_simulate()` function requires supplying the coefficients using the `coef_est` argument and specifying the model using arguments `distr`, `param`, `scaling`, `regress`, `p`, `q`, `par_static`, `par_link`, `par_init`, and, in the case of multivariate models, the dimension `n`:

```
gas_simulate(t_sim = 1L, x_sim = NULL, distr, param = NULL, scaling = "unit",
  regress = "joint", n = NULL, p = 1L, q = 1L, par_static = NULL,
  par_link = NULL, par_init = NULL, coef_est = NULL)
```

Alternatively, only a gas object containing a model estimated by the `gas()` function can be provided using the `gas_object` argument:

```
gas_simulate(gas_object, t_sim = 1L, x_sim = NULL)
```

The number of observations to simulate can be specified using the `t_sim` argument. If exogenous variables are utilized, their values must be provided for the simulation sample using the `x_sim` argument.

The function returns a list of S3 class `gas_simulate` with three components: `data`, `model`, `simulation`. The data component contains the exogenous variables, if supplied. The model component contains the specification of the model structure and size. The simulation component contains the simulated time series, time-varying parameters, and scores. Available generic functions are `summary()` and `plot()`.

3.4 Bootstrapping

To compute standard deviations and confidence intervals of the estimated coefficients in GAS models, the package provides the `gas_bootstrap()` function. This function employs the bootstrapping technique to estimate the uncertainty associated with the coefficients. The parametric method involves repeatedly simulating time series using the parametric model and re-estimating the coefficients based on the simulated data. The `simple_block`, `moving_block`, and `stationary_block` methods execute the circular block bootstrap with fixed non-overlapping blocks, fixed overlapping blocks, and randomly sized overlapping blocks, respectively.

The `gas_bootstrap()` function requires an estimated GAS model as input. The best way is to simply supply a gas object to the `gas_object` argument:

```
gas_bootstrap(gas_object, method = "parametric", rep_boot = 1000L,
  block_length = NULL, quant = c(0.025, 0.975), optim_function =
  wrapper_optim_nloptr, optim_arguments = list(opts = list(algorithm =
  "NLOPT_LN_NELDERMEAD", xtol_rel = 0, maxeval = 1e+04)), parallel_function =
  NULL, parallel_arguments = list())
```

Alternatively, the individual arguments including the data, model specification, and estimated coefficients can be provided:

```
gas_bootstrap(method = "parametric", rep_boot = 1000L, block_length = NULL,
  quant = c(0.025, 0.975), y, x = NULL, distr, param = NULL, scaling = "unit",
  regress = "joint", p = 1L, q = 1L, par_static = NULL, par_link = NULL,
  par_init = NULL, lik_skip = 0L, coef_fix_value = NULL, coef_fix_other = NULL,
  coef_fix_special = NULL, coef_bound_lower = NULL, coef_bound_upper = NULL,
  coef_est = NULL, optim_function = wrapper_optim_nloptr, optim_arguments =
  list(opts = list(algorithm = "NLOPT_LN_NELDERMEAD", xtol_rel = 0, maxeval =
  1e+04)), parallel_function = NULL, parallel_arguments = list())
```

The bootstrapping method is determined by the `method` argument. The number of bootstrap samples is specified by the `rep_boot` argument. For the `simple_block` and `moving_block` methods, the fixed size of blocks must be specified by the `block_length` argument. For the `stationary_block` method, the mean size of blocks must be specified by the `block_length` argument. The desired quantiles can be specified using the `quant` argument. As bootstrapping can be computationally very demanding, parallelization is achievable by employing the `parallel_function` argument, which expects a function similar to `lapply()`, allowing the application of a function over a list. Two wrapper functions are available for convenience: `wrapper_parallel_multicore()`, which utilizes the multi-core parallelization functionality from the `parallel` package, and `wrapper_parallel_snow()`, which utilizes the snow parallelization functionality from the `parallel` package. Additional arguments can be passed to the parallelization function as a list using the `parallel_arguments` argument. If `parallel_function` is set to `NULL`, no parallelization is employed and `lapply()` is used.

The function returns a list of S3 class `gas_bootstrap` with three components: `data`, `model`, `bootstrap`. The data component contains the supplied time series and exogenous variables. The model component contains the specification of the model structure and size. The bootstrap component contains the full set of bootstrapped coefficients as well as the basic statistics derived from them. Available generic functions are `summary()`, `plot()`, `coef()`, and `vcov()`.

3.5 Filtered parameters

The filtered time-varying parameters of an estimated model can be directly obtained from the output of the `gas()` function. However, to investigate the uncertainty associated with these parameters, the `gas_filter()` function can be used. This function also supports forecasting and provides two methods. The `simulated_coefs` method calculates a path of time-varying parameters for each simulated coefficient set, assuming asymptotic normality with a given variance-covariance matrix. See [Blasques et al. \(2016b\)](#) for more details on this method. The `given_coefs` method computes a path of time-varying parameters for each supplied coefficient set. Suitable sets of coefficients can be obtained, for example, through the use of the `gas_bootstrap()` function.

An estimated GAS model can be supplied as a `gas` object to the `gas_object` argument:

```
gas_filter(gas_object, method = "simulated_coefs", coef_set = NULL,
  rep_gen = 1000L, t_ahead = 0L, x_ahead = NULL, rep_ahead = 1000L,
  quant = c(0.025, 0.975))
```

Alternatively, the individual arguments including the data, model specification, and estimated coefficients with variance-covariance matrix can be provided:

```
gas_filter(method = "simulated_coefs", coef_set = NULL, rep_gen = 1000L,
  t_ahead = 0L, x_ahead = NULL, rep_ahead = 1000L, quant = c(0.025, 0.975), y,
  x = NULL, distr, param = NULL, scaling = "unit", regress = "joint", p = 1L,
  q = 1L, par_static = NULL, par_link = NULL, par_init = NULL,
  coef_fix_value = NULL, coef_fix_other = NULL, coef_fix_special = NULL,
  coef_bound_lower = NULL, coef_bound_upper = NULL, coef_est = NULL,
  coef_vcov = NULL)
```

The method argument determines the approach for capturing uncertainty. For the `given_coefs` method, the `coef_set` argument in the form of a numeric matrix of coefficient sets in rows must be provided. For the `simulated_coefs` method, the `rep_gen` argument representing the number of generated coefficient sets must be provided. If forecasting is desired, the number of observations to forecast can be specified using the `t_ahead` argument, values of exogenous variables for the forecasted period can be provided using the `x_ahead` argument, and the number of simulation repetitions in the forecasted sample can be controlled using the `rep_ahead` argument. The desired quantiles can be specified using the `quant` argument.

The function returns a list of S3 class `gas_filter` with three components: `data`, `model`, `filter`. The `data` component contains the supplied time series and exogenous variables. The `model` component contains the specification of the model structure and size. The `filter` component contains in-sample and possibly out-of-sample means, standard deviations, and quantiles of the time-varying parameters and scores. Available generic functions are `summary()` and `plot()`.

3.6 Supplementary functions for distributions

The `distr()` function can be utilized to retrieve a list of distributions and their parametrizations supported by the `gas()` function. To narrow down the output and focus on specific distributions, arguments such as `filter_type` or `filter_dim`, among others, can be specified. The output is in the form of a `data.frame` with columns providing information on the distributions such as the data type, dimension, orthogonality, and default parameterization.

To work with individual distributions, the [gasmodel](#) package offers several functions. The `distr_density()` function computes the density of a given distribution, the `distr_mean()` function computes the mean of a given distribution, the `distr_var()` function computes the variance of a given distribution, the `distr_score()` function computes the score of a given distribution, the `distr_fisher()` function computes the Fisher information of a given distribution, and the `distr_random()` function generates random observations from a given distribution. Each of these functions can be supplied with arguments specifying the distribution and the parametrization, namely `distr`, `param`, `par_link`. It is important to note that while the `gas()` function may automatically set the logarithmic/logistic link for time-varying parameters, it must be set manually for the distribution functions. Additionally, a vector of parameter values must be provided to the `f` argument. Some functions may also require an observation to be provided to the `y` argument. For detailed usage instructions, please refer to the documentation for each individual function.

4 Example usage

4.1 Analysis of toilet paper sales

To demonstrate the practical application of the package, we conduct an analysis on the dynamics of toilet paper sales. The dataset `toilet_paper_sales` includes the daily number of toilet paper packs sold in a European store during the years 2001 and 2002, along with a promo variable indicating whether the product was featured in a campaign. In addition to the promo dummy variable, we utilize dummy variables for each day of the week, which are already supplied in the dataset.

There are some missing values, corresponding to the days when the store was closed. It is not necessary to remove these values as the `gas()` function is capable of handling them. When missing observations occur, the time-varying parameters are reinitialized—parameter values are set to their initial value (either the unconditional value or the value supplied by the `par_init` argument), and the score is set to zero. Naturally, missing observations are excluded from the evaluation of the likelihood function.

```
data("toilet_paper_sales")
y <- toilet_paper_sales$quantity
x <- as.matrix(toilet_paper_sales[3:9])
```

Given that our primary variable of interest is in the form of counts, we employ a count distribution. The `distr()` function can be used to obtain a list of appropriate distributions.

```
distr(filter_type = "count", filter_dim = "uni", filter_default = TRUE)$distr

#> [1] "dpois"      "geom"       "negbin"     "pois"       "zigeom"     "zinegbin"   "zipois"
```

We start our analysis by utilizing the Poisson distribution, which is the customary initial choice in count data analysis. The promo and day of the week dummy variables are included as exogenous variables. The dynamics are specified as a regression model with dynamic errors. We estimate the model by the `gas()` function.

```
est_pois <- gas(y = y, x = x, distr = "pois", regress = "sep")
est_pois

#> GAS Model: Poisson Distribution / Mean Parametrization / Unit Scaling
#>
#> Coefficients:
#>
#>      Estimate Std. Error  Z-Test Pr(>|Z|)
#> log(mean)_omega  2.6517736  0.0436953  60.6878 < 2e-16 ***
#> log(mean)_beta1 -0.0015755  0.0264302  -0.0596  0.95247
#> log(mean)_beta2 -0.0792304  0.0271167  -2.9218  0.00348 **
#> log(mean)_beta3 -0.0108654  0.0265633  -0.4090  0.68251
#> log(mean)_beta4  0.0554222  0.0257702   2.1506  0.03151 *
#> log(mean)_beta5 -0.8179358  0.0352252 -23.2202 < 2e-16 ***
#> log(mean)_beta6 -1.7635113  0.0538292 -32.7612 < 2e-16 ***
#> log(mean)_beta7  0.7063463  0.0361533  19.5375 < 2e-16 ***
#> log(mean)_alpha1  0.0142384  0.0012708  11.2042 < 2e-16 ***
#> log(mean)_phi1   0.9818186  0.0121071  81.0946 < 2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Log-Likelihood: -2111.266, AIC: 4242.532, BIC: 4288.128
```

The output of the `gas()` function follows the following convention for naming coefficients. The first part of a coefficient name (before the underscore) refers to the parameter of the distribution and its transformation. The names of the individual distribution parameters are listed in the distributions vignette. For nonnegative parameters, the logarithmic transformation, `log()`, can be applied, while for nonnegative parameters less than 1, the logit transformation, `logit()`, can be used. The second part of the coefficient name (after the underscore) refers to either the constant (`omega`), the regression coefficient (`beta`), the score coefficient (`alpha`), or the autoregressive coefficient (`phi`), as described in (3), (4), and (5). For `beta`, the number denotes the corresponding exogenous variable. For `alpha` and `phi`, the number denotes the lag.

The Poisson distribution can be quite restrictive in practice as it necessitates equidispersion, meaning that the mean of the variable is equal to the variance. Subsequently, we opt for a more versatile distribution—the negative binomial distribution, which accommodates overdispersion, allowing the variance to be greater than the mean.

```
est_negbin <- gas(y = y, x = x, distr = "negbin", regress = "sep")
est_negbin

#> GAS Model: Negative Binomial Distribution / NB2 Parametrization / Unit Scaling
#>
#> Coefficients:
#>
#>      Estimate Std. Error  Z-Test  Pr(>|Z|)
#> log(mean)_omega  2.6384526  0.0562032  46.9449 < 2.2e-16 ***
#> log(mean)_beta1 -0.0100824  0.0357634  -0.2819  0.77800
#> log(mean)_beta2 -0.0772088  0.0366240  -2.1081  0.03502 *
#> log(mean)_beta3 -0.0155342  0.0361813  -0.4293  0.66767
#> log(mean)_beta4  0.0452483  0.0357004   1.2674  0.20500
#> log(mean)_beta5 -0.8240699  0.0430441 -19.1448 < 2.2e-16 ***
#> log(mean)_beta6 -1.7736613  0.0589493 -30.0879 < 2.2e-16 ***
#> log(mean)_beta7  0.7037864  0.0481655  14.6118 < 2.2e-16 ***
#> log(mean)_alpha1  0.0256734  0.0035329   7.2670 3.676e-13 ***
#> log(mean)_phi1   0.9769718  0.0175857  55.5549 < 2.2e-16 ***
#> dispersion      0.0349699  0.0051488   6.7918 1.107e-11 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Log-Likelihood: -2059.834, AIC: 4141.668, BIC: 4191.824
```

We compare the models based on the Poisson and negative binomial distributions using the Akaike information criterion (AIC).

```
AIC(est_pois, est_negbin)
```

```
#>      df      AIC
#> est_pois  10 4242.532
#> est_negbin 11 4141.668
```

Based on the AIC, the negative binomial model is preferred, as the addition of the overdispersion parameter distinctly improves the fit. Nevertheless, the coefficient estimates, standard errors, and p-values are quite similar in both models. The coefficients for working days are close to zero. Note that Monday is treated as the baseline. In contrast, Saturdays and, even more so, Sundays exhibit a significant drop in sales. Promoting the product in a campaign significantly increases sales. The coefficient estimates themselves need to be interpreted with respect to the utilized logarithmic link function. The score and autoregressive coefficients have the expected positive signs. Furthermore, the autoregressive coefficient is close to one, suggesting high persistence. In Figure 1, we visualize the time-varying mean with the logarithmic transformation using the `plot()` generic function.

```
plot(est_negbin)
```

Next, we project sales for the following year utilizing the `forecast()` function with the default `mean_path` method. This method computes time-varying parameters assuming zero score and subsequently generates the mean of the time series. By manipulating the `promo` dummy variable, we can perform a what-if analysis. In Figure 2, we illustrate the forecasted sales, considering the promotion of the product throughout the entire year.

```
x_ahead <- cbind(kronecker(matrix(1, 53, 1), diag(7)), 1)[3:367, -1]
fcst_negbin <- gas_forecast(est_negbin, t_ahead = 365, x_ahead = x_ahead)
plot(fcst_negbin)
```

The estimated coefficients are subject to uncertainty. The `gas()` function reports standard errors and p-values based on the empirical Hessian justified by the asymptotic theory. However, if the sample size is small, it may be appropriate to derive the standard errors and p-values using the bootstrap method. The `gas_bootstrap()` function offers both parametric and block bootstrap options. This

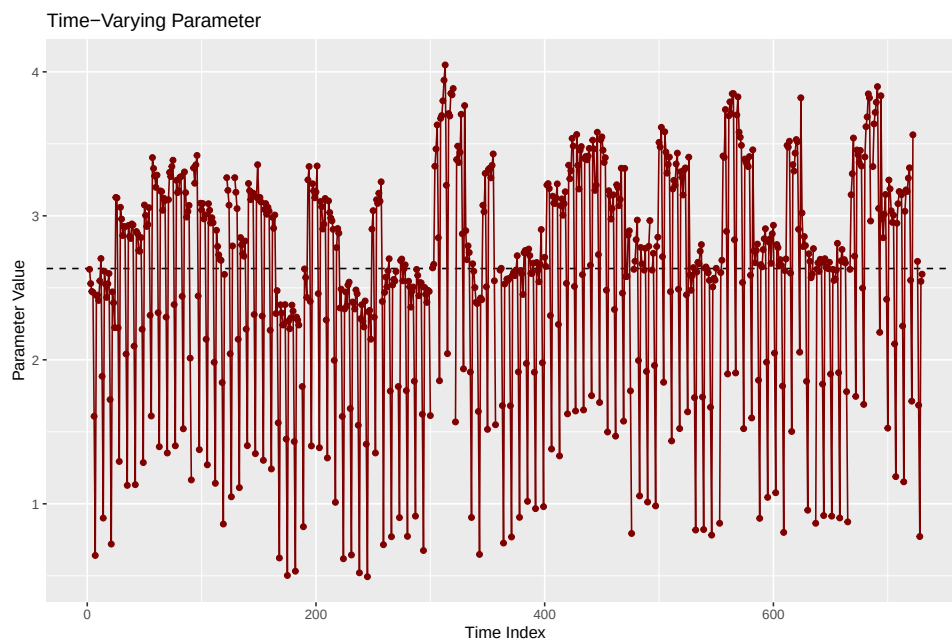


Figure 1: The fitted logarithm of the mean. The horizontal dashed line represents the unconditional value of the logarithm of the mean.

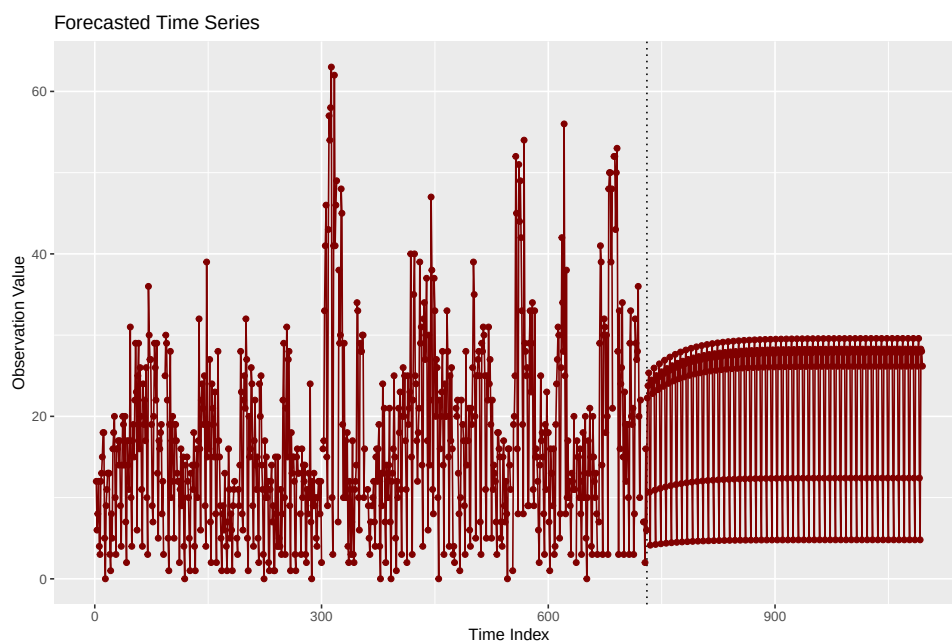


Figure 2: The forecasted toilet paper sales for the next year. The product is assumed to be always promoted. The vertical dotted line divides the in-sample and out-of-sample periods.

process could be computationally intensive, depending on factors such as the number of repetitions, the quantity of observations, the complexity of the model structure, and the optimizer used. To expedite the computation, the function supports parallelization through the arguments `parallel_function` and `parallel_arguments`. As a simple illustration, we demonstrate the parametric bootstrap with only 10 repetitions. To obtain more accurate estimates of the tails and confidence intervals, it is recommended to use a few thousand repetitions.

```
set.seed(42)
boot_negbin <- gas_bootstrap(est_negbin, rep_boot = 10)
```

We have a relatively large sample, and the bootstrapped standard errors closely align with those obtained from the empirical Hessian. Note that standard deviations can also be obtained using the `vcov()` generic function for both `est_negbin` and `boot_negbin`.

```
est_negbin$fit$coef_sd - boot_negbin$bootstrap$coef_sd

#> log(mean)_omega log(mean)_beta1 log(mean)_beta2 log(mean)_beta3
#> -0.0126389898 0.0062104317 -0.0020216686 -0.0040419594
#> log(mean)_beta4 log(mean)_beta5 log(mean)_beta6 log(mean)_beta7
#> 0.0040558932 0.0186693180 -0.0017401598 0.0033140955
#> log(mean)_alpha1 log(mean)_phi1 dispersion
#> -0.0009915868 0.0058394839 0.0008697134
```

As the estimated coefficients are subject to uncertainty, the fitted values of time-varying parameters are also uncertain. To acquire the standard errors and confidence bands of the logarithm of the mean, we employ the `gas_filter()` function with the default `simulated_coefs` method. This method calculates a path of time-varying parameters for each simulated coefficient set under the assumption of asymptotic normality with a given variance-covariance matrix.

```
set.seed(42)
flt_negbin <- gas_filter(est_negbin, rep_gen = 100)
```

In our case, the confidence bands are relatively narrow, averaging 0.16 in width.

```
mean(diff(t(flt_negbin$filter$par_tv_quant[, 1, ])))

#> [1] 0.1560328
```

4.2 Other case studies

For another example of using the package, we refer to the `case_durations` vignette, which loosely follows the paper by [Tomanová and Holý \(2021\)](#). This case study examines the timing of bookshop orders in the fashion of autoregressive conditional duration (ACD) models. It employs the generalized gamma distribution with a dynamic scale parameter.

The `case_rankings` vignette replicates the case study by [Holý and Zouhar \(2022\)](#). It analyzes the annual results of the Ice Hockey World Championships, which take the form of time-varying rankings. The analysis utilizes both stationary and random walk specifications, along with the Plackett–Luce distribution incorporating dynamic worth parameters.

4.3 Limitations and customization

There are many GAS models with distributions or structures that are not supported by the `gasmodel` package. These include copula models ([De Lira Salvatierra and Patton, 2015](#)), matrix models ([Opschoor et al., 2018](#)), censoring models ([Harvey and Liao, 2023](#)), models with parameter interactions ([Holý and Tomanová, 2022](#)), sports rating models ([Gorgi et al., 2019](#)), semiparametric models ([Blasques et al., 2016a](#)), Markov regime-switching models ([Bazzi et al., 2017](#)), and spatio-temporal models ([Catania and Billé, 2017](#)). The vignette customization further discusses these limitations and demonstrates how the package can be customized to accommodate some of these models.

5 Conclusion

The purpose of the **gasmodel** package is to provide researchers, analysts, and data scientists with a versatile toolkit for a broad spectrum of GAS models in R. While it is important to note that not all GAS models found in the literature are supported by the package due to their diverse nature, the package still provides a solid foundation. For some specific GAS models, modifications of the package may be required, or an alternative specialized package/code may prove to be a better option. Nevertheless, the **gasmodel** package offers considerable flexibility for specifying dynamics, and it boasts an extensive array of probability distribution options.

6 Acknowledgments

The work on this paper was supported by the Czech Science Foundation under project 23-06139S and the personal and professional development support program of the Faculty of Informatics and Statistics, Prague University of Economics and Business.

References

- D. Ardia, K. Boudt, and L. Catania. Downside risk evaluation with the R package GAS. *The R Journal*, 10(2):410–421, 2018. doi: 10.32614/rj-2018-064. [p23]
- D. Ardia, K. Boudt, and L. Catania. Generalized autoregressive score models in R: The GAS package. *Journal of Statistical Software*, 88(6):1–28, 2019. doi: 10.18637/jss.v088.i06. [p23]
- M. Artemova, F. Blasques, J. van Brummelen, and S. J. Koopman. Score-driven models: Methods and applications. *Oxford Research Encyclopedia of Economics and Finance*, 2022. doi: 10.1093/acrefore/9780190625979.013.671. [p27]
- M. Bazzi, F. Blasques, S. J. Koopman, and A. Lucas. Time-varying transition probabilities for Markov regime switching models. *Journal of Time Series Analysis*, 38(3):458–478, 2017. doi: 10.1111/jtsa.12211. [p35]
- F. Blasques, S. J. Koopman, and A. Lucas. Stationarity and ergodicity of univariate generalized autoregressive score processes. *Electronic Journal of Statistics*, 8(1):1088–1112, 2014. doi: 10.1214/14-ejs924. [p27]
- F. Blasques, S. J. Koopman, and A. Lucas. Information-theoretic optimality of observation-driven time series models for continuous responses. *Biometrika*, 102(2):325–343, 2015. doi: 10.1093/biomet/asu076. [p27]
- F. Blasques, J. Ji, and A. Lucas. Semiparametric score driven volatility models. *Computational Statistics & Data Analysis*, 100(2013):58–69, 2016a. doi: 10.1016/j.csda.2015.04.003. [p35]
- F. Blasques, S. J. Koopman, K. Łasak, and A. Lucas. In-sample confidence bands and out-of-sample forecast bands for time-varying parameters in observation-driven models. *International Journal of Forecasting*, 32(3):875–887, 2016b. doi: 10.1016/j.ijforecast.2015.11.018. [p29, 31]
- F. Blasques, P. Gorgi, S. J. Koopman, and O. Wintenberger. Feasible invertibility conditions and maximum likelihood estimation for observation-driven models. *Electronic Journal of Statistics*, 12(1): 1019–1052, 2018. doi: 10.1214/18-ejs1416. [p27]
- F. Blasques, A. Lucas, and A. C. van Vlodrop. Finite sample optimality of score-driven volatility models: Some Monte Carlo evidence. *Econometrics and Statistics*, 19:47–57, 2021. doi: 10.1016/j.ecosta.2020.03.010. [p27]
- F. Blasques, J. van Brummelen, S. J. Koopman, and A. Lucas. Maximum likelihood estimation for score-driven models. *Journal of Econometrics*, 227(2):325–346, 2022. doi: 10.1016/j.jeconom.2021.06.003. [p27]
- F. Blasques, V. Holý, and P. Tomanová. Zero-inflated autoregressive conditional duration model for discrete trade durations with excessive zeros. *Studies in Nonlinear Dynamics and Econometrics*, 28(5): 673–702, 2024. doi: 10.1515/snde-2022-0008. [p27]
- S. Blazsek and A. Licht. Dynamic conditional score models: A review of their applications. *Applied Economics*, 52(11):1181–1199, 2020. doi: 10.1080/00036846.2019.1659498. [p27]

- G. Bodin, R. Saavedra, C. Fernandes, and A. Street. ScoreDrivenModels.jl: A Julia package for generalized autoregressive score models. 2020. [p23]
- T. Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3): 307–327, 1986. doi: 10.1016/0304-4076(86)90063-1. [p27]
- H. W. Borchers. Package ‘pracma’, 2023. URL <https://cran.r-project.org/package=pracma>. [p29]
- F. Calvori, F. Cipollini, and G. M. Gallo. Go with the flow: A GAS model for predicting intra-daily volume shares. 2013. [p27]
- L. Catania and A. G. Billé. Dynamic spatial autoregressive models with autoregressive and heteroskedastic disturbances. *Journal of Applied Econometrics*, 32(6):1178–1196, 2017. doi: 10.1002/jae.2565. [p35]
- D. Creal, S. J. Koopman, and A. Lucas. A general framework for observation driven time-varying parameter models. 2008. URL <https://www.tinbergen.nl/discussion-paper/2649>. [p24]
- D. Creal, S. J. Koopman, and A. Lucas. A dynamic multivariate heavy-tailed model for time-varying volatilities and correlations. *Journal of Business & Economic Statistics*, 29(4):552–563, 2011. doi: 10.1198/jbes.2011.10070. [p27]
- D. Creal, S. J. Koopman, and A. Lucas. Generalized autoregressive score models with applications. *Journal of Applied Econometrics*, 28(5):777–795, 2013. doi: 10.1002/jae.1279. [p23, 24, 25]
- R. A. Davis, W. T. M. Dunsmuir, and S. B. Street. Observation-driven models for Poisson counts. *Biometrika*, 90(4):777–790, 2003. doi: 10.1093/biomet/90.4.777. [p27]
- I. De Lira Salvatierra and A. J. Patton. Dynamic copula models and high frequency data. *Journal of Empirical Finance*, 30:120–135, 2015. doi: 10.1016/j.jempfin.2014.11.008. [p27, 35]
- R. F. Engle and J. R. Russell. Autoregressive conditional duration: A new model for irregularly spaced transaction data. *Econometrica*, 66(5):1127–1162, 1998. doi: 10.2307/2999632. [p27]
- R. V. Fonseca and F. Cribari-Neto. Bimodal Birnbaum-Saunders generalized autoregressive score model. *Journal of Applied Statistics*, 45(14):2585–2606, 2018. doi: 10.1080/02664763.2018.1428734. [p27]
- P. Gilbert and R. Varadhan. Package ‘numDeriv’, 2022. URL <https://cran.r-project.org/package=numDeriv>. [p29]
- P. Gorgi, S. J. Koopman, and R. Lit. The analysis and forecasting of tennis matches by using a high dimensional dynamic model. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 182(4):1393–1409, 2019. doi: 10.1111/rssa.12464. [p27, 35]
- P. R. Hansen, P. Janus, and S. J. Koopman. Realized Wishart-GARCH: A score-driven multi-asset volatility model. 2016. [p27]
- A. Harvey, S. Hurn, D. Palumbo, and S. Thiele. Modelling circular time series. *Journal of Econometrics*, 239(1):105450, 2024. doi: 10.1016/j.jeconom.2023.02.016. [p25, 27]
- A. C. Harvey. *Dynamic Models for Volatility and Heavy Tails: With Applications to Financial and Economic Time Series*. Cambridge University Press, New York, 2013. doi: 10.1017/cbo9781139540933. [p23, 24]
- A. C. Harvey. Score-driven time series models. *Annual Review of Statistics and Its Application*, 9(1): 321–342, 2022. doi: 10.1146/annurev-statistics-040120-021023. [p27]
- A. C. Harvey and T. Chakravarty. Beta-t(E)GARCH. 2008. [p24, 27]
- A. C. Harvey and R. Ito. Modeling time series when some observations are zero. *Journal of Econometrics*, 214(1):33–45, 2020. doi: 10.1016/j.jeconom.2019.05.003. [p27]
- A. C. Harvey and Y. Liao. Dynamic tobit models. *Econometrics and Statistics*, 26:72–83, 2023. doi: 10.1016/j.ecosta.2021.08.012. [p27, 35]
- V. Holý. Analyzing and forecasting success in the Men’s Ice Hockey World (Junior) Championships using a dynamic ranking model. *Journal of Quantitative Analysis in Sports*, 21(3):211–235, 2025. doi: 10.1515/jqas-2024-0137. [p26]
- V. Holý and P. Tomanová. Modeling price clustering in high-frequency prices. *Quantitative Finance*, 22(9):1649–1663, 2022. doi: 10.1080/14697688.2022.2050285. [p27, 35]

- V. Holý and J. Zouhar. Modelling time-varying rankings with autoregressive and score-driven dynamics. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 71(5):1427–1450, 2022. doi: 10.1111/rssc.12584. [p27, 35]
- R. J. Hyndman. The ARIMAX model muddle, 2010. URL <https://robjhyndman.com/hyndsight/arimax/>. [p26]
- S. J. Koopman and R. Lit. Forecasting football match results in national league competitions using score-driven time series models. *International Journal of Forecasting*, 35(2):797–809, 2019. doi: 10.1016/j.ijforecast.2018.10.011. [p27]
- S. J. Koopman, A. Lucas, and M. Scharth. Predicting time-varying parameters with parameter-driven and observation-driven models. *Review of Economics and Statistics*, 98(1):97–110, 2016. doi: 10.1162/rest_a_00533. [p27]
- S. J. Koopman, R. Lit, A. Lucas, and A. Opschoor. Dynamic discrete copula models for high-frequency stock price changes. *Journal of Applied Econometrics*, 33(7):966–985, 2018. doi: 10.1002/jae.2645. [p27]
- R. Lit, S. J. Koopman, and A. C. Harvey. *Time Series Lab - Score Edition*, 2021. URL <https://timeserieslab.com>. [p23]
- A. Opschoor, P. Janus, A. Lucas, and D. van Dijk. New HEAVY models for fat-tailed realized covariances and returns. *Journal of Business & Economic Statistics*, 36(4):643–657, 2018. doi: 10.1080/07350015.2016.1245622. [p35]
- G. Sucarrat. betategarch: Simulation, estimation and forecasting of Beta-Skew-t-EGARCH models. *The R Journal*, 5(2):137–147, 2013. doi: 10.32614/rj-2013-034. [p23]
- R. Taylor. *PyFlux: An Open Source Time Series Library for Python*, 2018. URL <https://github.com/rjt1990/pyflux>. [p23]
- P. Tomanová and V. Holý. Clustering of arrivals in queueing systems: Autoregressive conditional duration approach. *Central European Journal of Operations Research*, 29(3):859–874, 2021. doi: 10.1007/s10100-021-00744-7. [p35]
- J. Ypma and S. G. Johnson. Package ‘nloptr’, 2020. URL <https://cran.r-project.org/package=nloptr>. [p29]

Vladimír Holý
Prague University of Economics and Business
Department of Econometrics
Winston Churchill Square 1938/4, 130 67 Prague 3, Czechia
ORCID: 0000-0003-0416-0434
vladimir.holy@vse.cz

QTLEMM: An R Package for QTL Mapping and Hotspot Detection

by Ping-Yuan Chung, You-Tsz Guo, Hsiang-An Ho, Hsin-I Lee, Po-Ya Wu, Man-Hsia Yang, Miao-Hui Zeng, and Chen-Hung Kao

Abstract This paper introduces an R package QTLEMM that implements commonly used and popular statistical methods for QTL mapping and QTL hotspot detection. For QTL mapping, QTLEMM offers a suite of functions for simulating data, computing significance thresholds, and estimating QTL parameters using single-QTL or multiple-QTL methods in diverse experimental populations. These methods encompass linear regression, permutation tests, Gaussian stochastic process, normal mixture models, and truncated normal mixture models. Moreover, QTLEMM accommodates both complete genotyping and selective genotyping data, and enables the fitting and comparison of different models in the analysis. For QTL hotspot detection, QTLEMM devises the statistical framework that can handle both the data from genetical genomics experiments and summarized data, mitigate the underestimation of hotspot thresholds, and also identify various types of hotspots at a very low computational cost. QTLEMM provides numerical and graphical results, and can facilitate the discovery of more significant results in the analysis of quantitative traits in biological studies.

1 Introduction

Many biologically and economically important traits in organisms are quantitative rather than qualitative. These include traditional traits (such as yields and quality in rice, weight and body fat percentage in animals, and diabetes and hypertension in humans) and molecular traits (such as gene expression and protein levels). Quantitative traits typically exhibit continuous variation in a population, so there is no easy way to categorize them. They are likely to be affected by numerous genes each with modest effects and easily affected by environmental factors (Falconer and Mackay, 1996). Consequently, traditional methods such as the Mendelian segregation ratio analysis, mean and variance analyses, covariance studies, and the examination of familial correlations are very difficult for detecting the individual genes contributing to these traits. The genes responsible for quantitative traits are referred to as quantitative trait loci (QTL). For a long time, researchers have tried to obtain individual QTL information for exploring the genetic mechanisms underlying quantitative traits and further to manipulate them for improving the traits. With the availability of fine-scale genetic marker data along the genomes for various organisms, it has become possible to systematically map for and detect individual QTL (QTL mapping) by using more sophisticated statistical methods. Understanding the genetic mechanisms of quantitative traits using QTL mapping remains a major challenge and considerable issue in broad areas of biological studies (Chen et al., 2021; Kumar et al., 2024; Meng et al., 2024; Mackay and Anholt, 2024).

Statistical methods for QTL mapping have been well established (Lander and Botstein, 1989; Haley and Knott, 1992; Zeng, 1993; Jansen, 1993; Zeng, 1994; Xu and Atchley, 1995; Kao et al., 1999; Kao, 2000; Sen and Churchill, 2001; Broman et al., 2003; Kao and Zeng, 2002; Kao, 2004, 2006; Verbyla et al., 2007; Li et al., 2008; Kao and Zeng, 2009, 2010; Kao and Ho, 2012; Lee et al., 2014; Wang et al., 2016). These methods analyze the marker and trait data from well-designed experimental populations to estimate the QTL parameters, including the numbers, positions, various gene effects (additive, dominance, and interactive), variance components, heritabilities, etc. The experimental populations include the most commonly used populations, such as the backcross and F_2 populations, and other more advanced populations, such as recombinant inbred (RI) populations, advanced intercross (AI) populations, intermated recombinant inbred (IRI) populations, and immortalized F_2 populations (Kao and Zeng, 2009). The statistical methods are applied to analyze the QTL

mapping data and tackle several central issues, including the estimation of QTL parameters, determination of threshold values and selective genotyping, in the QTL mapping studies. These studies have provided important insights into the genetic mechanisms governing quantitative traits in various organisms, such as rice, maize, alfalfa, Atlantic salmon, trout, etc. (Vaughan et al., 2007; Chen et al., 2021; Kumar et al., 2024; Meng et al., 2024; Mackay and Anholt, 2024).

QTL hotspots, characterized by genomic locations enriched in QTL, represent a common and notable feature when collecting numerous QTL for various traits in various biological studies (Chardon et al., 2004; West et al., 2007; Breitling et al., 2008; Wu et al., 2008; Yang et al., 2019; Meng et al., 2024). These hotspots are significant and appealing due to their high informativeness and potential harboring for genes related to quantitative traits. Presently, both the data containing original marker genotypes and numerous molecular traits for each individual (referred to as individual-level data hereafter) from genetical genomics experiments and the summarized QTL data from public QTL databases can provide the data sets with numerous QTL for hotspot analysis. Statistical methods using either type of data for detecting QTL hotspots have been proposed, and they are mainly based on the permutation test approach (Wu et al., 2008; Li et al., 2010; Breitling et al., 2008; Neto et al., 2012; Yang et al., 2019; Wu et al., 2021). Among these methods, the statistical framework outlined by Yang et al. (2019) and Wu et al. (2021) has the notable features of being able to handle both types of data, addresses several challenges and saves computational cost in the process of QTL hotspot detection.

Statistical QTL mapping software packages such as MapMaker/QTL (Lincoln et al., 1993), WinQTLCart (Wang, 2000), R/**qtl** (Broman et al., 2003), QTLNetwork (Yang et al., 2008; Taylor and Verbyla, 2011; Verbyla et al., 2012), QTL.gCIMapping.GUI (Zhang et al., 2020) have been developed and documented in the literature. Among them, while most packages consider fixed effect models, the R/**wgaim** package developed in the R system offers linear mixed effects and random effects models, respectively, for QTL mapping in the backcross, DH and RIL populations. It has the feature of being able to simultaneously incorporate the whole genome into the model, and hence is relatively simple in selecting and detecting QTL during the analysis (Taylor and Verbyla, 2011). Notably, R/**qtl** is a free and powerful R package that provides a broad range of methods, which include single-marker analysis, interval mapping (Lander and Botstein, 1989), regression interval mapping (Haley and Knott, 1992), multiple QTL mapping (Jansen, 1993), composite interval mapping (Zeng, 1994) for a wide variety of experimental populations for QTL mapping. Permutation test (Churchill and Doerge, 1994) is used to determine significance thresholds for QTL mapping in R/**qtl**. Here, we introduce an R package **QTLEMM** (QTL EM algorithm mapping) that implements commonly used and popular statistical methods for both QTL mapping and QTL hotspot detection. For QTL mapping analysis, in addition to providing most of the methods in R/**qtl**, **QTLEMM** also offers multiple interval mapping (Kao et al., 1999) to fit multiple QTL directly in the model for a wide range of experimental populations. Furthermore, **QTLEMM** can perform novel tasks that the existing R packages lack such as simulating and handling the complete or selective genotyping data, computing the significance threshold values based on Gaussian stochastic process, and providing the asymptotic variance-covariance matrix for the QTL estimates (Kao and Zeng, 1997; Kao and Ho, 2012; Lee et al., 2014). **QTLEMM** also distinguishes itself by uniquely offering the statistical framework of Yang et al. (2019) and Wu et al. (2021) using either the individual-level data or summarized data to proceed with the QTL hotspot detection analysis. We provide a comprehensive overview of the primary R functions in the **QTLEMM** package. Results from analyses are presented through numerical and graphical outputs, facilitating interpretation and visualization of findings. The **QTLEMM** package provides researchers with statistical tools to find more significant results in exploring the network among expression of genes, QTL hotspots, and quantitative traits in genes, genomes, and genetics studies.

2 Statistical methods

Identifying individual QTL (QTL mapping) is a crucial endeavor aimed at understanding the genetic basis and architecture of quantitative traits, thereby facilitating trait manipulation and improvement. Since the specific locations of QTLs are unknown prior to mapping and they could potentially be located anywhere along the genome, the primary objectives of statistical methods are centered around searching for individual QTLs and subsequently fitting them all into statistical model for the estimation of QTL parameters.

2.1 QTL mapping models

Lander and Botstein (1989) were the first to propose a QTL mapping procedure known as interval mapping, which systematically searches the entire genome for QTLs. The interval mapping approach utilizes one marker interval (one flanking marker pair) at a time to establish a putative QTL at a specific position. It models the relationship between a quantitative trait and the putative QTL at that position, subsequently testing for the presence of the QTL. For a putative QTL, denoted as Q , at a specific fixed position x along the genome, the statistical model for individual i with a phenotypic trait value y_i can be expressed as follows:

$$y_i = G_i + \varepsilon_i \quad (1)$$

where G_i represents the genotypic value of individual i , and ε_i is a residual assumed to follow a normal distribution with mean 0 and variance σ^2 . For the individuals in a population derived from two inbred lines, such as the F_2 population, the genotypes of their Q can be one of the three possible genotypes, P_1 homozygote (QQ), heterozygote (Qq) or P_2 homozygote (qq). Several different genetic models have been proposed to characterize the relationship between genotypic values and gene effects (Cockerham, 1954; Van Der Veen, 1959; Weir and Cockerham, 1977; Kao and Zeng, 2002). According to Cockerham's model (Kao and Zeng, 2002), the relationship between the three genotypic values and the QTL effects can be modeled as $G_{QQ} = \mu + a - d/2$, $G_{Qq} = \mu + d/2$ and $G_{qq} = \mu - a - d/2$, respectively, where μ is the mean genotypic value, a and d represent the additive and dominance effects of the QTL, respectively. We then can construct an equivalent model of equation 1 for individual i as follows:

$$y_i = \mu + ax_i + dz_i + \varepsilon_i \quad (2)$$

where $(x_i, z_i) = (1, -1/2)$, $(0, 1/2)$ or $(-1, -1/2)$ if the QTL genotype of individual i is QQ , Qq or qq . Equation 2 builds the relationship between the genotypic values and QTL genotypes. If the putative QTL is located at the marker, the model is a regression model. However, if the putative QTL is positioned at x within the marker interval (M,N), the genotypes of the QTL are not directly observable and must be inferred from its flanking markers M and N. In this scenario, the statistical model typically becomes a normal mixture model and is called interval mapping (IM) model. Given data with n individuals, the likelihood function of the IM model for $\theta = (\mu, a, d, \sigma^2)$ can be expressed as follows:

$$L(\theta|Y, X) = \prod_{i=1}^n \left[\sum_{j=1}^3 p_{ij} \times f(y_i|\mu_j, \sigma^2) \right] \quad (3)$$

where $f(y_i|\mu_j, \sigma^2)$ represents a normal probability density function with mean μ_j and variance σ^2 . The μ_j 's correspond to the genotypic values of the three different QTL genotypes ($\mu_1 = G_{QQ}, \mu_2 = G_{Qq}, \mu_3 = G_{qq}$), while p_{ij} 's denote the mixing proportions (conditional probabilities) of the three QTL genotypes inferred from the two flanking markers (refer to Kao and Zeng, 2009, for obtaining p_{ij} 's in various experimental populations). By treating the normal mixture model as an incomplete-data problem, the EM algorithm (Dempster

et al., 1977) can be readily implemented to obtain the maximum likelihood estimates (MLE) of the parameters. Subsequently, a likelihood ratio test (LRT) can be performed to test the null hypothesis of no QTL ($H_0 : a = 0$ and $d = 0$) at the position x . With a fine-scale genetic marker map throughout the genome, the IM model can be conducted at all positions covered by markers to produce a continuous LRT statistic profile along chromosomes. By setting a predetermined LRT threshold, the position with the significantly largest LRT statistic in a chromosome region is considered the estimated QTL location. This method enables the systematic search and identification of QTLs at the genome-wide level, thereby facilitating the estimation of QTL parameters. However, since the search process for QTL needs to be performed at every position of the genome, the iterative expectation-maximization (EM) algorithm can become computationally expensive for QTL mapping (Haley and Knott, 1992; Kao, 2000). Haley and Knott (1992) introduced regression interval mapping (REG IM) as an approximation to the IM model, aimed at reducing computational costs. In REG IM, the quantitative trait value is regressed on the conditional expected genotypic value, providing a computationally efficient alternative to IM (Haley and Knott, 1992), although the approximation may not be satisfactory in all cases (Kao, 2000; Sen and Churchill, 2001).

The approach of IM model focuses on one putative QTL at a time within the model. However, it may introduce bias in the identification and estimation of QTLs when multiple QTLs are present in the same linkage group (Lander and Botstein, 1989; Haley and Knott, 1992; Zeng, 1994). To address this issue, composite interval mapping (CIM, Zeng, 1994) and multiple QTL mapping (MQM) model (Jansen, 1993), which combines interval mapping with multiple regression analysis, was proposed. During the test for a putative QTL, they both involve using other markers as covariates to mitigate the interference of other QTLs and reduce residual variance, thereby improving the accuracy of the test. The **QTLEMM** package provides the IM, REG IM, CIM and MQM models for the interval mapping QTL analysis. To further enhance QTL mapping, Kao et al. (1999) introduced the multiple interval mapping (MIM) approach. The MIM approach aims to leverage multiple marker intervals concurrently to incorporate multiple putative QTLs into the model for QTL mapping. For instance, considering m putative QTLs, $Q_1, Q_2, \dots, Q_m$, located at given positions within m separate marker intervals, $(M_1, N_1), (M_2, N_2), \dots, (M_m, N_m)$, respectively, the statistical model fitted these m putative QTLs can be expressed as follows:

$$y_i = \mu + \sum_{j=1}^m (a_j x_{ij} + d_j z_{ij}) + \varepsilon_i \quad (4)$$

For m putative QTLs in the model, there are 3^m possible QTL genotypes, and the likelihood of the model for $\theta = (\mu, a_1, d_1, a_2, d_2, \dots, a_m, d_m, \sigma^2)$ becomes a mixture of 3^m normal

$$L(\theta|Y, X) = \prod_{i=1}^n \left[\sum_{j=1}^{3^m} p_{ij} \times f(y_i | \mu_j, \sigma^2) \right] \quad (5)$$

under the normal assumption, where p_{ij} 's are the conditional probabilities of the 3^m possible QTL genotypes given the flanking marker genotypes. The statistical model (equation 4) with normal mixture likelihood (equation 5) is called MIM model. The general formulas by Kao and Zeng (1997), formulated based on the EM algorithm, can be used to estimate the parameters of the MIM model. To avoid using the iterative EM algorithm, alternative approximate methods considering multiple QTLs in the model include REG IM (Haley and Knott, 1992) and multiple imputation by Sen and Churchill (2001). While the two approximate methods offer faster computational speeds, their differences compared to the MIM model in the QTL analysis can be significant in certain situations, as discussed by Kao (2000) and Sen and Churchill (2001), and demonstrated through empirical examples (not shown). The R/**qtl** package provides these two approximate methods for QTL mapping. Subsequently, Kao (2004), Kao (2006) and Kao and Zeng (2009) extended the MIM model to a wide range of advanced populations for QTL mapping, considering specific genome structures present in advanced populations. In addition, Lee et al. (2014) further developed

the MIM model for the selective genotyping design, a topic we discuss below. The MIM approach indeed offers enhanced precision and power in QTL mapping. Also, it enables the analysis and estimation of epistasis between QTL, more accurate prediction of genotypic values of individuals, and estimation of heritabilities of quantitative traits. The **QTLEMM** package provides the MIM model and the REG IM method (considering multiple QTLs) to deal with the situations of multiple QTLs in the QTL mapping analysis.

Determination of threshold values

In the interval mapping procedure, a series of null hypotheses, both correlated and uncorrelated, are tested using LRT statistics across all genomic positions. Given the multiplicity of tests, controlling genome-wide error rates is crucial when determining threshold values for claiming significant QTL detection. It has been recognized that various factors, such as the number and size of intervals, population genome structures, and marker density, are involved and should be considered in determining the threshold value of QTL detection. To address this challenge, several analytical, empirical, and numerical approaches have been proposed to obtain the threshold values. These include methods like Bonferroni adjustment, Ornstein-Uhlenbeck process, numerical simulation, permutation test, and Gaussian process. Each offers unique insights and advantages in obtaining threshold values tailored to the specific characteristics of the QTL mapping study (Lander and Botstein, 1989; Churchill and Doerge, 1994; Rebai et al., 1994; Piepho, 2001; Zou, 2004; Chang et al., 2009; Guo, 2011; Kao and Ho, 2012). It has been known that numerical methods like permutation tests or numerical simulations are computationally intensive, and analytical methods like Gaussian processes offer a more efficient alternative with lower computational costs. The Gaussian process approaches by Chang et al. (2009), Guo (2011) and Kao and Ho (2012) can stand out as particularly efficient, as it is much faster than the permutation test in obtaining thresholds. This significant feature in computational speed makes the Gaussian process method a highly practical and attractive option as far as the computational efficiency is concerned in determining the threshold values for QTL detection.

Chang et al. showed that the asymptotic distribution of the score test statistics, denoted as $u(x_i)$ for $i = 1, 2, \dots, k$, at all the k sequential positions in the genome, follows a Gaussian stochastic process. Furthermore, as the squared score statistic $u^2(x)$ is asymptotically equivalent to the LRT statistic (Cox and Hinkley, 1979; Chang et al., 2009), the distribution of the supremum of $u^2(x)$ along the genome under the null hypothesis can be used to assess the threshold value of the LRT statistic in QTL mapping. Based upon this concept, Guo (2011) and Kao and Ho (2012) extended Chang et al.'s methodology by deriving more general score test statistics and Gaussian processes tailored for evaluating threshold values in the backcross, F_2 , RI F_t and AI F_t populations. These advancements provide researchers with statistical tools to determine the significance thresholds for QTL mapping analyses in diverse experimental populations. In the scenario of the F_2 population, each of the k positions is linked with two score test statistics: one for the additive effect and the other for the dominance effect. Let U represent a vector whose components are the score test statistics at the k genomic positions. Therefore, the vector U has length of $2k$. The asymptotic distribution of U follows a Gaussian stochastic process, denoted as $U \sim N(\mu, \Sigma)$, which is a multivariate normal distribution. The variance-covariance matrix Σ captures the variability and correlations among the score test statistics across different genomic positions. If the QTL are located at markers, the genotypic distributions of one and two genes are needed to compute their variances and covariances. If the QTL are located in the marker intervals, the genotypic distributions of two, three and four genes are required to obtain their variances and covariances (see Kao and Ho 2012 for details). The transition equations proposed by Haldane and Waddington (1931), Geiringer (1944), and Kao and Zeng (2010) provide valuable tools for deriving genotypic frequencies of two, three, and four genes, facilitating the construction of the variance-covariance matrix. These equations offer insights into the genotypic distribution of a wide variety of experimental populations, enabling a deeper understanding of variance-covariance structures between genes. The general frameworks of the score test statistics and Gaussian processes introduced by Guo (2011) and Kao and

Table 1: Comparison of the threshold values at level $\alpha = 0.05$ obtained using numerical simulation, Gaussian process and permutation test in the backcross, F_2 and RIL populations.

Threshold	No. of markers						
	2	3	6	11	21	51	101
Backcross population							
Simulation ^a	5.54	6.13	6.48	7.45	7.80	8.21	8.28
Gaussian p. ^b	5.48	6.79	7.27	7.64	7.96	8.23	8.36
Permutation ^d	5.12	5.56	6.75	7.26	7.75	8.43	8.91
F ₂ population							
Simulation ^a	8.13	8.98	10.00	10.48	11.20	11.75	12.32
Gaussian p. ^c	8.10	8.97	9.85	10.97	11.15	11.73	11.96
Permutation ^d	7.36	8.10	9.72	10.26	11.01	11.61	12.68
RIL population							
Simulation ^a	5.74	6.41	7.31	7.87	8.64	9.21	9.53
Gaussian p. ^c	5.56	6.30	6.96	7.93	8.56	9.10	9.32
Permutation ^d	6.34	6.97	7.50	8.14	8.95	9.80	10.54

Markers are evenly placed on a 100-cM Chromosome. The F_2 population considers both additive and dominance effects. R/**qtl** is used to perform the permutations tests.

^a based on 10,000 simulated data sets each containing 200 individuals from the null distribution.

^b based on 10,000 simulations (Guo, 2011).

^c based on 10,000 simulations (Kao and Ho, 2012).

^d based on 1000 permutations of one simulated data set containing 200 individuals.

Ho (2012) can be used to obtain the threshold values of QTL mapping for genomes with different sizes and marker densities in the backcross, F_2 , RI F_i and AI F_i populations.

The permutation test has been an appealing approach to obtain the thresholds because it is robust to departures from distributional assumptions and can reflect the peculiarity of the data at hand. To justify the Gaussian process approaches, we perform the permutation tests using R/**qtl** to compute the thresholds at $\alpha = 0.05$ level for different numbers of equally spaced markers on a 100-cM chromosome in the backcross, F_2 and RIL populations. These thresholds are then compared with those obtained from the same set-up by using the numerical simulations and Gaussian processes in Guo (2011) and Kao and Ho (2012). The three sets of threshold values are tabulated and compared in Table 1. In general, it shows that the threshold values obtained by the three approaches are similar to each other in each population, although the thresholds from the permutation tests tend to be slightly larger as compared to those from the other two approaches in the RIL population. Also, the thresholds are higher in denser marker maps as expected. Besides, we also apply the permutation test and Gaussian process to obtain the thresholds for the two simulated and real examples (Sections 3.2.2 and 3.3.2), showing that the two sets of threshold values are also close to each other. The different approaches yield similar thresholds mainly because the normality assumption is met for the data. The above investigations justify the Gaussian process in the computation of the threshold values. Moreover, we found that the Gaussian process is approximately 16,800 times faster than the permutation test (without parallel computing) in obtaining thresholds. The computation cost of Gaussian process is much cheaper than the permutation test and numerical simulation. The R/**qtl** uses parallel computing techniques to perform the permutation tests so as to be significantly quicker in obtaining the thresholds. Even so, note that the issue of computational cost still remains in the permutation tests. Importantly, the Gaussian process methods have very low computational costs, making them practical for large-scale analyses. In practice, when given a specific significance level and genome size, threshold values should be adjusted to account for denser marker maps and more advanced populations. This adjustment ensures that the statistical analysis appropriately controls for multiple testing and accounts for the complexities inherent in different genetic backgrounds and experimental designs. The QTLEMM package implements the Gaussian processes derived by Guo (2011) and Kao and Ho (2012) for computing significant thresholds of QTL mapping.

Selective genotyping

The cost of conducting QTL mapping experiments includes both phenotyping and genotyping expenses. In situations where budget constraints are not a primary concern, researchers usually choose complete genotyping, wherein all individuals in the sample undergo both genotyping and phenotyping procedures. However, despite recent reductions in genotyping costs, researchers frequently encounter insufficient budgets that prevent them from fully covering the expenses of complete genotyping. In situations where budgets are insufficient, researchers may explore alternative cost-saving approaches. Selective genotyping has been known as a cost-saving strategy to reduce genotyping work and can still maintain nearly equivalent efficiency to complete genotyping in QTL mapping (Lebowitz et al., 1987; Lander and Botstein, 1989; Xu and Vogl, 2000; Lee et al., 2014). This method involves selecting individuals from the high and low extremes of the trait distribution for genotyping, while leaving the remaining individuals ungenotyped within the entire sample. By focusing genotyping on individuals with extreme trait values, researchers can still capture most of the genetic variation in the sample to maintain efficiency. Overall, selective genotyping allows researchers to balance between budget constraints and mapping efficiency in QTL detection analysis.

Suppose that the sample consists of n individuals, out of which n_s individuals with extreme trait values ($n_s/2$ each from the upper and lower extremes) are selected for marker genotyping. The remaining $n_u = n - n_s$ individuals are not genotyped. Statistical QTL mapping methods for analyzing selective genotyping data can either consider all the n individuals (full data) or consider just the n_s genotyped individuals (genotyping data) in their models for QTL detection. If only the genotyping data are utilized in the analysis, data of this sort are called centrally truncated data. Xu and Vogl (2000) and Lee et al. (2014) introduced the truncated model within the mixture framework of interval mapping procedure, presenting a truncated normal mixture model for QTL analysis. For n_s genotyped individuals, the likelihood function for θ in the m QTL model can be expressed as follows:

$$L(\theta|Y, X) = \prod_{i=1}^{n_s} \left[\sum_{j=1}^{3^m} p_{ij} \times \frac{f(y_{si}|\mu_j, \sigma^2)}{U_j} \right] \quad (6)$$

where y_{si} is the trait value of the i th genotyped individual, and

$$U_j = \int_{-\infty}^{T_L} f(y_{si}|\mu_j, \sigma^2) dy_{si} + \int_{T_R}^{\infty} f(y_{si}|\mu_j, \sigma^2) dy_{si} \quad (7)$$

is the cumulative density with trait values greater than T_R (right truncated point) and lower than T_L (left truncated point), such that $P(y_{si} > T_R) = P(y_{si} < T_L) = n_s/2n$. Further details on the EM algorithm for obtaining the MLE of the parameters in the truncated normal mixture model are provided in Lee et al. (2014). If the full data are fitted into the statistical model for QTL analysis, the model likelihood can be expressed as follows:

$$L(\theta|Y, X) = \prod_{i=1}^{n_s} \left[\sum_{j=1}^{3^m} p_{ij} \times f(y_{si}|\mu_j, \sigma^2) \right] \times \prod_{i=1}^{n_u} \left[\sum_{j=1}^{3^m} q_j \times f(y_{ui}|\mu_j, \sigma^2) \right] \quad (8)$$

where the first term represents the likelihood for the n_s genotyped individuals, while the second term accounts for the n_u ungenotyped individuals, and y_{ui} is the trait value of i th ungenotyped individual.

In equation 8, note that p_{ij} 's are derived from the conditional probabilities of the QTL genotypes given their flanking marker genotypes, and q_j 's represent the proportions of QTL genotypes in the ungenotyped individuals (Lee et al., 2014). In the parameter estimation, the same EM algorithm employed for complete genotyping (Kao and Zeng, 1997) can be directly applied to obtain the MLE. Studies have indicated that the analysis utilizing full data by the model in equation 8 outperforms that utilizing only genotyping data by the model in equation 6 because additional information from the ungenotyped individuals is

incorporated into the analysis (Xu and Vogl, 2000; Lee et al., 2014). Additionally, selective genotyping using larger genotyping proportions, such as $n_s/n = 0.5$, may maintain roughly equivalent power to complete genotyping, whereas using smaller genotyping proportions presents difficulties in achieving the same level of power (Lee et al., 2014). These current selective genotyping methods mainly focus on the backcross and F_2 populations. Herein, we have substantially extended and modified the MIM models in equations 6 and 8 for selective genotyping in other advanced populations by considering their specific population genome structures. The **QTLEMM** package provides the MIM models to deal with the selective genotyping data (full or genotyping data) from the F_2 population and the more advanced populations.

2.2 QTL hotspot detection

Genome-wide QTL hotspot detection typically requires datasets containing numerous QTL to proceed with the analysis. Currently, genetical genomics experiments and public QTL databases serve as two feasible sources of such data. These two data sources have different structures. Genetical genomics experiments provide individual-level data, enabling the detection of thousands of QTLs in a single experiment. On the other hand, public databases such as GRAMENE, Q-TARO, Rice TOGO browser, PeanutBase, and MaizeGDB curate thousands of summarized QTL data. These databases curate the information from numerous independent QTL experiments across various traditional traits, and contain detected QTL, trait names, and reference sources but lack individual-level data. Utilizing both individual-level data from genetical genomics experiments or summarized QTL data from public databases, several statistical methods, primarily based on permutation tests, have been proposed to detect QTL hotspots. West et al. (2007), Wu et al. (2008), Li et al. (2010), Breitling et al. (2008) and Neto et al. (2012) have developed statistical methods to detect QTL hotspots for genetical genomics experiments. These methods for detecting QTL hotspots may suffer from several problems, including ignoring the correlation structure among traits, neglecting the magnitude of LOD scores of the QTLs, or incurring a very high computational cost. These problems often lead to the detection of excessive spurious hotspots, failure to discover biologically interesting hotspots composed of a small to moderate number of QTLs with strong LOD scores, and computational intractability, respectively, during the detection process. Solving these problems is crucial for improving the accuracy and efficiency of QTL hotspot detection.

The statistical framework developed by Yang et al. (2019) and Wu et al. (2021) can accommodate both individual-level data and summarized data, and it can also address the aforementioned problems at a time in QTL hotspot detection. The statistical framework first summarizes the QTL for all traits using an EQF (expected QTL frequency) matrix. The EQF matrix has column dimension equivalent to the genome size and row dimension corresponding to the number of traits. As the statistical framework directly operates on the EQF matrix, it has a very cheap computational cost. Then it lets $\gamma_{t,\alpha}$ represent the EQF threshold for assessing at least t spurious hotspots at level α in the EQF matrix. Two special devices, trait grouping and top $\gamma_{t,\alpha}$ threshold, are deployed to handle the remaining problems. The trait grouping groups the tightly linked and/or pleiotropic traits together to account for the correlation structure among traits, and then can be used as an option to obtain much stricter EQF thresholds and to dismiss spurious QTL hotspots. The top $\gamma_{t,\alpha}$ threshold is defined as the highest EQF threshold (with the smallest t) necessary for a genomic position to be significant as a QTL hotspot in the EQF matrix, and can be used to outline the LOD-score pattern of QTL in a hotspot across the different EQF matrices. The pattern of the top $\gamma_{t,\alpha}$ thresholds of a QTL hotspot across the different EQF matrices can profile its relative significance status as compared to other hotspots, so as to have the ability to identify the small and moderate hotspots with strong LOD scores in detecting QTL hotspots. The statistical framework of Yang et al. and Wu et al. has a very low computational cost and hence is particularly suitable for obtaining the QTL hotspot architectures for all the transcriptions within a reasonable time and cost frame in the genetical genomics experiments as compared to the approaches by permuting the individual-level data. Please refer to

Yang et al. (2019) and Wu et al. (2021) for further details on the statistical framework. The **QTLEMM** package provides their proposed statistical framework for QTL hotspot detection.

3 Using QTLEMM for QTL mapping analysis

The functions for the QTL mapping analysis in the **QTLEMM** package are capable of handling the data from backcross, F_2 , AI, RI, IRI and immortalized F_2 populations. For each population, the package considers both complete genotyping data and selective genotyping data for the QTL mapping analysis. The functions within the package enable the utilization of several methods including linear regression, IM, REG IM, CIM and MQM and MIM models for QTL mapping analysis, and they are outlined in Table 2. The progeny() function generates simulated trait and genotype data for diverse experimental populations. These data are then input into the IM.search() function to search the genome for potential QTLs. Additionally, the MIM.search() function can search for an additional QTL given other identified QTLs. The best position can be further obtained by using the MIM.points() function. Subsequently, the D.make() and Q.make() functions are employed to create the genetic design matrix of the QTL effects and the conditional probability matrix of the QTL genotypes, respectively. These two matrices are then utilized in the EM.MIM() function to estimate the parameters in the MIM model. Figure 1 is the flow chart of using the above functions for the QTL mapping analysis. Below, we demonstrate the application of these QTL mapping functions using both simulated and real examples.

3.1 Inputs

The QTL mapping data typically consist of two components: phenotypic trait values and marker genotypes observed in the individuals under study. To initiate QTL mapping analysis using the **QTLEMM** package, four essential arguments are required: markers (marker), genotypes (geno), phenotypes (y) and QTL (QTL). The marker argument is a $k \times 2$ matrix containing marker information, where k is the number of markers. In the marker argument, the first column labels the chromosomes where the markers are located, while the second column indicates the marker positions in Morgan (M) or centimorgan (cM). Table 3 provides an example of the marker argument, displaying that the first three markers of the first chromosome are positioned at 0, 24, and 40 cM, respectively. The QTL argument is a $q \times 2$ matrix containing QTL information, where q is the number of QTLs. Its format is the same as that of the marker argument. The geno argument is an $n \times k$ matrix containing the marker genotypes of n individuals. Genotypes for P_1 homozygote (MM), heterozygote (Mm) and P_2 homozygote (mm) are encoded as 2, 1 and 0, respectively, while missing genotypes are coded as NA. Table 4' provides an example of the geno matrix, where each row represents the genotypes of the k markers of an individual. The y argument is an $n \times 1$ vector containing the trait values of n individuals.

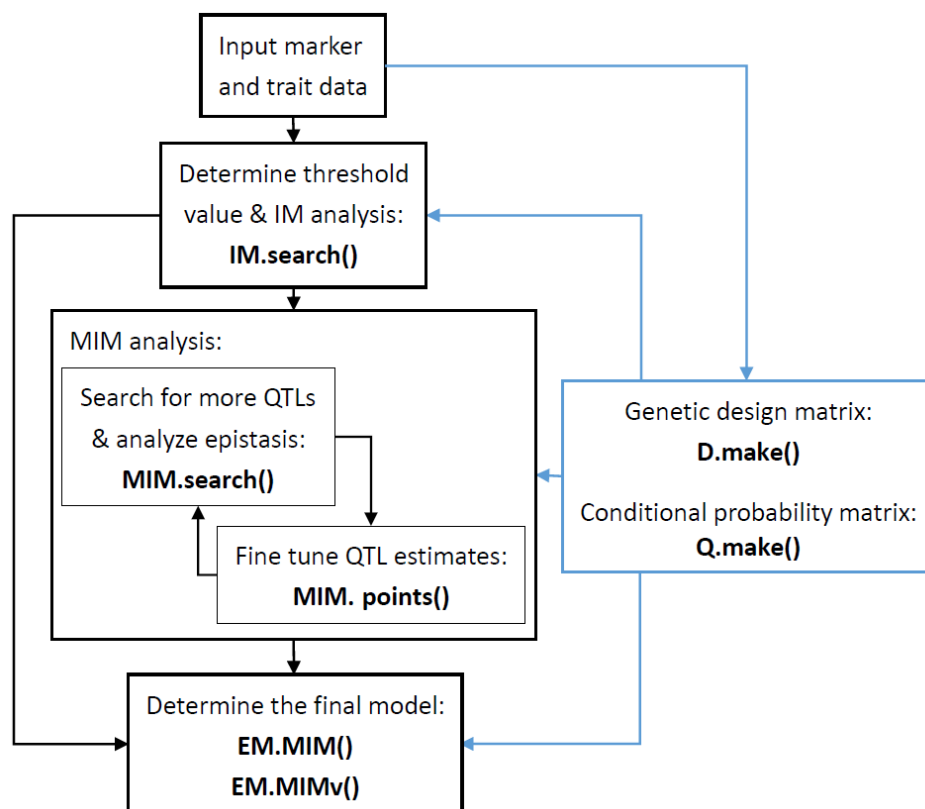
3.2 Simulation examples

Below a simulation example is presented to demonstrate the usage of the **QTLEMM** package. Initially, it is necessary to install and load the **QTLEMM** package and set an arbitrary random number seed, such as 8000, for data simulation in the R environment. Below are the codes.

```
> install.packages("QTLEMM")
> library(QTLEMM)
> set.seed(8000)
> options(digits = 3)
```

Table 2: List of the functions for QTL mapping in the QTLEMM package

Function	Description
Major function	
EM.MIM()	MIM to estimate the parameters.
EM.MIMv()	MIM to estimate the parameters and their variances.
IM.search()	IM to search for the possible QTL.
MIM.search()	MIM to search for one additional QTL given the identified QTLs in the model.
MIM.points()	MIM to fine tune the QTL parameters by a multidimensional search around the regions of the identified QTL in the model.
Minor function	
progeny()	Generate the simulated phenotype and genotype data.
D.make()	Generate the genetic design matrix.
Q.make()	Generate the conditional probability matrix.
LRTthre()	The LRT threshold for QTL detection based on Gaussian stochastic process.

**Figure 1:** Flowchart of using the QTL mapping functions in the QTLEMM package.**Table 3:** The format example of marker/QTL information data

chromosome	position_cM
1	0
1	24
1	40
...	...
12	72
12	126

Table 4: The format example of genotype data

	<i>marker</i> ₁	<i>marker</i> ₂	<i>marker</i> ₃	<i>marker</i> ₄	<i>marker</i> ₅	...	<i>marker</i> _{<i>k</i>}
<i>ind</i> ₁	2	1	1	2	0	...	2
<i>ind</i> ₂	2	1	0	0	1	...	1
<i>ind</i> ₃	2	2	NA	1	1	...	0
<i>ind</i> ₄	0	0	1	0	NA	...	2
...	...	...	...	...	...	...	...
<i>ind</i> _{<i>n</i>}	1	1	0	0	1	...	0

Data generation

The `progeny()` function can simulate marker genotype and phenotype (trait) data from experimental populations for QTL mapping analysis. This function accepts several key arguments: the `E` vector argument represents the effects of the QTL; the `ng` argument specifies the generation number; the `h2` argument sets the heritability; the `size` argument contains the sample size; the `type` argument is used to specify the population type, which includes backcross (`type="BC"`), advanced intercross population (`type="AI"`), and recombinant inbred population (`type="RI"`). Now consider the scenario that a simulated dataset consists of 200 F_2 individuals with three chromosomes, each with eleven 10-cM equally spaced markers. Three QTLs are positioned at [1,23] (the 23 cM of the 1st chromosome), [1,77] and [2,55], respectively, and their effects are assumed to be -10, 12, and 8, respectively. The 1st and 3rd QTLs have an additive-by-additive effect of 1. The heritability is set at 0.5. Below are the commands used to generate such a dataset. The command of defining the QTL effects is as follows:

```
> eff <- c("a1" = -10, "a2" = 12, "a3" = 8, "a1:a3" = 1)
```

If other effects, such as dominance effect of 3 for the 2nd QTL and additive-by-dominance effect of 2 for the 1st and 2nd QTLs, are considered, the arguments in the command is `d2=3` and `a2:d1=2`. Please refer to the [QTLEMM](#) document in CRAN for more detailed instructions. The commands for setting the specified positions of QTLs and markers are as follows:

```
> marker <- cbind(rep(1:3,each = 11), rep(seq(0, 100, 10), 3))
> QTL <- cbind(c(1, 1, 2), c(23, 77, 55))
```

Then, the `progeny()` function can use the above commands to generate the QTL mapping data of 200 F_2 individuals with heritability 0.5.

```
> testdata <- progeny(QTL, marker, type = "RI", ng = 2, E.vector = eff, h2 = 0.5,
  size = 200)
> names(testdata)
[1] "phe" "E.vector" "marker.prog" "QTL.prog" "genetic.value" "VG" "VE"

> y <- testdata$phe
> geno <- testdata$marker.prog
```

The `progeny()` function outputs a dataset into the object named `testdata`. This file contains four parts: phenotypes (`phe`), QTL effects (`E.vector`), marker genotypes (`marker.prog`), and QTL genotypes (`QTL.prog`). The markers and trait values of the 200 individuals in the `testdata` file are further extracted and organized into the `geno` matrix and `y` vector for QTL mapping analysis.

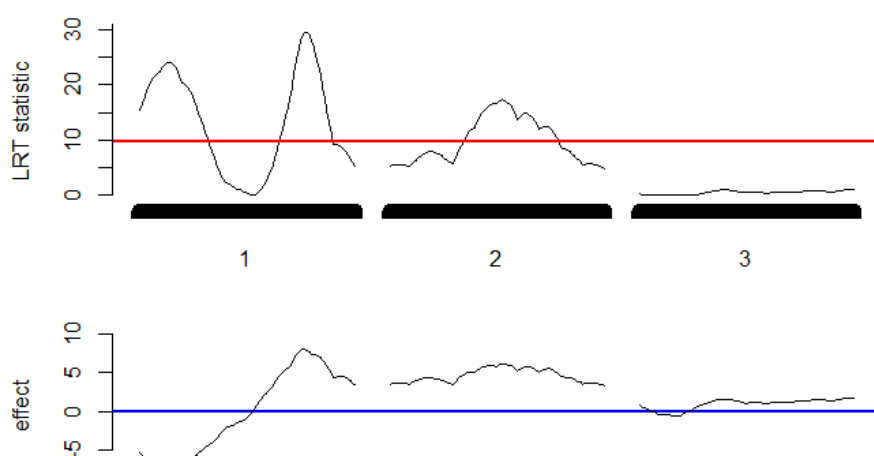


Figure 2: The graphical output generated by the `IM.search()` function. The upper plot shows the profile of LRT statistics, while the lower plot exhibits the profile of effects. The red line represents the threshold value of 9.62 obtained by using Gaussian process.

Interval mapping

The `IM.search()` function is designed to implement the IM model for the QTL mapping analysis. Its arguments include: the `type` argument specifies the population type (BC, AI, and RI population); the `ng` argument represents the generation number; the `speed` argument determines the walking speed of the IM analysis (in cM); the `d.eff` argument indicates if the dominant effect will be considered or not (for AI or RI); the `QTLdist` argument specifies the minimum distance (in cM) between the detected QTL; the `plot.all` and `plot.chr` arguments indicate whether plots of the LRT statistic profile will be generated or not. Below are the codes of using the `IM.search()` function to perform the IM analysis on the simulated dataset without considering any dominance effect.

```
> IMtest <- IM.search(marker, geno, y, type = "RI", ng = 2, speed = 1, d.eff = FALSE,
  QTLdist = 15, plot.all = TRUE, plot.chr = FALSE, console = FALSE)
> names(IMtest)
[1] "effect" "LRT.threshold" "detect.QTL" "model" "inputdata"

> IMtest$LRT.threshold
 95%
9.62
```

The outputs of the `IM.search()` function (`IMtest`) include: estimated effects at all positions (`effect`); LRT threshold (`LRT.threshold`) obtained using Gaussian process; numerical results of the detected QTLs (`detect.QTL`); graphical outputs. The `IM.search()` function can also perform the REG IM model for the QTL mapping analysis by setting the argument `method="REG"` (default `method="EM"`). Figure 2 is the graphical output of the `IM.search()` function. It illustrates the profiles of the LRT statistics and effects across the three chromosomes. The LRT profile shows three significant peaks, indicating three QTLs are detected, on two of the three chromosomes. For this dataset, the LRT threshold of considering both additive and dominance effects for assessing the significance of QTL detection is 12.58 (12.15) by using Gaussian process (permutation test in R/[qtl](#)). The LRT threshold value of considering only additive effect is 9.62 (9.82) by using Gaussian process (using our own permutation program, as R/[qtl](#) does not provide the option of considering only additive

effect in the F2 population). The numerical results of the detected QTLs can be listed using the following commands.

```
> detQTL <- IMtest$detect.QTL
> detQTL
      chr cM   a1  LRT    R2
14     1 14 -7.00 24.1 0.1064
77     1 77  8.03 29.6 0.1324
153    2 53  6.14 17.3 0.0787
```

The IM analysis concludes that the three QTLs are detected at [1,14], [1,77] and [2,53] with effects of -7.00, 8.03 and 6.14, respectively. They contribute approximately 10.64%, 13.24%, and 7.87% of the trait variation, respectively.

Multiple interval mapping

The analysis of the IM model can be further improved using the MIM approach by jointly fitting the three QTL into the model so as to obtain more precise and accurate estimates of QTL parameters. The EM.MIM() function is designed to perform the MIM model analysis. Before conducting the EM.MIM() function, two matrices, the genetic design matrix (D.matrix) and the conditional probability matrix (cp.matrix), must be constructed first. The D.make() and Q.make() functions are utilized to generate the two matrices, respectively. The commands in the D.make(), Q.make() and EM.MIM() functions for the MIM model fitting the three QTL at [1,14], [1,77] and [2,53] with an additive by additive effect (between the QTLs at [1,14] and [2,53]) are given below respectively.

```
> dQTL <- detQTL[,1:2]
> D.matrix <- D.make(nQTL = 3, type = "RI", a = TRUE, d = 0, aa = c(1, 3))
```

The first argument of the D.make() function is the number of QTL in the MIM model and is nQTL=3 in this case; the second argument specifies the population type and is type="RI"; the arguments a and d indicate if additive or dominance effects will be considered and they are a=TRUE and d=0 since only the additive effects are considered; the arguments aa, dd, and ad specify the epistatic effects between QTLs and is aa=c(1, 3) since the additive by additive effect between the 1st and 3rd QTLs is considered. The dimension of D.matrix matrix for this three-QTL MIM model is 27×4 , and the elements of first six rows are shown below.

```
> dim(D.matrix)
[1] 27 4

> head(D.matrix)
      a1 a2 a3 a1:a3
222  1  1  1     1
221  1  1  0     0
220  1  1 -1    -1
212  1  0  1     1
211  1  0  0     0
210  1  0 -1    -1
```

The arguments in the Q.make() function for generating the conditional probability matrix of the three-QTL MIM model in this case are shown below. The dimension of the cp.matrix matrix for this three-QTL MIM model is 200×27 .

```
> cp.matrix <- Q.make(dQTL, marker, geno, type = "RI", ng = 2)$cp.matrix
> dim(cp.matrix)
[1] 200 27
```

Three inputs are required for driving the EM.MIM() function to perform the MIM analysis: the genetic design matrix (D.matrix); the conditional probability matrix (cp.matrix); the phenotypic values (y). The outputs from the EM.MIM() function include a vector containing the estimated QTL effects (E.vector), the mean (beta), the residual variance (variance), the posterior probabilities matrix (PI.matrix), the log likelihood value (log.likelihood), the LRT statistics (LRT), the coefficient of determination (R2), the estimated trait values (y.hat), and the iteration time (iteration.time) as shown below.

```
> MIMtest <- EM.MIM(D.matrix = D.matrix, cp.matrix = cp.matrix, y = y, console = FALSE)
> names(MIMtest)
[1] "QTL" "E.vector" "beta" "variance" "PI.matrix" "log.likelihood" "LRT" "R2"
[9] "y.hat" "yu.hat" "iteration.number" "model"

> MIMtest$E.vector
      a1      a2      a3 a1:a3
-9.61 10.29 6.35  1.66

> c(MIMtest$log.likelihood, MIMtest$LRT, MIMtest$R2)
[1] -772.192 145.114 0.411
```

The log likelihood of the MIM model fitting the three QTL with epistasis is approximately -772. The estimated QTL effects are approximately -9.61, 10.29 and 6.35 (true values being -10, 12, and 8), respectively, and the estimated epistatic effect is approximately 1.66. The estimated heritability (R2) is 0.411, while the true heritability is 0.50. The above MIM-related functions can also perform the REG IM model (multiple QTL version) by setting the argument method="REG". Besides using the original marker and trait data, note especially that all the MIM-related functions can utilize the results from the analysis of IM or MIM functions, such as the IM.search(), MIM.search(), and MIM.points() functions, as input to conduct the analysis. Also, the marker and trait data and the detected QTL information in the previous analysis can be used in the subsequent analysis. Below are the codes of using the results from the IM analysis (IMtest) to conduct the EM.MIM() function..

```
> MIMtest_ <- EM.MIM(D.matrix = D.matrix, IMresult = IMtest, console = FALSE)
> MIMtest_.$E.vector
      a1      a2      a3 a1:a3
-9.61 10.29 6.35  1.66
```

The EM.MIMv() function can provide the asymptotic variance-covariance matrix of the QTL estimates. The inputs in the EM.MIMv() function include: QTL information about the QTL effects and positions (QTL); marker information (marker); genotypes (geno); genetic design matrix (D.matrix); conditional probability matrix (cp.matrix); phenotypic values (y). If the argument cp.matrix is set to NULL, the conditional probability matrix is constructed from the input QTL information and marker information. If the estimated QTL positions coincide with markers, the asymptotic variance-covariance matrix is not available. Below are the arguments of the EM.MIMv() function using the marker and trait data to produce the variance-covariance matrix for the MIM model fitting the three detected QTLs at [1,14], [1,77], and [2,53].

```
> MIMv <- EM.MIMv(dQTL, marker, geno, D.matrix, cp.matrix = NULL, y, console = FALSE)
> # MIMv <- EM.MIMv(D.matrix = D.matrix, IMresult = IMtest, console = FALSE)
> names(MIMv)
[1] "E.vector" "beta" "variance" "PI.matrix" "log.likelihood" "LRT" "R2" "y.hat"
[9] "iteration.number" "avc.matrix" "EMvar"
```

The avc.matrix is the asymptotic variance-covariance matrix, and the EMvar contains the asymptotic variances of the estimates. They are listed below.

```
> round(MIMv$avc.matrix, 3)
      QTL1  QTL2  QTL3  a1    a2    a3  a1:a3 residual.var    mu
QTL1    0.015  0.017  0.013 -0.003  0.000  0.014  0.076    -0.073 -0.003
QTL2    0.017  0.004 -0.006 -0.023  0.021 -0.003  0.041    -0.191  0.002
QTL3    0.013 -0.006  0.065 -0.034  0.035  0.036  0.134    -0.688  0.009
a1      -0.003 -0.023 -0.034  1.417 -0.354 -0.091  0.038     1.775  0.006
a2       0.000  0.021  0.035 -0.354  1.585 -0.039  0.154    -2.462 -0.096
a3       0.014 -0.003  0.036 -0.091 -0.039  1.463  0.257    -1.185 -0.034
a1:a3    0.076  0.041  0.134  0.038  0.154  0.257  3.724    -2.787 -0.096
variance -0.073 -0.191 -0.688  1.775 -2.462 -1.185 -2.787    179.411  0.030
X1      -0.003  0.002  0.009  0.006 -0.096 -0.034 -0.096     0.030  0.650

> round(MIMv$EMvar, 3)
      QTL1  QTL2  QTL3  a1    a2    a3  a1:a3 residual.var    mu
0.015 0.004 0.065 1.417 1.585 1.463 3.724    179.411  0.650
```

The asymptotic variances of the estimated QTL positions and effects are 0.015, 0.004, 0.065, 1.417, 1.585, 1.463 and 3.724, respectively. The asymptotic variances of the estimated mean and residual variance are 0.650 and 179.411, respectively.

The `MIM.search()` function is devised to fit the detected QTLs into the model to search the genome for other possible QTL. The arguments in the `MIM.search()` function include the detected QTL (denoted by `dQTL2` in this example), marker (for marker information), `geno` (for genotypes), `y` (for phenotypes), `type` (for population type), `ng` (for the generation number), `D.matrix` (for the genetic design matrix), `speed` (for the walking speed in cM), `QTLdist` (for the minimum distance between detected QTLs). The outputs of the `MIM.search()` function include information about the estimates of all search positions (`effect`), the best QTL positions with the largest log likelihood (`QTL.best`), and the estimated QTL effects at the best QTL positions (`effect.best`). For demonstration purposes, assume that the two detected QTLs located at [1,14] and [1,77] are fitted into the MIM model to search for the next (third) QTL considering the additive by additive effect (the design matrix will be the same as that in the above `EM.MIM()` function). Below are the commands of the `MIM.search()` function to conduct the search for the third QTL given the two detected QTLs:

```
> dQTL2 <- cbind(c(1, 1), c(14, 77))
> MIMs <- MIM.search(dQTL2, marker, geno, y, type = "RI", ng = 2, D.matrix = D.matrix,
  speed = 1, QTLdist = 15, console = FALSE)
> names(MIMs)
[1] "effect" "QTL.best" "effect.best" "model" "inputdata"

> MIMs$QTL.best
      chromosome position(cM)
QTL 1           1           14
QTL 2           1           77
QTL new         2           54

> MIMs$effect.best
      a1    a2    a3  a1:a3    LRT log.likelihood    R2
-9.619 10.302 6.385  1.806 145.876    -772.129 0.412
```

The third QTL is detected at the position [2,54] with an estimated effect of approximately 6.385. The log likelihood of the MIM model is about -772.129. The LRT statistic for testing the significance of the effects jointly is about 145.876.

Another function related to the MIM analysis is the `MIM.points()` function, which is used to fine tune the estimation of QTL parameters by multidimensional search around the detected QTLs. The fine-tuning ranges around the detected QTLs are defined using the `scope` argument, while the other arguments are the same as those in the `MIM.search()` function.

Below is the command of the `MIM.search()` function for performing a three-dimensional search on the 10-cM range on both sides of the three QTL at [1,14], [1,77] and [2,54] (with additive by additive effect).

```
> MIMp <- MIM.points(dQTL, marker, geno, y, type = "RI", ng = 2, D.matrix = D.matrix,
  speed = 2, scope = 10, console = FALSE)
> # MIMp <- MIM.points(D.matrix = D.matrix, speed = 2, scope = 10, IMresult = IMtest,
  console = FALSE)
> names(MIMp)
[1] "effect" "QTL.best" "effect.best" "model" "inputdata"

> MIMp$QTL.best
      chromosome position(cM)
[1,]           1           24
[2,]           1           75
[3,]           2           53

> MIMp$effect.best
      a1      a2      a3  a1:a3      LRT log.likelihood      R2
-10.846 11.994 6.503  3.725 181.130      -765.371 0.464
```

The results show that the largest model log likelihood becomes -765.371, and the estimated heritability is 0.464. After fine-tuning, the detected positions are closer to the true positions [1,23], [1,77] and [2,55], compared to the estimated positions [1,14], [1,77] and [2,53] before fine-tuning. With these estimates, other composite genetic parameters such as heritability and variance components of a quantitative trait can be estimated. Additionally, the response to selection can be predicted based on these estimates.

3.3 The yeast dataset example

The yeast dataset (Brem et al., 2005) consists of 112 backcross individuals with 5740 traits and 1072 markers. The raw data are reprocessed into a new dataset called 'yeast.process', which can be loaded from **QTLEMM** package using the following command:

```
> load(system.file("extdata", "yeast.process.RDATA", package = "QTLEMM"))
```

The `yeast.process` dataset comprises three lists: the list of marker genotypes (`yeast.process$geno`) that contains the marker genotypes of the 112 individuals; the list of trait values (`yeast.process$pheno`) that contains the trait values of the 112 individuals; the list of marker information (`yeast.process$marker`) that includes the marker map (distances) of the 1072 markers on the 16 chromosomes.

```
> geno <- yeast.process$geno
> marker <- yeast.process$marker
> pheno <- yeast.process$pheno
```

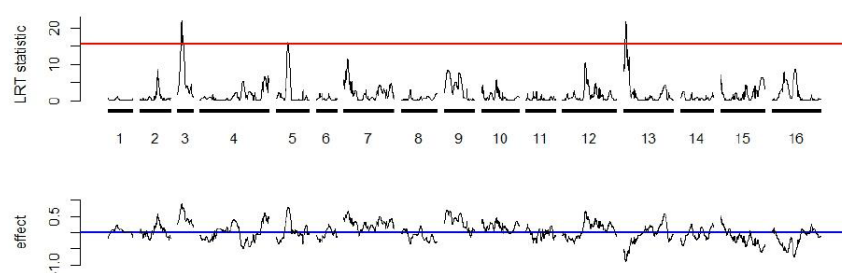
Selective genotyping data

For the demonstration of analyzing selective genotyping data, we selected the 3590th trait from the dataset and intentionally deleted the marker genotypes of the individuals with medium trait values to produce selective genotyping data for QTL mapping analysis. Specifically, one half of the individuals with extreme trait values (comprising one quarter each from the upper and lower extremes) are chosen to keep their marker genotypes and trait values, and the marker genotypes of the remaining individuals are deleted and ignored in the analysis. Below are the codes for generating the selective genotyping dataset.

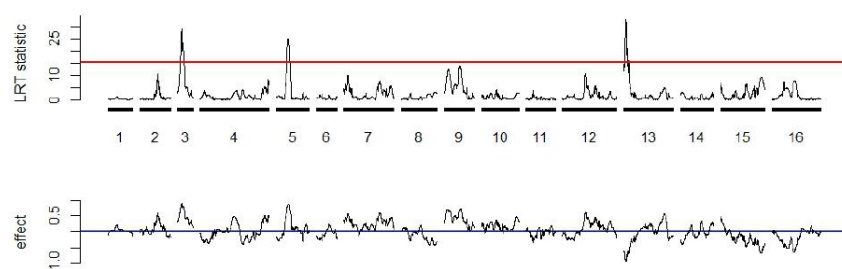
```

> y0 <- pheno[, 3590]
> y <- y0[y0>quantile(y0)[4] | y0<quantile(y0)[2]]
> yu <- y0[y0 >= quantile(y0)[2] & y0 <= quantile(y0)[4]]
> geno.s <- geno[y0 > quantile(y0)[4] | y0 < quantile(y0)[2],]

```



(a) selective genotyping data



(b) complete genotyping data

Figure 3: The profiles of the LRT statistics and estimated effects along the genome by using the `IM.search2()` function to analyze the selective genotyping data of the 3590th trait (part a), and using the `IM.search()` function to analyze the complete genotyping data (part b). The red line indicates the LRT threshold obtained by using Gaussian process for assessing the significance of QTL detection.

The vector `y` contains the trait values of the individuals with marker genotypes (the upper and lower 25% individuals), and the `geno.s` argument consists of their marker genotypes. The vector `yu` contains the trait values of individuals without marker genotypes. The `IM.search()` function can also perform several selective genotyping QTL mapping methods, which encompass the Lee et al. model (`sele.g="f"`), the truncated model (`sele.g="t"`), and the population frequency-based model (`sele.g="p"`), to analyze the selective genotyping dataset (see Lee et al. 2014 for details). If `sele.g="n"`, the function is used to analyze the complete genotyping data.

Selective genotyping IM and MIM analysis

The followings are the codes of the `IM.search()` function to analyze the selective genotyping data of the 3590th trait. The random number seed 8000 is used to set up the Gaussian process to compute threshold values for assessing the significance of QTLs.

```

> library(QTLEMM)
> set.seed(8000)
> IMtest2 <- IM.search(marker, geno.s, y, yu, sele.g = "f", type = "BC", ng = 1,
  plot.all = TRUE, plot.chr = FALSE, console = FALSE)
> IMtest2$detect.QTL
  chr  cM   a1  LRT   R2
626   3  53 0.893 22.0 0.1128

```



```

1579    5 112  0.753 16.0 0.0749
4523   13  22 -0.882 21.7 0.1048
> IMtest2$LRT.threshold
95%
15.4

```

Figure 3(a) presents the profiles of the LRT statistics and estimated effects along the genome. It shows that three QTL are detected at [3,53], [5,112] and [13,22], respectively. The LRT threshold value is 15.40 (15.36) by the Gaussian process (permutation test in R/[qtl](#)). For comparison, we use the `IM.search()` function to conduct complete genotyping analysis for the 3590th trait.

```

> IMcon <- IM.search(marker, geno, y0, type = "BC", ng = 1, LRT.thre = 15.4,
  plot.all = TRUE, plot.chr = FALSE, console = FALSE)
> IMcon$detect.QTL
      chr  cM    a1  LRT    R2
624     3  53  0.904 29.0 0.210
1580    5 117  0.877 25.0 0.171
4511   13  22 -0.945 33.1 0.231

```

The profiles of the LRT statistics and estimated effects along the genomes are presented in Figure 3(b). It shows that three QTL are detected at [3,53], [5,115] and [13,22], respectively. Both the selective and complete genotyping IM analyses produce similar LRT statistic profiles and estimates of positions and effects. For each detected QTL, the complete genotyping data analysis produces larger LRT statistics and R^2 's as compared to the selective genotyping analysis.

Using the `IM.search()` function, the estimates of QTL effects and positions, model likelihoods and model R^2 values were obtained individually. Certainly, we would like to further fit these detected QTLs simultaneously into a multiple-QTL model (the MIM Model). This allows the QTLs to be jointly fitted and controlled in the model to explain more genetic variation of the quantitative traits and obtain more precise estimates. Below are the commands of the `EM.MIM()` function to perform the selective genotyping MIM model analysis that considers the three detected QTLs and their all possible epistasis.

```

> D.matrix <- D.make(3, type = "BC", aa = TRUE)
> MIMtest2 <- EM.MIM(D.matrix = D.matrix, IMresult = IMtest2, console = FALSE)
> MIMtest2$E.vector
      a1    a2    a3  a1:a2  a1:a3  a2:a3
0.818 0.744 -0.954 -0.641  0.371 -0.423

> c(MIMtest2$log.likelihood, MIMtest2$LRT, MIMtest2$R2)
[1] -115.253  79.117  0.512

```

The model R^2 and likelihood are 0.512 and -115.41, respectively. The estimated marginal and epistatic QTL effects are 0.818, 0.744, -0.954, -0.641, 0.371 and -0.423, respectively. The `MIM.points()` function can be further used to perform a multi-dimensional search around the 5-cM regions of the detected QTL positions (at [3,53], [5,112] and [13,22]) to fine-tune the QTL estimates (using the argument of `scope=5`). Below are the codes of the `MIM.points()` function to perform the multi-dimensional search and the fine-tuning results.

```

> MIMp <- MIM.points(D.matrix = D.matrix, scope = 5, IMresult = IMtest2, console = FALSE)
> MIMp$QTL.best
      chromosome position(cM)
[1,]           3           58
[2,]           5          111
[3,]          13           21

```

```
> MIMp$effect.best
      a1      a2      a3  a1:a2  a1:a3  a2:a3    LRT log.likelihood    R2
0.678 0.758 -0.964 -0.866  0.673 -0.499 82.214      -113.705 0.524
```

The model with the largest log likelihood (-113.705) occurs at positions [3,58], [5,111] and [13,21], and the estimated effects are 0.678, 0.758, -0.964, -0.866, 0.673, -0.499, respectively. The model R^2 (estimated heritability) improves from 0.512 to 0.524.

4 Using QTLEMM for QTL hotspot detection

The analysis of QTL hotspot detection has been a pivotal step towards unraveling the genetic architectures of quantitative traits in the study of genes, genomes and genetics (Breitling et al., 2008; Fu et al., 2009; Neto et al., 2012; Wang et al., 2014; Yang et al., 2019; Wu et al., 2021). The genetical genomics experiments and public QTL databases are two feasible sources to provide data with many QTLs for the detection of QTL hotspots. The statistical framework of QTL hotspot detection proposed by Yang et al. (2019) and Wu et al. (2021) is capable of accommodating both types of data. The framework addresses various challenges, including handling the correlation structure among traits, identifying different types of hotspots, and ensuring computational efficiency, thereby making it practical for QTL hotspot detection. The QTLEMM package offers the statistical framework by Yang et al. and Wu et al. for the detection of QTL hotspots. The functions for detecting QTL hotspots using the framework are summarized in Table 5. Below, we present the analyses of two real examples, the yeast genetic genomics dataset and the GRAMENE rice database, as demonstration of using these functions for detecting QTL hotspots.

4.1 The yeast genetic genomics dataset example

There are 5740 molecular traits in the yeast dataset (Brem et al., 2005). The QTL mapping procedure employed for the 3590th trait using the IM.search() function can be applied to analyze the remaining 5739 traits, obtaining their LRT statistics at all positions along the genome. These LRT statistics can then be converted into LOD scores using the formula $LOD = LRT/4.6$. Subsequently, the LOD scores are organized into a LOD matrix for QTL hotspot detection, following the methods outlined by Yang et al. (2019) and Wu et al. (2021). The LOD.QTLdetect() function is constructed to detect QTL hotspots. It requires two input datasets: the LOD matrix and the bin information on the chromosomes. The LOD matrix is a $t \times p$ matrix, where t and p are the numbers of traits and numbers of bins on the chromosomes, respectively. The LOD matrix contains the LOD scores of all bins for all traits (refer to Table 6). The bin information is an $n \times 2$ matrix, where n is the number of chromosomes, and it contains the information about the bin number on each chromosome. The first column denotes the chromosomes, and the second column denotes the numbers of bins on the chromosomes (refer to Table 7).

Table 5: List of the functions for QTL hotspot detection in the QTLEMM package

Function	Description
LOD.QTLdetect()	Detect QTL by LOD matrix.
EQF.permu()	EQF matrix cluster permutation process for QTL hotspot detection.
EQF.plot()	Depict the EQF plot by the result of permutation process.
Qhot()	This function generates both numerical and graphical summaries of the QTL hotspot detection in the genomes.
Qhot.EQF()	Convert the QTL flanking marker data to EQF matrix and carry out the EQF matrix cluster permutation process.

The LOD matrix of the yeast data can be downloaded from GitHub using the commands below. Users can combine the four files (yeast.LOD.1.RDATA, yeast.LOD.2.RDATA, yeast.LOD.3.RDATA, yeast.LOD.4.RDATA) to obtain the complete LOD matrix.

Table 6: The format of LOD matrix

	bin_1	bin_2	bin_3	bin_4	bin_5	...	bin_n
$trait_1$	0.047	0.116	0.209	0.313	0.342	...	0.358
$trait_2$	0.095	0.176	0.274	0.376	0.301	...	0.342
$trait_3$	0.798	0.67	0.533	0.394	0.342	...	0.284
$trait_4$	0.363	0.321	0.272	0.219	0.192	...	0.149
$trait_5$	0.017	0.01	0.005	0.002	0.001	...	0
...	...	...	...	...	...	...	...
$trait_t$	0.683	0.593	0.471	0.336	0.304	...	0.271

Table 7: The format example of bin information

chromosome	number_of_bin
1	256
2	324
3	160
4	723
...	...
15	463
16	513

```

> load(url("https://github.com/py-chung/QTLEMM/raw/main/inst/extdata/yeast.LOD.1.RDATA"))
> load(url("https://github.com/py-chung/QTLEMM/raw/main/inst/extdata/yeast.LOD.2.RDATA"))
> load(url("https://github.com/py-chung/QTLEMM/raw/main/inst/extdata/yeast.LOD.3.RDATA"))
> load(url("https://github.com/py-chung/QTLEMM/raw/main/inst/extdata/yeast.LOD.4.RDATA"))
> load(url("https://github.com/py-chung/QTLEMM/raw/main/inst/extdata/yeast.LOD.bin.RDATA"))
> LOD <- rbind(yeast.LOD.1, yeast.LOD.2, yeast.LOD.3, yeast.LOD.4)
> bin <- yeast.LOD.bin

```

Once the LOD matrix is available, the `LOD.QTLdetect()` function can be applied to detect QTL hotspots. The function's arguments include LOD for the LOD matrix (refer to Table 6), bin for the numbers of bins on each chromosome (refer to Table 7), `thre` for the threshold value (in terms of LOD) of QTL detection, and `QTLdist` for specifying the minimum distance (cM) between the detected QTL. The numerical results of this function will be output to the `LOD.QTLdetect.result` file.

```

> library(QTLEMM)
> set.seed(8000)
> LOD.QTLdetect.result <- LOD.QTLdetect(LOD, bin, thre = 3, QTLdist = 20, console = FALSE)
> names(LOD.QTLdetect.result)
[1] "detect.QTL.number" "QTL.matrix" "EQF.matrix" "linkage.QTL.number"
[5] "LOD.threshold" "bin"

```

The `LOD.QTLdetect.result` file is a data list comprising several components: `detect.QTL.number` contains the number of detected QTL of each trait; `QTL.matrix` holds the QTL positions, where elements marked as 1 represent the QTL positions, elements marked as 0 represent bins with LOD scores under the LOD threshold, and other positions are designated as NA; `EQF.matrix` contains the EQF value of each bin; `linkage.QTL.number` indicates the number of linked QTL among all detected QTLs; `LOD.threshold` and `bin` remain the same as those in the input data. With these information, the `EQF.permu()` function embedding the permutation analysis (with trait grouping; Wu et al., 2021) can be applied to detect QTL hotspots. The arguments in the `EQF.permu()` function involve inputting the output data from `LOD.QTLdetect()`, specifying the permutation time (`npermu`), and using

the genome-wide error rate (GWER) of a given level α to carry out the permutation analysis. Additionally, the `Q=TRUE` argument is to perform permutation analysis without trait grouping.

```
> EQF.permu.result <- EQF.permu(LOD.QTLdetect.result, npermu = 1000, alpha = 0.05,
  Q = TRUE, console = FALSE)
> names(EQF.permu.result)
[1] "EQF.matrix" "bin" "LOD.threshold" "cluster.number" "cluster.id" "cluster.matrix"
[7] "permu.matrix.cluster" "permu.matrix.Q" "EQF.threshold"
```

The output of the `EQF.permu()` function includes several components. The `EQF.matrix`, `bin`, and `LOD.threshold` lists represent the EQF matrix, bin information matrix, and the LOD threshold respectively, which are the same as those in the input data. The `cluster.number` contains the number of QTLs in each trait group. The `cluster.id` contains the serial number of traits in each trait group. The `cluster.matrix` includes the reduced EQF matrix after trait grouping. The `permu.matrix.cluster` contains the result of permutation with trait grouping, sorted by order. Similarly, the `permu.matrix.Q` contains the result of the permutation without trait grouping, also sorted by order. The `EQF.threshold` represents the EQF threshold calculated from the permutation analysis. Moreover, the `EQF.plot()` function is designed to plot the figure of the EQF architecture of the genome (see Figure 4) using the output of the `EQF.permu()` function. Below are the codes.

```
> EQF.plot(EQF.permu.result, plot.all = TRUE, plot.chr = FALSE)
```

The command of `plot.all = TRUE` is to draw the EQF architecture of the entire genome (16 chromosomes) in a single figure (Figure 4). If `plot.chr = TRUE`, the EQF architectures of genome is drawn separately by chromosomes in different figures.

4.2 The GRAMENE rice database example

The `Qhot()` function manages summarized QTL data collected from public QTL databases to detect QTL hotspots. Below, we demonstrate the use of the `Qhot()` function to detect QTL hotspots using the public GRAMENE rice database. First the QTL data in the GRAMENE rice database can be loaded from [QTLEMM](#) package using the following command:

```
> load(system.file("extdata", "gramene.chr.RDATA", package = "QTLEMM"))
> load(system.file("extdata", "gramene.QTL.RDATA", package = "QTLEMM"))
> head(gramene.chr)
```

	CHR	Center.cM.	Length.cM.
1	1	74.2	184
2	2	55.5	161
3	3	84.3	166
4	4	19.7	133
5	5	51.8	121
6	6	66.8	127

```
> head(gramene.QTL) # 9 trait categories in the second column
```

	X	Trait	chr	L	R
1	1	Biochemical	1	54.1	54.1
2	2	Vigor	1	147.4	147.4
3	3	Vigor	1	147.4	147.4
4	4	Vigor	1	147.4	147.4
5	5	Vigor	1	147.4	158.6
6	6	Vigor	1	54.1	54.1

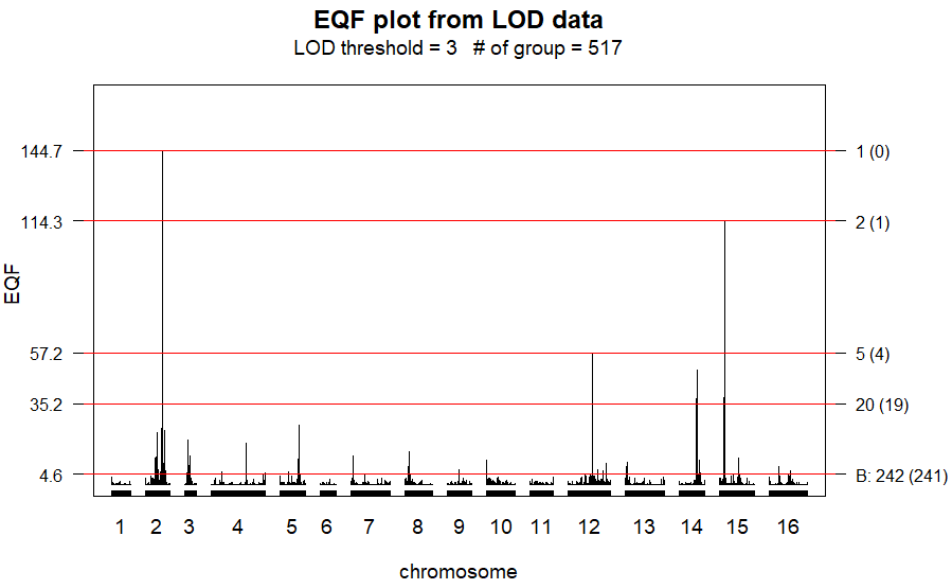


Figure 4: The 3-LOD EQF architecture of the yeast 16 chromosomes and the hotspots detected under different EQF thresholds at GWER of 5% by using the `EQF.plot()` function with the output of the `EQF.permu()` function. The EQF architecture are constructed by the uniform method with bin size of 0.5 cM.

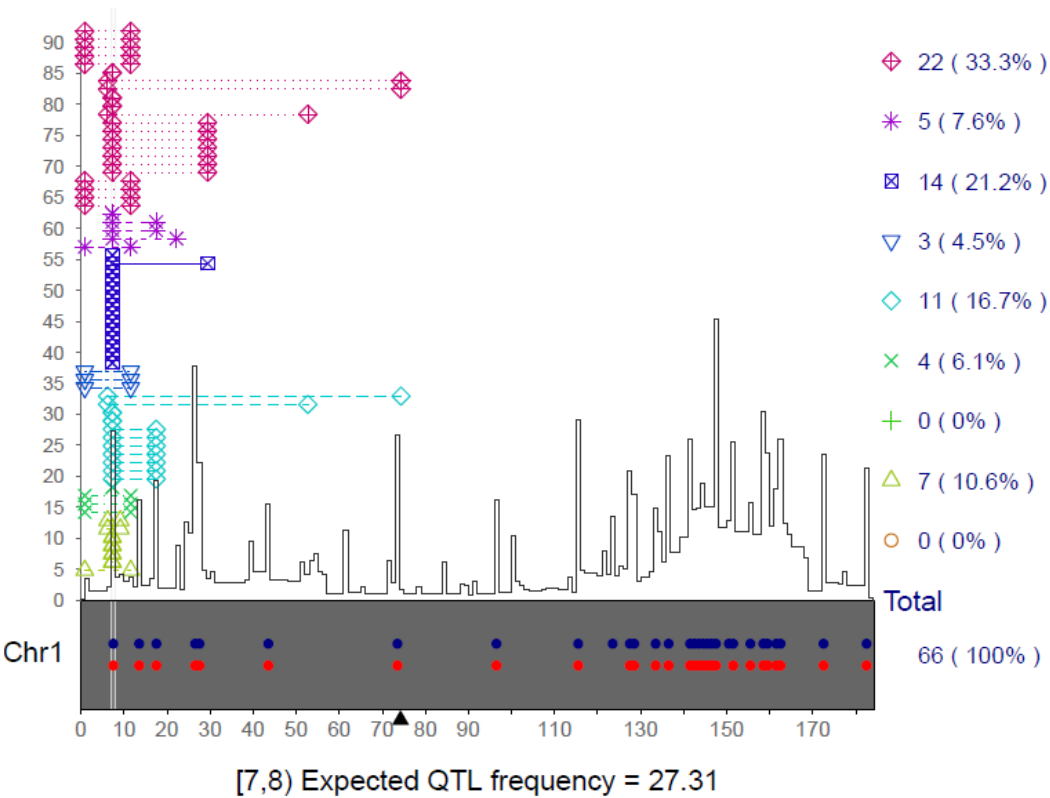


Figure 5: The plot of EQF architecture of the 1st chromosome and breakdown of QTL composition at bin [7,8) in the PDF file produced by using the `Qhot()` function with `save.pdf=TRUE`. The x-axis denotes the 1st chromosome, the y-axis denotes the EQF values. The black triangle denotes the position of centromere. The blue (red) dots denote the QTL hotspots detected by the [Yang et al. \(2019\)](#) method (the Q method). The 9 different colored symbols denote the QTLs responsible for the 9 different trait categories (see [Yang et al., 2019](#)). The dotted lines denote the lengths of the marker intervals flanking the QTLs (QTL intervals). In total, 66 QTLs contribute probabilities to the EQF value of 27.31 at bin [7,8). The numbers of contributive QTLs of the 9 different trait categories are 22, 5, 14, 3, 11, 4, 0, 7, and 0, respectively.

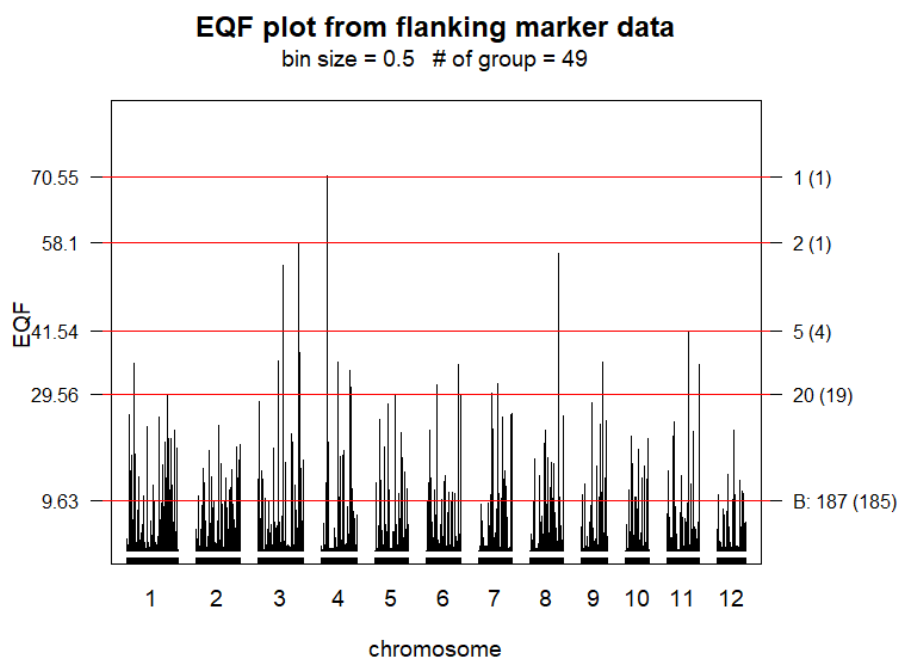


Figure 6: The EQF architectures of the rice 12 chromosomes and the hotspots detected under different EQF thresholds at GWER of 5% by using the `EQF.plot()` function with the output of the `Qhot.EQF()` function. The EQF architecture are constructed by the uniform method with bin size of 0.5 cM.

The `gramene.chr` is a data frame containing the information about the chromosomes, including their numbers, midpoint positions (in cM), and lengths. The `gramene.QTL` is a data frame for the information about QTLs, including their serial numbers, trait names, the chromosomes on which they are located, and positions of their flanking markers (in cM). Then the `Qhot()` function can utilize the information about chromosomes and QTLs to detect the QTL hotspots and output the analysis results. Below are the codes.

```
> library(QTLEMM)
> Qhot.result <- Qhot(gramene.QTL, gramene.chr, save.pdf = TRUE)
> names(Qhot.result)
[1] "EQF" "P.threshold" "Q.threshold" "nHot"
```

The analysis results include the EQF values at every bin of chromosomes (EQF), EQF thresholds obtained by the citeauthorYang-2019 method (`P.threshold`), EQF thresholds obtained using the Q method (`Q.threshold`), and the numbers of detected hotspots in each chromosome by the citeauthorYang-2019 method and Q method (`nHot`). The `save.pdf=T` command is to generate a PDF file that contains the plots of QTL composition at every bin. Figure 5 shows the plot of QTL composition at bin [7,8) of the first chromosome. It outlines the EQF architecture of the 1st chromosome, the QTL intervals, and the composition of QTLs responsible for different traits in the hotspot at bin [7,8). The `Qhot.EQF()` function is designed to draw the EQF architecture of the genome. The inputs of the `Qhot.EQF()` function include the information about chromosomes, QTLs and traits. Below are the codes of the `Qhot.EQF()` and `EQF.plot()` functions to draw the EQF architecture of the rice genome.

```
> load(system.file("extdata", "gramene.trait.RDATA", package = "QTLEMM"))
> head(gramene.trait) # 236 traits in the second column
  X      Trait chr    L    R
1 1 leaf nitrogen content  1  54.1  54.1
2 2      root dry weight  1 147.4 147.4
3 3      root dry weight  1 147.4 147.4
4 4      tiller number   1 147.4 147.4
5 5      tiller number   1 147.4 158.6
```



```

6 6          root number    1  54.1  54.1

> set.seed(8000)
> Qhot.EQF.result <- Qhot.EQF(gramene.trait, gramene.chr[,3], bin.size = 0.5,
  permu = TRUE, npermu = 1000, alpha = 0.05, Q = TRUE, console = FALSE)
> names(Qhot.EQF.result)
[1] "EQF.matrix" "bin" "bin.size" "EQF.trait" "EQF.detect" "EQF.nondetect"
[7] "cluster.matrix" "permu.matrix.cluster" "permu.matrix.Q" "EQF.threshold"

> EQF.plot(Qhot.EQF.result, plot.all = TRUE, plot.chr = FALSE)

```

The `EQF.plot()` function uses the output of the `Qhot.EQF()` function to draw the figure of the EQF architecture (see Figure 6).

5 Conclusion and discussion

In this paper we introduce the R package called **QTLEMM** for the analysis of QTL mapping and QTL hotspot detection, and attempt to provide a comprehensive overview of the functions in the package by analyzing the examples of both simulated and real data sets. The package offers several novel features and advantages:

- **QTLEMM** is designed to accommodate a wide range of experimental populations, including backcross, F_2 , AI F_t , RI F_t , IRI and immortalized F_2 populations. This versatility enables comprehensive QTL mapping analysis across different genetic backgrounds and breeding designs.
- Users can employ single-QTL or multiple-QTL models to estimate QTL parameters. It can accommodate a host of statistical models to be fitted and compared for QTL detection. The thresholds for claiming the QTL detection can be also determined in the backcross, F_2 and more advanced AI F_t and RI F_t populations.
- **QTLEMM** is unique in providing the asymptotic variance-covariance matrix for the estimates of QTL parameters.
- **QTLEMM** is unique in being able to handle both complete genotyping and selective genotyping data from diverse experimental populations in QTL mapping analysis.
- Results from QTL mapping and hotspot detection analyses are presented through numerical and graphical outputs, facilitating interpretation and visualization of findings.

The process of QTL mapping and hotspot detection usually involves the analysis of a large number of positions along the genomes. At each position, statistical models are applied to the estimation and testing for making decision, causing the process to be often time-consuming and computationally intensive in the analysis. We attempt to reduce the computational cost and speed up the analysis by eliminating unnecessary loops in writing the functions in this package. The **QTLEMM** package offers mature, effective, and commonly used statistical methods for QTL mapping and hotspot detection in the analysis of genetic architecture of quantitative traits. We envision the **QTLEMM** package will be valuable for finding more significant results in exploring the networks among genes, QTL hotspots and quantitative traits in broad areas of biological studies.

6 Availability

- The **QTLEMM** package is freely available from the Comprehensive R Archive Network at <https://cran.r-project.org/web/packages/QTLEMM/index.html>.
- The development website is available at <https://github.com/py-chung/QTLEMM>.

7 Acknowledgements

The authors thank the associate editor and four anonymous reviewers for helpful comments on this article and the software. This study was supported partly by grant MSTC 111-2118-M-001-005 from National Science and Technology Council, Taiwan, Republic of China.

References

- R. Breitling, Y. Li, B. M. Tesson, J. Fu, C. Wu, T. Wiltshire, A. Gerrits, L. V. Bystrykh, G. De Haan, A. I. Su, et al. Genetical genomics: spotlight on QTL hotspots. *PLoS genetics*, 4(10):e1000232, 2008. doi: 10.1371/journal.pgen.1000232. URL <https://journals.plos.org/plosgenetics/article?id=10.1371/journal.pgen.1000232>. [p40, 46, 57]
- R. B. Brem, J. D. Storey, J. Whittle, and L. Kruglyak. Genetic interactions between polymorphisms that affect gene expression in yeast. *Nature*, 436(7051):701–703, 2005. doi: 10.1038/nature03865. URL <https://www.nature.com/articles/nature03865>. [p54, 57]
- K. W. Broman, H. Wu, S. Sen, and G. A. Churchill. R/qtl: QTL mapping in experimental crosses. *bioinformatics*, 19(7):889–890, 2003. doi: 10.1093/bioinformatics/btg112. URL <https://academic.oup.com/bioinformatics/article/19/7/889/197785>. [p39, 40]
- M. N. Chang, R. Wu, S. S. Wu, and G. Casella. Score statistics for mapping quantitative trait loci. *Statistical Applications in Genetics and Molecular Biology*, 8(1), 2009. doi: 10.2202/1544-6115.1386. URL <https://www.degruyter.com/document/doi/10.2202/1544-6115.1386/html>. [p43]
- F. Chardon, B. Virlon, L. Moreau, M. Falque, J. Joets, L. Decousset, A. Murigneux, and A. Charcosset. Genetic architecture of flowering time in maize as inferred from quantitative trait loci meta-analysis and synteny conservation with the rice genome. *Genetics*, 168(4):2169–2185, 2004. doi: 10.1534/genetics.104.032375. URL <https://academic.oup.com/genetics/article/168/4/2169/6059422>. [p40]
- J. Chen, L. Leach, J. Yang, F. Zhang, Q. Tao, Z. Dang, Y. Chen, and Z. Luo. A tetrasomic inheritance model and likelihood-based method for mapping quantitative trait loci in autotetraploid species. *New Phytologist*, 230(1):387–398, 2021. doi: 10.1111/nph.16413. URL <https://nph.onlinelibrary.wiley.com/doi/full/10.1111/nph.16413>. [p39, 40]
- G. A. Churchill and R. Doerge. Empirical threshold values for quantitative trait mapping. *Genetics*, 138(3):963–971, 1994. doi: 10.1093/genetics/138.3.963. URL <https://academic.oup.com/genetics/article/138/3/963/6012760>. [p40, 43]
- C. C. Cockerham. An extension of the concept of partitioning hereditary variance for analysis of covariances among relatives when epistasis is present. *Genetics*, 39(6):859, 1954. doi: 10.1093/genetics/39.6.859. URL <https://academic.oup.com/genetics/article/39/6/859/6033465>. [p41]
- D. R. Cox and D. V. Hinkley. *Theoretical statistics*. CRC Press, 1979. ISBN 0412161605, 9780412161605. URL https://www.google.com/books?hl=zh-TW&lr=&id=ppoujo-BInsC&oi=fnd&pg=PP11&dq=Theoretical+statistics&ots=yS7Q5aRcB3&sig=Cr-o-TYVTMPodX4_QHFedZEwKq0. [p43]
- A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society: series B (methodological)*, 39(1):1–22, 1977. doi: 10.1111/j.2517-6161.1977.tb01600. URL <https://academic.oup.com/jrsssb/article-abstract/39/1/1/7027539>. [p41]
- D. Falconer and F. Mackay. *Introducttion to Quantitative Genetics*. 1996. URL <https://pesquisa.bvsalud.org/portal/resource/pt/biblio-934993>. [p39]

- J. Fu, J. J. Keurentjes, H. Bouwmeester, T. America, F. W. Verstappen, J. L. Ward, M. H. Beale, R. C. De Vos, M. Dijkstra, R. A. Scheltema, et al. System-wide molecular evidence for phenotypic buffering in Arabidopsis. *Nature genetics*, 41(2):166–167, 2009. doi: 10.1038/ng.308. URL <https://www.nature.com/articles/ng.308>. [p57]
- H. Geiringer. On the probability theory of linkage in Mendelian heredity. *The Annals of Mathematical Statistics*, 15(1):25–57, 1944. doi: 10.1214/aoms/1177731313. URL <https://www.jstor.org/stable/2236210>. [p43]
- Y.-T. Guo. A study of assessing genome-wise statistical significance of QTL mapping in the advanced backcross populations. Master's thesis, National Taiwan University, Taiwan, ROC, 2011. URL <https://www.airitilibrary.com/Article/Detail/U0001-2607201123494500>. [p43, 44]
- J. Haldane and C. Waddington. Inbreeding and linkage. *Genetics*, 16(4):357, 1931. doi: 10.1093/genetics/16.4.357. URL <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1201105/>. [p43]
- C. S. Haley and S. A. Knott. A simple regression method for mapping quantitative trait loci in line crosses using flanking markers. *Heredity*, 69(4):315–324, 1992. doi: 10.1038/hdy.1992.131. URL <https://www.nature.com/articles/hdy1992131>. [p39, 40, 42]
- R. C. Jansen. Interval mapping of multiple quantitative trait loci. *Genetics*, 135(1):205–211, 1993. doi: 10.1093/genetics/135.1.205. URL <https://academic.oup.com/genetics/article/135/1/205/6011202>. [p39, 40, 42]
- C.-H. Kao. On the differences between maximum likelihood and regression interval mapping in the analysis of quantitative trait loci. *Genetics*, 156(2):855–865, 2000. doi: 10.1093/genetics/156.2.855. URL <https://academic.oup.com/genetics/article/156/2/855/6051393>. [p39, 42]
- C.-H. Kao. Multiple-interval mapping for quantitative trait loci controlling endosperm traits. *Genetics*, 167(4):1987–2002, 2004. doi: 10.1534/genetics.103.021642. URL <https://academic.oup.com/genetics/article/167/4/1987/6050433>. [p39, 42]
- C.-H. Kao. Mapping quantitative trait loci using the experimental designs of recombinant inbred populations. *Genetics*, 174(3):1373–1386, 2006. doi: 10.1534/genetics.106.056416. URL <https://academic.oup.com/genetics/article/174/3/1373/6065211>. [p39, 42]
- C.-H. Kao and H.-A. Ho. A score-statistic approach for determining threshold values in QTL mapping. *Front. Biosci. E*, 4:2670–2682, 2012. doi: 10.2741/e582. URL <https://www.imrpess.com/journal/FBE/4/8/10.2741/E582>. [p39, 40, 43, 44]
- C.-H. Kao and M.-H. Zeng. A study on the mapping of quantitative trait loci in advanced populations derived from two inbred lines. *Genetics Research*, 91(2):85–99, 2009. doi: 10.1017/S0016672309000081. URL <https://www.cambridge.org/core/journals/genetics-research/article/study-on-the-mapping-of-quantitative-trait-loci-in-advanced-populations-derived-from-two-inbred-lines/2DB5AAAA88FEC461512F094518FC4281>. [p39, 41, 42]
- C.-H. Kao and M.-H. Zeng. An investigation of the power for separating closely linked QTL in experimental populations. *Genetics Research*, 92(4):283–294, 2010. doi: 10.1017/S0016672310000273. URL <https://www.cambridge.org/core/journals/genetics-research/article/an-investigation-of-the-power-for-separating-closely-linked-qt1-in-experimental-populations/114B514E1BECBC2AEB5731094CDB12BC>. [p39, 43]
- C.-H. Kao and Z.-B. Zeng. General formulas for obtaining the MLEs and the asymptotic variance-covariance matrix in mapping quantitative trait loci when using the EM algorithm. *Biometrics*, 53(2):653–665, 1997. doi: 10.2307/2533965. URL <https://www.jstor.org/stable/2533965>. [p40, 42, 45]

- C.-H. Kao and Z.-B. Zeng. Modeling epistasis of quantitative trait loci using cockerham's model. *Genetics*, 160(3):1243–1261, 2002. doi: 10.1093/genetics/160.3.1243. URL <https://academic.oup.com/genetics/article/160/3/1243/6052511>. [p39, 41]
- C.-H. Kao, Z.-B. Zeng, and R. D. Teasdale. Multiple interval mapping for quantitative trait loci. *Genetics*, 152(3):1203–1216, 1999. doi: 10.1093/genetics/152.3.1203. URL <https://academic.oup.com/genetics/article/152/3/1203/6094249>. [p39, 40, 42]
- P. Kumar, N. Longmei, M. Choudhary, M. Gupta, B. Kumar, B. Jat, B. Bhushan, M. C. Dagla, and S. K. Aggarwal. Enhancement of nutritional quality in maize grain through QTL-based approach. *Cereal Research Communications*, 52(1):39–55, 2024. doi: 10.1007/s42976-023-00378-2. URL <https://link.springer.com/article/10.1007/s42976-023-00378-2>. [p39, 40]
- E. S. Lander and D. Botstein. Mapping mendelian factors underlying quantitative traits using RFLP linkage maps. *Genetics*, 121(1):185–199, 1989. doi: 10.1093/genetics/121.1.185. URL <https://academic.oup.com/genetics/article/121/1/185/5997927>. [p39, 40, 41, 42, 43, 45]
- R. Lebowitz, M. Soller, and J. Beckmann. Trait-based analyses for the detection of linkage between marker loci and quantitative trait loci in crosses between inbred lines. *Theoretical and Applied Genetics*, 73:556–562, 1987. doi: 10.1007/BF00289194. URL <https://link.springer.com/article/10.1007/BF00289194>. [p45]
- H.-I. Lee, H.-A. Ho, and C.-H. Kao. A new simple method for improving QTL mapping under selective genotyping. *Genetics*, 198(4):1685–1698, 2014. doi: 10.1534/genetics.114.168385. URL <https://academic.oup.com/genetics/article/198/4/1685/5935979>. [p39, 40, 42, 45, 46, 55]
- H. Li, J.-M. Ribaut, Z. Li, and J. Wang. Inclusive composite interval mapping (ICIM) for digenic epistasis of quantitative traits in biparental populations. *Theoretical and Applied Genetics*, 116:243–260, 2008. doi: 10.1007/s00122-007-0663-5. URL <https://link.springer.com/article/10.1007/s00122-007-0663-5>. [p39]
- S. Li, Q. Lu, and Y. Cui. A systems biology approach for identifying novel pathway regulators in eQTL mapping. *Journal of biopharmaceutical statistics*, 20(2):373–400, 2010. doi: 10.1080/10543400903572803. URL <https://www.tandfonline.com/doi/full/10.1080/10543400903572803>. [p40, 46]
- S. Lincoln, M. Daly, and E. Lander. Mapping genes controlling quantitative traits using mapmaker/qlt version 1.1: a tutorial and reference manual. whitehead Institute. Cambridge, Mass, 1993. URL <https://home.cc.umanitoba.ca/~psgendb/doc/mapmaker/mapmakerQTL.tutorial.pdf>. [p40]
- T. F. Mackay and R. R. Anholt. Pleiotropy, epistasis and the genetic architecture of quantitative traits. *Nature Reviews Genetics*, pages 1–19, 2024. doi: 10.1038/s41576-024-00711-3. URL <https://www.nature.com/articles/s41576-024-00711-3>. [p39, 40]
- Q.-L. Meng, C.-G. Qiang, J.-L. Li, M.-F. Geng, N.-N. Ren, Z. Cai, M.-X. Wang, Z.-H. Jiao, F.-M. Zhang, X.-J. Song, et al. Genetic architecture of ecological divergence between *Oryza rufipogon* and *Oryza nivara*. *Molecular Ecology*, page e17268, 2024. doi: 10.1111/mec.17268. URL <https://onlinelibrary.wiley.com/doi/full/10.1111/mec.17268>. [p39, 40]
- E. C. Neto, M. P. Keller, A. F. Broman, A. D. Attie, R. C. Jansen, K. W. Broman, and B. S. Yandell. Quantile-based permutation thresholds for quantitative trait loci hotspots. *Genetics*, 191(4):1355–1365, 2012. doi: 10.1534/genetics.112.139451. URL <https://academic.oup.com/genetics/article/191/4/1355/5935078>. [p40, 46, 57]
- H.-P. Piepho. A quick method for computing approximate thresholds for quantitative trait loci detection. *Genetics*, 157(1):425–432, 2001. doi: 10.1093/genetics/157.1.425. URL <https://academic.oup.com/genetics/article/157/1/425/6048348>. [p43]

- A. Rebai, B. Goffinet, and B. Mangin. Approximate thresholds of interval mapping tests for QTL detection. *Genetics*, 138(1):235–240, 1994. doi: 10.1093/genetics/138.1.235. URL <https://academic.oup.com/genetics/article/138/1/235/6012691>. [p43]
- Ś. Sen and G. A. Churchill. A statistical framework for quantitative trait mapping. *Genetics*, 159(1):371–387, 2001. doi: 10.1093/genetics/159.1.371. URL <https://academic.oup.com/genetics/article/159/1/371/6049651>. [p39, 42]
- J. Taylor and A. Verbyla. R package wgaim: QTL analysis in bi-parental populations using linear mixed models. *Journal of Statistical Software*, 40:1–18, 2011. doi: 10.18637/jss.v040.i07. URL <https://www.jstatsoft.org/article/view/v040i07/0>. [p40]
- J. Van Der Veen. Tests of non-allelic interaction and linkage for quantitative characters in generations derived from two diploid pure lines. *Genetica*, 30:201–232, 1959. doi: 10.1007/BF01535675. URL <https://link.springer.com/article/10.1007/BF01535675>. [p41]
- D. Vaughan, E. Balazs, and J. Heslop-Harrison. From crop domestication to super-domestication. *Annals of botany*, 100(5):893–901, 2007. doi: 10.1093/aob/mcm224. URL <https://academic.oup.com/aob/article/100/5/893/137414>. [p40]
- A. P. Verbyla, B. R. Cullis, and R. Thompson. The analysis of QTL by simultaneous use of the full linkage map. *Theoretical and Applied Genetics*, 116(1):95–111, 2007. doi: 10.1007/s00122-007-0650-x. URL <https://link.springer.com/article/10.1007/s00122-007-0650-x>. [p39]
- A. P. Verbyla, J. D. Taylor, and K. L. Verbyla. RWGAIM: an efficient high-dimensional random whole genome average (QTL) interval mapping approach. *Genetics Research*, 94(6):291–306, 2012. doi: 10.1017/S0016672312000493. URL <https://doi.org/10.1017/S0016672312000493>. [p40]
- J. Wang, H. Yu, X. Weng, W. Xie, C. Xu, X. Li, J. Xiao, and Q. Zhang. An expression quantitative trait loci-guided co-expression analysis for constructing regulatory network using a rice recombinant inbred line population. *Journal of experimental botany*, 65(4):1069–1079, 2014. doi: 10.1093/jxb/ert464. URL <https://academic.oup.com/jxb/article/65/4/1069/498411>. [p57]
- S. Wang. Simulation study on the methods for mapping quantitative trait loci in inbred line crosses. *Hangzhou, Zhejiang, China*, 2000. URL <https://brcwebportal.cos.ncsu.edu/zeng/Wang-Shengchu-Thesis.pdf>. [p40]
- S.-B. Wang, Y.-J. Wen, W.-L. Ren, Y.-L. Ni, J. Zhang, J.-Y. Feng, and Y.-M. Zhang. Mapping small-effect and linked quantitative trait loci for complex traits in backcross or DH populations via a multi-locus GWAS methodology. *Scientific Reports*, 6(1):29951, 2016. doi: 10.1038/srep29951. URL <https://www.nature.com/articles/srep29951>. [p39]
- B. Weir and C. C. Cockerham. *Two-locus theory in quantitative genetics*. 1977. ISBN 0-8138-1895-8. URL <https://www.cabidigitallibrary.org/doi/full/10.5555/19780134831>. [p41]
- M. A. West, K. Kim, D. J. Kliebenstein, H. Van Leeuwen, R. W. Michelmore, R. Doerge, and D. A. St. Clair. Global eQTL mapping reveals the complex genetic architecture of transcript-level variation in *Arabidopsis*. *Genetics*, 175(3):1441–1450, 2007. doi: 10.1534/genetics.106.064972. URL <https://academic.oup.com/genetics/article/175/3/1441/6061820>. [p40, 46]
- C. Wu, D. L. Delano, N. Mitro, S. V. Su, J. Janes, P. McClurg, S. Batalov, G. L. Welch, J. Zhang, A. P. Orth, et al. Gene set enrichment in eQTL data identifies novel annotations and pathway regulators. *PLoS genetics*, 4(5):e1000070, 2008. doi: 10.1371/journal.pgen.1000070. URL <https://journals.plos.org/plosgenetics/article?id=10.1371/journal.pgen.1000070>. [p40, 46]

- P.-Y. Wu, M.-H. Yang, and C.-H. Kao. A statistical framework for QTL hotspot detection. *G3*, 11(4):jkab056, 2021. doi: 10.1093/g3journal/jkab056. URL <https://academic.oup.com/g3journal/article/11/4/jkab056/6151767>. [p40, 46, 47, 57, 58]
- S. Xu and W. R. Atchley. A random model approach to interval mapping of quantitative trait loci. *Genetics*, 141(3):1189–1197, 1995. doi: 10.1093/genetics/141.3.1189. URL <https://academic.oup.com/genetics/article/141/3/1189/6013542>. [p39]
- S. Xu and C. Vogl. Maximum likelihood analysis of quantitative trait loci under selective genotyping. *Heredity*, 84(5):525–537, 2000. doi: 10.1046/j.1365-2540.2000.00653. URL <https://www.nature.com/articles/6886530>. [p45, 46]
- J. Yang, C. Hu, H. Hu, R. Yu, Z. Xia, X. Ye, and J. Zhu. QTLNetwork: mapping and visualizing genetic architecture of complex traits in experimental populations. *Bioinformatics*, 24(5): 721–723, 2008. doi: 10.1093/bioinformatics/btm494. URL <https://academic.oup.com/bioinformatics/article-abstract/24/5/721/200448>. [p40]
- M.-H. Yang, D.-H. Wu, and C.-H. Kao. A statistical procedure for genome-wide detection of QTL hotspots using public databases with application to rice. *G3: Genes, Genomes, Genetics*, 9(2):439–452, 2019. doi: 10.1534/g3.118.200922. URL <https://academic.oup.com/g3journal/article/9/2/439/6026674>. [p40, 46, 47, 57, 60]
- Z.-B. Zeng. Theoretical basis for separation of multiple linked gene effects in mapping quantitative trait loci. *Proceedings of the National Academy of Sciences*, 90(23):10972–10976, 1993. doi: 10.1073/pnas.90.23.10972. URL [https://www.pnas.org/doi/10.1073/pnas.90.23.10972](https://www.pnas.org/doi/10.1073/pnas.90.23.10972?doi=10.1073/pnas.90.23.10972). [p39]
- Z.-B. Zeng. Precision mapping of quantitative trait loci. *Genetics*, 136(4):1457–1468, 1994. doi: 10.1093/genetics/136.4.1457. URL <https://academic.oup.com/genetics/article/136/4/1457/6012232>. [p39, 40, 42]
- Y.-W. Zhang, Y.-J. Wen, J. M. Dunwell, and Y.-M. Zhang. QTL. gCIMapping. GUI v2.0: An R software for detecting small-effect and linked QTLs for quantitative traits in bi-parental segregation populations. *Computational and structural biotechnology journal*, 18:59–65, 2020. doi: 10.1016/j.csbj.2019.11.005. URL <https://www.sciencedirect.com/science/article/pii/S2001037019302971>. [p40]
- G. Zou. A modified poisson regression approach to prospective studies with binary data. *American journal of epidemiology*, 159(7):702–706, 2004. doi: 10.1093/aje/kwh090. URL <https://academic.oup.com/aje/article/159/7/702/71883>. [p43]

Ping-Yuan Chung

Institute of Statistical Science, Academia Sinica
Taipei 11529, Taiwan, Republic of China
pychung@stat.sinica.edu.tw

You-Tsz Guo

Institute of Statistical Science, Academia Sinica
Taipei 11529, Taiwan, Republic of China
yowyow220@hotmail.com

Hsiang-An Ho

Institute of Statistical Science, Academia Sinica
Taipei 11529, Taiwan, Republic of China
r92221020@ntu.edu.tw

Hsin-I Lee

Institute of Statistical Science, Academia Sinica

Taipei 11529, Taiwan, Republic of China
hilee@stat.sinica.edu.tw

Po-Ya Wu
Institute of Statistical Science, Academia Sinica
Taipei 11529, Taiwan, Republic of China
debbywu@stat.sinica.edu.tw

Man-Hsia Yang
Crop Science Division, Taiwan Agricultural Research Institute, Council of Agriculture
Taichung 41362, Taiwan, Republic of China
ymh@tari.gov.tw

Miao-Hui Zeng
Institute of Statistical Science, Academia Sinica
Taipei 11529, Taiwan, Republic of China
miaomiao@stat.sinica.edu.tw

Chen-Hung Kao
Institute of Statistical Science, Academia Sinica
Taipei 11529, Taiwan, Republic of China
chkao@stat.sinica.edu.tw

hdbayes: An R Package for Bayesian Analysis of Generalized Linear Models Using Historical Data

by Ethan M. Alt¹, Xinxin Chen*, Luiz M. Carvalho and Joseph G. Ibrahim

Abstract There has been increased interest in the use of historical data to formulate informative priors in regression models. While many such priors for incorporating historical data have been proposed, adoption is limited due to access to software. Where software does exist, the implementations between different methods could be vastly different, making comparisons between methods difficult. In this paper, we introduce the R package *hdbayes*, an implementation of the power prior, normalized power prior, Bayesian hierarchical model, robust meta-analytic prior, commensurate prior, and latent exchangeability prior for generalized linear models. The bulk of the package is written in the Stan programming language, with user-friendly R wrapper functions to call samplers.

1 Introduction

A common occurrence in practical applications of Bayesian modeling is the use of historical data to inform the prior for the analysis of the data set at hand, which we refer to as the current data. For example, in clinical trials, one may be in possession of a Phase III data set in addition to the Phase II trial already completed. In such cases, it is often desirable to elicit an informative prior for the regression coefficients based on the historical data.

Many methods for incorporating historical data have been proposed, including the power prior (Ibrahim and Chen, 2000), the normalized power prior (Duan et al., 2006; Carvalho and Ibrahim, 2021), the Bayesian hierarchical model, commensurate priors (Hobbs et al., 2012), and robust meta-analytic predictive priors (Schmidli et al., 2014). Unfortunately, software implementations of these methods may be difficult to come across. Where they do exist, the implementations can differ so substantially that it becomes troublesome to utilize multiple packages to compare methods. For example, the *RBesT* package (Weber et al., 2021), which implements the robust meta-analytic predictive prior, accepts data in an aggregate format where each row represents a study or group and uses a single function that accommodates Gaussian, binomial, and Poisson models. In contrast, the *NPP* package (Han et al., 2021), which implements the normalized power prior, provides separate functions for different outcome types, each requiring a distinct data format. For instance, `NPP : LMNPP_MCMC()` requires users to provide vectors of individual-level responses for the current and historical data, whereas `NPP : BerNPP_MCMC()` requires a vector of two elements (number of trials and number of successes) for each data set. Furthermore, many existing implementations rely on Metropolis-type samplers, which can be difficult to tune.

The R package *hdbayes* aims to fill this gap. In particular, *hdbayes* offers user-friendly R functions to conduct Bayesian analysis for generalized linear models (GLMs) using historical data, with consistent syntax across methods. The backbone of the package is written in the Stan programming language (Carpenter et al., 2017), implemented in the *cmdstanr* package (Gabry et al., 2024). Although *cmdstanr* is not available on CRAN, *hdbayes* uses the *instantiate* package (Landau, 2024), which depends on *cmdstanr*, to create pre-compiled code. Stan utilizes a highly efficient Markov chain Monte Carlo (MCMC) method known as Hamiltonian Monte Carlo (HMC), which requires little-to-no tuning from the user's perspective. In particular, Stan implements a highly optimized variant of the No U-Turn Sampler (NUTS) algorithm (Hoffman et al., 2014).

¹The first two authors contributed equally to this manuscript.

1.1 1.1 Installation

The package **hdbayes** is available on CRAN. By default, R installs binary builds on Windows and macOS. In these cases, the Stan models included with the package are not compiled during installation, and users may encounter a “model not compiled” error when using **hdbayes**. To ensure compilation, we recommend installation from source:

```
install.packages("hdbayes", type = "source")
```

Using **hdbayes** also requires the package **cmdstanr** and CmdStan (the command-line interface to Stan). Detailed instructions for installing both are provided in the **cmdstanr** documentation.

The remainder of this paper proceeds as follows. In Section 2, we review methods for prior elicitation using historical data that are implemented in the **hdbayes** package, indicating any existing publicly available implementations for each prior. We provide methodology and code examples for model selection via marginal likelihoods in Section 3. In Section 4, we illustrate the utility of our package via analyses of real data sets in AIDS clinical trials, comparing posterior results across all implemented priors in **hdbayes**. We close with some discussion in Section 5.

2 Prior elicitation with historical data

In this section, we review the priors implemented in the **hdbayes** package. Where applicable, we also discuss existing software implementations for each prior. We focus on prior elicitation for generalized linear models (GLMs) (McCullagh and Nelder, 1989), whose likelihood function is given by

$$L(\boldsymbol{\beta}, \phi | \mathbf{y}, \mathbf{X}) \propto \prod_{i=1}^n \exp \left\{ \frac{1}{a_i(\phi)} [y_i \theta_i - b(\theta_i)] + c(y_i, \phi) \right\}, \quad (1)$$

where $\theta_i = \theta(\mathbf{x}_i' \boldsymbol{\beta})$, $\theta(\cdot)$ is referred to as the θ -link function, and $b(\cdot)$ and $c(\cdot, \phi)$ are determined by the underlying probability distribution. The vector $\boldsymbol{\beta}$ is a p -dimensional vector of regression coefficients corresponding to the $p \times 1$ vector of covariates $\mathbf{x}_i$ (with $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)'$), both of which may include an intercept. The response variable is denoted by y_i (and $\mathbf{y} = (y_1, \dots, y_n)'$), and $\phi > 0$ is a dispersion parameter, which is known and equal to 1 for binomial and Poisson models. For ease of exposition, we assume $a_i(\phi) = \phi$. Note that GLMs are sometimes parameterized in terms of the mean of the i^{th} observation, μ_i , using the μ -link function $g(\mu_i) = \mathbf{x}_i' \boldsymbol{\beta}$. In this case, $\theta(\cdot) = (\hat{b}^{-1} \circ g^{-1})(\cdot)$, where $\hat{f}$ denotes the first derivative of the function f . When $a_i(\phi) = \phi$, the likelihood in (1) can be expressed in matrix form as

$$L(\boldsymbol{\beta}, \phi | \mathbf{y}, \mathbf{X}) \propto \exp \left\{ \frac{1}{\phi} [\mathbf{y}' \theta(\mathbf{X} \boldsymbol{\beta}) - \mathbf{1}_n' b(\theta(\mathbf{X} \boldsymbol{\beta}))] + \mathbf{1}_n' c(\mathbf{y}, \phi) \right\}, \quad (2)$$

where $\mathbf{1}_n = (1, \dots, 1)'$ is an $n \times 1$ vector of ones, and the functions $\theta(\cdot)$, $b(\cdot)$, and $c(\cdot, \phi)$ are evaluated elementwise.

Let the current data set be denoted by $D = \{(y_i, \mathbf{x}_i) : i = 1, \dots, n\}$, where n is the sample size of the current data. Suppose we have H historical data sets, with the h^{th} historical data set denoted by $D_{0h} = \{(y_{0hi}, \mathbf{x}_{0hi}) : i = 1, \dots, n_{0h}\}$ for $h = 1, \dots, H$, where n_{0h} is the corresponding sample size. Let $D_0 = \{D_{01}, \dots, D_{0H}\}$ denote the collection of all historical data sets. In what follows, we describe the priors implemented in **hdbayes** and compare them with several other R packages that provide related functionality for incorporating historical data.

2.1 Basic syntax

The **hdbayes** package is designed to enable users familiar with the R programming language (R Core Team, 2024), particularly its `glm()` function in the **stats** package, to apply historical data borrowing priors in a unified and user-friendly manner. To this end, **hdbayes** provides a basic syntax that is common across all implemented priors. In particular, each prior takes the form `glm.prior(formula, family, data.list, prior.args, ...)`, where `formula` is a two-sided formula object, `family` is a family object containing a distribution-link function pair, and `data.list` is a list of data.frames, with the first element corresponding to the current data set and the remaining elements treated as historical data sets. The argument `prior.args` serves as a placeholder for prior-specific arguments (e.g., hyperparameters), and the ellipsis (...) passes additional arguments to the sampler in **cmdstanr** (e.g., number of chains, warm-up iterations, etc.). Note that the first two arguments of `glm.prior()` are identical to those of `stats::glm()`, while the third argument differs: `glm.prior()` takes a list of data frames, whereas `stats::glm()` uses a single data frame. Each implemented prior in **hdbayes** also provides sensible default values for `prior.args`, making it easy for users to apply the methods. A summary of the functionality of **hdbayes** compared to other packages is presented in Table 1. Our comparison is restricted to packages that implement GLMs, although some of them also support survival analysis functionality.

	Package Name				
	hdbayes	NPP	BayesPPD	psborrow2	RBest
Power prior	X				
Normalized power prior	X	X	X		
Normalized asymptotic power prior	X				
Robust meta-analytic predictive prior	X				X
Bayesian hierarchical model	X				X
Commensurate prior	X			X	
LEAP	X				
All models in <code>stats::glm()</code>	X				
> 1 historical data set	X	X	X		X
Marginal likelihood calculation	X				
Survival capabilities	X			X	

Table 1: Features present in various R packages providing support for prior elicitation on the basis of historical data.

2.2 Power prior

The power prior (PP) of Ibrahim and Chen (2000), developed for settings with a single historical data set, involves discounting the likelihood of the historical data by a value $a_{01} \in [0, 1]$ (often referred to as the *discounting parameter*) along with eliciting an initial prior π_0 . We may express this mathematically as

$$\pi_{\text{PP}}(\boldsymbol{\beta}, \phi | D_{01}, a_{01}, \pi_0) = \frac{L(\boldsymbol{\beta}, \phi | D_{01})^{a_{01}} \pi_0(\boldsymbol{\beta}, \phi)}{Z(a_{01})} \propto L(\boldsymbol{\beta}, \phi | D_{01})^{a_{01}} \pi_0(\boldsymbol{\beta}, \phi), \quad (3)$$

where $Z(a_{01}) = \int_{\mathbb{R}^p} \int_0^\infty L(\boldsymbol{\beta}, \phi | D_{01})^{a_{01}} \pi_0(\boldsymbol{\beta}, \phi) d\phi d\boldsymbol{\beta}$ is a normalizing constant, whose exact value is unimportant when a_{01} is fixed.

For fixed a_{01} , the effective sample size of the PP (i.e., the number of observations that the prior is “worth”) is given by $a_{01}n_{01}$, which is easy to compute. The initial prior π_0 is typically chosen to be non-informative, since the goal is for the prior to be primarily informed by the historical data. When $a_{01} = 0$, the PP reduces to the initial prior, and when $a_{01} = 1$ the PP is the posterior of the historical data. The PP thus provides a flexible way to incorporate historical information and quantify the informativeness of the prior. Ibrahim et al. (2015)

provides an overview of how to select a_{01} . In general, it is recommended to try several values of a_{01} to see how sensitive the posterior is to the choice of a_{01} .

In **hdbayes**, we extend the traditional PP to accommodate multiple historical data sets by allowing users to specify a vector of discounting parameters $\mathbf{a}_0 = (a_{01}, \dots, a_{0H}) \in [0, 1]^H$. Mathematically, we may express this PP as

$$\pi_{\text{PP}}(\boldsymbol{\beta}, \phi | D_0, \mathbf{a}_0, \pi_0) \propto \left[\prod_{h=1}^H L(\boldsymbol{\beta}, \phi | D_{0h})^{a_{0h}} \right] \pi_0(\boldsymbol{\beta}, \phi), \quad (4)$$

where the initial prior is specified as

$$\begin{aligned} \beta_j &\sim N(\mu_{0j}, \sigma_{0j}^2) \text{ for } j = 1, \dots, p, \\ \phi &\sim N^+(\alpha_0, \gamma_0^2), \end{aligned} \quad (5)$$

and $N^+(\mu, \sigma^2)$ denotes a normal distribution with mean μ and variance σ^2 , truncated from below at zero. The half-normal distribution is the special case when $\mu = 0$. This specification assumes independence between $\boldsymbol{\beta}$ and ϕ in the initial prior, while dependence is induced through the historical data whenever at least one $a_{0h} > 0$.

The hyperparameters $\boldsymbol{\mu}_0 = (\mu_{01}, \dots, \mu_{0p})'$, $\boldsymbol{\sigma}_0 = (\sigma_{01}, \dots, \sigma_{0p})'$, α_0 , and γ_0 can be elicited by the user, though **hdbayes** provides non-informative defaults. In particular, the defaults are, $\boldsymbol{\mu}_0 = \mathbf{0}_p$, $\boldsymbol{\sigma}_0 = 10 \cdot \mathbf{1}_p$, $\alpha_0 = 0$, and $\gamma_0 = 10$, where $\mathbf{0}_q$ denotes the q -dimensional vector of zeros. This corresponds to independent normal initial priors for the components of $\boldsymbol{\beta}$ with mean 0 and variance 100, and a half-normal initial prior for ϕ , i.e., $\pi_0(\phi) \propto \varphi(\phi|0, 100) \cdot 1\{\phi > 0\}$, where $\varphi(\cdot|\mu, \sigma^2)$ denotes the normal density with mean μ and standard deviation σ , and $1\{A\}$ is the indicator function taking value 1 if A is true and 0 otherwise.

The PP is also implemented in the **BayesPPD** package (Shen et al., 2023) via the function `glm.fixed.a0()`. **BayesPPD** uses Gibbs sampling where feasible (e.g., the normal linear model) and slice sampling (Neal, 2003) otherwise. Slice samplers are easier to tune than Metropolis-type samplers, but it is difficult to implement multivariate versions of slice samplers. As a result, most practical implementations conduct slice sampling on the full conditional distributions. Unfortunately, these samplers can be slow to converge and exhibit poor mixing in high-dimensional or strongly correlated settings (Neal, 2003; Murray et al., 2010; Bloem-Reddy and Cunningham, 2016).

The **BayesPPD** implementation supports binomial models with the number of trials exceeding 1, which is not currently supported by the **hdbayes** implementation `glm.pp()` (although one could always de-collapse the data). Both **BayesPPD** and **hdbayes** allow for multiple historical data sets. However, **BayesPPD** does not support inverse-Gaussian or gamma outcomes and therefore does not cover all GLMs. In addition, the syntax of `BayesPPD::glm.fixed.a0()` is less user-friendly for the novice R user, as it does not utilize the formula class to construct the response variable and design matrix nor does it use the convenient family class to provide the distribution and link function. Finally, the link functions in **BayesPPD** are not as exhaustive as those offered in the `link-glm` class (e.g., the `cauchit` link is not available).

A second package by the same authors as **BayesPPD**, **BayesPPDSurv**, implements the normalized power prior for time-to-event outcomes in both the analysis and design of clinical trials, using a proportional hazards model with piecewise constant baseline hazards (referred to as the PWEPH model). However, its syntax closely follows that of **BayesPPD**. The current version of **hdbayes** does not provide functionality for trial designs but supports several survival models for data analysis, including the PWEPH model, accelerated failure time (AFT) models, and a mixture cure rate model.

2.3 Normalized power prior

The PP in (4) can be sensitive to the choice of a_0 . Because we are generally uncertain about what values a_0 should take, one way to mitigate this sensitivity is to treat a_0 as random. However, when a_0 is treated as random, a normalizing constant must be estimated; otherwise, the resulting posterior violates the likelihood principle (Duan et al., 2006; Neuen-schwander et al., 2009). This leads to the normalized power prior (NPP), which, for multiple historical data sets, is given by

$$\begin{aligned}\pi_{\text{NPP}}(\boldsymbol{\beta}, \phi, \mathbf{a}_0 | D_0, \pi_0) &= \prod_{h=1}^H \pi_{\text{PP}}(\boldsymbol{\beta}, \phi | D_{0h}, a_{0h}, \pi_0^{1/H}) \pi(a_{0h}), \\ &= \prod_{h=1}^H \frac{L(\boldsymbol{\beta}, \phi | D_{0h})^{a_{0h}} \pi_0(\boldsymbol{\beta}, \phi)^{1/H}}{Z_h(a_{0h})} \pi(a_{0h}), \\ &= \left[\prod_{h=1}^H \frac{L(\boldsymbol{\beta}, \phi | D_{0h})^{a_{0h}}}{Z_h(a_{0h})} \pi(a_{0h}) \right] \pi_0(\boldsymbol{\beta}, \phi)\end{aligned}\quad (6)$$

where $Z_h(a_{0h}) = \int_{\mathbb{R}^p} \int_0^\infty L(\boldsymbol{\beta}, \phi | D_{0h}) \pi_0(\boldsymbol{\beta}, \phi)^{1/H} d\boldsymbol{\beta} d\phi$ is a normalizing constant, $\pi(a_{0h})$ is a prior on a_{0h} (implemented as a Beta prior), and the remaining notation follows that of Section 2.2. In most cases, the function $Z_h(\cdot)$ is analytically intractable and must be estimated numerically.

The approach taken in the **hdbayes** package is a simplified version of the two-step approach described by Carvalho and Ibrahim (2021). The procedure is summarized in Algorithm 1. Bridge sampling in Algorithm 1 is conducted using the **bridgesampling** package (Gronau et al., 2020). Because bridge sampling cannot be directly implemented in Stan, a grid of discounting parameter values $\boldsymbol{\alpha} = \{0 = \alpha_1 < \alpha_2 < \dots < \alpha_T = 1\}$ and the corresponding estimated normalizing constants $\widehat{Z_h(\alpha_1)}, \dots, \widehat{Z_h(\alpha_T)}$ are passed to Stan as data. Linear interpolation is then performed within Stan to approximate $Z_h(a)$ for arbitrary $a \in [0, 1]$. This piecewise linear approximation makes gradient evaluation feasible, which is required for HMC methods.

Algorithm 1: Posterior Sampling under the Normalized Power Prior

Data: Current data D ; historical data $D_0 = \{D_{0h} : h = 1, \dots, H\}$; grid of points $\boldsymbol{\alpha} = \{0 = \alpha_1 < \alpha_2 < \dots < \alpha_T = 1\}$; number of PP MCMC samples M_h ; number of posterior samples M .

Result: M posterior samples under the NPP in (6).

```

1 for  $h \leftarrow 1$  to  $H$  do
2   for  $t \leftarrow 1$  to  $T$  do
3     Obtain  $M_h$  MCMC samples from the PP in (4) using historical data  $D_{0h}$ , initial
       prior  $\pi_0^{1/H}$ , and discounting parameter  $\alpha_t$ .
4     Conduct bridge sampling to obtain an estimate  $\widehat{Z_h(\alpha_t)}$  of  $Z_h(\alpha_t)$  using the  $M_h$ 
       samples from the prior.
5   end
6   Use linear interpolation on  $\{\widehat{Z_h(\alpha_1)}, \dots, \widehat{Z_h(\alpha_T)}\}$  to obtain a piecewise linear
       approximation  $\hat{Z}_h(\cdot)$  of  $Z_h(\cdot)$ .
7   Draw  $M$  posterior samples under the NPP by substituting  $\hat{Z}_h(\cdot)$  for  $Z_h(\cdot)$  in (6).
8 end
```

The function `glm.npp.lognc()` in the **hdbayes** package estimates the logarithm of the normalizing constant for a single value α_t of the discounting parameter, with syntax similar to `stats::glm()` function. Specifically, sampling from (4) is conducted via **cmdstanr** and then the logarithm of the normalizing constant, $\log Z_h(\alpha_t)$ is estimated using the **bridge-sampling** package. We note that one may utilize parallel computing in order to obtain the estimated normalizing constants faster by using the **parallel** package, which is included

with base R. After estimating the normalizing constants $Z_h(\alpha_t)$ over a grid of discounting parameter values, posterior samples under the NPP (6) can be obtained using the `glm.npp()` function. The syntax for this function is nearly identical to that of `glm.pp()` described in Section 2.2, except that users must supply values for the arguments `a0.lognc` (the grid points) and `lognc` (the estimated logarithm of the normalizing constants) obtained as described above.

The package **NPP** offers an implementation of the NPP for GLMs. **NPP** uses independence or random-walk Metropolis-Hastings proposals for posterior sampling of the discounting parameters, which are fast but can be difficult to tune, often resulting in highly correlated samples (Roberts and Rosenthal, 2001). Moreover, the **NPP** package uses a Laplace approximation (Tierney and Kadane, 1986) to estimate the normalizing constant, which is faster than the bridge sampling approach in **hdbayes** but can be inaccurate in small-sample or high-dimensional settings (Shun and McCullagh, 1995). Finally, each distribution in the exponential family corresponds to a separate function in **NPP**, making the syntax less streamlined and somewhat cumbersome to use.

The aforementioned **BayesPPD** package also offers an implementation of the NPP. The approach is a two-step approach similar to the algorithm above. The main difference between **BayesPPD** and **hdbayes** besides those mentioned in Section 2.2 is that **BayesPPD** conducts slice sampling to sample from the prior density (4), while **hdbayes** relies on HMC.

It is worth noting that under a conjugate (multivariate normal-inverse-gamma) initial prior, the normalizing constant of the PP for the normal linear model is known and does not need to be estimated. When users call `glm.npp()` with `family = gaussian('identity')`, the function requires them to input a grid of estimated normalizing constants. Alternatively, **hdbayes** provides the `lm.npp()` function to sample from the posterior of a normal linear model under the NPP, which is a one-step approach that does *not* require estimation of the normalizing constant before posterior sampling.

2.4 Normalized asymptotic power prior

Ibrahim and Chen (2000) showed that, under large samples of the historical data set, the PP in (3) converges to a multivariate normal density, i.e.,

$$\pi_{PP}(\beta, \phi | D_{0h}, a_{0h}, \pi_0) \xrightarrow{n_{0h} \rightarrow \infty} \varphi \left(\beta, \phi \left| (\hat{\beta}'_{0h}, \hat{\phi}_{0h})', a_{0h}^{-1} \left[I(\hat{\beta}_{0h}, \hat{\phi}_{0h} | D_{0h}) \right]^{-1} \right. \right), \quad (7)$$

where $\varphi(\cdot | \mu, \Sigma)$ is the multivariate normal density function with mean μ and covariance matrix Σ . Here, $\hat{\beta}_{0h}$ and $\hat{\phi}_{0h}$ are the maximum likelihood estimates (MLEs) of (β, ϕ) under the historical data D_{0h} , and $I(\cdot | D_{0h})$ is the Fisher information matrix (i.e., the expectation of the negative Hessian matrix of the log-likelihood) based on the GLM likelihood for historical data D_{0h} . The right-hand side of (7) has been referred to as the “asymptotic power prior” (Ibrahim et al., 2015).

Because the right-hand side of (7) is properly normalized, there is no need to estimate a normalizing constant if we treat a_{0h} as random. However, since the dispersion parameter ϕ is restricted to be positive, the normal approximation may require a large historical sample size to perform well due to potential skewness. To improve the approximation, we take the transformation $\tau = \log \phi$. By the invariance property of MLEs, $\hat{\tau}_{0h} = \log \hat{\phi}_{0h}$, and the Jacobian matrix of this transformation is given by

$$J(\beta, \tau) = \begin{pmatrix} \frac{\partial \beta}{\partial \beta'} & \frac{\partial \beta}{\partial \tau} \\ \frac{\partial \phi}{\partial \beta'} & \frac{\partial \phi}{\partial \tau} \end{pmatrix} = \begin{pmatrix} I & \mathbf{0}_p \\ \mathbf{0}'_p & \exp\{\tau\} \end{pmatrix},$$

so that the Fisher information for historical data D_{0h} is $I(\beta, \tau) = J(\beta, \tau)' I(\beta, \exp\{\tau\} | D_{0h}) J(\beta, \tau)$.

The **hdbayes** package offers an implementation of what we call the normalized asymptotic power prior (NAPP) using this transformation $\tau = \log \phi$. Let $\theta = (\beta', \tau)'$ and

$\hat{\theta}_{0h} = (\hat{\beta}'_{0h}, \hat{\tau}_{0h})'$. The NAPP is given by

$$\pi_{\text{NAPP}}(\theta, a_0 | D_0) = \prod_{h=1}^H \varphi \left(\theta \mid \hat{\theta}_{0h}, a_{0h}^{-1} [I(\hat{\theta}_{0h} | D_{0h})]^{-1} \right) \pi(a_{0h}),$$

where we take an independent Beta prior for each a_{0h} , $h = 1, \dots, H$. The primary advantage of the NAPP over the NPP is that there is no need to estimate a normalizing constant, so that the implementation is a one-step approach. However, when a normal distribution is a poor approximation to the likelihood function, the NAPP might be overly informative. Posterior samples under the NAPP can be obtained using the `glm.napp()` function, which only requires a formula, family, and a list of data.frames giving the current and historical data sets. To our knowledge, there are no other R packages that implement the NAPP.

2.5 Bayesian hierarchical model

The Bayesian hierarchical model (BHM) is arguably the most widely used Bayesian model for informative prior elicitation. The BHM assumes that the parameters for the current and historical data sets are different but come from the same distribution whose hyperparameters themselves are treated as random.

Let $(\beta', \phi)'$ denote the GLM parameters for the current data set, where $\beta = (\beta_1, \dots, \beta_p)'$. Let $(\beta'_{0h}, \phi_{0h})'$ denote the GLM parameters for the historical data set D_{0h} , $h = 1, \dots, H$, where $\beta_{0h} = (\beta_{0h1}, \dots, \beta_{0hp})'$. The BHM as implemented in `hdbayes` may be expressed hierarchically as

$$\begin{aligned} \mu_j | \mu_{0j}, \sigma_{0j} &\sim N(\mu_{0j}, \sigma_{0j}^2), \quad j = 1, \dots, p, \\ \sigma_j | m_j, s_j &\sim N^+(m_j, s_j^2), \quad j = 1, \dots, p, \\ \beta_j, \beta_{0hj} | \mu_j, \sigma_j &\stackrel{\text{i.i.d.}}{\sim} N(\mu_j, \sigma_j^2), \quad h = 1, \dots, H, \quad j = 1, \dots, p, \\ \phi | m_0, s_0 &\sim N^+(m_0, s_0^2), \\ \phi_{0h} | m_{0h}, s_{0h} &\sim N^+(m_{0h}, s_{0h}^2), \quad h = 1, \dots, H, \\ y_i | \beta, \phi &\sim f(y_i | \beta, \phi), \quad i = 1, \dots, n, \\ y_{0hi} | \beta_{0h}, \phi_{0h} &\sim f(y_{0hi} | \beta_{0h}, \phi_{0h}), \quad h = 1, \dots, H, \quad i = 1, \dots, n_{0h}, \end{aligned} \quad (8)$$

where $f(\cdot | \beta, \phi)$ is the density (or mass) function corresponding to the GLM likelihood in (1). Here, μ_j is referred to as the global (or meta-analytic) mean, σ_j is the global standard deviation (which measures the heterogeneity of parameters across data sets), and $\xi = (m_0, s_0, \{(m_{0h}, s_{0h}) : h = 1, \dots, H\}, \{(\mu_{0j}, \sigma_{0j}, m_j, s_j) : j = 1, \dots, p\})$ denotes the collection of elicited hyperparameters. The hyperparameters s_j are the most crucial ones, as they must often be chosen to be somewhat subjective and reflect most of the borrowing properties under the BHM.

Let θ denote all parameters to be sampled in (8). The posterior density of (8) can be expressed as

$$p_{\text{BHM}}(\theta | D, D_0, \xi) \propto L(\beta, \phi | D) \left[\prod_{h=1}^H L(\beta_{0h}, \phi_{0h} | D_{0h}) \right] \pi_{\text{BHM}}(\theta), \quad (9)$$

where

$$\begin{aligned} \pi_{\text{BHM}}(\theta) &= \varphi^+(\phi | m_0, s_0^2) \left[\prod_{h=1}^H \varphi^+(\phi_{0h} | m_{0h}, s_{0h}^2) \right] \cdot \\ &\quad \left\{ \prod_{j=1}^p \left[\varphi(\mu_j | \mu_{0j}, \sigma_{0j}^2) \varphi^+(\sigma_j | m_j, s_j^2) \varphi(\beta_j | \mu_j, \sigma_j^2) \prod_{h=1}^H \varphi(\beta_{0hj} | \mu_j, \sigma_j^2) \right] \right\}, \end{aligned} \quad (10)$$

and $\varphi^+(\cdot|a, b^2)$ denotes the density of a truncated normal distribution $N^+(a, b^2)$.

In **hdbayes**, posterior inference under the BHM can be conducted using the `glm.bhm()` function. The default hyperparameters values are $\mu_{0j} = 0$, $\sigma_{0j} = 10$, $m_j = 0$, $s_j = 1$, $m_0 = m_{0h} = 0$, $s_0 = s_{0h} = 10$ for $j = 1, \dots, p$ and $h = 1, \dots, H$. The value $s_j = 1$ is moderately informative but is designed to encourage information borrowing. In general, s_j cannot be chosen to be large unless there are many (e.g. ≥ 10) historical data sets, which is rarely the case.

Several R packages provide related functionality for hierarchical modeling. The R package **historicalborrow** implements hierarchical models for borrowing information from historical studies. However, it assumes continuous (normal) outcomes and focuses on borrowing information only on the mean outcome of the control arm, whereas **hdbayes** supports a broad class of GLMs and enables leveraging information on both control and treatment arms as well as on covariate effects. A BHM similar to that implemented in **hdbayes** can also be fitted using the `gMAP()` function in the **RBesT** package. Although the documentation and vignettes of **RBesT** primarily illustrate `gMAP()` for deriving a prior from historical data, the same function can, in principle, be used to obtain posterior inference from a BHM by including the current trial as an additional study within the hierarchical model. **RBesT** supports GLMs for normal, binomial, and Poisson outcomes, but it is designed for aggregate (study-level) data and does not directly accommodate covariate adjustment. In contrast, **hdbayes** accommodates individual-level data and naturally incorporates covariates as in `stats::glm()` function.

2.6 Robust meta-analytic predictive prior

The *robust* meta-analytic predictive (RMAP) prior (Schmidli et al., 2014) extends the BHM framework introduced in Section 2.5 by incorporating a vague (non-informative) component to mitigate prior-data conflict. As shown by Schmidli et al. (2014), the BHM in (9) induces a prior for the regression coefficients of the current study, referred to as the meta-analytic predictive (MAP) prior, which is given by

$$\pi_{\text{MAP}}(\beta|D_0) = \int \int \left[\prod_{j=1}^p \varphi(\beta_j|\mu_j, \sigma_j^2) \right] \pi(\mu, \sigma|D_0) d\mu d\sigma, \quad (11)$$

where $\mu = (\mu_1, \dots, \mu_p)$, $\sigma = (\sigma_1, \dots, \sigma_p)$, and $\pi(\mu, \sigma|D_0)$ is the posterior density of (μ, σ) obtained by fitting a BHM to the historical data only, i.e.,

$$\begin{aligned} \pi(\mu, \sigma|D_0) &= \int \int p_{\text{BHM}}(\mu, \sigma, \beta_0, \phi_0|D_0) d\beta_0 d\phi_0, \\ &\propto \int \int \left\{ \prod_{h=1}^H L(\beta_{0h}, \phi_{0h}|D_{0h}) \varphi^+(\phi_{0h}|m_{0h}, s_{0h}^2) \prod_{j=1}^p \varphi(\beta_{0hj}|\mu_j, \sigma_j^2) \right\} \\ &\quad \cdot \left\{ \prod_{j=1}^p \varphi(\mu_j|\mu_{0j}, \sigma_{0j}^2) \varphi^+(\sigma_j|m_j, s_j^2) \right\} d\beta_0 d\phi_0, \end{aligned} \quad (12)$$

where $p_{\text{BHM}}(\cdot|D_0)$ denotes the posterior density of the BHM for the historical data given in (9), $\beta_0 = (\beta'_{01}, \dots, \beta'_{0H})'$, and $\phi_0 = (\phi_{01}, \dots, \phi_{0H})'$.

The RMAP prior is constructed as a two-component mixture of the MAP prior and a vague prior. For an arbitrary parameter vector θ , the RMAP prior is defined as

$$\pi_{\text{RMAP}}(\theta|D_0, \gamma) = \gamma \pi_{\text{MAP}}(\theta|D_0) + (1 - \gamma) \pi_v(\theta),$$

where $\gamma \in [0, 1]$ is an elicited hyperparameter that controls the degree of borrowing from the historical data, and π_v denotes the vague prior. When $\gamma = 0$, the RMAP prior reduces to the vague prior, whereas when $\gamma = 1$, it is identical to the MAP prior, yielding the same posterior distribution as the BHM when combined with the current data likelihood.

Although Schmidli et al. (2014) recommends approximating the MAP prior using a finite mixture of conjugate priors, it can be difficult and time-consuming to identify an appropriate approximation. The **RBesT** package implements this approach using the expectation–maximization (EM) algorithm, which enables fast analytic posterior computation and facilitates the evaluation of operating characteristics in trial designs based on the RMAP prior. In contrast, **hdbayes** does not rely on finite-mixture approximations. Instead, it computes the posterior under the RMAP prior by directly evaluating the marginal likelihoods of the vague and MAP priors. Specifically, the posterior under the RMAP prior is given by

$$\begin{aligned} p_{\text{RMAP}}(\boldsymbol{\beta}, \phi | D, D_0, \gamma) &= \frac{L(\boldsymbol{\beta}, \phi | D) [\gamma \pi_I(\boldsymbol{\beta}, \phi | D_0) + (1 - \gamma) \pi_V(\boldsymbol{\beta}, \phi)]}{\int \int L(\boldsymbol{\beta}^*, \phi^* | D) [\gamma \pi_I(\boldsymbol{\beta}^*, \phi^* | D_0) + (1 - \gamma) \pi_V(\boldsymbol{\beta}^*, \phi^*)] d\boldsymbol{\beta}^*, d\phi^*}, \\ &= \tilde{\gamma} p_I(\boldsymbol{\beta}, \phi | D, D_0) + (1 - \tilde{\gamma}) p_V(\boldsymbol{\beta}, \phi | D), \end{aligned} \quad (13)$$

where $p_I(\boldsymbol{\beta}, \phi | D, D_0) = L(\boldsymbol{\beta}, \phi | D) \pi_I(\boldsymbol{\beta}, \phi | D_0) / Z_I(D, D_0)$ is the posterior density under the informative (MAP) prior, $p_V(\boldsymbol{\beta}, \phi | D) = L(\boldsymbol{\beta}, \phi | D) \pi_V(\boldsymbol{\beta}, \phi) / Z_V(D)$ is the posterior density under the vague prior, and

$$\tilde{\gamma} = \frac{\gamma Z_I(D, D_0)}{\gamma Z_I(D, D_0) + (1 - \gamma) Z_V(D)} \quad (14)$$

is the updated mixture weight. The normalizing constants $Z_I(D, D_0)$ and $Z_V(D)$ are estimated via the **bridgesampling** package within **hdbayes**. This approach obviates the need for finite-mixture approximations of the MAP prior and is often more computationally convenient. Details for computing the normalizing constants are provided in Section 3. The procedure for obtaining posterior samples under the RMAP prior is summarized in Algorithm 2.

Algorithm 2: Posterior Sampling under the Robust Meta-Analytic Predictive Prior

Data: Current data D ; historical data $D_0 = \{D_{0h} : h = 1, \dots, H\}$; number of posterior samples M ; hyperparameters ξ .

Result: M posterior samples under the RMAP prior.

- 1 Obtain M posterior samples under the BHM using the current and historical data, $p_I(\boldsymbol{\theta} | D, D_0)$.
 - 2 Obtain M posterior samples under a vague prior using the current data only, $p_V(\boldsymbol{\theta} | D)$.
 - 3 Compute the normalizing constants $Z_I(D, D_0)$ and $Z_V(D)$ via **bridgesampling**, then compute $\tilde{\gamma}$ in (14).
 - 4 For each $m = 1, \dots, M$, draw an indicator $j_m \sim \text{Bernoulli}(\tilde{\gamma})$. If $j_m = 1$, sample (with replacement) $\boldsymbol{\theta}^{(m)}$ from the posterior samples corresponding to $p_I(\cdot | D, D_0)$; otherwise, sample (with replacement) $\boldsymbol{\theta}^{(m)}$ from those corresponding to $p_V(\cdot | D)$.
-

2.7 Commensurate prior

The commensurate prior (CP) of Hobbs et al. (2012) is a hierarchical prior. In the traditional CP, the current data regression coefficients, $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)'$, are assumed to follow normal distributions centered at the historical data regression coefficients, $\boldsymbol{\beta}_0 = (\beta_{01}, \dots, \beta_{0p})'$, with hierarchical precision parameters. Specifically, the CP assumes $\beta_j \sim N(\beta_{0j}, \tau_j^{-1})$, where τ_j is referred to as the “commensurability parameter” for the j^{th} regression coefficient. The parameter τ_j measures how compatible the current and historical data are based on the j^{th} covariate, with larger values indicating a higher degree of commensurability.

To our knowledge, a CP for multiple historical data sets has not yet been formally developed. In **hdbayes**, we implement the CP by assuming that all historical data sets share the same regression coefficients (but may have different dispersion parameters, if

applicable). Expressed hierarchically, the CP as implemented in **hdbayes** is given by

$$\begin{aligned}
 \beta_{0j} | \mu_{0j}, \sigma_{0j} &\sim N(\mu_{0j}, \sigma_{0j}^2), \quad j = 1, \dots, p, \\
 \phi | m_0, s_0 &\sim N^+(m_0, s_0^2), \\
 \phi_{0h} | m_{0h}, s_{0h} &\sim N^+(m_{0h}, s_{0h}^2), \quad h = 1, \dots, H, \\
 \beta_j | \beta_{0j}, \tau_j &\sim N(\beta_{0j}, \tau_j^{-1}), \quad j = 1, \dots, p, \\
 \tau_j | p_{\text{spike}}, \mu_{\text{spike}}, \sigma_{\text{spike}}, \mu_{\text{slab}}, \sigma_{\text{slab}} &\sim p_{\text{spike}} N^+(\mu_{\text{spike}}, \sigma_{\text{spike}}^2) + (1 - p_{\text{spike}}) N^+(\mu_{\text{slab}}, \sigma_{\text{slab}}^2), \\
 y_i | \beta, \phi &\sim f(y_i | \beta, \phi), \quad i = 1, \dots, n, \\
 y_{0hi} | \beta_0, \phi_{0h} &\sim f(y_{0hi} | \beta_0, \phi_{0h}), \quad h = 1, \dots, H, \quad i = 1, \dots, n_{0h}.
 \end{aligned} \tag{15}$$

Following [Hobbs et al. \(2012\)](#), we elicit a spike-and-slab prior on the commensurability parameters τ_j . When there is only one historical data set ($H = 1$), the CP implemented in **hdbayes** corresponds to the traditional CP of [Hobbs et al. \(2012\)](#).

The CP is implemented in **hdbayes** via the function `glm.commensurate()`. The default hyperparameters are

$$\begin{aligned}
 \mu_{0j} &= 0, \quad j = 1, \dots, p, \\
 \sigma_{0j} &= 10, \quad j = 1, \dots, p, \\
 m_0 &= m_{0h} = 0, \quad h = 1, \dots, H, \\
 s_0 &= s_{0h} = 10, \quad h = 1, \dots, H, \\
 p_{\text{spike}} &= 0.1, \\
 \mu_{\text{spike}} &= 200, \\
 \sigma_{\text{spike}} &= 0.1, \\
 \mu_{\text{slab}} &= 0, \\
 \sigma_{\text{slab}} &= 5.
 \end{aligned}$$

The default “spike” component approximates a point mass at $\tau_j = 200$, encouraging a high degree of borrowing, whereas the default “slab” component is a half-normal distribution with scale 5, which places most of its mass on smaller values of τ_j , allowing a smaller amount of borrowing. In general, eliciting hyperparameters for the spike-and-slab prior on the τ_j ’s is problem specific.

The R packages **psborrow** ([Gower-Page et al., 2025](#)) and **psborrow2** ([Secrest and Gravestock, 2025](#)) provide propensity score-based implementations of the CP, with the former relying on **rjags** to conduct MCMC sampling and the latter relying on **cmdstanr** (as does **hdbayes**). The **psborrow2** package offers more flexibility in specifying initial priors than **hdbayes**. However, since initial priors are usually taken to be non-informative, the class of prior distributions has little impact on the analysis results in the vast majority of applications. Moreover, this increased flexibility comes at the cost of a more complicated syntax, while the syntax in **hdbayes** is similar to the `glm` function in the **stats** package.

2.8 Latent exchangeability prior

The latent exchangeability prior (LEAP), developed by [Alt et al. \(2024\)](#), assumes that the historical data are generated from a finite mixture model consisting of $K \geq 2$ components, with the current data generated from one component of this mixture (taken to be the first component, without loss of generality). For a single historical data set, the posterior under

the LEAP can be expressed hierarchically as

$$\begin{aligned}
 \gamma &\sim \text{Dirichlet}(\alpha_0), \\
 \beta, \beta_{0k} &\stackrel{\text{i.i.d.}}{\sim} N(\mu_0, \Sigma_0), \quad k = 2, \dots, K, \\
 \phi, \phi_{0k} &\stackrel{\text{i.i.d.}}{\sim} N^+(m_0, s_0^2), \quad k = 2, \dots, K, \\
 y_i | \beta, \phi &\sim f(\cdot | \beta, \phi), \quad i = 1, \dots, n, \\
 y_{0i} | \beta, \beta_0, \phi, \phi_0, \gamma &\sim \gamma_1 f(\cdot | \beta, \phi) + \sum_{k=2}^K \gamma_k f(\cdot | \beta_{0k}, \phi_{0k}), \quad i = 1, \dots, n_0,
 \end{aligned} \tag{16}$$

where $\gamma = (\gamma_1, \dots, \gamma_K)$ are the mixing probabilities, and $\alpha_0 = (\alpha_{01}, \dots, \alpha_{0K})'$ is the concentration hyperparameter of the Dirichlet prior. The prior mean and covariance for the K regression coefficients are denoted by μ_0 and Σ_0 , respectively, while m_0 and s_0 are the location and scale parameters of the truncated normal prior on the K dispersion parameters. The default hyperparameters in **hdbayes** are $K = 2$, $\alpha_0 = \mathbf{1}_K$, $\mu_0 = \mathbf{0}_p$, $\Sigma_0 = \text{diag}\{\sigma_{0j}^2, j = 1, \dots, p\}$ with $\sigma_{0j} = 10$, corresponding to non-informative initial priors.

Unlike the priors described above, which conduct blanket discounting of the historical data, the LEAP conducts discounting at the individual level of the historical data. Since the LEAP has not been developed for multiple historical data sets, the approach taken by **hdbayes** is to stack all H historical data sets into a single combined data set D_0 with $n_0 = \sum_{h=1}^H n_{0h}$ observations. The LEAP is implemented via the `glm.leap()` function in **hdbayes**, which provides the first implementation of the LEAP in a publicly available R package.

3 Model selection via marginal likelihoods

Canonical Bayesian model selection proceeds by calculating the marginal likelihood (also referred to as the “evidence”). For example, suppose the model space $\mathcal{M} = \{M_1, M_2\}$ consists of two candidate models. Kass and Raftery (1995) shows that the preference of M_2 over M_1 can be expressed in terms of the Bayes factor, given by

$$\text{BF}(D) = \frac{Z_2(D)}{Z_1(D)} = \frac{\int L_2(\theta_2|D) \pi_2(\theta_2) d\theta_2}{\int L_1(\theta_1|D) \pi_1(\theta_1) d\theta_1}, \tag{17}$$

where $L_j(\cdot|D)$ is the likelihood corresponding to model M_j with parameters θ_j and prior π_j , and $Z_j(D)$ is the normalizing constant of the posterior for model M_j and prior π_j (i.e., the marginal likelihood for model M_j).

In **hdbayes**, many of the implemented priors are not normalized. While this does not present an issue for accurate MCMC sampling, it generally leads to marginal likelihoods that are only accurate up to a constant of proportionality. When comparing models with different priors, regression coefficients, etc., this can result in incorrect Bayes factors. Let $\pi(\theta) = \tilde{\pi}(\theta)/C$ denote a prior for θ , where $\tilde{\pi}$ is the unnormalized prior, and $C = \int \tilde{\pi}(\theta) d\theta$ is its normalizing constant. The marginal likelihood may be computed as

$$Z(D) = \int L(\theta|D) \pi(\theta) d\theta = \frac{\int L(\theta|D) \tilde{\pi}(\theta) d\theta}{\int \tilde{\pi}(\theta^*) d\theta^*} = \frac{\tilde{Z}(D)}{C}, \tag{18}$$

where $\tilde{Z}(D) = \int L(\theta|D) \tilde{\pi}(\theta) d\theta$.

It follows that the marginal likelihood can be computed in a two-step process. First, the normalizing constant of the unnormalized prior is estimated, for example, via bridge sampling using MCMC samples from the prior. Second, MCMC samples are drawn from the posterior under $\tilde{\pi}$, and the normalizing constant of the posterior is then estimated, again via bridge sampling. The ratio of these two estimates gives the marginal likelihood $Z(D)$.

We give an example of these techniques in Appendix A, where we illustrate how to select between the logit and probit link functions for the HIV data set (see Section 4.1) under the PP using the marginal likelihood functionality in **hdbayes**.

3.1 Example: LEAP

To illustrate our approach, consider the LEAP. Let $L(\theta_1|D)$ denote the likelihood for the current data, and let $L_0(\theta, \gamma|D_0)$ denote the likelihood pertaining to the mixture model for the historical data, where $\theta = (\theta'_1, \dots, \theta'_K)'$ and $\gamma = (\gamma_1, \dots, \gamma_K)'$. Also, let $\pi_0(\theta, \gamma)$ denote the initial prior. The marginal likelihood under the LEAP can be expressed as

$$\begin{aligned} Z(D) &= \int L(\theta_1|D) \pi_{\text{LEAP}}(\theta_1|D_0) d\theta_1, \\ &= \int \int L(\theta_1|D) \frac{L_0(\theta, \gamma|D_0) \pi_0(\theta, \gamma)}{\int \int L_0(\theta^*, \gamma^*|D_0) d\theta^* d\gamma^*} d\theta d\gamma, \\ &= \frac{\int \int L(\theta_1|D) L_0(\theta, \gamma|D_0) \pi_0(\theta, \gamma) d\theta d\gamma}{\int \int L_0(\theta^*, \gamma^*|D_0) d\theta^* d\gamma^*}, \end{aligned} \quad (19)$$

which takes the same form as (18). To estimate the marginal likelihood under the LEAP in **hdbayes**, users first fit the LEAP using the `glm.leap()` function. The resulting fitted object is then passed to `glm.logml.leap()`, which internally performs two bridge-sampling steps. First, it draws MCMC samples from the prior and computes the normalizing constant of the prior (the denominator in (19)). Then, it estimates the normalizing constant of the posterior (the numerator in (19)). The ratio of these two quantities provides the marginal likelihood $Z(D)$ under the LEAP.

The function can be called as

```
glm.logml.leap(post.samples, bridge.args, ...)
```

where `post.samples` is the fitted object returned by `glm.leap()`, `bridge.args` is a list of arguments passed to `bridgesampling::bridge_sampler()`, and `...` includes additional arguments for the **cmdstanr** sampler used to draw prior samples for estimating the denominator in (19).

Analogous functions for other priors are available in **hdbayes**, including `glm.logml.pp()` (PP), `glm.logml.npp()` (NPP), `glm.logml.napp()` (NAPP), and `glm.logml.map()` (BHM). For the NPP and NAPP, the priors are already normalized, so no estimation of the prior normalizing constant is required. A separate function for the RMAP prior is not provided, but its marginal likelihood can be easily computed using the updated mixture weight in (14) (included in the output of `glm.rmap()`) and the marginal likelihoods under the MAP and vague priors.

4 Data analyses

4.1 Logistic regression example

We now illustrate the functionality in **hdbayes** by analyzing the data on two clinical trials presented in Section 4.2 of Chen et al. (1999) using logistic regression to model progression to HIV in a cohort of patients treated with zidovudine (AZT). More details and full output can be found in the vignette “AIDS-Progression” that accompanies **hdbayes**. The historical data come from the ACTG019 study (Volberding et al., 1990), which was a double-blind placebo-controlled clinical trial comparing AZT with a placebo in people with CD4 cell counts below 500. The sample size of complete observations for this study was $n_0 = 822$. The response variable (outcome) is binary, taking the value 1 for death, development of AIDS or AIDS-related complex (ARC) and 0 otherwise. The covariates considered are CD4 cell count (`cd4`), age, treatment and race.

Our goal is to analyze data from a more recent study, ACTG036 (Merigan et al., 1991), which includes $n = 183$ observations. We use the methods implemented in `hdbayes` to construct informative priors based on the ACTG019 study (historical data) and apply them to the analysis of ACTG036, using the same set of covariates. To facilitate computation, we center and scale the continuous covariates (age and cd4). In general, we recommend this preprocessing step in order to keep regression coefficients on roughly the same scale and thus avoid difficult posterior geometries for the Stan dynamic Hamiltonian Monte Carlo (dHMC) procedure to explore. The data preparation can be accomplished by calling

```
library(hdbayes)
library(posterior)
library(dplyr)
library(parallel)

data(actg019)
data(actg036)
age_stats <- with(actg036,
                  c('mean' = mean(age), 'sd' = sd(age)))
cd4_stats <- with(actg036,
                  c('mean' = mean(cd4), 'sd' = sd(cd4)))
actg036$age <- ( actg036$age - age_stats['mean'] ) / age_stats['sd']
actg019$age <- ( actg019$age - age_stats['mean'] ) / age_stats['sd']
actg036$cd4 <- ( actg036$cd4 - cd4_stats['mean'] ) / cd4_stats['sd']
actg019$cd4 <- ( actg019$cd4 - cd4_stats['mean'] ) / cd4_stats['sd']
```

Now, we set up the analysis by creating a formula and (GLM) family, as well as obtaining the maximum likelihood estimates (MLEs) of the regression coefficients in both current and historical data models using `stats::glm()`. We also specify a list of data sets, with the first element being the current data and the second element being the historical data. In `hdbayes`, we assume that the first specified data set in the list is the current data set, and any other data sets are historical.

```
formula <- outcome ~ age + race + treatment + cd4
p       <- length(attr(terms(formula), "term.labels")) ## number of predictors
family  <- binomial('logit')

fit.mle.cur  <- glm(formula, family, actg036)
fit.mle.hist <- glm(formula, family, actg019)

the.data <- list(actg036, actg019)
```

We can now briefly inspect the confidence intervals for the coefficients in the historical data (ACTG019) model:

```
round(confint(fit.mle.hist), 3)

              2.5 % 97.5 %
(Intercept) -6.901 -2.440
age          0.114  0.870
race        -0.026  4.424
treatment   -1.357 -0.158
cd4         -1.031 -0.396
```

and for the coefficients in the current data (ACTG036) model:

```
round(confint(fit.mle.cur), 3)
```

	2.5 %	97.5 %
(Intercept)	-7.237	-1.840
age	-0.513	0.813
race	-1.932	3.141
treatment	-1.575	1.317
cd4	-2.913	-0.926

From these confidence intervals, we observe (i) substantial uncertainty in the coefficient estimates from the current data model and (ii) notable discrepancies between the estimates from the current and historical data models, particularly for treatment, which is of primary interest. This motivates the use of informative priors that leverage historical data while accommodating potential prior-data conflict by allowing for discounting.

Next, we set up the computational specifications for our analysis, in which we will run four parallel Markov chains, each with 2,500 posterior samples following a burn-in period of 1,000 iterations (yielding a total of 10,000 post-burn-in samples):

```
ncores      <- 4
nchains     <- 4 ## number of Markov chains
iter_warmup <- 1000 ## warmup per chain for MCMC sampling
iter_sampling <- 2500 ## number of samples post warmup per chain
```

We are now prepared to fit the BHM (Section 2.5), which can be achieved by simply calling its dedicated function, `glm.bhm()`:

```
fit.bhm <- glm.bhm(
  formula, family, the.data,
  meta.mean.mean = 0, meta.mean.sd = 10,
  meta.sd.mean = 0, meta.sd.sd = 0.5,
  iter_warmup = iter_warmup, iter_sampling = iter_sampling,
  chains = nchains, parallel_chains = ncores,
  refresh = 0
)
```

The output from `glm.bhm()` is a `draws_df` object derived from the [posterior](#) package (Bürkner et al., 2025). One may utilize functions in the [posterior](#) package, like `summarise_draws()`, to acquire posterior inference and MCMC diagnostics. For instance, one can obtain posterior mean, standard deviation, and quantiles for both current and historical data regression coefficients as follows:

```
## function to pull out the summaries in a convenient form
## function to pull out the posterior summaries in a convenient form
get_summaries <- function(fit, pars.interest, digits = 3) {
  fit %>%
    select(all_of(pars.interest)) %>%
    summarise_draws(mean, sd, ~quantile(.x, probs = c(0.025, 0.5, 0.975)),
      .num_args = list(digits = digits))
}
## names of current and historical regression: coefficients
base.pars      <- c("(Intercept)", "age", "race", "treatment", "cd4")
base.pars.hist <- paste(base.pars, "hist", "1", sep="_")
get_summaries(fit = fit.bhm, pars.interest = c(base.pars, base.pars.hist))

# A tibble: 10 × 6
  variable      mean    sd `2.5%` `50%` `97.5%`
  <chr>          <dbl> <dbl> <dbl> <dbl> <dbl>
```

1 (Intercept)	-4.130	0.906	-6.151	-4.049	-2.601
2 age	0.262	0.261	-0.307	0.283	0.729
3 race	1.012	0.903	-0.593	0.958	2.956
4 treatment	-0.644	0.430	-1.496	-0.648	0.231
5 cd4	-1.247	0.369	-2.066	-1.214	-0.645
6 (Intercept)_hist_1	-3.917	0.828	-5.791	-3.841	-2.527
7 age_hist_1	0.456	0.185	0.097	0.455	0.822
8 race_hist_1	1.383	0.826	0.006	1.309	3.267
9 treatment_hist_1	-0.706	0.291	-1.287	-0.700	-0.139
10 cd4_hist_1	-0.776	0.164	-1.098	-0.777	-0.450

Although not shown above, the `posterior::summarise_draws()` function can also be used to compute common MCMC convergence and efficiency diagnostics, such as the R-hat statistic, bulk effective sample size (bulk-ESS), and tail effective sample size (tail-ESS). For example,

```
posterior::summarise_draws(fit.bhm, rhat, ess_bulk, ess_tail)
```

All model-fitting functions in **hdbayes** (e.g., `glm.bhm()`, `glm.pp()`, `glm.leap()`) return objects of class `draws_df`, which can be directly passed to functions in the **bayesplot** package (Gabry and Mahr, 2024) to visualize MCMC diagnostics, such as trace plots and autocorrelation (ACF) plots. This feature allows users to leverage existing tools to assess MCMC convergence and mixing without the need for additional diagnostic functions within **hdbayes**. For example, one can generate trace plots for selected coefficients using

```
bayesplot::mcmc_trace(fit.bhm, pars = c("age", "race", "treatment", "cd4"))
```

and ACF plots using

```
bayesplot::mcmc_acf(fit.bhm, pars = c("age", "race", "treatment", "cd4"))
```

Together, the functions in **posterior** and **bayesplot** provide a straightforward and comprehensive approach for conducting MCMC diagnostics for all models implemented in **hdbayes**.

Having used the BHM as an example of the type of output that can be obtained from functions available in **hdbayes**, we now illustrate how to fit the other models introduced in Section 2 to enable later comparison of their results. To fit the CP (Section 2.7), one can simply call `glm.commensurate()`. Posterior inference for the current and historical data regression coefficients can then be obtained in the same manner as in the BHM example.

```
fit.commensurate <- glm.commensurate(
  formula = formula, family = family, data.list = the.data,
  p.spike = 0.1, spike.mean = 200, spike.sd = 0.1,
  slab.mean = 0, slab.sd = 5,
  iter_warmup = iter_warmup, iter_sampling = iter_sampling,
  chains = nchains, parallel_chains = ncores,
  refresh = 0
)
base.pars.hist <- paste(base.pars, "hist", sep="_")
get_summaries(fit = fit.commensurate, pars.interest = c(base.pars, base.pars.hist))

# A tibble: 10 × 6
  variable      mean      sd `2.5%` `50%` `97.5%`
  <chr>         <dbl> <dbl> <dbl> <dbl> <dbl>
1 (Intercept) -4.197  0.902 -6.164 -4.121 -2.643
2 age          0.223  0.276 -0.332  0.224  0.748
3 race         0.980  0.892 -0.585  0.917  2.917
```

4 treatment	-0.605	0.472	-1.542	-0.602	0.325
5 cd4	-1.326	0.345	-2.046	-1.308	-0.704
6 (Intercept)_hist	-3.944	0.827	-5.783	-3.855	-2.553
7 age_hist	0.476	0.185	0.113	0.476	0.841
8 race_hist	1.403	0.822	0.031	1.324	3.235
9 treatment_hist	-0.706	0.294	-1.291	-0.702	-0.137
10 cd4_hist	-0.765	0.162	-1.084	-0.765	-0.445

Notice that here we specify a spike-and-slab prior on the commensurability parameters (τ_j 's).

Fitting the RMAP prior (Section 2.6) involves setting the hyperparameter γ (denoted as w in the function `glm.rmap()`), which controls the amount of borrowing from the historical data. The default value in `hdbayes` is $\gamma = 0.1$. To sample from the posterior distribution under the RMAP prior, we implement Algorithm 2 described in Section 2.6 as follows:

```
## fit RMAP
res.rmap <- glm.rmap(
  formula = formula, family = family, data.list = the.data,
  w = 0.1,
  iter_warmup = iter_warmup, iter_sampling = iter_sampling,
  chains = nchains, parallel_chains = ncores,
  refresh = 0
)
fit.rmap <- res.rmap[["post.samples"]]
get_summaries(fit.rmap, pars.interest = base.pars)

# A tibble: 5 × 6
  variable      mean      sd `2.5%` `50%` `97.5%`
  <chr>         <dbl> <dbl> <dbl> <dbl> <dbl>
1 (Intercept) -4.283  0.995 -6.517 -4.208 -2.595
2 age          0.236  0.286 -0.355  0.255  0.746
3 race         0.847  0.979 -1.003  0.802  2.936
4 treatment   -0.540  0.527 -1.556 -0.556  0.566
5 cd4         -1.472  0.438 -2.420 -1.445 -0.724
```

While the argument w could be omitted, we write it explicitly and encourage users to test different values. A similar situation occurs when using the PP discussed in Section 2.2, but here we need to set the discounting parameter a_{01} , for which no default value is provided in `hdbayes`. In the present analysis, we set $a_{01} = \frac{1}{2} \frac{n}{n_0} \approx 0.11$. The rationale for this choice is that when historical and current data sets have similar sample sizes, we typically set $a_{01} = 0.5$. However, when the current study has a much smaller sample size than the historical data, we discourage borrowing so as to avoid overwhelming the information contained in the current data.

```
n0      <- nrow(actg019)
n        <- nrow(actg036)
a0.star <- (n/n0) * 1/2
fit.pp  <- glm.pp(
  formula = formula, family = family, data.list = the.data,
  a0 = a0.star, ## discounting parameter
  iter_warmup = iter_warmup, iter_sampling = iter_sampling,
  chains = nchains, parallel_chains = ncores,
  refresh = 0
)
get_summaries(fit.pp, pars.interest = base.pars)
```

```
# A tibble: 5 × 6
  variable      mean      sd `2.5%` `50%` `97.5%`
  <chr>         <dbl> <dbl> <dbl> <dbl> <dbl>
1 (Intercept) -3.875  1.240 -6.745 -3.721 -1.884
2 age           0.249  0.281 -0.303  0.249  0.793
3 race          0.718  1.223 -1.258  0.578  3.509
4 treatment    -0.482  0.564 -1.624 -0.471  0.593
5 cd4          -1.280  0.326 -1.945 -1.275 -0.672
```

For the NPP (Section 2.3), it is necessary to specify a prior distribution for a_{01} . Following the same rationale as before, we elicit a Beta prior with parameters α and β such that the mean satisfies $\alpha/(\alpha + \beta) = (1/2)(n/n_0)$ and the coefficient of variation equals 1. This gives $\alpha \approx 0.77$ and $\beta \approx 6.21$. The resulting prior distribution is shown in Figure 1A.

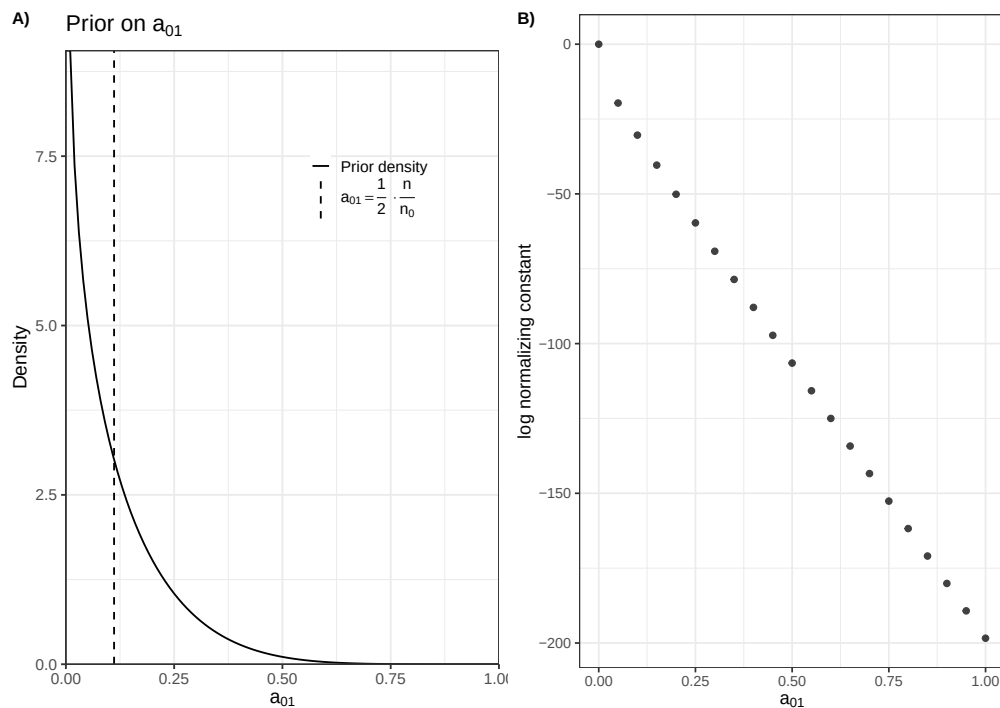


Figure 1: Beta prior distribution for a_{01} and estimated log-normalizing constants as a function of a_{01} for the AIDS progression logistic regression model. In panel A, the vertical dotted line marks the mean value $1/2(n/n_0)$. Panel B shows the estimated log-normalizing constant obtained using Algorithm 1. Note that $\log Z_h(0) = 0$ for all h .

As discussed in Section 2.3, implementing the NPP also requires estimating the normalizing constant $Z_1(a_{01})$ at several values of a_{01} in order to approximately sample from the joint posterior of a_{01} and θ , the model parameters. Fortunately, this task can be accomplished in parallel using the [parallel](#) package:

```
a0      <- seq(0, 1, length.out = 21)
histdata <- the.data[[2]]
## wrapper to obtain log normalizing constant in parallel package
logncfun <- function(a0, ...){
  hdbayes::glm.npp.lognc(
    formula = formula, family = family, histdata = histdata, a0 = a0, ...
  )
}
cl <- makeCluster(10)
clusterSetRNGStream(cl, 123)
clusterExport(cl, varlist = c('formula', 'family', 'histdata'))
```



```
## call created function
a0.lognc <- parLapply(
  cl = cl, X = a0, fun = logncfun, iter_warmup = 2*iter_warmup,
  iter_sampling = 2*iter_sampling, chains = nchains, refresh = 0
)
stopCluster(cl)
```

Here, we specify a longer warm-up period to improve convergence for small values of a_{01} , which can lead to flatter posteriors that are more challenging to sample from. The resulting data.frame includes the values of a_{01} , the corresponding estimated log-normalizing constants, and MCMC diagnostics, allowing users to assess the reliability of the results.

```
a0.lognc <- data.frame( do.call(rbind, a0.lognc) )
head(a0.lognc) %>%
  mutate(across(where(is.numeric), round, 3))
```

	a0	lognc	min_ess_bulk	max_Rhat
1	0.00	0.000	10492.866	1.001
2	0.05	-19.658	7016.144	1.001
3	0.10	-30.372	6495.816	1.001
4	0.15	-40.382	5753.908	1.001
5	0.20	-50.115	6753.503	1.001
6	0.25	-59.686	6865.135	1.001

Figure 1B displays the estimated log-normalizing constants evaluated at a grid of 21 equally-spaced values of $a_{01} \in [0, 1]$. We are now ready to fit the NPP using the computations stored in `a0.lognc`.

```
fit.npp <- glm.npp(
  formula = formula, family = family, data.list = the.data,
  a0.lognc = a0.lognc$a0,
  lognc = matrix(a0.lognc$lognc, ncol = 1),
  a0.shape1 = beta.pars$a, a0.shape2 = beta.pars$b, ## Beta prior on a_{01}
  iter_warmup = iter_warmup, iter_sampling = iter_sampling,
  chains = nchains, parallel_chains = ncores,
  refresh = 0
)
get_summaries(fit = fit.npp, pars.interest = c(base.pars, "a0_hist_1"))
```

```
# A tibble: 6 × 6
  variable      mean    sd `2.5%` `50%` `97.5%`
  <chr>         <dbl> <dbl> <dbl> <dbl> <dbl>
1 (Intercept) -3.836 1.179 -6.546 -3.709 -1.932
2 age          0.278 0.265 -0.251  0.279  0.781
3 race         0.802 1.170 -1.166  0.690  3.437
4 treatment   -0.539 0.534 -1.594 -0.546  0.529
5 cd4         -1.193 0.339 -1.944 -1.158 -0.615
6 a0_hist_1    0.187 0.115  0.050  0.158  0.478
```

The NAPP (Section 2.4) can be fitted using very similar syntax:

```
fit.napp <- glm.napp(
  formula = formula, family = family, data.list = the.data,
  a0.shape1 = beta.pars$a, a0.shape2 = beta.pars$b,
  iter_warmup = iter_warmup, iter_sampling = iter_sampling,
  chains = nchains, parallel_chains = ncores,
```

```

    refresh = 0
  )
  get_summaries(fit = fit.napp, pars.interest = c(base.pars, "a0_hist_1"))

# A tibble: 6 × 6
  variable      mean    sd `2.5%` `50%` `97.5%`
  <chr>         <dbl> <dbl> <dbl> <dbl> <dbl>
1 (Intercept) -3.644  1.173 -6.204 -3.541 -1.645
2 age          0.302  0.252 -0.202  0.306  0.780
3 race         0.729  1.167 -1.288  0.632  3.234
4 treatment    -0.543  0.503 -1.530 -0.547  0.468
5 cd4         -1.113  0.322 -1.840 -1.081 -0.568
6 a0_hist_1     0.188  0.128  0.026  0.158  0.502

```

Note that no reference to a_0 .lognc or lognc is necessary, as the normalizing constant for the NAPP is known in closed form.

As described in Section 2.8, fitting the LEAP requires specifying the hyperparameters K and α_0 , corresponding to the arguments `K` and `prob.conc` in the `glm.leap()` function. Posterior samples under the LEAP can be obtained as follows:

```

fit.leap <- glm.leap(
  formula = formula, family = family, data.list = the.data,
  K = 2, prob.conc = rep(1, 2),
  iter_warmup = iter_warmup, iter_sampling = iter_sampling,
  chains = nchains, parallel_chains = ncores,
  refresh = 0
)
get_summaries(fit = fit.leap, pars.interest = c(base.pars, "gamma"))

# A tibble: 6 × 6
  variable      mean    sd `2.5%` `50%` `97.5%`
  <chr>         <dbl> <dbl> <dbl> <dbl> <dbl>
1 (Intercept) -4.206  1.025 -6.432 -4.083 -2.643
2 age          0.300  0.183 -0.077  0.305  0.645
3 race         1.154  1.021 -0.415  1.038  3.434
4 treatment    -0.668  0.425 -1.507 -0.680  0.228
5 cd4         -0.941  0.256 -1.507 -0.916 -0.514
6 gamma        0.948  0.059  0.763  0.966  0.997

```

In the output above, `gamma` represents the posterior probability that an individual in the historical data set is exchangeable with those in the current data, which corresponds to γ_1 in (16). The posterior mean of `gamma` suggests a high degree of exchangeability between the current and historical data sets.

We are now ready to compare the performance of all methods investigated here. A graphical summary of the posterior estimates is presented in Figure 2. As shown in Figure 2, there is considerable variation in the coefficient estimates across different methods. For instance, both the BHM and the RMAP prior pull the estimated treatment effect more strongly towards the historical MLE than the NAPP and PP do; however, the opposite pattern occurs for the estimated coefficient of `cd4`. For all covariates, the coefficients estimated under the LEAP are the closest to the historical MLEs. We note, however, that we did not limit the amount of borrowing for the LEAP, which could, in principle, be controlled by imposing a truncated prior on γ_1 in (16). While some methods, such as the PP, lead to more uncertain estimates of the treatment effect, the BHM and LEAP yield smaller uncertainties. Importantly, all credible intervals include zero, indicating that despite incorporating information from a historical study with a significant treatment effect, the analysis of the current data suggests that the treatment effect could be null.

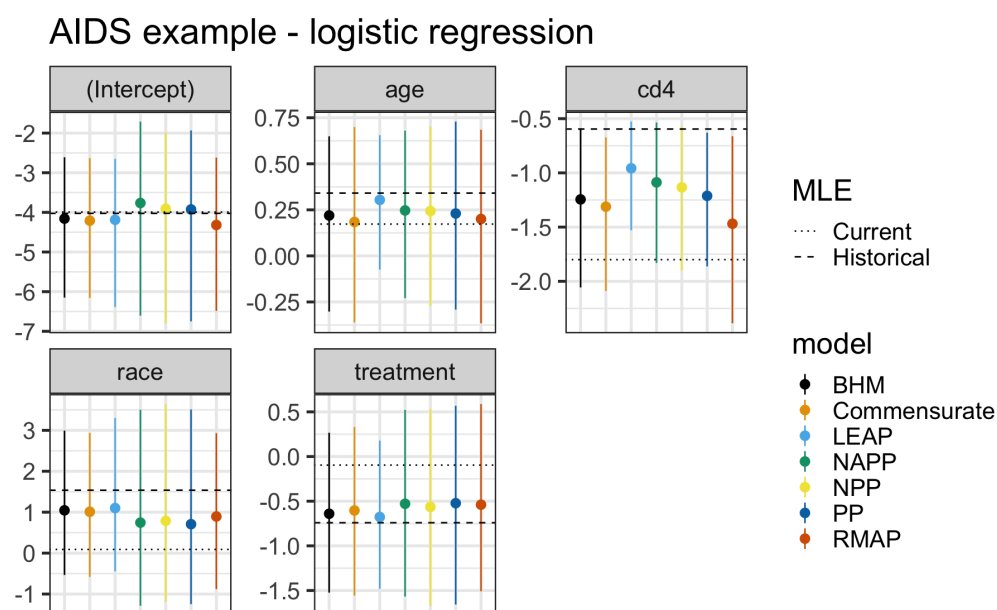


Figure 2: Coefficient estimates for the AIDS progression logistic regression model. For each coefficient (vertical panels), we show the posterior mean (dot) and the 95% equal-tailed credible interval (solid vertical line) for each of the models considered. Horizontal lines (dotted: ACTG036; dashed: ACTG019) indicate the MLE for each coefficient.

5 Discussion

The package **hdbayes** provides a cohesive and user-friendly framework for implementing commonly used priors that incorporate historical data within GLMs. In addition, **hdbayes** supports several survival models, including the PWEPH model, AFT models, and a mixture cure rate model, although we primarily focus on analyses using GLMs in this paper. The full suite of GLMs and link functions available in the **stats** package is supported in **hdbayes**, addressing an important gap in existing statistical software.

In the future, we aim to make several extensions to the **hdbayes** package. For example, more complex data types (e.g., longitudinal data) could be included. Moreover, we aim to accommodate more flexible types of borrowing, such as partial borrowing techniques and methods for handling missing data. Finally, other types of historical data priors, such as propensity score-integrated priors, could be included to provide a comprehensive suite of tools for Bayesian dynamic borrowing within a single package.

References

- E. M. Alt, X. Chang, X. Jiang, Q. Liu, M. Mo, H. A. Xia, and J. G. Ibrahim. LEAP: The latent exchangeability prior for borrowing information from historical data. *Biometrics*, 80(3): ujae083, 2024. ISSN 0006-341X. doi: 10.1093/biomtc/ujae083. [p78]
- B. Bloem-Reddy and J. Cunningham. Slice sampling on Hamiltonian trajectories. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 3050–3058. PMLR, 2016. URL <https://proceedings.mlr.press/v48/bloem-reddy16.html>. [p72]
- P.-C. Bürkner, J. Gabry, M. Kay, and A. Vehtari. posterior: Tools for working with posterior distributions, 2025. URL <https://mc-stan.org/posterior/>. R package version 1.6.1. [p82]

- B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software*, 76(1), 2017. URL <https://doi.org/10.18637/jss.v076.i01>. [p69]
- L. M. Carvalho and J. G. Ibrahim. On the normalized power prior. *Statistics in Medicine*, 40(24):5251–5275, 2021. URL <https://doi.org/10.1002/sim.9124>. [p69, 73]
- M.-H. Chen, J. G. Ibrahim, and C. Yiannoutsos. Prior elicitation, variable selection and Bayesian computation for logistic regression models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(1):223–242, 1999. URL <https://doi.org/10.1111/1467-9868.00173>. [p80]
- Y. Duan, K. Ye, and E. P. Smith. Evaluating water quality using power priors to incorporate historical information. *Environmetrics: The Official Journal of the International Environmetrics Society*, 17(1):95–106, 2006. URL <https://doi.org/10.1002/env.752>. [p69, 73]
- J. Gabry and T. Mahr. bayesplot: Plotting for Bayesian models, 2024. URL <https://mc-stan.org/bayesplot/>. R package version 1.11.1. [p83]
- J. Gabry, R. Češnovar, A. Johnson, and S. Bröder. cmdstanr: R Interface to ‘CmdStan’, 2024. URL <https://mc-stan.org/cmdstanr/>. R package version 0.8.1, <https://discourse.mc-stan.org>. [p69]
- C. Gower-Page, Y. Lu, and A. Lin. psborrow: Bayesian Dynamic Borrowing with Propensity Score, 2025. URL <https://CRAN.R-project.org/package=psborrow>. R package version 0.2.4. [p78]
- Q. F. Gronau, H. Singmann, and E.-J. Wagenmakers. bridgesampling: An R package for estimating normalizing constants. *Journal of Statistical Software*, 92(10):1–29, 2020. URL <https://doi.org/10.18637/jss.v092.i10>. [p73]
- Z. Han, T. Bai, and K. Ye. NPP: Normalized Power Prior Bayesian Analysis, 2021. URL <https://CRAN.R-project.org/package=NPP>. R package version 0.4.0. [p69]
- B. P. Hobbs, D. J. Sargent, and B. P. Carlin. Commensurate priors for incorporating historical information in clinical trials using general and generalized linear models. *Bayesian Analysis (Online)*, 7(3):639, 2012. URL <https://doi.org/10.1214/12-BA722>. [p69, 77, 78]
- M. D. Hoffman, A. Gelman, et al. The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014. URL <http://jmlr.org/papers/v15/hoffman14a.html>. [p69]
- J. G. Ibrahim and M.-H. Chen. Power prior distributions for regression models. *Statistical Science*, pages 46–60, 2000. URL <https://doi.org/10.1214/ss/1009212673>. [p69, 71, 74]
- J. G. Ibrahim, M.-H. Chen, Y. Gwon, and F. Chen. The power prior: theory and applications. *Statistics in Medicine*, 34(28):3724–3749, 2015. URL <https://doi.org/10.1002/sim.6728>. [p71, 74]
- R. E. Kass and A. E. Raftery. Bayes factors. *Journal of the American Statistical Association*, 90(430):773–795, 1995. URL <https://doi.org/10.1080/01621459.1995.10476572>. [p79, 92]
- W. M. Landau. instantiate: Pre-Compiled ‘CmdStan’ Models in R Packages, 2024. URL <https://CRAN.R-project.org/package=instantiate>. R package version 0.2.2. [p69]
- P. McCullagh and J. A. Nelder. *Generalized linear models*. Chapman & Hall, 1989. [p70]
- TC. Merigan, DA. Amato, J. Balsley, M. Power, WA. Price, S. Benoit, A. Perez-Michael, A. Brownstein, AS. Kramer, and D. Brettler. Placebo-controlled trial to evaluate zidovudine in treatment of human immunodeficiency virus infection in asymptomatic patients with hemophilia. NHF-ACTG 036 Study Group. *Blood*, 78(4):900–906, Aug. 1991. ISSN 0006-4971. URL <https://doi.org/10.1182/blood.V78.4.900.900>. [p81]

- I. Murray, R. Adams, and D. MacKay. Elliptical slice sampling. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9, pages 541–548. JMLR Workshop and Conference Proceedings, 2010. URL <https://proceedings.mlr.press/v9/murray10a.html>. [p72]
- R. M. Neal. Slice sampling. *The Annals of Statistics*, 31(3):705–767, 2003. URL <https://doi.org/10.1214/aos/1056562461>. [p72]
- B. Neuenschwander, M. Branson, and D. J. Spiegelhalter. A note on the power prior. *Statistics in Medicine*, 28(28):3562–3566, 2009. URL <https://doi.org/10.1002/sim.3722>. [p73]
- R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2024. URL <https://www.R-project.org/>. [p71]
- G. O. Roberts and J. S. Rosenthal. Optimal scaling for various Metropolis-Hastings algorithms. *Statistical Science*, 16(4):351–367, 2001. ISSN 0883-4237, 2168-8745. doi: 10.1214/ss/1015346320. [p74]
- H. Schmidli, S. Gsteiger, S. Roychoudhury, A. O’Hagan, D. Spiegelhalter, and B. Neuenschwander. Robust meta-analytic-predictive priors in clinical trials with historical control information. *Biometrics*, 70(4):1023–1032, 2014. URL <https://doi.org/10.1111/biom.12242>. [p69, 76, 77]
- M. Secrest and I. Gravestock. *psborrow2: Bayesian Dynamic Borrowing Analysis and Simulation*, 2025. URL <https://CRAN.R-project.org/package=psborrow2>. R package version 0.0.4.0. [p78]
- Y. Shen, M. A. Psioda, and J. G. Ibrahim. *BayesPPD: Bayesian Power Prior Design*, 2023. URL <https://CRAN.R-project.org/package=BayesPPD>. R package version 1.1.2. [p72]
- Z. Shun and P. McCullagh. Laplace approximation of high dimensional integrals. *Journal of the Royal Statistical Society: Series B (Methodological)*, 57(4):749–760, 1995. ISSN 2517-6161. doi: 10.1111/j.2517-6161.1995.tb02060.x. [p74]
- L. Tierney and J. B. Kadane. Accurate approximations for posterior moments and marginal densities. *Journal of the American Statistical Association*, 81(393):82–86, 1986. URL <https://doi.org/10.1080/01621459.1986.10478240>. [p74]
- P. A. Volberding, S. W. Lagakos, M. A. Koch, C. Pettinelli, M. W. Myers, D. K. Booth, H. H. Balfour Jr, R. C. Reichman, J. A. Bartlett, M. S. Hirsch, et al. Zidovudine in asymptomatic human immunodeficiency virus infection: a controlled trial in persons with fewer than 500 CD4-positive cells per cubic millimeter. *New England Journal of Medicine*, 322(14): 941–949, 1990. URL <https://doi.org/10.1056/NEJM199004053221401>. [p80]
- S. Weber, Y. Li, J. W. Seaman, T. Kakizume, and H. Schmidli. Applying meta-analytic-predictive priors with the R Bayesian evidence synthesis tools. *Journal of Statistical Software*, 100(19):1–32, 2021. doi: 10.18637/jss.v100.i19. [p69]

Ethan M. Alt

University of North Carolina at Chapel Hill

135 Dauer Drive, Chapel Hill, NC

USA

ORCID: 0000-0002-6112-9030

ethanal@live.unc.edu

Xinxin Chen

University of North Carolina at Chapel Hill

135 Dauer Drive, Chapel Hill, NC

USA

ORCID: 0000-0003-1759-7811

xchen2.unc.edu

Luiz M. Carvalho

Getulio Vargas Foundation

Rio de Janeiro

Brazil

University of North Carolina at Chapel Hill

135 Dauer Drive, Chapel Hill, NC

USA

luiz.fagundes@fgv.br

Joseph G. Ibrahim

University of North Carolina at Chapel Hill

135 Dauer Drive, Chapel Hill, NC

USA

ibrahim@bios.unc.edu

A. Link selection in binary regression with the power prior

When employing GLMs, the choice of link function can be crucial, as it controls the degree of non-linearity between the conditional mean and the linear predictor. Here we illustrate how to use **hdbayes** to compute marginal likelihoods (and thus Bayes factors) under the PP, comparing the logit and probit links in a binary regression model.

Let $\mathbf{y}_0$ and $\mathbf{y}$ denote the historical and current binary response vectors, respectively. We compare the following logistic and probit models for the probability μ_i that $y_i = 1$:

$$\mathcal{M}_1 : \mu_i = \frac{\exp(\mathbf{x}'_i \boldsymbol{\beta})}{1 + \exp(\mathbf{x}'_i \boldsymbol{\beta})}, \quad (20)$$

$$\mathcal{M}_2 : \mu_i = \Phi(\mathbf{x}'_i \boldsymbol{\beta}), \quad (21)$$

where Φ is the cumulative distribution function of a standard normal random variable. We use the PP with a fixed discounting parameter a_0 to elicit priors for both models and employ bridge sampling to approximate the marginal likelihoods for $i = 1, 2$:

$$m_i(a_0) := \int_{\mathbb{R}^p} L_i(\boldsymbol{\beta}|D) \pi_{\text{PP}}(\boldsymbol{\beta}|D_0, a_0, \pi_0, \mathcal{M}_i) d\boldsymbol{\beta}, \quad (22)$$

$$\pi_{\text{PP}}(\boldsymbol{\beta}|D_0, a_0, \pi_0, \mathcal{M}_i) := \frac{L_i(\boldsymbol{\beta}|D_0)^{a_0} \pi_0(\boldsymbol{\beta})}{\int_{\mathbb{R}^p} L_i(\boldsymbol{\beta}|D_0)^{a_0} \pi_0(\boldsymbol{\beta}) d\boldsymbol{\beta}}, \quad (23)$$

where $D_0 = (\mathbf{y}_0, \mathbf{x}_0)$ and $D = (\mathbf{y}, \mathbf{x})$ denote the historical and current data sets, respectively, and L_i and $\pi_{\text{PP}}(\cdot | \dots, \mathcal{M}_i)$ are the likelihood and PP under model $\mathcal{M}_i$. For simplicity, we specify the same initial prior for the regression coefficients $\boldsymbol{\beta}$ in both models, assigning independent $N(0, 10^2)$ distributions, which correspond to the default prior in the PP implementation of **hdbayes**.

Notice that this computation necessitates two passes of the bridge sampling algorithm: one to estimate the PP normalizing constant in (23), and another to estimate the normalizing constant of the posterior. The Bayes factor is then given by $\text{BF}_{12}(a_0) = m_1(a_0)/m_2(a_0)$. It is interesting to understand how this quantity changes with different values of a_0 using the HIV data in Section 4.1.

We show the results in Figure S1, where the logarithm of $\text{BF}_{12}(a_0)$ is plotted over a regular grid of discounting parameter values ranging from $a_0 = 0$ to $a_0 = 1$ in increments of 0.1 is shown. Following Kass and Raftery (1995), we mark the $\text{BF}_{12}(a_0) \geq 1$ (dashed) and $\text{BF}_{12}(a_0) \geq 3$ (dotted) which correspond to weak and substantial evidence, respectively (see Section 3.2 therein). It is clear that the logit link is generally preferred over the probit link. However, the relative support is not very strong, as evidenced by the fact that $\text{BF}_{12}(a_0) \geq 3$ is observed only for $a_0 < 0.1$. This suggests that one would need to introduce very little borrowing in order to observe strong support for the logit link.

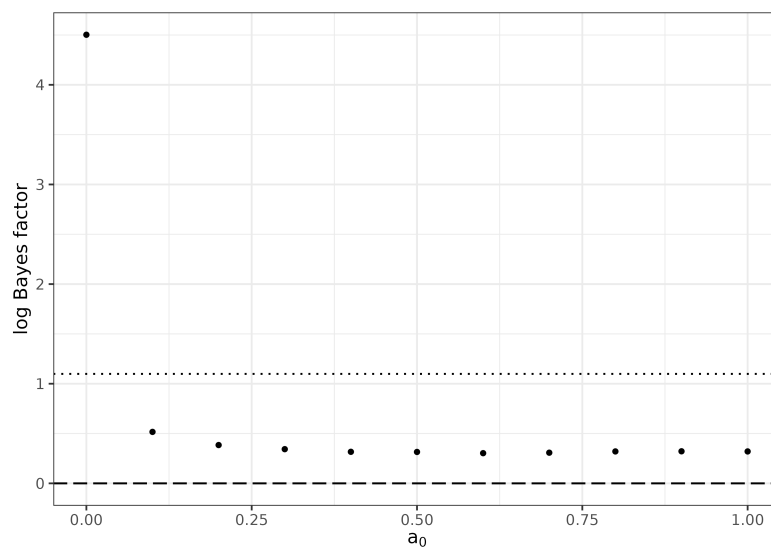


Figure S1: Log Bayes factor comparing logit (Model 1, $\mathcal{M}_1$) and probit ($\mathcal{M}_2$) links for various values of the discounting parameter a_0 . We show $\log(m_1(a_0)/m_2(a_0))$. Horizontal lines mark the $\log \text{BF}_{12} \geq 0$ (dashed) and $\log \text{BF}_{12} \geq \log(3)$ (dotted) thresholds.

mpitbR: An R Package for Calculating Multidimensional Poverty Indices

by Ignacio Girela

Abstract Multidimensional poverty measurement is a vital tool for assessing the deprivations and well-being of people across different dimensions, such as health, education, and living standards. However, existing packages for multidimensional poverty measurement have some limitations, such as ignoring the complex survey design of microdata. In this paper, we present `mpitbR`, a package for calculating multidimensional poverty indices based on the Alkire-Foster measurement approach, which accounts for the survey design, and offers various options and features for users. This package is the R version of the Stata `mpitb` package, which reproduces the workflow of the global Multidimensional Poverty Index developed by the United Nations Development Programme and Oxford University. The package provides functions for estimating the Alkire-Foster measures, as well as for analyzing poverty changes over time. The usage of the main functions and features of the `mpitbR` package is described and illustrated with an application over a synthetic data that has a typical household survey design.

1 Introduction

The first goal of the 2030 Agenda for Sustainable Development ambitiously postulates *Ending poverty in all its forms everywhere*. This call to global action recognizes poverty as not merely a lack of income but as a complex phenomenon with multiple dimensions that affect individuals' well-being.

To capture these diverse aspects of poverty, the last two decades have witnessed the development of various methodologies for multidimensional poverty measurement. The [Alkire and Foster \(2011\)](#) (AF) method, employing a *dual-cut-off-counting* approach, has gained wide adoption in both academic research and policy-making spheres due to its simplicity and rigorous theoretical foundation. Notably, the AF method allows disaggregation into various partial indices (AF measures), which serve as powerful tools for policy analysis and design ([Alkire, 2021](#)). The global Multidimensional Poverty Index (MPI), published annually by the United Nations Development Programme (UNDP) and the Oxford Poverty and Human Development Initiative (OPHI), exemplifies the worldwide relevance of the AF method ([UNDP and OPHI, 2022](#)). Furthermore, there is an increasing number of countries that have built their national MPIs based on the AF framework tailored to their local contexts in order to monitor and evaluate their poverty reduction strategies ([OPHI](#)).

Although the AF method has become a prominent framework for measuring multidimensional poverty, practical challenges arise in calculating these measures. On one hand, the construction of an MPI naturally hinges on the careful consideration and justification of its dimensions, indicators, and parameters, such as the weights and poverty lines, to ensure its accurate reflection of poverty's complexities within a specific context. On the other hand and very importantly, poverty measurement often relies on household survey data, and ignoring the complex survey design in poverty analysis can lead to biased estimates and erroneous statistical inferences, particularly when comparing poverty levels across subgroups or over time.

Therefore, reliable multidimensional poverty estimation requires tools that facilitate easy parameter management and account for complex survey designs. Existing Stata packages like `mpitb` ([Suppa, 2023](#)) address this need by providing functionalities to estimate AF measures and their associated standard errors. This Stata package provides a user-friendly way to replicate the widely-used global MPI methodology, establishing a solid guideline for researchers and practitioners in multidimensional poverty analysis. However, users without Stata expertise or who prefer the R programming environment lack a comparable solution in R.

The main contribution of this paper is to introduce the `mpitbR` package ([Girela, 2025](#)), a novel and valuable tool for multidimensional poverty measurement and analysis in R. `mpitbR` package builds upon the global MPI workflow, replicating the functionality of the aforementioned Stata package. Furthermore, unlike existing R packages, namely `MPI` ([Kukiattikun and Chainarong, 2022](#)) and `mpindex` ([Abdulsamad, 2024](#)), this package accounts for the complex design of household surveys by means of the `survey` package methods ([Lumley, 2004, 2010, 2024](#)). This ensures reliable estimation of AF measures, as well as their associated standard errors and confidence intervals, which are crucial for robust statistical inference. By addressing the survey design, `mpitbR` fills a critical gap in the R ecosystem for poverty researchers. The paper demonstrates the practical functionality of `mpitbR` with examples of common multidimensional poverty analysis exercises using real-world household survey data.

The rest of this paper is organized as follows. Section 2 briefly introduces the AF method and the

MPI calculation. Section 3 describes the **mpitbR** package and its main functions. Section 4 illustrates the usage of the package with some examples and applications. Section 5 concludes with a summary and provides some suggestions for further extensions of the package.

2 Measuring multidimensional poverty

Alkire and Foster (2011) proposed a multidimensional poverty measurement method (AF method) based on a *dual-cut-off-counting-approach* for the identification and aggregation of the poor. Constructing an MPI based on the AF method consists of the following steps (Alkire et al., 2015):

1. Determine a set of dimensions of poverty, $\mathcal{D}$, that are considered relevant for human development in a specific context (e.g., the global MPI chooses dimensions of health, education, and living standards, but other dimensions can be chosen depending on the context and goals).
2. Select d indicators that represent deprivations in each dimension (e.g., child mortality and undernutrition are the two indicators that represent the health dimension in the global MPI).
3. Assign weights to each dimension and indicator, reflecting their relative importance, where w_j represents the weight of the j -th indicator for $j = 1, \dots, d$ such that $\sum_{j=1}^d w_j = 1$. In practice, an equal nested weighting scheme is used: dimensions are weighted equally as well as each indicator within the dimension.
4. Set the indicators deprivation cut-offs, which define the minimum level of achievement required to be considered non-deprived in each indicator (for example, in the global MPI, a person is considered deprived in Years of Schooling indicator if she has less than six years of formal education).
5. Apply the deprivation cut-offs vector to each of the n observations (individuals or households) and build the $n \times d$ deprivation matrix, $\mathbf{g}^0$. Each element of this matrix, $[\mathbf{g}^0]_{ij}$, is a binary variable. If $[\mathbf{g}^0]_{ij} = 1$, the i -th observation is deprived in indicator j , and the opposite if $[\mathbf{g}^0]_{ij} = 0$.
6. Build the weighted deprivation matrix, $\mathbf{g}^0$, assigning the corresponding weight to each indicator (i.e., $\mathbf{g}^0 = \mathbf{g}^0 \times \text{diag}(w_1, \dots, w_d)$) and calculate the deprivations score for each observation, c_i , which is the weighted sum of the deprivations, $c_i = \sum_{j=1}^d w_j g_{ij}^0$.
7. Identify who is poor by setting a unique poverty cut-off, k , meaning the minimum proportion of weighted deprivations that an individual needs to experience to be considered multidimensional poor. This cut-off is compared with the deprivations score, c_i . Therefore, if $c_i \geq k$, the i -th observation in the sample is classified as multidimensional poor (e.g., the global MPI uses a cut-off, k , of 33.3% or $1/3$, meaning that people are considered to be poor if they experience deprivations in one-third or more of the weighted indicators).
8. Censor data of the non-poor to obtain the so-called censored (weighted) deprivation matrix, $\mathbf{g}^0(k)$, and censored deprivations scores, $c(k)$, where $c_i(k) = c_i$ if $c_i \geq k$, and $c_i(k) = 0$ otherwise.
9. Compute the M_0 by taking the mean of the censored deprivations score, $c(k)$:

$$M_0 = \frac{1}{n} \sum_{i=1}^n c_i(k). \quad (1)$$

The M_0 measure, also known as *Adjusted Headcount Ratio*, is sometimes equated with the MPI. Hereafter, we will use both terms interchangeably. Furthermore, the M_0 measure can be re-expressed as a function of other partial measures, which we refer to as AF measures. For instance,

- The M_0 measure can be decomposed into two key components: the *Incidence* (H) and the *Intensity* (A) of poverty. Formally, this is represented as:

$$M_0 = \frac{q}{n} \times \frac{1}{q} \sum_{i=1}^n c_i(k) = H \times A$$

Here, q is the number of people identified as multidimensional poor, $H = \frac{q}{n}$ is the proportion of multidimensionally poor people (the incidence), and $A = \frac{1}{q} \sum_{i=1}^n c_i(k)$ represents the average proportion of weighted deprivations experienced by the poor (the intensity).

- The MPI can also be calculated as the weighted sum of the *Censored Indicators Headcounts Ratios*, $h_j(k)$, for $j = 1, \dots, d$. This is clearer by rearranging the order of aggregation in the original M_0 equation (1). This is,

$$M_0 = \frac{1}{n} \sum_{i=1}^n c_i(k) = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^d g_{ij}^0(k) \quad (2)$$

$$= \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^d w_j g_{ij}^0(k) = \sum_{j=1}^d w_j \frac{1}{n} \sum_{i=1}^n g_{ij}^0(k) \quad (3)$$

$$= \sum_{j=1}^d w_j h_j(k) \quad (4)$$

Then, $h_j(k)$ represents the proportion of people who are deprived in indicator j and are multidimensional poor.

- From the uncensored deprivations matrix, $\mathbf{g}^0$, we can obtain their uncensored versions, or the *Uncensored Headcount Ratio* for each indicator $j = 1, \dots, d$, denoted as h_j , which is the proportion of people deprived in indicator j . The comparison between the censored and uncensored indicators headcount ratios is insightful for targeted interventions (Alkire, 2021).
- We can also decompose the MPI by different population subgroups, such as age, regions, sex, etc.. Formally,

$$M_0 = \sum_{l=1}^L \phi^l M_0^l$$

where ϕ^l is the population share of the l -th level for $l = 1, \dots, L$, and L is the total number of levels in the subgroup. For instance, the population subgroup “sex” has two levels of analysis: male and female, encoded as $l = 1, 2$. This property also applies to other AF measures (H , A , h_j , $h_j(k)$).

- The *absolute* and *proportional contribution* of each indicator to poverty are usually reported, calculated as $w_j h_j(k)$ and $w_j h_j(k) / M_0$, respectively.

Changes over time

Alkire et al. (2017) proposed four measures to assess pro-poor multidimensional poverty reduction between two periods of time using repeated cross-sectional data. The following measures are explained using the Adjusted Headcount Ratio, M_0 , however, all the changes-over-time measures can be extended to the aforementioned AF measures (H , A , h_j , $h_j(k)$).

Let t^1 and t^2 denote the initial and final period, respectively. Then, $M_0^{t^1}$ and $M_0^{t^2}$ are their corresponding Adjusted Headcount Ratios. Note that for comparability purposes, these two poverty measures must have the same set of parameters (indicators, weights, deprivations and poverty cut-offs).

We define the *absolute rate of change* (Δ) as the difference in M_0 between the final t^2 and the initial period t^1 :

$$\Delta(M_0) = M_0^{t^2} - M_0^{t^1}$$

The *relative rate of change* (δ) is defined as the difference in M_0 , expressed as a percentage of the initial poverty level t^1 , i.e.,

$$\delta(M_0) = \frac{M_0^{t^2} - M_0^{t^1}}{M_0^{t^1}} \times 100$$

On the other hand, the annualized versions of the absolute and relative rates of change are used in order to compare changes over time across countries with different periods of reference.

The *annualized absolute rate of change* ($\bar{\Delta}$) is the absolute rate of change (as defined above) divided by the difference between the two time periods:

$$\bar{\Delta}(M_0) = \frac{M_0^{t^2} - M_0^{t^1}}{(t^2 - t^1)}$$

Finally, the *annualized relative rate of change* ($\bar{\delta}$) is defined as the compound annual rate of reduction in M_0 between the initial and the final periods, i.e.,

$$\bar{\delta}(M_0) = \left[\left(\frac{M_0^{t^2}}{M_0^{t^1}} \right)^{\frac{1}{t^2-t^1}} - 1 \right]$$

An explanation of the usage of the main functions for estimating AF measures and their changes over time using the **mpitbR** package is provided in the following section.

3 Overview of **mpitbR** package

mpitbR adapts the Stata *mpitb* framework for R users, ensuring consistency with the original workflow. The package offers two main functions, `mpitb.set` and `mpitb.est`, that mirror the functionality of the key Stata subcommands. This means that users familiar with Stata will find similar input parameters and output formats in **mpitbR**, facilitating a smooth transition between the two packages. The estimation of AF measures is based on these core functions. In addition, **mpitbR** includes built-in R functions, such as `coef`, `confint`, and `summary`, to allow users to examine the results (i.e., estimated coefficients, confidence intervals, and summary statistics of the AF measures). As mentioned above, **mpitbR** mainly depends upon the R package **survey** to account for the complex survey design during estimation.

3.1 `mpitb.set` function

This function, mirroring the `mpitb set` subcommand in Stata, serves as the setup step for **mpitbR**. It prepares the data for AF measure estimation by specifying the deprivation indicators and incorporating the survey design.

The usage and input arguments of function `mpitb.set` are summarized as follows:

```
mpitb.set(data, indicators, ...,
          name = "unnamed", desc = "desc.")
```

Arguments:

- **data**: A dataset containing household survey information. To incorporate survey design (e.g., sampling weights, strata and primary sampling units), use the `svydesign` function from the **survey** package and provide a “survey.design2”-class object to this argument. If data is a regular data frame, **mpitbR** coerces it to a “survey.design2”-class object. However, this process assumes a simple random sampling design, which is often inappropriate for household surveys and can lead to biased results. The function does issue a warning message to alert users about this assumption and the potential need to specify a more complex survey design if their data warrants it.
- **indicators**: This argument specifies the deprivation indicators (columns) in data that define the MPI dimensions. It can be provided in two formats:
 - a list: This is useful for defining the set of dimensions and their corresponding indicators. Each element in the list represents a dimension with a character vector containing its indicator names. This format facilitates automatic calculation of equal nested weights during estimation (up to 10 dimensions allowed). Under this format, users can also specify weights for unequal weighting of dimensions.
 - a character vector: This is more convenient when indicators are not grouped by dimensions. In this case, users need to specify weights manually during the AF measure estimation.

Details for the arguments:

- The deprivation indicators specified in data should be binary variables (0/1), i.e., it is assumed that users are working with the deprivation matrix, $\mathbf{g}^0$.
- The data should not contain missing values in these indicator columns. While the **survey** package can handle missing values for point estimates, it is crucial to note that it cannot calculate standard errors and confidence intervals when missing data are present.

Optional arguments:

- **name**: a short project name for identification (maximum 10 characters).

- desc: A brief description of the project setting.

Output:

The function returns an object of class "mpitb_set" containing:

- Survey design data (from the [survey](#) package).
- Names of the deprivation indicators.
- Project name and description (if provided).

3.2 mpitb.est function

Building upon the information from `mpitb.set`, the `mpitb.est` function performs the core calculations for AF measure estimation. Here users define all the parameters and the desired measures to be calculated.

```
mpitb.est(set, klist = NULL, weights = "equal",
  measures = c("M0", "H", "A"),
  indmeasures = c("hd", "hdk", "actb", "pctb"),
  indklist = NULL, over = NULL, ...,
  cotyear = NULL, tvar = NULL,
  cotmeasures = c("M0", "H", "A", "hd", "hdk"),
  ann = FALSE, cotklist = NULL,
  cotoptions = "total",
  noraw = FALSE, nooverall = FALSE,
  level = 0.95,
  multicore = getOption("mpitb.multicore"),
  verbose = TRUE)
```

Main arguments:

- `set` (required): An object of class "mpitb_set" created by the `mpitb.set` function. This object encapsulates the previously prepared data and settings for estimation.
- `klist` (required): A numeric vector of poverty cut-offs (k) as percentages (between 1 and 100) for multidimensional poverty measurement. Default values are not specified.
- `weights`: This argument defines the weighting scheme for the deprivation indicators. It can be provided in two formats:
 - a character (default): If `weights = "equal"`, it automatically calculates equal nested weights.
 - a numeric vector: In order to provide the weights manually, users can input a numeric vector of values between 0 and 1 such that all sum up to 1. Their values should match the order of the indicators in the `set` argument.
- `measures`: A character vector that allows selecting the specific AF measures to be calculated (M_0 , H , and A). By default, `mpitb.est` calculates each one of these measures (`c("M0", "H", "A")`).
- `indmeasures`: A character vector that allows selecting the (censored and uncensored) headcount ratio of each indicator and their contribution (absolute and proportional). By default, `mpitb.est` calculates each one of these measures (`c("hd", "hdk", "actb", "pctb")`).

Optional arguments:

- `indklist`: A numeric vector with poverty cut-offs (numbers between 1 and 100) for the indicator-specific measures.
- `over`: A character vector specifying the column names of the population subgroups (e.g., sex) in data, for which AF measures will be estimated across their respective levels of analysis (i.e., male and female).

Further details:

- The `measures` and `indmeasures` arguments allow users to avoid unnecessary estimations. If any of these arguments is `NULL`, `mpitb.est()` skips these measures. For instance, if `measures = c("M0", "H", "A")` and `indmeasures = NULL`, only the M_0 , H , and A will be estimated.

- If `indmeasures` is specified, but `indklist` is not, `indklist` defaults to the poverty cut-offs provided in `klist`.
- The `svycoprop` function from the [survey](#) package is used to estimate confidence intervals, treating the measures as proportions. This function employs the ‘logit’ method, which is often preferred for proportions as it helps to keep the confidence intervals within the valid 0 to 1 range. The ‘logit’ method involves fitting a logistic regression model and calculating a Wald-type interval on the log-odds scale, which is then transformed back to the probability scale.

Changes over time analysis

[mpitbR](#) allows users to analyze how multidimensional poverty changes across different time periods in the survey data. This requires specifying a few additional arguments:

- `tvar`: A character vector indicating the column name in the data that identifies the time period or survey round. This argument is always required for changes over time analysis.
- `cotyear` (optional): A character vector indicating the column name in the data that contains the year information for each survey round (decimal values allowed). This is required for annualized measures.
- `cotmeasures` (optional): A character vector specifying the particular AF measures to estimate for changes over time. By default, all AF measures are calculated. However, users can save computational time by specifying preferred measures. For example, if `cotmeasures = c("M0", "H", "A")` only changes in M_0 , H and A will be calculated.
- `cotklist` (optional): A numeric vector specifying poverty cut-offs for the changes over time analysis. If not provided, it defaults to the cut-offs provided in `klist`.
- `cotoptions` (optional): This argument is used only if users have more than two survey rounds. It determines how changes are estimated:
 - “total” (default): Estimates changes over the entire observation period.
 - “insequence”: Estimates year-to-year changes.

Further details:

The standard errors of changes-over-time measures are estimated using the Delta method (detailed in `svycontrast()` function from [survey](#) package).

Additional control arguments

- `ann` and `noraw` (optional): Logical values that control whether to estimate annualized and non-annualized changes-over-time measures. By default, both are FALSE (non-annualized).
 - If `cotyear` is provided (year information), `ann` is automatically set to TRUE (annualized).
 - If `cotyear` has values but `ann` is FALSE, only non-annualized measures are estimated.
 - If only annualized measures are desired, users can set `noraw` to TRUE in order to avoid non-annualized calculations.
- `nooverall` (optional): A logical value that controls whether to estimate poverty measures for the overall data (e.g., national level). By default, it is FALSE (estimates overall). Setting `nooverall` to TRUE skips overall estimates.
- `level` (optional): Specifies the confidence level for estimated confidence intervals (defaults to 95%). Users can adjust this value.
- `multicore` (optional): Enables parallel computations for all measures and poverty cut-offs. This argument uses the “Forking” method. Hence, this option is only available on Unix-like operating systems.
- `verbose` (optional): A logical value that controls whether to print messages or not. By default, it is TRUE.

Table 1: Comparison of R Packages for Multidimensional Poverty Calculation

Feature	mpitbR	mpindex	MPI
Estimate AF disaggregated measures	Yes	Yes	Yes
Include changes over time analysis	Yes	No	No
Account for survey design	Yes	No	No
Flexibility	Medium (mirrors UNPD methodology)	High (permits building the deprivations matrix from data)	Medium (works directly with the deprivations matrix)
Ease of use	Medium (few functions with large number of arguments)	Low (MPI specifications from external files (.txt, .csv, .json))	High (few functions with small number of arguments)
Include R build-in functions for analyzing results	Yes	No	No

3.3 Other functions

The output of `mpitb.est()` function mirrors the structure of the Stata package. It is a two-element list where each element is a `data.frame` containing the estimated values. These data frames follow the same format as the Stata package output.

Then, users can apply functions such as `coef`, `confint`, and `summary`. Since the output data may contain multiple AF measures, by cut-offs and levels of the subgroups and possibly by different time periods, it is recommended to filter the data for specific analyses with these functions.

`summary` function performs a t-test on the estimates, inheriting the confidence level in the `level` argument of `mpitb.est` function. This function is particularly helpful for the changes-over-time estimates, where the user can infer if there has been a statistically significant reduction in multidimensional poverty between two time periods. However, when conducting changes-over-time analysis, the `summary` function requires filtering the output to a single AF measure, one change-over-time measure, and one poverty cut-off to perform a distinct statistical test.

The next section provides examples of how to implement these functions in practice using a household survey.

3.4 Comparison with other packages

While `mpindex` and `MPI` also calculate AF-based multidimensional poverty indices, including their disaggregation properties, a crucial advantage of `mpitbR` lies in the fact that it accounts for the household survey design. This allows for rigorous statistical inference, a feature conspicuously absent in `mpindex` and `MPI`. The ability to perform such analysis becomes particularly crucial for understanding poverty dynamics over time, a functionality uniquely offered within the `mpitbR` ecosystem.

Due to its dependence on the `survey` package functions and its aim to mirror the Stata `mpitb` workflow, `mpitbR` suffers from a degree of inflexibility. However, this is largely compensated by its consistency with the official calculation methodology used by the United Nations. Conversely, `mpindex` and `MPI` offer greater flexibility in their implementation. In addition, `mpindex` allows for building the deprivations matrix from data within its workflow, unlike the other two packages which require a predefined deprivations matrix for the calculations.

However, the `mpindex` package can be harder to implement since all the specifications of the MPI calculation (dimensions, indicators, weights, and poverty cut-offs) have to be set out from the R environment (using `.txt`, `.csv`, or `.json` files). In contrast, `MPI` and `mpitbR` packages are fairly easier to use, although `mpitbR` demands a steeper learning curve due to the large number of arguments.

Last but not least, `mpitbR` leverages R’s built-in functions for analyzing results, such as `coef()`, `confint()`, and `summary()`, a feature that is totally absent in the other two packages. Table 1 summarizes these comparisons.

4 How to use the package

To bridge the gap between theory and practice, this section provides illustrative examples aiming to showcase the application of the `mpitbR` package for common exercises encountered by researchers in real-world multidimensional poverty analysis. We use the `syn_cdta` dataset, a synthetic dataset with a household survey structure, also used in the original `mpitb` Stata package examples by Suppa (2023). This allows for a direct comparison between the Stata `mpitb` framework and the `mpitbR` package.

Table 2: A synthetic household survey data

d_nutr	d_cm	d_satt	d_educ	d_elct	d_sani	d_wtr	d_hsg	d_ckfl	d_asst	area	region	stratum	psu	weight	year	t
0	0	1	0	0	0	0	1	0	0	1	4	401	401000	1	2010	1
0	0	0	0	1	0	0	1	1	1	1	1	104	104003	1	2010	1
0	0	0	0	1	0	0	0	0	0	1	20	2002	2002005	1	2010	1
0	0	0	0	0	0	0	1	0	0	1	20	2004	2004002	1	2010	1
0	0	1	0	1	0	0	1	0	0	1	18	1805	1805000	1	2010	1
0	0	1	0	1	1	0	1	0	1	0	18	1803	1803001	1	2010	1

4.1 The syn_cdta Dataset

The syn_cdta dataset included in the mpitbR package comprises 15,000 observations, representing individuals, and includes 17 variables. These variables can be categorized as follows:

- **Ten Binary Indicator Variables:** These variables represent different dimensions typically measured in the Multidimensional Poverty Index (MPI). Examples might include Nutrition, Child Mortality, School Attendance, and so on. For detailed descriptions of the specific MPI dimensions included, please refer to the [UNDP and OPHI \(2022\)](#).
- **Two Subgroup Variables:** The dataset includes two categorical variables, “area” and “region”, that can be used for subgroup analysis.
- **Survey design variables:** Three additional variables, “psu”, “stratum”, and “weight”, define the complex survey design.
- **Survey Round and Year:** Two additional variables, “t” and “year” respectively, identify the survey round and year of data collection for each observation.

For a detailed list of variables refer to Table 2.

4.2 Example 1: Cross-Sectional Estimations for a Single Country

First of all, we load the required packages.

```
library(survey)
library(mpitbR)
```

The analysis is restricted to the first survey round (i.e., subsetting data with `t == 1`), and observations with missing values in some deprivation indicators are removed, as the [survey](#) package methods do not support missing values for standard error calculations.

```
# Keep observation from the first survey round
first_round <- subset(syn_cdta, t == 1)
# Drop NA in deprivation indicators columns
first_round <- na.omit(first_round)
```

Next, we define the survey design using the `svydesign` function from the [survey](#) package, considering the primary sampling units (psu), weights (weight), and strata (stratum) information in the data.

```
# Define survey design
svydata <- svydesign(id=~psu, weights = ~weight, strata = ~stratum, data = first_round)
```

To specify the MPI estimation settings, we use the `mpitb.set` function. The MPI typically considers three core dimensions: Health (hl), Education (ed), and Living Standards (ls). Here, we define indicators for each dimension:

- **Health:** `d_nutr` (deprivation in Nutrition indicator) and `d_cm` (child mortality indicator)
- **Education:** `d_satt` (deprivation in School Attendance indicator) and `d_educ` (deprivation in Years of Schooling indicator)
- **Living Standards:** `d_elct` (deprivation in Electricity Access indicator), `d_sani` (deprivation in Improved Sanitation indicator), `d_wtr` (deprivation in Drinkable Water Access indicator), `d_hsg` (deprivation in Housing Conditions indicator), `d_ckfl` (deprivation in Cooking Fuel indicator), and `d_asst` (deprivation in Assets Ownership indicator)

```
# Define the dimensions of poverty with their corresponding indicators
indicators <- list(hl = c("d_nutr", "d_cm"),
                  ed = c("d_satt", "d_educ"),
                  ls = c("d_elct", "d_sani", "d_wtr", "d_hsg", "d_ckfl", "d_asst"))
# Set the multidimensional poverty project
set <- mpitb.set(svydata, indicators = indicators,
                 name = "trial01", desc = "pref. spec")
```

Finally, we define a short name ("trial01") and description ("pref. spec") for our project using `mpitb.set`.

For this first round, assume we want to estimate all the AF measures. These include the aggregate measures (M_0 , H , and A) and the indicator-specific measures (h_d , $h_d(k)$, $actb$, and $pctb$). We prefer an equal-nested weighting scheme (equal weights for all dimensions and equal indicator weights within dimensions). Additionally, for each measure, we set three poverty cut-offs: 20%, 33%, and 50%. Finally, we want to calculate the disaggregated measures by subnational regions and living areas (urban and rural), accounting for the complex design of the survey.

This information is specified in the `mpitb.est` function as follows:

```
est <- mpitb.est(set = set, klist = c(20, 33, 50),
                 weights = "equal",
                 measures = c("M0", "H", "A"),
                 indmeasures = c("hd", "hdk", "actb", "pctb"),
                 over = c("area", "region"))

#>          ***** SPECIFICATION *****
#> Call:
#> mpitb.est.mpitb_set(set = set, klist = c(20, 33, 50), weights = "equal",
#>   measures = c("M0", "H", "A"), indmeasures = c("hd", "hdk",
#>   "actb", "pctb"), over = c("area", "region"))
#> Name: trial01
#> Weighting scheme: equal
#> Description: pref. spec
#> -----
#>
#> Dimension 1: hl 0.333 (d_nutr, d_cm)
#> Dimension 2: ed 0.333 (d_satt, d_educ)
#> Dimension 3: ls 0.333 (d_elct, d_sani, d_wtr, d_hsg, d_ckfl, d_asst)
#> -----
#>
#> Indicator 1: d_nutr 0.1667
#> Indicator 2: d_cm 0.1667
#> Indicator 3: d_satt 0.1667
#> Indicator 4: d_educ 0.1667
#> Indicator 5: d_elct 0.0556
#> Indicator 6: d_sani 0.0556
#> Indicator 7: d_wtr 0.0556
#> Indicator 8: d_hsg 0.0556
#> Indicator 9: d_ckfl 0.0556
#> Indicator 10: d_asst 0.0556
#>
#>          ***** ESTIMATION *****
#> -----
#> Partial AF measures: ' M0 H A ' under estimation... DONE
#>
#> -----
#> Indicator-specific measures: ' hd hdk actb pctb ' under estimation... DONE
#>
#>          ***** RESULTS *****
#> -----
#> Parameters
#> Subgroups: 3
#> Poverty cut-offs (k): 3
#>
#> *Notes:
```

```
#> Confidence level: 95 %
#> Parallel estimations: FALSE
```

The `mpitb_set`-class object is provided in the `set` argument. The poverty cut-offs are specified in `klist` and the population subgroups in `over`. The equal-nested weighting scheme is specified in the `weights` argument by passing the character string `"equal"`. The `measures` argument specifies the aggregate measures, and `indmeasures` specifies the indicator-specific measures we want to estimate. However, by default, all these measures and equal-nested weights are calculated, so these arguments can be omitted in this case.

Users can verify the specification of their project using the messages returned by `mpitb.est`. These messages report:

1. The function call, verifying correct argument assignment.
2. The dimensions with their assigned indicators and corresponding weights.
3. Which measures are being estimated.
4. The estimation parameters (number of poverty cut-offs and subgroups).
5. Other features such as the confidence level and whether parallel estimation has been used.

All these messages can be suppressed using `verbose = FALSE` in `mpitb.est` function.

Recall that the `mpitb.est` function returns a two-element list (of class `"mpitb_est"`), where each element is a data frame containing all the estimates. The first element, named `"lframe"`, includes all the cross-sectional estimates for each level of analysis. The second element, named `"cotframe"`, contains all the changes over time measures for each level of analysis. In this first example, since we are not analyzing changes over time, `"cotframe"` will be `NULL`.

For this instance, the first rows of the `lframe` data frame can be examined using `head(est$lframe)`.

```
head(est$lframe)

#>      b      se      ll      ul subg  loa measure ctype k
#> 1 0.2169686 0.002363434 0.2123628 0.2216461 nat  nat      M0 lev 20
#> 2 0.2165175 0.003493254 0.2097356 0.2234567 0   area      M0 lev 20
#> 3 0.2172704 0.002873528 0.2116799 0.2229667 1   area      M0 lev 20
#> 4 0.2229081 0.011585856 0.2009836 0.2464864 1 region      M0 lev 20
#> 5 0.2111428 0.010886855 0.1905533 0.2333157 2 region      M0 lev 20
#> 6 0.2302469 0.013024076 0.2056679 0.2568137 3 region      M0 lev 20
#> indicator
#> 1      <NA>
#> 2      <NA>
#> 3      <NA>
#> 4      <NA>
#> 5      <NA>
#> 6      <NA>
```

Each row in this data frame represents an estimate for a specific measure and population subgroup. In this example, with three poverty cut-offs, two subgroups, and seven measures (three aggregate and four indicator-specific), we have a total of 2507 estimates. This structure closely mirrors the output of the Stata package. The first columns include the most important features such as the point estimate (`b`), its corresponding standard error (`se`) that accounts for the complex survey design, and the lower and upper confidence limits (`ll` and `ul`) calculated by default at a 95% confidence level. The remaining columns specify the corresponding AF measure (`measure`), its corresponding indicator (if applicable), its poverty cut-off (`k`), its subgroup level of analysis (`loa`) and its subgroup category (`subg`).

Since the elements of the list are data frames, users can easily subset the `'lframe'` or `'cotframe'` elements to examine point estimates or confidence intervals of specific subgroups and measures of interest. For instance, suppose we want to see the confidence intervals for the incidence (*H*) measure for the "area" subgroups at the 33% poverty cut-off. The code for this would be:

```
confint(subset(est$lframe, measure == "H" & loa == "area" & k == 33))

#> Subgroup Level of analysis Cut-off Lower Bound (95%) Upper Bound (95%)
#> 1      0      area      33      0.325      0.358
#> 2      1      area      33      0.317      0.345
```

Finally, `summary` can be used to calculate a t-test statistic for each point estimate in the data frames, along with the corresponding p-value and significance level. This is particularly useful for inferring whether a change-over-time measure is statistically different from zero. We will provide an example of this functionality below.

4.3 Example 2: Saving Time with Package Options

Efficiently prioritizing estimates can save time when dealing with numerous parameters like poverty cut-offs and population subgroups. As observed in the previous example, the number of estimates can increase rapidly even with a small number of parameters.

One way to optimize estimations is to consider the characteristics of the measures themselves. For instance, deprivation scores, c_i and $c_i(k)$, reach a finite number of values for fixed weights and cut-offs. Therefore, it is important to be aware of the included cut-off values to avoid redundant estimations. This becomes even more relevant when incorporating population subgroups, which significantly contribute to the growth in estimates.

The `mpitbR` package provides options to control which measures are estimated and the poverty cut-offs used. The `klist` argument specifies cut-offs for aggregate measures, while `indklist` allows defining a separate list for indicator-specific measures.

In the following code, we replicate the previous example with a few adjustments to demonstrate these time-saving options:

```
mpitb.est(set = set, klist = c(20, 33, 50),
          weights = "equal", measures = c("M0"),
          indmeasures = c("hd", "hdk"), indklist = c(33),
          over = c("area", "region"), nooverall = TRUE)
```

Here, we focus on the M_0 measure and exclude the H and A measures, along with contribution measures. Additionally, we specify a different cut-off value (33%) for indicator-specific measures and avoid national level estimates. As a result, the number of estimates is significantly reduced compared to the previous example (946 vs. 2507).

4.4 Example 3: Specify Alternative Weighting Schemes

In the previous examples, we assumed equal weights across dimensions and indicators (equal-nested weights). This can be achieved by passing the indicators grouped in a list to the `mpitb.set` function and setting `weights = "equal"` in the `mpitb.est` function (the default).

An alternative approach is to specify weights as a numeric vector in the `weights` argument. This can be more efficient for defining complex weighting schemes. For example, the previously employed equal nested weights, using a numerical vector, can be expressed as follows:

```
mpitb.est(set = set, klist = c(33),
          measures = c("M0"), indmeasures = NULL,
          weights = c(1/6, 1/6, 1/6, 1/6,
                     1/18, 1/18, 1/18, 1/18, 1/18, 1/18))
```

We can also assign different weights to each dimension. For instance, if we assign a 50% weight to the Health dimension and a 25% weight to each of the remaining dimensions, with equal weights within each dimension, the weights vector will be:

```
mpitb.est(set = set, klist = c(33),
          measures = c("M0"), indmeasures = NULL,
          weights = c(1/4, 1/4, 1/8, 1/8,
                     1/24, 1/24, 1/24, 1/24, 1/24, 1/24))
```

This approach allows users to readily assess the impact of adding/dropping indicators, merging indicators, or using different deprivation thresholds. Specifying alternative weighting schemes is often used during MPI construction. This approach serves two purposes: exploring parsimony of the measure and comparing orderings under different weighting scenarios (Santos and Villatoro, 2016; Alkire et al., 2022).

The following code demonstrates how to modify the MPI calculations by removing the electricity indicator (`d_elct`) from the analysis.

```
indicators <- list(hl = c("d_nutr", "d_cm"),
                  ed = c("d_satt", "d_educ"),
                  ls = c("d_sani", "d_wtr", "d_hsg", "d_ckfl", "d_asst"))
```

```
set <- mpitb.set(syn_cdta, indicators = indicators,
               name = "trial02", desc = "w/o electricity")

est <- mpitb.est(set = set, klist = c(20, 33, 50), weights = "equal",
               over = c("area", "region"))
```

4.5 Example 4: Changes over time analysis

In order to analyse pro-poor changes over time, we will use the entire data set to define our survey design with the difference that we have to specify which column indexes the survey rounds in the `tvar` argument of the `mpitb.est` function.

```
# Drop NA in deprivation indicators columns
syn_cdta <- na.omit(syn_cdta)
# Define survey design
svydata <- svydesign(id=~psu, weights = ~weight, strata = ~stratum, data = syn_cdta)

# Define list of indicators
indicators <- list(hl = c("d_nutr", "d_cm"),
                  ed = c("d_satt", "d_educ"),
                  ls = c("d_elct", "d_sani", "d_wtr", "d_hsg", "d_ckfl", "d_asst"))

# Specify MPI estimates setting
set <- mpitb.set(svydata, indicators = indicators,
               name = "trial01", desc = "Estimate changes over time")
```

We are interested in estimating AF measures for each population subgroup across both years and assessing whether poverty has decreased significantly. The following code estimates M_0 , H , and A measures for each year, along with measures of poverty changes over time, both at the regional and national levels. Setting `indmeasure = NULL` avoids estimating indicator-specific measures by levels. We restrict the analysis of changes over time to the AF measure calculated by each year: `c(M0, H, A)`. Remind that for estimating annualized measures, we need to specify the year column in `cotyear` argument.

```
# Estimation
est <- mpitb.est(set, klist = c(1, 33, 50),
               indmeasures = NULL,
               cotmeasures = c("M0", "A", "H"),
               over = c("region"), tvar = "t", cotyear = "year")

#>          ***** SPECIFICATION *****
#> Call:
#> mpitb.est.mpitb_set(set = set, klist = c(1, 33, 50), indmeasures = NULL,
#>   over = c("region"), cotyear = "year", tvar = "t", cotmeasures = c("M0",
#>   "A", "H"))
#> Name: trial01
#> Weighting scheme: equal
#> Description: Estimate changes over time
#> -----
#>
#> Dimension 1: hl 0.333 (d_nutr, d_cm)
#> Dimension 2: ed 0.333 (d_satt, d_educ)
#> Dimension 3: ls 0.333 (d_elct, d_sani, d_wtr, d_hsg, d_ckfl, d_asst)
#> -----
#>
#> Indicator 1: d_nutr 0.1667
#> Indicator 2: d_cm 0.1667
#> Indicator 3: d_satt 0.1667
#> Indicator 4: d_educ 0.1667
#> Indicator 5: d_elct 0.0556
#> Indicator 6: d_sani 0.0556
#> Indicator 7: d_wtr 0.0556
#> Indicator 8: d_hsg 0.0556
```

```
#> Indicator 9: d_ckfl 0.0556
#> Indicator 10: d_asst 0.0556
#>
#> ***** ESTIMATION *****
#> -----
#> Partial AF measures: ' M0 H A ' under estimation... DONE
#>
#> -----
#> Estimate changes over time over ' M0 A H ' measures... DONE
#>
#> ***** RESULTS *****
#> -----
#> Parameters
#> Number of time periods: 2
#> Subgroups: 2
#> Poverty cut-offs (k): 3
#>
#> *Notes:
#> Confidence level: 95 %
#> Parallel estimations: FALSE
```

Suppose we want to examine if the number of poor people has decreased overall for a poverty cut-off of 33%. We can observe the coefficients of the incidence of poverty at the national level.

```
coef(subset(est$lframe, k == 33 & measure == "H" & loa == "nat"))
```

```
#> Subgroup Level of analysis Cut-off Coefficient
#> 1 nat nat 33 0.335
#> 2 nat nat 33 0.231
```

In this case, we observe that the national Incidence of poverty, *H*, decreased from 33.5% to 23.1%. Examining the confidence intervals, if they do not overlap, it suggests a significant reduction of poverty (assuming a 95% confidence level by default). To do so, we use the `confint` function.

```
confint(subset(est$lframe, k == 33 & measure == "H" & loa == "nat"))
```

```
#> Subgroup Level of analysis Cut-off Lower Bound (95%) Upper Bound (95%)
#> 1 nat nat 33 0.324 0.346
#> 2 nat nat 33 0.222 0.240
```

If we want to investigate if this national-level tendency holds at each regional level, we can use the `summary` function. This allows us to explore the statistical significance of the changes-over-time measures (here, we focus on non-annualized absolute and relative measures, as annualized measures are typically used for cross-country comparisons).

```
summary(subset(est$otframe, k == 33 &
  measure == "H" & loa == "region" &
  ann == 0 & ctype == "abs"))

#>
#> Measure: H
#> Coefficients:
#> Non-annualized Absolute change over time measures:
#> Poverty cut-off: 33 %
#> Estimate Std.Err t-value Pr(>|t|)
#> 1.region -0.08364 0.03035 -2.755 0.005862 **
#> 10.region -0.13474 0.03205 -4.205 2.62e-05 ***
#> 11.region -0.08914 0.03375 -2.641 0.008259 **
#> 12.region -0.07353 0.03510 -2.095 0.036194 *
#> 13.region -0.08019 0.03333 -2.406 0.016137 *
#> 14.region -0.09188 0.03130 -2.935 0.003334 **
#> 15.region -0.11548 0.03929 -2.939 0.003290 **
#> 16.region -0.14710 0.03152 -4.667 3.06e-06 ***
```

```

#> 17.region -0.09310 0.03159 -2.948 0.003203 **
#> 18.region -0.03016 0.03137 -0.961 0.336415
#> 19.region -0.15718 0.02635 -5.965 2.44e-09 ***
#> 2.region -0.10215 0.02797 -3.653 0.000259 ***
#> 20.region -0.10135 0.02623 -3.864 0.000112 ***
#> 3.region -0.13947 0.03469 -4.020 5.81e-05 ***
#> 4.region -0.07307 0.03677 -1.987 0.046893 *
#> 5.region -0.13783 0.03228 -4.270 1.95e-05 ***
#> 6.region -0.10612 0.02689 -3.946 7.94e-05 ***
#> 7.region -0.07524 0.03829 -1.965 0.049445 *
#> 8.region -0.10993 0.03931 -2.797 0.005164 **
#> 9.region -0.16716 0.03465 -4.824 1.41e-06 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

summary(subset(est$coef, k == 33 &
               measure == "H" & loa == "region" &
               ann == 0 & ctype == "rel"))

#>
#> Measure: H
#> Coefficients:
#> Non-annualized Relative change over time measures:
#> Poverty cut-off: 33 %
#>      Estimate Std.Err t-value Pr(>|t|)
#> 1.region    -22.887   7.277  -3.145 0.001660 **
#> 10.region   -35.796   6.540  -5.474 4.41e-08 ***
#> 11.region   -28.953   9.485  -3.052 0.002270 **
#> 12.region   -22.727   9.480  -2.397 0.016510 *
#> 13.region   -25.838   8.915  -2.898 0.003751 **
#> 14.region   -25.986   7.566  -3.434 0.000594 ***
#> 15.region   -36.113  10.161  -3.554 0.000380 ***
#> 16.region   -43.751   6.901  -6.340 2.30e-10 ***
#> 17.region   -29.186   8.550  -3.413 0.000641 ***
#> 18.region   -10.463  10.395  -1.007 0.314166
#> 19.region   -43.284   5.593  -7.739 1.00e-14 ***
#> 2.region    -32.303   6.917  -4.670 3.01e-06 ***
#> 20.region   -31.130   7.023  -4.433 9.31e-06 ***
#> 3.region    -39.850   7.541  -5.285 1.26e-07 ***
#> 4.region    -24.271  10.657  -2.277 0.022760 *
#> 5.region    -40.165   7.565  -5.309 1.10e-07 ***
#> 6.region    -28.963   6.378  -4.541 5.60e-06 ***
#> 7.region    -23.366  10.622  -2.200 0.027816 *
#> 8.region    -32.528   9.030  -3.602 0.000315 ***
#> 9.region    -42.857   6.797  -6.305 2.88e-10 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

These results suggest that there have been statistically significant reductions in the incidence of poverty between the two time periods (with different levels of confidence) in each region, with the exception of region 18, where we cannot conclude that the number of multidimensional poor people has been reduced.

5 Summary

Multidimensional Poverty Indices (MPIs) are crucial tools for measuring poverty beyond sheer income, considering multiple deprivations individuals face. This paper has introduced the **mpitbR** package, a user-friendly toolkit for estimating MPIs based on the Alkire-Foster method. This package offers researchers and practitioners a comprehensive framework for multidimensional poverty measurement.

mpitbR is tailored to the specific needs of the global MPI workflow, reflecting the functionality of the *mpitb* Stata package. This includes data requirements, standard outputs, and relevant analysis tools. The unified output structure, while mirroring the original package, constrains the development

of straightforward built-in plot functions for visualizing the estimated results, forcing a trade-off between simplicity and the extensive conditional logic required to adapt generic functions to various Alkire-Foster measures included in the output. However, **mpitbR** holds significant potential for further development.

Future versions of the package will incorporate functions for robustness analysis (to assess the sensitivity of the results to specific choices, such as poverty cut-offs or weights), interlinkages analysis (to explore which combinations of deprivations are more prevalent among the poor) and visualizing results (to explore data and communicate findings, maximizing user convenience despite the output complexity).

Additionally, implementing functions for panel data analysis within the Alkire-Foster framework would be valuable for longitudinal studies. Including novel methods for analyzing inequality among the poor within the AF method would also be highly beneficial.

Ultimately, the future development of **mpitbR** will also be guided by user needs, ongoing research advancements, and available resources. In sum, by providing a powerful and adaptable toolkit, **mpitbR** actively contributes to the field of multidimensional poverty measurement.

6 Acknowledgements

I would like to express my sincere thanks both to Rodrigo García Arancibia and José Vargas, my mentors, for their invaluable guidance, feedback, and encouragement throughout the development of this package. I also wish to acknowledge Nicolai Suppa for his generous support and enthusiasm in adapting his Stata package to R users. Without his constructive suggestions this package would not have been possible. Finally, I would like to thank every instructor from the OPHI Summer School 2022, for their thorough lectures on the Alkire-Foster method for multidimensional poverty measurement and the provision of the Stata source code which mainly motivated this project.

References

- B. Abdulsamad. *mpindex: Multidimensional Poverty Index (MPI)*, 2024. URL <https://CRAN.R-project.org/package=mpindex>. R package version 0.2.1. [p94]
- S. Alkire. *Multidimensional Poverty Measures as Policy Tools*, chapter Dimensions of Poverty: Measurement, Epistemic Injustices, Activism, pages 197–214. Philosophy and Poverty. Springer, 2021. ISBN 9783030317102. [p94, 96]
- S. Alkire and J. Foster. Counting and multidimensional poverty measurement. *Journal of Public Economics*, 95(7-8):476–487, 08 2011. doi: 10.1016/j.jpubeco.2010.11.006. URL <http://dx.doi.org/10.1016/j.jpubeco.2010.11.006>. [p94, 95]
- S. Alkire, J. Foster, S. Seth, M. E. Santos, J. M. Roche, and P. Ballon. *Multidimensional Poverty Measurement and Analysis*. Oxford University Press, 2015. URL <https://doi.org/10.1093/acprof:oso/9780199689491.001.0001>. [p95]
- S. Alkire, J. M. Roche, and A. Vaz. Changes Over Time in Multidimensional Poverty: Methodology and results for 34 countries. *World Development*, 94:232–249, 2017. ISSN 0305-750X. doi: 10.1016/j.worlddev.2017.01.011. URL <https://www.sciencedirect.com/science/article/pii/S0305750X17300256>. [p96]
- S. Alkire, U. Kanagaratnam, R. Nogales, and N. Suppa. Revising the global multidimensional poverty index: Empirical insights and robustness. *Review of Income and Wealth*, 68(S2), 03 2022. doi: 10.1111/roiw.12573. URL <http://dx.doi.org/10.1111/roiw.12573>. [p104]
- I. Girela. *mpitbR: Calculate Alkire-Foster Multidimensional Poverty Measures*, 2025. URL <https://CRAN.R-project.org/package=mpitbR>. R package version 1.0.1. [p94]
- K. Kukiattikun and C. Chainarong. *MPI: Computation of Multidimensional Poverty Index (MPI)*, 2022. URL <https://CRAN.R-project.org/package=MPI>. R package version 0.1.0. [p94]
- T. Lumley. Analysis of Complex Survey Samples. *Journal of Statistical Software*, 9(8):1–19, 2004. doi: 10.18637/jss.v009.i08. [p94]
- T. Lumley. *Complex Surveys: A Guide to Analysis Using R: A Guide to Analysis Using R*. John Wiley and Sons, 2010. [p94]

- T. Lumley. *survey: Analysis of Complex Survey Samples*, 2024. URL <https://CRAN.R-project.org/package=survey>. R package version 4.4. [p94]
- OPHI. National MPI and policy. <https://ophi.org.uk/national-mpi-and-policy>. URL <https://ophi.org.uk/national-mpi-and-policy>. Accessed: 2024-06-15. [p94]
- M. E. Santos and P. Villatoro. A multidimensional poverty index for Latin America. *Review of Income and Wealth*, 64(1):52–82, 11 2016. doi: 10.1111/roiw.12275. URL <http://dx.doi.org/10.1111/roiw.12275>. [p104]
- N. Suppa. mpitb: A toolbox for multidimensional poverty indices. *The Stata Journal*, 23(3):625–657, 2023. doi: 10.1177/1536867X231195286. URL <https://doi.org/10.1177/1536867X231195286>. [p94, 100]
- UNDP and OPHI. 2022 Global Multidimensional Poverty Index (MPI). *UNDP (United Nations Development Programme)*, 2022. URL <https://hdr.undp.org/content/2022-global-multidimensional-poverty-index-mpi>. [p94, 101]

Ignacio Girela
CONICET, Universidad Nacional de Córdoba
Facultad de Ciencias Económicas, Bv. Enrique Barros s/n
Córdoba, Argentina
<https://github.com/girelaignacio>
ORCID: 0000-0003-3297-3854
ignacio.girela@unc.edu.ar

BayesPPDSurv: An R Package for Bayesian Sample Size Determination and Analysis Using the Power and Normalized Power Prior for Time-To-Event Data

by Yueqi Shen, Matthew A. Psioda and Joseph G. Ibrahim

Abstract The **BayesPPDSurv** (Bayesian Power Prior Design for Survival Data) R package supports Bayesian power and type I error calculations and model fitting using the power and normalized power priors incorporating historical data for the analysis of time-to-event outcomes. The package implements the stratified proportional hazards regression model with piecewise constant hazard within each stratum. The package allows the historical data to inform the treatment effect parameter, parameter effects for other covariates in the regression model, as well as the baseline hazard parameters. The use of multiple historical datasets is supported. A novel algorithm is developed for computationally efficient use of the normalized power prior. In addition, the package supports the use of arbitrary sampling priors for computing Bayesian power and type I error rates, and has built-in features that semi-automatically generate sampling priors from the historical data. We demonstrate the use of **BayesPPDSurv** in a comprehensive case study for a melanoma clinical trial design.

1 Introduction: BayesPPDSurv

The incorporation of historical information in clinical trial design has become increasingly popular due to its potential for making trials more efficient. If the historical trial is sufficiently similar to the current trial, one can achieve more accurate point estimates and increased power (Viele et al., 2014). One natural way of integrating historical information is through informative priors in a Bayesian framework. The power prior (Ibrahim and Chen, 2000) is a popular class of informative priors that allow the incorporation of historical data through a discounted likelihood. It is constructed by raising the historical data likelihood to a power a_0 , where $0 \leq a_0 \leq 1$. The discounting parameter a_0 can be fixed or modeled as random. When it is modeled as random and estimated jointly with other parameters of interest (denoted by θ), the normalized power prior (Duan et al., 2006) is recommended, as normalization is critical to enabling the prior to factor into a conditional distribution of θ given a_0 and a marginal distribution of a_0 . Some challenges of adopting methods for borrowing from historical data include the potential for type I error inflation with prior-data conflict and the difficulty in calibrating how much to borrow, e.g., the choice of a_0 in the case of the power prior.

The power prior is widely used due to its easy construction and intuitive interpretation (Ibrahim et al., 2015). The theoretical properties of the power prior and the normalized power prior have been extensively studied. For example, Ibrahim et al. (2003) show that the power prior is an optimal class of informative priors in the sense that it minimizes a convex sum of the Kullback–Leibler (KL) divergences between two posterior densities, in which one density is based on no incorporation of historical data, and the other density is based on pooling the historical and current data. Shen et al. (2026) show that the marginal posterior for the discounting parameter converges to a point mass at zero if there is any discrepancy between the historical and current data. They also show that the marginal posterior for a_0 does not converge to a point mass at one when the datasets are fully compatible, and yet, for an i.i.d. normal model and finite sample size, the marginal posterior for a_0 always has most of its mass around one when the datasets are fully compatible. Chen and Ibrahim (2006) and Shen et al. (2024a) establish the analytic connection between the power and the normalized power priors

and Bayesian hierarchical models, respectively. The normalizing constant in the normalized power prior is analytically intractable when there are covariates, except in the case of the normal linear model. [Carvalho and Ibrahim \(2021\)](#) propose a bisection-type algorithm for computing the normalizing constant. In our package, **BayesPPDSurv** (Bayesian Power Prior Design for Survival Data), we propose a novel algorithm that provides an arbitrarily accurate approximation to the normalized power prior for θ itself, avoiding the need to compute the normalizing constant and thus providing significant computational efficiency.

The focus of the **BayesPPDSurv** package is on applying the power prior and normalized power prior to time-to-event outcomes for sample size determination and/or data analysis. In particular, we implement the proportional hazards model ([Cox, 1972](#)) with piecewise constant baseline hazard ([Friedman, 1982](#)). Modeling the baseline hazard with a piecewise constant function is a widely used approach in Bayesian analysis ([Ibrahim et al., 2001](#)). Several R packages on CRAN implement the piecewise constant hazard model. The **pch** package ([Frumento, 2024](#)) implements the piecewise constant hazard model for censored and truncated data. The **gsDesign** package ([Anderson, 2024](#)) supports the piecewise constant hazard model for group sequential clinical trial design. These packages do not allow the inclusion of historical data. There are several R packages that implement the power prior or variations of the power prior for the purpose of model estimation or design. The **BayesCTDesign** package ([Eggleston et al., 2021](#)) supports two-arm randomized Bayesian trial design using historical control data with the power prior for a variety of outcome models, including the Weibull and piecewise constant hazard models for time-to-event outcomes. However, it does not allow using the historical data to inform the treatment effect parameter or parameter effects for other covariates. The **bayesDP** package ([Balcome et al., 2021](#)) implements the discounted power prior for single arm and two-arm clinical trials where the discounting parameter is determined by a discounting function estimated based on a measure of prior-data conflict. While it accommodates borrowing historical information for the treatment effect with the piecewise constant hazard model, it does not allow additional covariates, and it must be used in conjunction with the package **bayesCT** ([Chandereng et al., 2020](#)) for trial design. There are three R packages that implement the normalized power prior where a_0 is modeled as random. The **NPP** package ([Han et al., 2021](#)) supports posterior sampling using the normalized power prior for Bernoulli, normal, multinomial and Poisson models, as well as for the normal linear model. The **hdbayes** ([Alt, 2022](#)) package implements several methods that leverage historical data for generalized linear models, including the power prior and the normalized power prior. These two packages do not accommodate models for time-to-event outcomes, nor do they perform sample size determination. The R package **BayesPPD** ([Shen et al., 2023a](#)) supports trial analysis and design using the power prior and the normalized power prior for generalized linear models, but it does not support models for time-to-event outcomes.

Our package **BayesPPDSurv** ([Shen et al., 2024b](#)) addresses an important gap by providing a suite of functions for Bayesian power and type I error rate calculations and model fitting after incorporating historical data with the power prior and the normalized power prior for time-to-event data. It implements the stratified proportional hazards regression model with piecewise constant hazard within each stratum. The package allows the historical data to inform the treatment effect parameter, parameter effects for other covariates in the hazard ratio regression model, as well as the baseline hazard parameters. The time interval partition for the piecewise baseline hazards can be stratified. The discounting parameter a_0 can be fixed or modeled as random. The use of multiple historical datasets is supported. For sample size determination, we consider the simulation-based method developed in [Chen et al. \(2011\)](#) utilizing the sampling and fitting priors ([Wang and Gelfand, 2002](#)) as applied in [Psioda and Ibrahim \(2019\)](#). The package supports the use of arbitrary sampling priors for computing Bayesian power and type I error rates, and has built-in features that semi-automatically generate sampling priors from the historical data. **BayesPPDSurv** is computationally efficient. It implements the slice sampler ([Neal, 2003](#)) with Rcpp ([Eddelbuettel and Francois, 2011](#)), and functions for analysis take less than a minute to execute in most instances.

The rest of the article proceeds as follows. In section 2.2, we describe the theoretical details of the methods implemented in the package. In section 2.3, we provide details on using the package and its various features. In section 2.4, we present a comprehensive case study for a melanoma clinical trial design with example code. The article is concluded with a brief discussion.

2 Theoretical Framework

2.1 The Power Prior and the Normalized Power Prior

Let D denote data from the current study and D_0 denote data from a historical study. Let θ denote the model parameters and $L(\theta|D)$ denote a general likelihood function associated with a given outcome model, such as a generalized linear model (GLM) or a survival model. The power prior (Ibrahim and Chen, 2000) is defined as

$$\pi(\theta|D_0, a_0) \propto L(\theta|D_0)^{a_0} \pi_0(\theta),$$

where $0 \leq a_0 \leq 1$ is a discounting parameter for the historical data likelihood, and $\pi_0(\theta)$ is the initial prior for θ . The parameter a_0 allows one to control the influence of the historical data on the posterior distribution. When $a_0 = 0$, historical information is disregarded and the power prior becomes equivalent to the initial prior $\pi_0(\theta)$. Conversely, when $a_0 = 1$, the power prior corresponds to the posterior distribution of θ given the historical data and the initial prior.

The power prior can easily accommodate multiple historical datasets. Suppose there are J historical datasets denoted by D_{0j} for $j = 1, \dots, J$ and let $D_0 = (D_{01}, \dots, D_{0J})$. The power prior becomes

$$\pi(\theta|D_0, a_0) \propto \prod_{j=1}^J L(\theta|D_{0j})^{a_{0j}} \pi_0(\theta),$$

where $a_0 = (a_{01}, \dots, a_{0J})'$ are dataset-specific discounting parameters and $0 \leq a_{0j} \leq 1$ for $j = 1, \dots, J$.

Modeling a_0 as random allows one to represent uncertainty in how much the historical data should be discounted. Duan et al. (2006) introduce the *normalized power prior*, given by

$$\pi(\theta, a_0|D_0) = \pi(\theta|D_0, a_0) \pi(a_0) = \frac{L(\theta|D_0)^{a_0} \pi_0(\theta)}{c(a_0)} \pi(a_0),$$

where $\pi(a_0)$ is the initial prior for a_0 . The normalized power prior specifies a conditional prior for θ given a_0 and a marginal prior for a_0 . The normalizing constant,

$$c(a_0) = \int L(\theta|D_0)^{a_0} \pi_0(\theta) d\theta,$$

is often analytically intractable and requires Monte Carlo methods for estimation. In **BayesPPDSurv**, we implement a novel and computationally viable method for using the normalized power prior that avoids the need to compute the normalizing constant.

2.2 The Piecewise Constant Hazard Proportional Hazards (PWCH-PH) Model

In **BayesPPDSurv**, we implement the stratified proportional hazards model with piecewise constant baseline hazard within each stratum, which is a common approach for Bayesian analysis of time-to-event data (Ibrahim et al., 2001).

In Table 1, we provide a summary of the notation used in our model. In our implementation, we allow the incorporation of a stratification variable with S levels, where $s = 1, \dots, S$ is the stratum index. Let $\lambda_s(t)$ denote the baseline hazard and $\Lambda_s(t)$ denote the cumulative baseline hazard for stratum s . Let t_i denote the time to event and c_i denote the time to

censorship for subject i , $i = 1, \dots, n$. Suppose $\nu_i = I[t_i \leq c_i]$ denotes the indicator that an event is observed for subject i , $y_i = \min(t_i, c_i)$ denotes the individual's observation time, and $x_i = (x_{i1}, \dots, x_{iP})'$ denotes a $P \times 1$ vector of covariates, where without loss of generality we assume x_{i1} is the treatment indicator. If there are no additional covariates, x_i will consist of the treatment indicator only. The current data consists of $D = \{(y_i, x_i, \nu_i), i = 1, \dots, n\}$. Further, $\beta = (\beta_1, \dots, \beta_P)'$ denotes a $P \times 1$ vector of regression coefficients, where β_1 is the coefficient for the treatment indicator, and λ denotes the set of all baseline hazard parameters. Then, the likelihood of a stratified proportional hazards model is given by

$$L(\beta, \lambda|D) \propto \prod_{i=1}^n \{\lambda_{[s_i]}(y_i) \exp(x_i' \beta)\}^{\nu_i} \exp\{-\Lambda_{[s_i]}(y_i) \exp(x_i' \beta)\}.$$

Indices	
$s = 1, \dots, S$	stratum index
$i = 1, \dots, n$	subject index
$k = 1, \dots, K_s$	time interval index
$j = 1, \dots, J$	historical dataset index
G_s	the set of indices corresponding to subjects from stratum s .
Subject-level variables	
t_i	time to event for subject i
c_i	time to censorship for subject i
$\nu_i = I[t_i \leq c_i]$	the indicator that an event is observed for subject i
$y_i = \min(t_i, c_i)$	observation time for subject i
$x_i = (x_{i1}, \dots, x_{iP})'$	covariates for subject i
Interval-level variables	
$t_{s,0} < \dots < t_{s,K_s}$	change points for stratum s
$I_{s,k} = (t_{s,k-1}, t_{s,k}]$	the k -th interval for stratum s
ν_{ik}	the indicator that the event occurred in interval $I_{s_i,k}$
r_{ik}	the subject's time at risk in interval $I_{s_i,k}$
Parameters	
λ_{sk}	constant hazard over interval $I_{s,k}$ for stratum s
$\beta = (\beta_1, \dots, \beta_P)'$	regression coefficients
$a_0 = (a_{01}, \dots, a_{0J})$	the discounting parameters for the J historical datasets

Table 1: Summary of Notation

The package allows the baseline hazard and time interval partition to vary across the S levels of the stratification variable. For stratum s , we partition time into K_s intervals, where $k = 1, \dots, K_s$ is the interval index, with change points $0 = t_{s,0} < t_{s,1} < \dots < t_{s,K_s} = \infty$. Suppose $\lambda_{sk} > 0$ denotes the constant hazard over interval $I_{s,k} = (t_{s,k-1}, t_{s,k}]$ for stratum s , ν_{ik} denotes the indicator that the event occurred in interval $I_{s_i,k}$, r_{ik} denotes the subject's time at risk in interval $I_{s_i,k}$, and G_s denotes the set of indices corresponding to subjects from stratum s . Then the likelihood of a PWCH-PH model can be re-written as

$$L(\beta, \lambda|D) \propto \prod_{s=1}^S \prod_{k=1}^{K_s} \lambda_{sk}^{\sum_{i \in G_s} \nu_{ik}} \exp \left\{ -\lambda_{sk} \left(\sum_{i \in G_s} \exp(x_i' \beta) r_{ik} \right) \right\} \times \prod_{i=1}^n \exp(x_i' \beta)^{\nu_i}. \quad (1)$$

2.3 Power Prior for the PWCH-PH Model

Suppose there are J historical datasets, $j = 1, \dots, J$, each with sample size n_{0j} . Analogously, the j^{th} historical dataset consists of $D_{0j} = \{(y_{0ji}, x_{0ji}, \nu_{0ji}), i = 1, \dots, n_{0j}\}$, where y_{0ji} , x_{0ji} and ν_{0ji} correspond to the individual's observation time, covariate vector and event indicator, respectively, for subject i in historical data j . Suppose G_{sj} denotes the set of indices corresponding to subjects from stratum s in historical dataset j . Let ν_{0ik} denote the indicator that the event occurred in interval $I_{s_i,k}$ and r_{0ik} denote the subject's time at risk

in interval $I_{s_i,k}$ for subjects in the historical datasets. The baseline hazard for the historical data for stratum s and interval k is denoted by λ_{0sk} , and λ_0 denotes the set of all baseline hazard parameters for the historical data. Let $a_0 = (a_{01}, \dots, a_{0J})$ denote the discounting parameters for the J historical datasets, where a_{0j} discounts the j^{th} historical dataset. Then, the power prior for β is given by

$$\pi(\beta|D_0, \lambda_0, a_0) \propto \prod_{j=1}^J L(\beta|\lambda_0, D_{0j})^{a_{0j}} \times \pi_0(\beta) \quad (2)$$

$$\propto \prod_{j=1}^J \prod_{s=1}^S \prod_{k=1}^{K_s} \lambda_{0sk}^{a_{0j} \left(\sum_{i \in G_{sj}} \nu_{0ik} \right)} \exp \left\{ -a_{0j} \lambda_{0sk} \left(\sum_{i \in G_{sj}} \exp(x'_{0i} \beta) r_{0ik} \right) \right\} \times \prod_{j=1}^J \prod_{i=1}^{n_{0j}} \exp(x'_{0i} \beta)^{\nu_{0i} a_{0j}} \times \pi_0(\beta). \quad (3)$$

In this formulation, the current and historical datasets have different baseline hazard parameters (λ in (1) and λ_0 in (3)), so the historical data does not directly inform the inference for λ . If we assume the current and historical data share the same set of baseline hazard parameters, then $\lambda_0 = \lambda$, and the historical information is used to estimate λ directly. In **BayesPPDSurv**, when the power prior is used with functions `phm.fixed.a0` and `power.phm.fixed.a0`, one can use the `shared.blh` argument to choose whether the baseline hazard parameters are shared. By default, `shared.blh=FALSE`. When the normalized power prior is used with functions `phm.random.a0` and `power.phm.random.a0`, the baseline hazard parameters have to be unshared.

For the initial prior on β , the package allows several choices of priors, including the uniform improper prior and independent normal priors for each component of β . For the priors on the λ_{sk} 's, the package allows independent Gamma priors, independent normal priors on $\log(\lambda_{sk})$, and the improper prior $\pi(\lambda) \sim \prod_{s=1}^S \prod_{k=1}^K \lambda_{sk}^{-1}$. If the baseline hazard parameters are not shared between the current and historical data, the prior choices supported for λ_0 are independent Gamma priors, independent normal priors on $\log(\lambda_{0sk})$, and the improper prior $\pi(\lambda_0) \sim \prod_{s=1}^S \prod_{k=1}^K \lambda_{0sk}^{-1}$.

2.4 Implementing the Normalized Power Prior for the PWCH-PH Model

BayesPPDSurv supports the use of a normalized power prior for the case where a_0 is modeled as random and the baseline hazard parameters are *unshared* between the current and historical data (i.e., $\lambda \neq \lambda_0$). We use independent normal initial priors for β , $N(\mu_p, \sigma_p^2)$ for each β_p , independent gamma priors for λ and λ_0 and independent beta priors for a_0 . Specifically, we use

$$\lambda_{sk} \sim \text{Gamma}(a_{sk}, b_{sk}),$$

$$\lambda_{0sk} \sim \text{Gamma}(c_{sk}, d_{sk}),$$

and

$$a_{0j} \sim \text{beta}(u_j, v_j),$$

where a_{sk} , b_{sk} , c_{sk} , d_{sk} , u_j and v_j are user-specified hyperparameters. The joint prior for β , λ , λ_0 and a_0 can now be written as

$$\pi(\beta, \lambda, \lambda_0, a_0|D_0) \propto \pi(\lambda_0|\beta, a_0, D_0) \times \pi(\beta|D_0, a_0) \times \pi(a_0) \times \pi(\lambda).$$

We first factor $\pi(\beta, \lambda_0 | D_0, a_0)$ into $\pi(\beta | D_0, a_0) \times \pi(\lambda_0 | \beta, a_0, D_0)$. We have

$$\begin{aligned} \pi(\beta, \lambda_0 | D_0, a_0) &\propto \prod_{j=1}^J L(\beta, \lambda_0 | D_{0j})^{a_{0j}} \pi_0(\beta) \pi(\lambda_0) \\ &= \prod_{s=1}^S \prod_{k=1}^{K_s} \lambda_{0sk}^{\sum_{j=1}^J a_{0j} \left(\sum_{i \in G_{sj}} \nu_{0ik} \right)} \exp \left\{ - \sum_{j=1}^J a_{0j} \lambda_{0sk} \left(\sum_{i \in G_{sj}} \exp(x'_{0i} \beta) r_{0ik} \right) \right\} \frac{d_{sk}^{c_{sk}}}{\Gamma(c_{sk})} \lambda_{0sk}^{c_{sk}-1} \exp(-d_{sk} \lambda_{0sk}) \\ &\quad \times \prod_{p=1}^P N(\mu_p, \sigma_p^2) \times \prod_{j=1}^J \prod_{i=1}^{n_{0j}} \exp(x'_{0i} \beta)^{\nu_{0i} a_{0j}} \\ &= \prod_{p=1}^P N(\mu_p, \sigma_p^2) \prod_{j=1}^J \prod_{i=1}^{n_{0j}} \exp(x'_{0i} \beta)^{\nu_{0i} a_{0j}} \prod_{s=1}^S \prod_{k=1}^{K_s} \frac{d_{sk}^{c_{sk}}}{q_{sk}^{p_{sk}}} \Gamma(p_{sk}) \times \text{Gamma}(\lambda_0; p_{sk}, q_{sk}), \end{aligned}$$

where $p_{sk} = \sum_{j=1}^J a_{0j} \left(\sum_{i \in G_{sj}} \nu_{0ik} \right) + c_{sk}$ and $q_{sk} = \sum_{j=1}^J a_{0j} \left(\sum_{i \in G_{sj}} \exp(x'_{0i} \beta) r_{0ik} \right) + d_{sk}$.

Then

$$\pi(\lambda_{0sk} | \beta, a_0, D_0) = \text{Gamma}(\lambda_{0sk}; p_{sk}, q_{sk}),$$

and

$$\pi(\beta | D_0, a_0) \propto \prod_{s=1}^S \prod_{k=1}^{K_s} q_{sk}^{-p_{sk}} \prod_{j=1}^J \prod_{i=1}^{n_{0j}} \exp(x'_{0i} \beta)^{\nu_{0i} a_{0j}} \prod_{p=1}^P N(\mu_p, \sigma_p^2). \quad (4)$$

When using the normalized power prior in **BayesPPDSurv**, for simplicity of computation, we focus on inferences for β and λ only, so we marginalize over λ_0 and a_0 to obtain

$$\begin{aligned} \pi(\beta, \lambda | D_0) &= \int \int \pi(\lambda_0 | \beta, a_0, D_0) \times \pi(\beta | D_0, a_0) \times \pi(a_0) \times \pi(\lambda) d\lambda_0 da_0 \\ &= \int \pi(\beta | D_0, a_0) \pi(a_0) da_0 \times \pi(\lambda). \end{aligned}$$

The normalizing constant for equation (4) is analytically intractable. In **BayesPPD**, we implemented an algorithm (Shen et al., 2023b) that approximates the normalizing constant with the partition weighted kernel estimator (Wang et al., 2018). In **BayesPPDSurv**, instead of approximating the normalizing constant, we approximate the normalized power prior for β itself, $\pi(\beta | D_0) = \int \pi(\beta | D_0, a_0) \pi(a_0) da_0$, with a multivariate normal distribution, utilizing the Bayesian Central Limit Theorem (Chen, 1985). We propose a novel algorithm to approximate $\pi(\beta | D_0)$ as follows:

1. Sample $a_{01}, \dots, a_{0L}$ from $\pi(a_0)$, where $L = 10,000$, for example. Note that each a_{0l} is a $J \times 1$ vector corresponding to the J historical datasets.
2. Given each a_{0l} , use slice sampling (Neal, 2003) to obtain a sample of β_l (after burn-in) based on the kernel of $\pi(\beta | a_{0l}, D_0)$ in equation (4).
3. Given $\beta_1, \dots, \beta_L$, the distribution $\pi(\beta | D_0)$ can be approximated by a multivariate normal distribution or a mixture of multivariate normals. The mixture of multivariate normals can lead to an arbitrarily accurate approximation of the samples with a large enough number of components, which is a well-known property of finite mixture models (Titterton et al., 1985; Nguyen et al., 2020). To approximate with a single multivariate normal, one can compute the mean $\bar{\beta}$ and covariance $\bar{\Sigma}$ of $\beta_1, \dots, \beta_L$, and approximate $\pi(\beta | D_0)$ with $N_P(\bar{\beta}, \bar{\Sigma})$.

By default, the function `phm.random.a0()` approximates $\pi(\beta | D_0)$ with a single multivariate normal distribution, $N_P(\bar{\beta}, \bar{\Sigma})$. The package allows the user to approximate $\pi(\beta | D_0)$ with any finite mixture of multivariate normal distributions as well. To achieve this, the user can take the output of the function `approximate.prior.beta()`, which are discrete samples of β_l , $l = 1, \dots, L$, and use external software to compute a mixture of multivariate normal

distributions that best approximates the samples. In Table 5 of section 2.4, we show that this method provides highly accurate approximations. One drawback of this approximation method is that the posterior samples of a_0 and λ_0 cannot be obtained. Further details are given in section 2.3.

2.5 Bayesian Sample Size Determination

To perform Bayesian sample size determination, we use the simulation-based procedure proposed in [Chen et al. \(2011\)](#). We assume the first column of the covariate matrix is the treatment indicator, and the corresponding parameter is β_1 , the log hazard ratio. By default, the hypotheses are given by

$$H_0 : \beta_1 \geq \delta$$

and

$$H_1 : \beta_1 < \delta,$$

where δ is a prespecified value. To test hypotheses in the opposite direction, i.e., $H_0 : \beta_1 \leq \delta$ and $H_1 : \beta_1 > \delta$, one can set the parameter `nullspace.ineq` to "<".

Now we discuss the simulation-based procedure used to estimate the Bayesian type I error rate and power. Let Θ_0 and Θ_1 denote the parameter spaces corresponding to H_0 and H_1 . Let $y^{(n)}$ denote the simulated current data associated with a sample size of n , $\theta = (\beta, \lambda, \lambda_0)$ denotes the model parameters, $\pi^{(s)}(\theta)$ denotes the sampling prior, and $\pi^{(f)}(\theta)$ denotes the fitting prior. The sampling prior is used to generate the hypothetical data while the fitting prior is used to fit the model after the data are generated. Let $\pi_0^{(s)}(\theta)$ denote the sampling prior that only puts mass in the null region, i.e., $\theta \in \Theta_0$, and $\pi_1^{(s)}(\theta)$ denotes the sampling prior that only puts mass in the alternative region, i.e., $\theta \in \Theta_1$. Let $q_0^{(n)}$ and $q_1^{(n)}$ denote the Bayesian type I error rate and power, respectively, associated with a sample size of n . Let B denote the number of simulated trials. To compute the Bayesian power or type I error, the following algorithm is used for each simulated trial b :

- Step 1: Generate $\theta^{(b)} \sim \pi_j^{(s)}(\theta)$, $j = 0, 1$. The choice of sampling priors is described in section 2.3.1.
- Step 2: Generate $y^{(b)} \sim f(y^{(b)}|\theta^{(b)})$. Data generation for the PWCH-PH model is detailed in section 2.2.6.
- Step 3: Estimate the posterior distribution $\pi(\theta|y^{(b)}, D_0, a_0)$ and the posterior probability $P(\beta_1 < \delta|y^{(b)}, \pi^{(f)}, D_0, a_0)$. Model fitting is performed using a custom slice sampler written in C++ via Rcpp ([Neal, 2003](#); [Eddelbuettel and Francois, 2011](#)).
- Step 4: Compute the indicator $r^{(b)} = I[P(\beta_1 < \delta|y^{(b)}, \pi^{(f)}, D_0, a_0) \geq \gamma]$, where the constant $\gamma > 0$ is a prespecified posterior probability threshold for rejecting the null hypothesis (e.g., 0.975).

The estimate of $q_j^{(n)}$ is given by $\frac{1}{B} \sum_{b=1}^B r^{(b)}$, where $j = 0, 1$. The quantity $q_0^{(n)}$ corresponding to $\pi^{(s)}(\theta) = \pi_0^{(s)}(\theta)$ is the Bayesian type I error rate, while $q_1^{(n)}$ corresponding to $\pi^{(s)}(\theta) = \pi_1^{(s)}(\theta)$ is the Bayesian power. To identify the desired sample size, we can compute $n_{\alpha_0} = \min\{n : q_0^{(n)} \leq \alpha_0\}$ and $n_{\alpha_1} = \min\{n : q_1^{(n)} \geq 1 - \alpha_1\}$ for given $\alpha_0 > 0$ and $\alpha_1 > 0$, and therefore, the sample size is taken to be $\max\{n_{\alpha_0}, n_{\alpha_1}\}$. Common choices of α_0 and α_1 include $\alpha_0 = 0.05$ and $\alpha_1 = 0.2$. These choices guarantee that the Bayesian type I error rate is at most 0.05 and the Bayesian power is at least 0.8. Type I error control can be achieved in two ways with the power prior. Typically, one can decrease the discounting parameter to obtain the desired type I error rate. Alternatively, one can increase γ to address potential type I error inflation after borrowing historical data.

2.6 Data Simulation for the PWCH-PH Model

Following [Psioda et al. \(2018\)](#), we describe the steps for simulating the observed data for the PWCH-PH model. We simulate the complete data for subject i through the following procedure:

1. Simulate the enrollment time r_i . The package allows choices of the uniform or the exponential distribution for the distribution for enrollment times.
2. Simulate the treatment indicator x_{i1} from a Bernoulli distribution with user-specified randomization probability. The default value is 0.5.
3. Simulate the additional covariates $x_{i2}, \dots, x_{iP}$. The package samples one row of covariates with replacement from the combined covariate matrix of all historical datasets.
4. Simulate the stratum index s_i . The package samples with replacement from the combined stratum vector of all historical datasets.
5. Compute $\phi_i = \exp(x_i' \beta)$ and $\theta_{i,k} = \lambda_{s_i,k} \phi_i$ for $k = 1, \dots, K_{s_i}$.
6. Simulate the event time t_i as follows:
 - (a) Set $k = 1$.
 - (b) Simulate $t_i \sim \text{Exponential}(\text{rate} = \theta_{i,k}) + t_{s_i,k-1}$.
 - (c) If $t_i > t_{s_i,k}$, then increment k by one and return to step (b), otherwise terminate.
7. Simulate the censorship time c_i . Choices for the distribution of censorship times include the uniform distribution, the exponential distribution, or a user-specified constant time.
8. Compute the observation time $\tilde{y}_i = \min(t_i, c_i)$ and the event indicator $\tilde{\nu}_i = I(t_i \leq c_i)$.
9. Compute the elapsed time $e_i = r_i + y_i$.

The above procedure yields a hypothetical complete dataset corresponding to a scenario where all subjects are followed until the event is observed or they drop out. One constructs the observed dataset from the complete dataset as follows:

1. Determine the time T of target event number ν in the complete dataset.
2. Remove subjects with $r_i \geq T$.
3. For the remaining subjects,
 - (a) If $e_i > T$, set $y_i = T - r_i$ and $\nu_i = 0$.
 - (b) If $e_i \leq T$, set $y_i = \tilde{y}_i$ and $\nu_i = \tilde{\nu}_i$.

In the above, the quantity ν represents the target number of events that is desired to trigger the primary analysis for the simulated trial. The user can also determine the minimum amount of time ($T_{\min}$) and maximum amount of time ($T_{\max}$) that subjects are followed. If $T < T_{\min}$, then T is replaced by $T_{\min}$. Administrative censoring occurs at $T_{\max}$.

3 Using [BayesPPDSurv](#)

[BayesPPDSurv](#) supports the incorporation of multiple historical datasets using the power prior with fixed a_0 or using the normalized power prior with a_0 modeled as random. It contains two categories of functions, functions for model fitting ("[phm.fixed.a0\(\)](#)") and "[phm.random.a0\(\)](#)") and functions for sample size determination ("[power.phm.fixed.a0\(\)](#)") and "[power.phm.random.a0\(\)](#)"). Functions for model fitting return posterior samples of the

parameters, while functions for sample size determination return estimates of the Bayesian power or type I error rate.

When a_0 is modeled as random, the function `approximate.prior.beta()` performs slice sampling to generate samples of β from the normalized power prior. By default, the function `phm.random.a0()` approximates the normalized power prior for β , $\pi(\beta|D_0)$, with a single multivariate normal distribution. To accommodate greater accuracy of approximation, the package allows the user to approximate $\pi(\beta|D_0)$ with a finite mixture of multivariate normal distributions as well. Specifically, the user can take the output of the `approximate.prior.beta()` function, which is a matrix of samples of β from the normalized power prior, and use external software to compute a mixture of multivariate normal distributions that best approximates the samples. The resulting mixture distribution can then be the input to the `prior.beta.mvn` argument of `phm.random.a0()`, which is a list of lists, where each list has three elements, consisting of the mean vector, the covariance matrix and the weight of each multivariate normal distribution. Then, $\pi(\beta|D_0)$ is approximated by the mixture of the multivariate normal distributions provided. Since we use λ_0 and a_0 as auxiliary variables for the approximation for $\pi(\beta|D_0)$, the user specifies the hyperparameters for the priors for λ_0 and a_0 in the function `approximate.prior.beta()`. The hyperparameters for the priors for λ and the initial prior for β are specified in the function `phm.random.a0()`.

The package allows the time interval partition to vary across the S levels of the stratification variable. The user must specify the total number of intervals for each stratum, `n.intervals`. Then for each stratum, by default, the change points are assigned so that an approximately equal number of events are observed in all the intervals for the pooled historical and current datasets. The user can also specify the change points for each stratum. When a_0 is fixed, by default, the baseline hazard parameters are unshared (i.e., $\lambda \neq \lambda_0$) between the current and historical data. If `shared.blh=TRUE`, baseline hazard parameters are shared and historical information is used to estimate the baseline hazard parameters. When a_0 is modeled as random, the package only supports unshared baseline hazards.

For study design applications, the user can manipulate many attributes of the data generation process, including the enrollment time distribution (uniform or exponential), the randomization probability for the treated group, the censorship time distribution (uniform, exponential or constant), the probability of subjects dropping out of the study (non-administrative censoring), the dropout time distribution (uniform), and the minimum and maximum amount of time that subjects are followed.

3.1 Sampling Priors

Our implementation in `BayesPPDSurv` does not assume any particular distribution for the sampling priors. The user specifies discrete approximations of the sampling priors by providing a matrix or list of parameter values and the algorithm samples with replacement from the matrix or the list as the first step of the data generation. The user must specify `samp.prior.beta`, a matrix of samples for β , and `samp.prior.lambda`, a list of matrices where each matrix represents the sampling prior for the baseline hazards for each stratum. The number of columns of each matrix must be equal to the number of intervals for that stratum.

Now we describe strategies to elicit the sampling priors, as detailed in Psioda and Ibrahim (2019). Suppose one wants to test the hypotheses

$$H_0 : \beta_1 \geq 0$$

and

$$H_1 : \beta_1 < 0.$$

To elicit the sampling prior for β_1 to compute power, one can simply sample from a truncated normal distribution with negative mean, so that the mass of the prior falls in the alternative space. Conversely, to compute the type I error rate, one can sample from a truncated normal distribution with positive mean, so that the mass of the prior falls in the null space. Next, to

generate the sampling prior for the other parameters, $\beta_2, \dots, \beta_P$ and λ (the same sampling prior is assumed for λ_0), one can use the posterior samples given the historical data as the discrete approximation to the sampling prior. The function `phm.fixed.a0()` generates such posterior samples if the `current.data` argument is set to `FALSE` and $a_{0j} = 1$ for $j = 1, \dots, J$. This method is illustrated in the case study in the following section.

4 Case Study: Melanoma Clinical Trial Design

We consider the high-risk melanoma trial design application in [Psioda and Ibrahim \(2019\)](#), and demonstrate how [BayesPPDSurv](#) can be used for coefficient estimation as well as power and type I error rate calculations for time-to-event data in Bayesian clinical trial designs that incorporate historical information.

Interferon Alpha-2b (IFN) is an adjuvant chemotherapy for deep primary or regionally metastatic melanoma. The E1684 trial ([Kirkwood et al., 1996](#)) and the E1690 trial ([Kirkwood et al., 2000](#)) were randomized controlled trials conducted to evaluate the efficacy of IFN for melanoma following surgery. The studies classified the subjects into four disease stage groups. Following [Psioda and Ibrahim \(2019\)](#), we restrict our attention to patients in disease stage four, i.e., regional lymph node recurrence at any interval after appropriate surgery for primary melanoma of any depth. The primary outcome is relapse-free survival. The number of positive lymph nodes at lymphadenectomy is used as a stratification variable (≤ 2 vs. ≥ 3) due to its prognostic value. We compare patients who received the IFN treatment to those who received observation (OBS). Table 2 summarizes the total number of events and the total risk time by treatment group and number of positive lymph nodes for the two studies.

Study	Treatment	# Nodes	Sample Size	# Events	Risk Time
E1684	OBS	≤ 2	37	26	88.4
		≥ 3	47	36	105.8
	IFN	≤ 2	44	21	176.3
		≥ 3	39	31	81.1
E1690	OBS	≤ 2	51	23	122.4
		≥ 3	53	42	78.9
	IFN	≤ 2	51	29	123.1
		≥ 3	59	36	137.0

Table 2: Relapse-free survival data by treatment group and number of positive nodes for trials E1684 and E1690. The number of positive lymph nodes is used as a stratification variable.

First, suppose we are interested in analyzing the relationship between relapse-free survival and the IFN treatment for the E1690 study after incorporating historical information from the E1684 study. We build a PWCH-PH model with two strata, stratum-specific baseline hazard parameters (unshared between current and historical datasets) and one covariate (the treatment indicator). For stratum 1 ($\# \text{ nodes} \leq 2$), we use four time intervals (selected using the deviance information criterion (DIC)). The baseline hazard parameters are $\lambda_{1,1}$, $\lambda_{1,2}$, $\lambda_{1,3}$, $\lambda_{1,4}$. For stratum 2 ($\# \text{ nodes} \geq 3$), we use three time intervals (also selected using DIC). The baseline hazard parameters are $\lambda_{2,1}$, $\lambda_{2,2}$, $\lambda_{2,3}$. We use the default change points, which are determined so that an approximately equal number of events are observed in each time interval for the pooled current and historical datasets. We use the default initial prior for β , a normal prior with mean zero and variance 10^3 . We use the default priors for λ and λ_0 , which are independent non-informative gamma priors with shape and rate parameters equal to 10^{-5} . The code below demonstrates the analysis with the E1684 study data as prior information incorporated through a power prior with a_0 fixed at 0.5. Here, `nMC` represents the number of MCMC iterations and `nBI` represents the number of burn-in samples.

```
data(melanoma)
hist <- melanoma[melanoma$study=="1684",]
```

```

current <- melanoma[melanoma$study=="1690",]
n.intervals <- c(4,3)
nMC <- 10000
nBI <- 200
historical <- list(list(time=hist$failtime, event=hist$rfscens,
                      X=as.matrix(hist[, "trt"]), S=hist$stratum))

set.seed(1)
result <- phm.fixed.a0(time=current$failtime, event=current$rfscens,
                      X=as.matrix(current[, "trt"]), S=current$stratum,
                      historical=historical, a0=0.5, n.intervals=n.intervals,
                      nMC=nMC, nBI=nBI)

> quantile(result$beta_samples)
      0%      25%      50%      75%     100%
-0.7950745 -0.3694794 -0.2727719 -0.1737096  0.2939300
> colMeans(result$lambda_samples[[1]])
[1] 0.4594304 0.5089287 0.3052143 0.1188887
> colMeans(result$lambda_samples[[2]])
[1] 1.0761066 0.7789043 0.2019115
> colMeans(result$lambda0_samples[[1]])
[1] 0.67388605 0.75062812 0.35459671 0.06557007
> colMeans(result$lambda0_samples[[2]])
[1] 1.3381514 0.7562455 0.1853991

```

Table 3 displays the posterior mean, standard deviation and 95% credible interval for β and elements of λ . There is weak evidence suggesting a negative association between IFN and time-to-relapse.

Parameter	Mean	SD	95% CI
β	-0.27	0.15	(-0.57, 0.01)
$\lambda_{1,1}$	0.46	0.14	(0.23, 0.77)
$\lambda_{1,2}$	0.51	0.15	(0.26, 0.83)
$\lambda_{1,3}$	0.31	0.09	(0.16, 0.50)
$\lambda_{1,4}$	0.12	0.03	(0.07, 0.19)
$\lambda_{2,1}$	1.08	0.23	(0.68, 1.57)
$\lambda_{2,2}$	0.78	0.16	(0.50, 1.12)
$\lambda_{2,3}$	0.20	0.04	(0.12, 0.30)

Table 3: Posterior mean, standard deviation, and 95% credible interval for β and elements of λ for the E1690 study incorporating historical data E1684 via a power prior with $a_0 = 0.5$. The parameter $\lambda_{s,k}$ represents baseline hazard for time interval k for stratum s .

Next, our goal is to design a new trial incorporating the E1684 study using the power prior and the normalized power prior. We first specify the characteristics of trial data simulation. Let ν be the number of events at which the trial will stop and let n be the total number of subjects enrolled. For each ν , we take $n = 3\nu$. The choice of $n = 3\nu$ here is for illustrative purposes only. In any trial, the sample size just needs to be larger than ν . In practice, the sample size and the targeted number of events will depend on many factors such as the enrollment rate, the desired follow-up duration, and the cost of enrolling patients. We assume a subject's enrollment time follows a uniform distribution over a 4-year period. We allocate 50% of the subjects to the treatment group. For stratum allocation, we sample from the stratum indices of the historical data with replacement. We assume there is only administrative censoring which occurs when ν events have accrued. In the data generation phase, we use the default change points which are determined so that an approximately equal number of events are observed in each time interval for the historical dataset. When analyzing the generated data, we use the default change points which are determined so that an approximately equal number of events are observed in each time interval for the pooled

current and historical datasets. The baseline hazard parameters are not shared between the current and historical data. The same set of default priors are used for β , λ and λ_0 as before.

We compute the Bayesian power and type I error rate for a few sample sizes for tests of the hypotheses

$$H_0 : \beta \geq 0$$

and

$$H_1 : \beta < 0.$$

We use two sets of sampling priors, the default null (DN) and default alternative (DA) sampling priors, and the frequentist null (FN) and point-mass alternative (PA) sampling priors. The default null sampling prior is defined as $\pi_0^{(s)} = \pi(\theta|D_0, \beta \geq 0)$. The default alternative sampling prior is defined as $\pi_0^{(s)} = \pi(\theta|D_0, \beta < 0)$. The FN sampling prior is a point-mass prior at $\beta = 0$, which results in a Bayesian type I error rate that is closely related to the standard frequentist type I error rate. The PA sampling prior is a point-mass prior centered on $\beta = -0.27$, the posterior mean of β for the E1684 dataset, which results in Bayesian power that is closely related to the standard frequentist notion of power (i.e., power at a point). The sampling prior for λ is a point-mass prior centered on the posterior mean of λ for the E1684 dataset. We first obtain the posterior samples of β and λ using only the historical data, and then subset to the posterior samples of β that are greater than zero for the DN sampling prior and to the samples that are less than zero for the DA sampling prior. The DN and DA sampling priors can be easily acquired with the following code with `current.data=FALSE` and `a0=1`. Here, `nMC` is set to 10,000, but depending on the dataset, `nMC` may need to be larger to ensure enough entries in the truncated sampling priors.

```
nMC <- 10000
nBI <- 200
set.seed(1)
samples <- phm.fixed.a0(historical=historical, a0=1, n.intervals=n.intervals,
                        current.data=FALSE, nMC=nMC, nBI=nBI)
beta_priors <- samples$beta_samples
DN_beta_samp_prior <- as.matrix(beta_priors[beta_priors[,1] >= 0, ])
DA_beta_samp_prior <- as.matrix(beta_priors[beta_priors[,1] < 0, ])
lambda_samp_prior <- samples$lambda_samples
```

Now we are ready to compute the power and type I error rate using the power prior via the function `power.phm.fixed.a0()`. The following code computes the power for $\nu = 350$ and $a_0 = 0.6$ using the default alternative sampling prior.

```
set.seed(1)
a0 <- 0.6
n.events <- 350
n.subjects <- n.events * 3
nMC <- 50000
nBI <- 200
N <- 10000
result <- power.phm.fixed.a0(historical=historical, a0=a0, n.subjects=n.subjects,
                             n.events=n.events, n.intervals=n.intervals,
                             samp.prior.beta=DA_beta_samp_prior, samp.prior.lambda=lambda_samp_prior,
                             dist.enroll="Uniform", param.enroll=4, nMC=nMC, nBI=nBI,
                             delta=0, nullspace.ineq=">", N=N)
```

Table 4 displays the power and type I error rates for $\nu = 350$ and $\nu = 710$ for a_0 values of 0, 0.2 and 0.6 using the default sampling priors and the point-mass sampling priors. We obtain 50,000 posterior samples using the package's custom slice sampler after 200 burn-ins for each of the 10,000 simulated datasets. The results in Table 4 are comparable to the results in Figure 2 in Psioda and Ibrahim (2019) (i.e., the same up to Monte Carlo error).

We observe that the power increases with a_0 and sample size as expected. We can also see that the default sampling priors yield average rates that are often lower than rates based on point-mass priors.

Null sampling prior/ Alt sampling prior	a_0	Number of events	Type I error rate	Power
DN/DA	0	350	0.0121	0.3947
		710	0.0094	0.5868
	0.2	350	0.0134	0.4280
		710	0.0113	0.6055
	0.6	350	0.0242	0.4917
		710	0.0154	0.6504
FN/PA	0	350	0.0239	0.3145
		710	0.0226	0.5682
	0.2	350	0.0298	0.3545
		710	0.0349	0.5983
	0.6	350	0.0508	0.4435
		710	0.0474	0.6851

Table 4: Power and type I error rates for $\nu = 350$ and $\nu = 710$ for a_0 values of 0, 0.2 and 0.6 using the default sampling priors and the point-mass sampling priors. We run 50,000 iterations of the slice sampler with 200 burn-ins for each of the 10,000 simulated datasets.

Finally, we demonstrate how to estimate operating characteristics for a new trial incorporating the E1684 study with the normalized power prior where a_0 is modeled as random. We compute power and type I error with the DA and DN sampling priors using the function `power.phm.random.a0()`. By default, the function `power.phm.random.a0()` approximates the normalized power prior for β , $\pi(\beta|D_0)$, with a single multivariate normal distribution. We now show how to approximate $\pi(\beta|D_0)$ with a mixture of multivariate normal distributions. We first use the `approximate.prior.beta()` function to acquire discrete samples of β from the normalized power prior, and then use the R package `mixtools` (Benaglia et al., 2009) to compute a mixture of multivariate normal distributions that best approximates the samples. In the code below, we put the output of the function `normalmixEM()`, a weighted mixture of two normal distributions, into a list, which then becomes the input to the parameter `prior.beta.mvn`. In Figure 1, we plot the density of the discrete samples of β produced by `approximate.prior.beta()` as well as the normal mixture and single normal approximation. We can see that the normal mixture is a much better approximation. In this case, the two-component normal mixture provides an adequate estimate; the `mixtools` package contains functions that calculate Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC) to help the user choose the number of mixture components.

```
set.seed(1)
nMC <- 10000
nBI <- 200
N <- 10000
prior.beta <- approximate.prior.beta(historical, n.intervals,
                                     prior.a0.shape1=1, prior.a0.shape2=1,
                                     nMC=nMC, nBI=nBI)

library(mixtools)
mix <- normalmixEM(prior.beta)
list_mixture <- list(list(mix$mu[1], as.matrix(mix$sigma[1]), mix$lambda[1]),
                    list(mix$mu[2], as.matrix(mix$sigma[2]), mix$lambda[2]))
```

```

result <- power.phm.random.a0(historical=historical, n.subjects=n.subjects,
  n.events=n.events, n.intervals=n.intervals,
  prior.beta.mvn=list_mixture,
  samp.prior.beta=DA_beta_samp_prior, samp.prior.lambda=lambda_samp_prior,
  dist.enroll="Uniform", param.enroll=4, nMC=nMC, nBI=nBI,
  delta=0, nullspace.ineq=">", N=N)

```

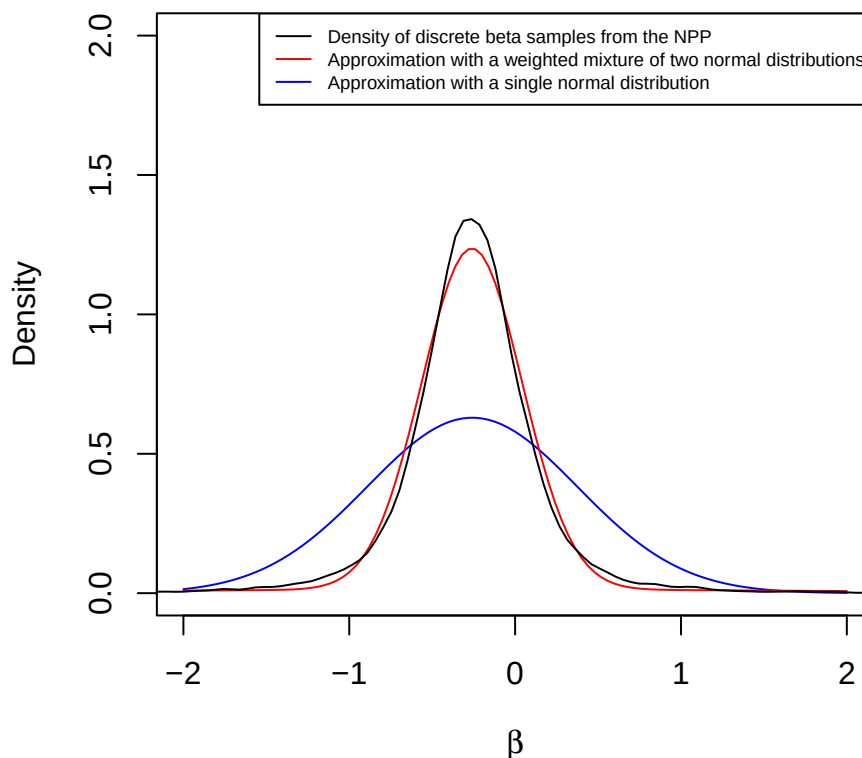


Figure 1: In this figure, the black curve represents the density of the discrete β samples from the normalized power prior. The red curve represents the weighted mixture of two normal distributions that approximates the β samples. The blue curve represents the single normal distribution that approximates the β samples. We can see that the mixture of normals is a much better approximation than the single normal distribution.

In Table 5, we compare the power and type I error rate calculations using the normalized power prior with a $\text{beta}(10^3, 10^3)$ prior for a_0 , with a power prior with $a_0 = 0.5$. We expect the results to be quite similar since the prior on a_0 for the normalized power prior reflects a high degree of certainty that the correct value is near 0.5. The DN/DA sampling priors are used, and we run 10,000 iterations of the slice sampler with 200 burn-ins for each of the 10,000 simulated datasets. The results are quite similar, indicating that our algorithm for approximating the normalized power prior provides reliable estimates.

Model	a_0	Type I error rate	Power
power prior	0.5	0.029	0.770
normalized power prior	$\text{beta}(10^3, 10^3)$	0.028	0.768

Table 5: Power and type I error rate computed with `power.phm.fixed.a0()` with a_0 fixed at 0.5, and `power.phm.random.a0()` with a $\text{beta}(10^3, 10^3)$ prior on a_0 . The DN sampling prior is used to compute the type I error rate and the DA sampling prior is used to compute power. The number of events used is 350. We run 10,000 iterations of the slice sampler with 200 burn-ins for each of the 10,000 simulated datasets.

5 Discussion

BayesPPDSurv facilitates Bayesian power and type I error rate calculations using the power and normalized power prior for time-to-event outcomes using a PWCH-PH model. We implement a flexible stratified version of the model, where the historical data can be used to inform the treatment effect, the effect of other covariates in the regression model, as well as the baseline hazard parameters. We develop a novel algorithm for approximating the normalized power prior that eliminates the need to compute the normalizing constant. The package also has features that semi-automatically generate the sampling priors from the historical data.

Future versions of the package will accommodate cure rate models. Another possible feature is the computation of optimal hyperparameters for the beta prior on a_0 to ensure that the normalized power prior adapts in a desirable way to prior-data conflict or prior-data agreement, based on the work of Shen et al. (2026).

References

- E. Alt. *hdbayes: Bayesian Analysis of Generalized Linear Models with Historical Data*, 2022. R package version 0.0.0.9000. [p]
- K. Anderson. *gsDesign: Group Sequential Design*, 2024. URL <https://keaven.github.io/gSDesign/>. R package version 3.6.1, <https://github.com/keaven/gSDesign>. [p]
- S. Balcome, D. Musgrove, T. Haddad, and G. L. Hickey. *bayesDP: Tools for the Bayesian Discount Prior Function*, 2021. URL <https://CRAN.R-project.org/package=bayesDP>. R package version 1.3.4. [p]
- T. Benaglia, D. Chauveau, D. R. Hunter, and D. Young. mixtools: An R package for analyzing finite mixture models. *Journal of Statistical Software*, 32(6):1–29, 2009. URL <https://www.jstatsoft.org/v32/i06/>. [p]
- L. M. Carvalho and J. G. Ibrahim. On the normalized power prior. *Statistics in Medicine*, 40(24):5251–5275, Jul 2021. doi: 10.1002/sim.9124. [p]
- T. Chandereng, D. Musgrove, T. Haddad, G. Hickey, T. Hanson, and T. Lystig. *bayesCT: Simulation and Analysis of Adaptive Bayesian Clinical Trials*, 2020. URL <https://CRAN.R-project.org/package=bayesCT>. R package version 0.99.3. [p]
- C.-F. Chen. On asymptotic normality of limiting density functions with Bayesian implications. *Journal of the Royal Statistical Society. Series B (Methodological)*, 47(3):540–546, 1985. [p]
- M.-H. Chen and J. Ibrahim. The relationship between the power prior and hierarchical models. *Bayesian Analysis*, 1, 09 2006. doi: 10.1214/06-BA118. [p]
- M.-H. Chen, J. G. Ibrahim, P. Lam, A. Yu, and Y. Zhang. Bayesian design of noninferiority trials for medical devices using historical data. *Biometrics*, 67(3):1163–1170, Sep 2011. doi: 10.1111/j.1541-0420.2011.01561.x. [p]

- D. R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(2):187–220, 1972. ISSN 00359246. URL <http://www.jstor.org/stable/2985181>. [p]
- Y. Duan, K. Ye, and E. P. Smith. Evaluating water quality using power priors to incorporate historical information. *Environmetrics (London, Ont.)*, 17(1):95–106, feb 2006. doi: 10.1002/env.752. [p]
- D. Eddelbuettel and R. Francois. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. [p]
- B. S. Eggleston, J. G. Ibrahim, B. McNeil, and D. Catellier. BayesCTDesign: an R package for Bayesian trial design using historical control data. *Journal of Statistical Software*, 100(21), 2021. doi: 10.18637/jss.v100.i21. [p]
- M. Friedman. Piecewise exponential models for survival data with covariates. *The Annals of Statistics*, 10(1):101–113, Mar 1982. doi: 10.1214/aos/1176345693. [p]
- P. Frumento. *pch: Piecewise Constant Hazard Models for Censored and Truncated Data*, 2024. URL <https://CRAN.R-project.org/package=pch>. R package version 2.1. [p]
- Z. Han, T. Bai, and K. Ye. *NPP: Normalized Power Prior Bayesian Analysis*, 2021. URL <https://CRAN.R-project.org/package=NPP>. R package version 0.4.0. [p]
- J. Ibrahim, M.-H. Chen, and D. Sinha. *Bayesian Survival Analysis*. Springer, New York, 2001. [p]
- J. G. Ibrahim and M.-H. Chen. Power prior distributions for regression models. *Statistical Science*, 15(1):46–60, feb 2000. doi: 10.1214/ss/1009212673. [p]
- J. G. Ibrahim, M.-H. Chen, and D. Sinha. On optimality properties of the power prior. *Journal of the American Statistical Association*, 98(461):204–213, mar 2003. doi: 10.1198/016214503388619229. [p]
- J. G. Ibrahim, M.-H. Chen, Y. Gwon, and F. Chen. The power prior: theory and applications. *Statistics in Medicine*, 34(28):3724–3749, dec 2015. doi: 10.1002/sim.6728. [p]
- J. M. Kirkwood, M. H. Strawderman, M. S. Ernstoff, T. J. Smith, E. C. Borden, and R. H. Blum. Interferon alfa-2b adjuvant therapy of high-risk resected cutaneous melanoma: the eastern cooperative oncology group trial EST 1684. *Journal of Clinical Oncology*, 14:7–17, 1996. [p]
- J. M. Kirkwood, J. G. Ibrahim, V. K. Sondak, J. Richards, L. E. Flaherty, M. S. Ernstoff, T. J. Smith, U. Rao, M. Steele, and R. H. Blum. High- and low-dose interferon alfa-2b in high-risk melanoma: first analysis of intergroup trial E1690/S9111/C9190. *Journal of Clinical Oncology*, 18(12):2444–2458, Jun 2000. doi: 10.1200/JCO.2000.18.12.2444. [p]
- R. M. Neal. Slice sampling. *Ann. Statist.*, 31(3):705–767, 2003. [p]
- T. T. Nguyen, H. D. Nguyen, F. Chamroukhi, and G. J. McLachlan. Approximation by finite mixtures of continuous density functions that vanish at infinity. *Cogent Mathematics & Statistics*, 7(1):1750861, 2020. doi: 10.1080/25742558.2020.1750861. [p]
- M. A. Psioda and J. G. Ibrahim. Bayesian clinical trial design using historical data that inform the treatment effect. *Biostatistics*, 20(3):400–415, Jul 2019. doi: 10.1093/biostatistics/kxy009. [p]
- M. A. Psioda, M. Soukup, and J. G. Ibrahim. A practical Bayesian adaptive design incorporating data from historical controls. *Statistics in Medicine*, 37(27):4054–4070, Nov 2018. doi: 10.1002/sim.7897. [p]
- Y. Shen, M. A. Psioda, and J. G. Ibrahim. *BayesPPD: Bayesian Power Prior Design*, 2023a. URL <https://CRAN.R-project.org/package=BayesPPD>. R package version 1.1.2. [p]

- Y. Shen, M. A. Psioda, and J. G. Ibrahim. BayesPPD: An R package for Bayesian sample size determination using the power and normalized power prior for generalized linear models. *The R Journal*, 14:335–351, 2023b. ISSN 2073-4859. doi: 10.32614/RJ-2023-016. <https://doi.org/10.32614/RJ-2023-016>. [p]
- Y. Shen, M. A. Psioda, L. M. Carvalho, and J. G. Ibrahim. ANPP: the adapted normalized power prior for borrowing information from multiple historical datasets in clinical trials, 2024a. arXiv preprint. [p]
- Y. Shen, M. A. Psioda, and J. G. Ibrahim. *BayesPPDSurv: Bayesian Power Prior Design for Survival Data*, 2024b. R package version 1.0.3. [p]
- Y. Shen, L. M. Carvalho, M. A. Psioda, and J. G. Ibrahim. Optimal priors for the discounting parameter of the normalized power prior. *Statistica Sinica*, 36, 2026. [p]
- D. Titterton, A. Smith, A. Smith, and U. Makov. *Statistical Analysis of Finite Mixture Distributions*. Applied section. Wiley, 1985. ISBN 9780471907633. URL <https://books.google.com/books?id=hZOQAQAIAAJ>. [p]
- K. Viele, S. Berry, B. Neuenschwander, B. Amzal, F. Chen, N. Enas, B. Hobbs, J. G. Ibrahim, N. Kinnersley, S. Lindborg, and et al. Use of historical control data for assessing treatment effects in clinical trials. *Pharmaceutical Statistics*, 13(1):41–54, Feb 2014. doi: 10.1002/pst.1589. [p]
- F. Wang and A. E. Gelfand. A simulation-based approach to Bayesian sample size determination for performance under a given model and for separating models. *Statistical Science*, 17(2):193–208, May 2002. doi: 10.1214/ss/1030550861. [p]
- Y.-B. Wang, M.-H. Chen, L. Kuo, and P. O. Lewis. A new monte carlo method for estimating marginal likelihoods. *Bayesian Analysis*, 13(2):311–333, 2018. [p]

Yueqi Shen

Department of Biostatistics

University of North Carolina at Chapel Hill

Currently affiliated with Merck & Co.

United States

angieshen6@gmail.com

Matthew A. Psioda

GSK

United States

matt.a.psioda@gsk.com

Joseph G. Ibrahim

Department of Biostatistics

University of North Carolina at Chapel Hill

United States

ibrahim@bios.unc.edu

Hyper-Poisson Regression Modeling via GAMLSS

by Freddy Hernández-Barajas, Jamiu S. Olumoh, Osho O. Ajayi, and Fernando Marmolejo-Ramos

Abstract Traditional Poisson regression assumes equal mean and variance, which is often violated in practice. Hyper-Poisson regression addresses this by accommodating under-dispersed and over-dispersed data. The proposed `DiscreteDists` R package integrates the hyper-Poisson distribution functions into the GAMLSS framework, enabling more accurate count data analysis. This study examines the capabilities of the `DiscreteDists` R package, which includes functions for both parameterizations of the hyper-Poisson distribution and its regression model. This paper demonstrates the implementation of hyper-Poisson regression using the `DiscreteDists` R package, offering enhanced tools for researchers. In addition, the study includes simulation studies and practical applications to demonstrate the performance and utility of the proposed R package.

1 Introduction

The conventional Poisson regression assumes that the mean and variance of the count data are equal; however, this assumption is often violated in practice. The hyper-Poisson regression proposed by Sáez-Castillo and Conde-Sánchez (2013) is a recent statistical modeling technique used to analyze count data where the variance is less than or greater than the mean. In many real-world scenarios, count data with under-dispersion (variance < mean) or over-dispersion (variance > mean) is common, and numerous examples abound in the fields of epidemiology, ecology, education, and finance, among others Cameron et al. (1988), Consul (1989), Winkelmann and Zimmermann (1991), Famoye (1993), Johnson et al. (2005), Sellers et al. (2012).

Generalized additive models for location, scale, and shape (GAMLSS) is a flexible regression framework that can handle a wide range of data distributions, including count data with under-dispersion or over-dispersion Stasinopoulos and Rigby (2008). GAMLSS is an extension of the generalized linear model (GLM) that allows for simultaneously modeling the mean, variance, skewness, and kurtosis of the response variable. This makes GAMLSS particularly useful for modeling complex data that exhibit non-standard patterns of variation Peluso et al. (2019).

In GAMLSS, the response variable is modeled using a set of additive predictor functions, each of which corresponds to different aspects or properties of the response variable. These quantities include the location (usually the mean or median), scale (usually the standard deviation or precision), and shape (such as skewness and kurtosis) parameters Marmolejo-Ramos et al. (2023). These predictor functions are typically constructed using smoothing functions that include splines and/or kernel smoothers because of their flexibility in describing the relationships between the response and predictor variable(s) El-Bachir and Davison (2018).

The hyper-Poisson distribution was initially proposed by Bardwell and Crow (1964). The first parameterization of the distribution has two parameters (λ for location and γ for dispersion), but neither of these parameters coincides with the distribution's mean value. For this reason, the authors presented a second parameterization of the hyper-Poisson distribution, also with two parameters (μ for location and γ for dispersion). In this new parameterization, the expected value coincides with the location parameter μ , making it attractive for creating a regression model to analyze real datasets. If one parameter matches the expected value, the regression model is appealing to any user due to the interpretability of this parameter.

Sáez-Castillo and Conde-Sánchez (2013) proposed the hyper-Poisson regression model and Sáez-Castillo et al. (2023) proposed the `DGLMExtPois` R package to fit the hyper-Poisson regression model. This package includes useful functions for estimating the effect of covariates on the parameters of the hyper-Poisson regression model (modeling μ and γ). Additionally, this package contains functions for the COM regression model Huang (2017) to perform residual analysis. Unfortunately, the `DGLMExtPois` R package lacks the basic functions to calculate probabilities and quantiles and generate random values for both parameterizations of the hyper-Poisson distribution.

In this paper, we describe the capabilities of our proposed `DiscreteDists` R package, which includes functions for both parameterizations of the hyper-Poisson distribution and the regression model. Our implementation is integrated within the GAMLSS framework, allowing us to model both distribution parameters using covariates. By using the `DiscreteDists` R package Hernández-Barajas et al. (2025), researchers can perform hyper-Poisson regression analyses more accurately and efficiently, enhancing their ability to uncover meaningful insights from count data with under-dispersion or over-dispersion.

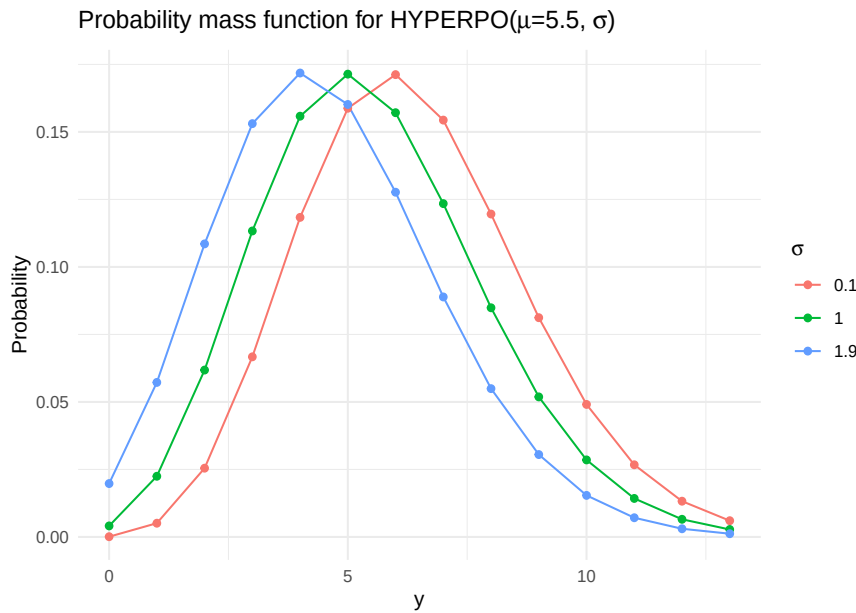


Figure 1: Probability mass function for the HYPERPO with three different values for the second parameter.

The remaining parts of this paper are organized as follows: Section 2 presents the hyper-Poisson distribution and the regression model based on the second parameterization. Section 3 explains the GAMLSS statistical framework. Section 4 introduces the main functions of the [DiscreteDists](#) R package. Section 5 presents the results of a simulation study to explore the performance of the proposed functions to estimate parameters, and Section 6 presents applications of the hyper-Poisson regression model to a real dataset. Finally, Section 7 presents the conclusions of the paper.

2 The hyper-poisson model

The attraction of the hyper-Poisson model is its ability to effectively capture the behavior of a count data set that is either under-dispersed or over-dispersed, or that satisfies the equality of the mean and variance as the basic Poisson distribution.

2.1 The hyper-Poisson distribution

The hyper-Poisson distribution was proposed by [Bardwell and Crow \(1964\)](#), and it is appropriate for over-dispersed, equi-dispersed, or under-dispersed discrete count data Y . Its mass function with location μ and dispersion σ parameters is given by:

$$f(y|\mu, \sigma) = \frac{\mu^y}{{}_1F_1(1; \sigma; \mu)} \frac{\Gamma(\sigma)}{\Gamma(\sigma + y)}, \quad y = 0, 1, 2, 3, \dots; \mu > 0; \sigma > 0 \quad (1)$$

This distribution is over-dispersed when $\sigma > 1$, under-dispersed when $\sigma < 1$, and equi-dispersed (i.e. Poisson) when $\sigma = 1$. The function ${}_1F_1$ is defined as:

$${}_1F_1(a; c; z) = \sum_{r=0}^{\infty} \frac{(a)_r}{(c)_r} \frac{z^r}{r!}, \quad (2)$$

with $(a)_r = \frac{\Gamma(a+r)}{\Gamma(a)}$ for real values $a > 0$ and r a positive integer. In particular, ${}_1F_1(1; c; z)$ is the confluent hypergeometric function with the first argument equal to unity.

To denote that a random variable Y follows the hyper-Poisson distribution with parameters μ and σ we will use the notation $Y \sim \text{HYPERPO}(\mu, \sigma)$. Figure 1 depicts an example of the probabilities for the $\text{HYPERPO}(\mu = 5.5, \sigma)$ and three possible values of the second parameter, $\sigma = 0.1$ (under-dispersed), $\sigma = 1.0$ (Poisson or equi-dispersed) and $\sigma = 1.9$ (over-dispersed).

Equation (1) is known to satisfy the recurrence relation:

$$(y + \mu)f_{y+1} = \sigma f_y, \quad (3)$$

which when summed over all values of the random variable Y , yields expressions $E(Y)$ for the mean and $V(Y)$ for the variance of the distribution:

$$E(Y) = \mu - (\sigma - 1) \frac{{}_1F_1(1; \sigma; \mu) - 1}{{}_1F_1(1; \sigma; \mu)} \quad (4)$$

$$V(Y) = \mu + [\mu - (\sigma - 1)] E(Y) - E^2(Y) \quad (5)$$

In the equations above, $E(Y)$ is a function of parameters μ and σ , whereas $V(Y)$ is a function of μ , σ and $E(Y)$. When the dispersion parameter $\sigma = 1$ (i.e., the equi-dispersion case), then clearly $E(Y) = \mu$ in equation (4) and hence the location parameter coincides with the theoretical expectation of a Poisson distribution assumption for the dataset. Using the same argument in equation (4) and setting $\sigma < 1$ leads to under-dispersion, while setting $\sigma > 1$ means over-dispersion.

The probability mass function for the hyper-Poisson distribution has been written here with parameters μ and σ and not using λ and γ as in [Bardwell and Crow \(1964\)](#) and [Sáez-Castillo and Conde-Sánchez \(2013\)](#); this was done mainly because those are the Greek letters used in the GAMLSS framework.

In the statistical literature, several traditional distributions (both continuous and discrete) have been reparameterized so that one of their parameters coincides with the expected value. For example, [Rigby et al. \(2019\)](#) (p. 84) introduced the distribution WEI3, a reparameterization of the Weibull distribution in which the first parameter μ corresponds to the mean. A discrete case can be found in [Ribeiro Jr et al. \(2020\)](#), where the authors proposed a new parameterization of the Conway–Maxwell–Poisson distribution to ensure that the first parameter represents the mean.

Following this idea, [Sáez-Castillo and Conde-Sánchez \(2013\)](#) proposed reexpressing the original HYPERPO distribution in the parameterization HYPERPO2, in which the first parameter μ coincides with the expected value. The approach is based on expression (4), substituting μ by λ , and then, for a given $E(Y)$ and σ , solving for λ in the equation.

$$E(Y) = \lambda - (\sigma - 1) \frac{{}_1F_1(1; \sigma; \lambda) - 1}{{}_1F_1(1; \sigma; \lambda)}. \quad (6)$$

Because λ appears inside the confluent hypergeometric function ${}_1F_1$, it cannot be isolated in a closed form, and must therefore be obtained numerically. Consequently, a closed expression for the PDF is not available.

Table 1: Comparative table between the two parametrizations of the hyper-Poisson distribution. The λ value in HYPERPO2 is obtained from the expression (6) given $E(Y)$ and σ .

Param.	pdf	$E(Y)$	$Var(Y)$
HYPERPO	Expression (1)	$\mu - (\sigma - 1) \frac{{}_1F_1(1; \sigma; \mu) - 1}{{}_1F_1(1; \sigma; \mu)}$	$\mu + [\mu - (\sigma - 1)] E(Y) - E^2(Y)$
HYPERPO2	Not available	μ	$\lambda + [\lambda - (\sigma - 1)] \mu - \mu^2$

As noted in [Sáez-Castillo and Conde-Sánchez \(2013\)](#), solving for λ in (6) represents a computational challenge during parameter estimation, since λ must be computed numerically at each evaluation of the log-likelihood.

2.2 The hyper-Poisson regression model

[Sáez-Castillo and Conde-Sánchez \(2013\)](#) extended the model of [Bardwell and Crow \(1964\)](#) to the situation in which the response variable Y is explained using the explanatory variables $X_1, X_2, \dots, X_p$. In this situation, y_i represents the observed value of Y and $x_{i1}, x_{i2}, x_{i3}, \dots, x_{ip}$ represents the observed values of the explanatory variables (with $i = 1, 2, \dots, n$). The approach proposed by [Sáez-Castillo and Conde-Sánchez \(2013\)](#) models the mean μ of Y and the dispersion parameter σ , rather than the canonical parameter λ . For this reason, we refer to this as the second parameterization of the hyper-Poisson distribution. The hyper-Poisson regression model under this parameterization can be expressed as:

$$\mu_i = \exp(W_i \beta), \quad (7)$$

$$\sigma_i = \exp(\mathbf{Z}_i \boldsymbol{\delta}). \quad (8)$$

The vectors $\mathbf{W}_i$ and $\mathbf{Z}_i$ are subsets of the explanatory variables $X_1, X_2, \dots, X_p$ and the vectors $\boldsymbol{\beta}$ and $\boldsymbol{\delta}$ are the effects vectors for the explanatory variables related to μ and σ , respectively. Given this definition, the model $hP(\mu_i(\boldsymbol{\beta}), \sigma_i(\boldsymbol{\delta}))$ now becomes a function of the parameters $\boldsymbol{\beta}$ and $\boldsymbol{\delta}$. These parameters are estimated using the maximum likelihood approach.

3 Generalized additive models for location, scale, and shape (GAMLSS)

Generalized Additive Models for Location, Scale, and Shape (GAMLSS) is a statistical framework that extends traditional generalized linear models (GLMs) to accommodate a wide range of response distributions, including non-Gaussian and heteroscedastic data (Rigby and Stasinopoulos, 2005). Within GAMLSS, the assumption of an exponential family distribution for the dependent (response) variable is relaxed and substituted with a general distribution family, which includes highly skewed and/or kurtosis, continuous and discrete distributions (Rigby and Stasinopoulos, 2005; Stasinopoulos and Rigby, 2008). GAMLSS expands the systematic component of the model, enabling the modeling not only of the location measure (such as mean, median, or mode) but also other parameters of the distribution of the dependent variable as linear and/or non-linear, parametric, and/or additive non-parametric functions of explanatory variables and/or random effects. Consequently, GAMLSS is particularly well-suited for modeling response variables that do not follow an exponential family distribution (Stasinopoulos and Rigby, 2008; Marmolejo-Ramos et al., 2022). This includes variables with characteristics such as leptokurtic or platykurtic distributions, positive or negative skewness, over-dispersed counts, or heterogeneity where the scale or shape of the distribution of the dependent variable changes with explanatory variable(s). Thus, GAMLSS enables modeling the relationships between predictors (covariates) and a response variable while also accounting for the distributional characteristics of the response variable. This makes GAMLSS a powerful tool for handling complex and diverse data types.

3.1 Model Frame

Consider a location regression model that establishes a relationship between a dependent (response) variable and one or more explanatory variables, defined by Nelder and Wedderburn (1972) as the general form of the Generalized Linear Model

$$g(\mu) = X\beta \quad (9)$$

where Y represents the response variable, μ denotes the expected value of Y , X is the matrix of predictor/explanatory variables (also known as the design matrix), β represents the vector of coefficients, and g is the link function. Equation (9) relates the linear predictor $X\beta$ to the expected value μ of the response variable through the link function g . The specific form of the link function and the choice of probability distribution for Y depend on the type of GLM used. The relationship between μ and $X\beta$ can be expressed using the inverse link function:

$$\mu = g^{-1}(X\beta) \quad (10)$$

When other statistical measures, such as variance, skewness, and kurtosis, are influenced by explanatory variables, the Generalized Additive Models for Location, Scale, and Shape (GAMLSS) framework becomes a plausible alternative to extend beyond mean regression models (Kneib, 2021; Ramires et al. (2021)). This extension generalizes both the concepts of Generalized Linear Models (Nelder and Wedderburn, 1972) and Generalized Additive Models (Hastie, 1990).

GAMLSS, a class of semi-parametric regression models, is designed to describe the response variable Y while accommodating diverse regression structures to explain any or all of its parameters using linear and/or nonlinear functions (Rigby and Stasinopoulos, 2005). Assuming that Y follows a distribution $D(\theta)$ with a probability density function as described in equation (1), and θ is its parameter vector associated with the explanatory variables, GAMLSS can be formulated as a function of regressors:

$$g_k(\theta_k) = \eta_k = X_k \beta_k + \sum_{j=1}^{J_k} s_{jk}(x_{jk}) \quad (11)$$

Here, $g_k(\cdot)$ refers to a suitable link function for the k -th parameter, generally determined by

the parameter range considered (Stasinopoulos et al., 2017). X_k is a known fixed $n \times (m_k + 1)$ model matrix, m_k denotes the number of explanatory variables related to the k -th parameter, $\beta_k = (\beta_{0k}, \beta_{1k}, \dots, \beta_{m_k k})^T$ represents a parameter vector of length $(m_k + 1)$, and $s_{jk}(\cdot)$ denotes smoothing functions that depend on variable X_{jk} , that is, $s_{jk}(x_{jk})$ is a vector that evaluates function s_{jk} at x_{jk} , the quantity J_k represents the number of explanatory variables X passed to the k -th parameter by the smoothing function. If the additive term $\sum_{j=1}^{J_k} s_{jk}(x_{jk}) = 0$ then, equation (11) reduces to parametric form. Furthermore, various additive terms like cubic splines, penalized splines, fractional polynomials, LOESS curves, ridge, principal component regression (PCR), simple random effects, varying coefficient, neural networks, kernels, regression trees, and LASSO and elastic net, etc, can be employed to address nonlinearity, structural changes, and other data peculiarities (Stasinopoulos et al., 2024).

Implementation of GAMLSS is available in R through packages such as **gamlss**, **gamlss.data**, and **gamlss.dist**, among others; these packages provide (penalized) likelihood-fitting algorithms, datasets, and distributions, respectively. Additionally, other packages can work in conjunction with GAMLSS; those packages are: the **gamboostLSS** package by Hofner et al. (2016) implements boosting methods for GAMLSS models, while the **bamlss** package by Umlauf et al. (2021) fits a GAMLSS model to data using the protocol of Bayesian MCMC. GAMLSS can also connect with machine learning techniques like neural networks, regression trees **pcr**, semi-structured distributional regression (Rügamer et al., 2024), and the Grammar of Graphics **ggplot2**.

3.2 Likelihood and link

Suppose that we have a dataset consisting of observations $y_1, y_2, \dots, y_n$ that is assumed to come from the hyper-Poisson distribution. The likelihood function $L(\mu, \sigma | y)$ for this data is the probability of observing the given data under the distribution in equation (1). The product of the individual PMF values for each observation is given as:

$$L(\mu, \sigma | y) = \prod_{i=1}^n \frac{\mu^{y_i}}{{}_1F_1(1; \mu; \sigma)} \frac{\Gamma(\sigma)}{\Gamma(y_i + \sigma)}. \quad (12)$$

To maximise the likelihood function and estimate the parameters μ and σ , we defined the log-likelihood function as:

$$\log L(\mu, \sigma | y) = \sum_{i=1}^n \left[y_i \log(\mu) - \log({}_1F_1(1; \mu; \sigma)) + \log\left(\frac{\Gamma(\sigma)}{\Gamma(y_i + \sigma)}\right) \right]. \quad (13)$$

4 Implementation of the hyper-poisson distribution in R

The **DiscreteDists** package implements the hyper-Poisson distribution within a GAMLSS framework. This package allows users to estimate the parameters for the distribution or estimate the effects of the regression model. The current version of the package is available on CRAN, and users can download it using the following code.

```
install.packages("DiscreteDists")
library(DiscreteDists)
```

In the **DiscreteDists** package, we implemented two parameterizations of the hyper-Poisson distribution:

- **HYPERP0(mu, sigma)**: this parameterization corresponds to the original proposal given in expression (1). The location parameter is $\mu > 0$, and the scale parameter $\sigma > 0$. In this parameterization the $E(Y)$ is a function of both parameters μ and σ , it is $E(Y) = \mu - (\sigma - 1)(1 - 1/{}_1F_1(1; \sigma; \mu))$.
- **HYPERP02(mu, sigma)**: this second parameterization represents an alternative parameterization. The location parameter is $\mu > 0$, and the scale parameter $\sigma > 0$. This parameterization is attractive because $E(Y) = \mu$, the expected value coincides with the location parameter μ .

In practice, users can use both parameterizations, **HYPERP0(mu, sigma)** or **HYPERP02(mu, sigma)**. The advantage of using the second parameterization is that the parameter μ coincides with the expected value $E(Y)$ of the response variable.

The **DiscreteDists** package provides the usual functions to calculate probabilities, cumulative probabilities, quantiles, and generate random samples. Table 2 summarizes the functions available for

each parameterization of the hyper-Poisson distribution. The main functions to estimate parameters within the GAMLSS framework are `HYPERPO` and `HYPERPO2`. Since the expected value and variance for the hyper-Poisson distribution do not have closed forms, the functions `mean_var_hp` and `mean_var_hp2` allow obtaining those values for both parametrizations. Additionally, Table 2 shows the related functions in the `DGLMExtPois` under each parametrization.

Table 2: Functions for the hyper-Poisson distribution in `DiscreteDists` and `DGLMExtPois` under two parametrizations.

Package	DiscreteDists		DGLMExtPois	
Function	First param.	Second param.	First param.	Second param.
Probability	dHYPERPO	dHYPERPO2	dhP	–
Cumulative	pHYPERPO	pHYPERPO2	phP	–
Quantile	qHYPERPO	qHYPERPO2	–	–
Random	rHYPERPO	rHYPERPO2	rhP	–
Mean-variance	mean_var_hp	mean_var_hp2	–	–
Estimation parameters	HYPERPO	HYPERPO2	–	glm_CMP

The proposed `DiscreteDists` R package integrates smoothly with the existing `gamlss` ecosystem of packages, such as `gamlss`, `gamlss.dist`, `gamlss.cens`, `bamlss`, among others. This ensures that both implementations of the hyper-Poisson distribution provided in the package can be used in the traditional way within the ecosystem by all users.

4.1 Examples for the first parameterization

This section provides examples of using the package functions for the first parameterization `HYPERPO`. Suppose we want to obtain probabilities for $HYPERPO(\mu = 5.5, \sigma = 0.1)$ when $y = 0, 1, \dots, 10$. To do this, we can use the following code:

```
library(DiscreteDists)
dHYPERPO(x=0:10, mu=5.5, sigma=0.1)

#> [1] 9.262073e-05 5.094140e-03 2.547070e-02 6.670898e-02 1.183546e-01
#> [6] 1.587684e-01 1.712208e-01 1.543795e-01 1.195897e-01 8.120290e-02
#> [11] 4.907867e-02
```

As mentioned, the hyper-Poisson distribution with $\sigma = 1$ coincides with the Poisson distribution. We can check this by comparing the results from $HYPERPO(\mu = 5.5, \sigma = 1)$ with $Poisson(\lambda = 5.5)$ as follows:

```
dHYPERPO(x=0:10, mu=5.5, sigma=1)

#> [1] 0.004086771 0.022477243 0.061812418 0.113322766 0.155818804 0.171400684
#> [7] 0.157117294 0.123449302 0.084871395 0.051865853 0.028526219

dpois(x=0:10, lambda=5.5)

#> [1] 0.004086771 0.022477243 0.061812418 0.113322766 0.155818804 0.171400684
#> [7] 0.157117294 0.123449302 0.084871395 0.051865853 0.028526219
```

Additionally to the `dHYPERPO` function, the `DiscreteDists` package contains the `pHYPERPO` function to obtain cumulative probabilities, `qHYPERPO` to obtain quantiles, and `rHYPERPO` to generate random values of the hyper-Poisson distributions. Below we present an example of how to simulate 500 values from $HYPERPO(\mu = 5.5, \sigma = 0.1)$.

```
set.seed(1234)
y <- rHYPERPO(n=500, mu=5.5, sigma=0.1)
y[1:15]

#> [1] 4 7 7 7 9 7 2 5 7 6 7 6 5 10 5
```

We can explore the empirical mean and variance from the `y` vector as follows:

```
mean(y)

#> [1] 6.478

var(y)

#> [1] 5.672862
```

The **DiscreteDists** package contains the function `mean_var_hp`, useful for obtaining the theoretical mean and variance for $HYPERP(\mu, \sigma)$ using the expressions (4) and (5). In the following, we can find the R code to obtain the theoretical mean and variance for $HYPERP(\mu = 5.5, \sigma = 0.1)$. We observe that the empirical mean and variance are close to the theoretical values.

```
mean_var_hp(mu=5.5, sigma=0.1)

#> $mean
#> [1] 6.399917
#>
#> $variance
#> [1] 5.500533
```

Recall that in the first parameterization of the hyper-Poisson distribution, the parameter μ is not the expected value $E(Y)$, for this reason, in this example $E(Y) = 6.399917$ whereas $\mu = 5.5$.

The **DiscreteDists** package contains the `HYPERP` function family, which allows for estimating parameters via the GAMLSS framework. Below we can find the R code to estimate μ and σ parameters using the `y` vector.

```
library(gamlss)
mod1 <- gamlss(y ~ 1, family=HYPERP)

#> GAMLSS-RS iteration 1: Global Deviance = 2261.591

summary(mod1)

#> *****
#> Family: c("HYPERP", "Hyper-Poisson")
#>
#> Call: gamlss(formula = y ~ 1, family = HYPERP)
#>
#> Fitting method: RS()
#>
#> -----
#> Mu link function: log
#> Mu Coefficients:
#> Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 1.71244 0.05607 30.54 <2e-16 ***
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> -----
#> Sigma link function: log
#> Sigma Coefficients:
#> Estimate Std. Error t value Pr(>|t|)
#> (Intercept) -2.748 4.559 -0.603 0.547
#>
#> -----
#> No. of observations in the fit: 500
#> Degrees of Freedom for the fit: 2
#> Residual Deg. of Freedom: 498
#> at cycle: 1
#>
#> Global Deviance: 2261.591
#> AIC: 2265.591
#> SBC: 2274.021
#> *****
```


From the last output we can find that $\hat{\mu} = \exp(1.71244) = 5.542469$ and $\hat{\sigma} = \exp(-2.748) = 0.06405584$, estimates that are close to the true values $\mu = 5.5$ and $\sigma = 0.1$.

In this last example, we simulate a dataset for the hyper-Poisson regression model with a response variable Y and two covariates X_1 and X_2 . The model to generate the dataset is given below.

$$\begin{aligned} y_i &\sim \text{HYPERPO}(\mu_i, \sigma_i), \\ \log(\mu_i) &= 1.21 - 3 \times X_{1i}, \\ \log(\sigma_i) &= 1.26 - 2 \times X_{2i}, \\ X_1 &\sim U(0, 1), \\ X_2 &\sim U(0, 1). \end{aligned} \tag{14}$$

In the following code, we are going to simulate 200 observations of the model and then we estimate the vector parameter $\Theta = (1.21, -3, 1.26, -2)^\top$ using the HYPERPO function family taking advantage of the GAMLSS framework.

```
# A function to simulate a data set with Y ~ HYPERPO2
gendat <- function(n) {
  x1 <- runif(n)
  x2 <- runif(n)
  mu    <- exp(1.21 - 3 * x1)
  sigma <- exp(1.26 - 2 * x2)
  y <- rHYPERPO(n=n, mu=mu, sigma=sigma)
  data.frame(y=y, x1=x1, x2=x2)
}

set.seed(1234)
dataset <- gendat(n=200)

mod2 <- gamlss(y~x1, sigma.fo=~x2, family=HYPERPO,
               control=gamlss.control(n.cyc=50, trace=FALSE),
               data=dataset)
```

Using the summary function, we can obtain a summary table with the usual information for a GAMLSS model.

```
summary(mod2)

#> *****
#> Family:  c("HYPERPO", "Hyper-Poisson")
#>
#> Call:  gamlss(formula = y ~ x1, sigma.formula = ~x2, family = HYPERPO,
#>      data = dataset, control = gamlss.control(n.cyc = 50, trace = FALSE))
#>
#> Fitting method: RS()
#>
#> -----
#> Mu link function:  log
#> Mu Coefficients:
#>      Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   1.0775     0.1255   8.587 2.62e-15 ***
#> x1           -2.5758     0.2935  -8.775 7.93e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> -----
#> Sigma link function:  log
#> Sigma Coefficients:
#>      Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   1.0599     0.2295   4.619 6.94e-06 ***
#> x2           -2.0979     0.4559  -4.601 7.49e-06 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#>
#> -----
#> No. of observations in the fit: 200
#> Degrees of Freedom for the fit: 4
#> Residual Deg. of Freedom: 196
#> at cycle: 45
#>
#> Global Deviance: 471.7851
#> AIC: 479.7851
#> SBC: 492.9784
#> *****
```

From the last summary we observe that $\hat{\Theta} = (1.0775, -2.5758, 1.0599, -2.0979)^\top$ while the true parameter vector is $\Theta = (1.21, -3, 1.26, -2)^\top$.

Since the examples for the second parameterization (HYPERPO2) follow the same structure as those already presented, they are omitted for brevity.

5 Simulation study

In this section, we showcase the outcomes of a Monte Carlo simulation study that examines parameter estimation for models with and without covariates under the second parameterization (HYPERPO2). To assess the effectiveness of estimating a general parameter θ , we employed the mean value and the Mean Squared Error (MSE) of the estimator $\hat{\theta}$. Those metrics are defined as follows:

$$\begin{aligned}\text{Mean value} &= \frac{\sum_{i=1}^k \hat{\theta}_i}{k} \\ \text{Mean Squared Error} &= \frac{\sum_{i=1}^k (\hat{\theta}_i - \theta)^2}{k}\end{aligned}$$

5.1 Model without covariates

During the first part of the simulation study, we utilized data generated from a $\text{HYPERPO2}(\mu, \sigma)$ distribution. The model employed for this segment can be succinctly summarized as follows:

$$\begin{aligned}y_i &\sim \text{HYPERPO2}(\mu, \sigma), \\ \mu &\in \{3, 7\}, \\ \sigma &\in \{0.5, 1, 1.5\}.\end{aligned}\tag{15}$$

For each value of sample size $n = 50, 100, 200, \dots, 1000$, we followed the next steps to simulate the data and estimate the parameters:

- Generate a sample of size n from a $\text{HYPERPO2}(\mu, \sigma)$.
- Obtain and store the estimations $\hat{\mu}$ and $\hat{\sigma}$ using the proposed procedure.
- Repeat steps 1 to 2 $k = 1,000$ times.

5.2 Model with covariates

In the second part of the simulation study, we considered data generated from the next model:

$$\begin{aligned}y_i &\sim \text{HYPERPO2}(\mu_i, \sigma_i), \\ \log(\mu_i) &= \beta_0 + \beta_1 \times X_{1i} + \beta_2 \times X_{11i}, \\ \log(\sigma_i) &= \gamma_0 + \gamma_1 \times X_{2i} + \gamma_2 \times X_{22i}, \\ X_1 &\sim U(0, 1), \\ X_{11} &\sim U(0, 1), \\ X_2 &\sim U(0, 1), \\ X_{22} &\sim U(0, 1).\end{aligned}\tag{16}$$

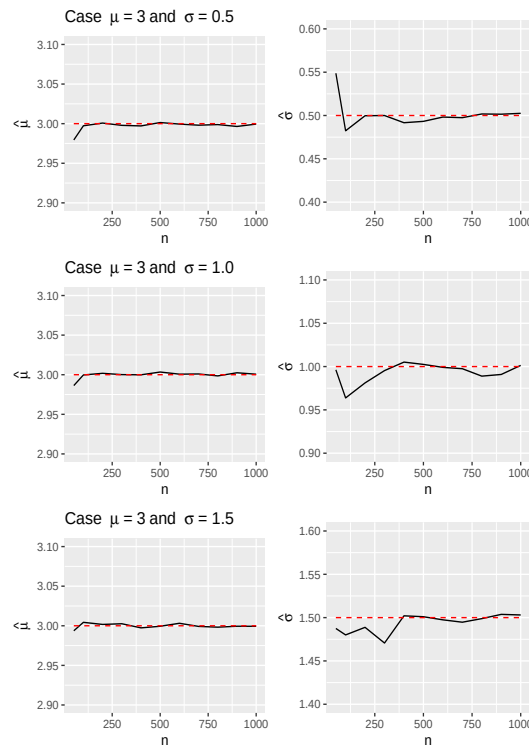


Figure 2: Mean for the estimated parameters $\hat{\mu}$ (on left) and $\hat{\sigma}$ (on right) versus n for $\mu = 3$ and $\sigma = 0.5, 1.0, 1.5$. Red horizontal lines correspond to the true values of μ and σ .

The parameter vector was fixed as $\theta = (\beta_0 = 1.21, \beta_1 = -3.0, \beta_2 = 2.0, \gamma_0 = 1.26, \gamma_1 = -2.0, \gamma_2 = 1.5)^\top$. From model (16), we have that $X_1 \sim U(0, 1)$ and $X_1 1 \sim U(0, 1)$, it implies that $\mu \approx \exp(1.21 - 3.0 \times 0.5 + 2.0 \times 0.5) = 2.03$. The same analysis can be done for the σ parameter to obtain $\sigma \approx 2.75$.

For each value of sample size $n = 50, 100, 200, \dots, 1000$, we followed the next steps to simulate the data and estimate the parameters:

- Generate a random sample of n values from a *HYPERPO2* distribution given in expression (16).
- Obtain and store the estimations $\hat{\beta}_0, \hat{\beta}_1, \hat{\gamma}_0$ and $\hat{\gamma}_1$ using the proposed procedure.
- Repeat steps 1 to 2 for $k = 1, 000$ times.

All scripts used in the simulation study are available as supplementary material in a GitHub repository at https://github.com/fhernanb/HYPERPO_paper.

5.3 Results for the simulations without covariates

Figures 2 and 3 depict the mean values of the estimated parameters $\hat{\mu}$ and $\hat{\sigma}$ plotted against the sample size n for several combinations of μ and σ . A general pattern observed is that as the sample size increases, the estimated parameters tend towards the true values, indicated by the horizontal red lines.

Figures 4 and 5 depict the Mean Squared Error (MSE) of the estimated parameters $\hat{\mu}$ and $\hat{\sigma}$ plotted against the sample size n for various combinations of μ and σ . A consistent trend observed is that with an increase in the sample size, the MSE converges toward zero, as expected.

5.4 Results for the simulations with covariates

Figures 6 and 7 illustrate the mean values and Mean Squared Error (MSE) of the estimated parameters $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, \hat{\gamma}_0, \hat{\gamma}_1$ and $\hat{\gamma}_2$ plotted against the sample size n . A prevailing trend observed is that with an increase in the sample size, the estimated parameters converge towards the true values, denoted by the horizontal red lines, while the MSE approaches zero.

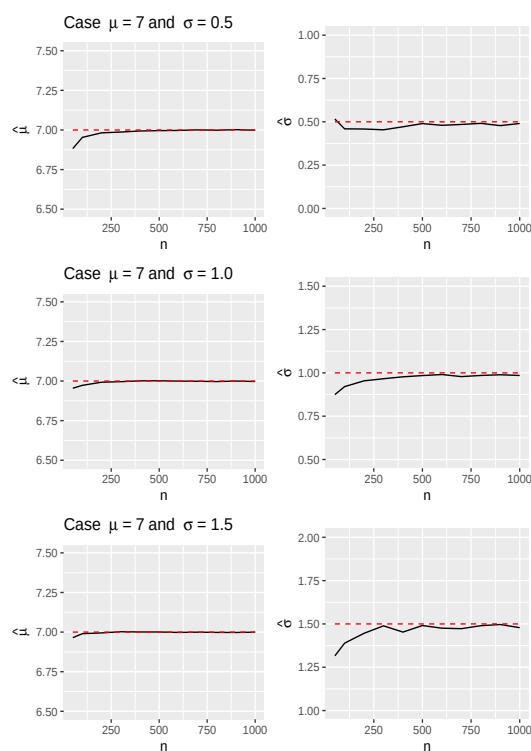


Figure 3: Mean for the estimated parameters $\hat{\mu}$ (on left) and $\hat{\sigma}$ (on right) versus n for $\mu = 7$ and $\sigma = 0.5, 1.0, 1.5$. Red horizontal lines correspond to the true values of μ and σ .

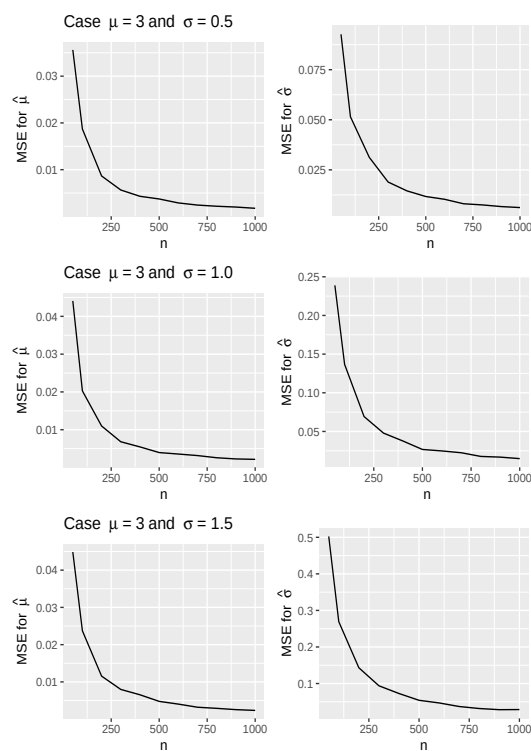


Figure 4: MSE for the estimated parameters $\hat{\mu}$ (on left) and $\hat{\sigma}$ (on right) versus n for $\mu = 3$ and $\sigma = 0.5, 1.0, 1.5$. Red horizontal lines correspond to the true values of μ and σ .

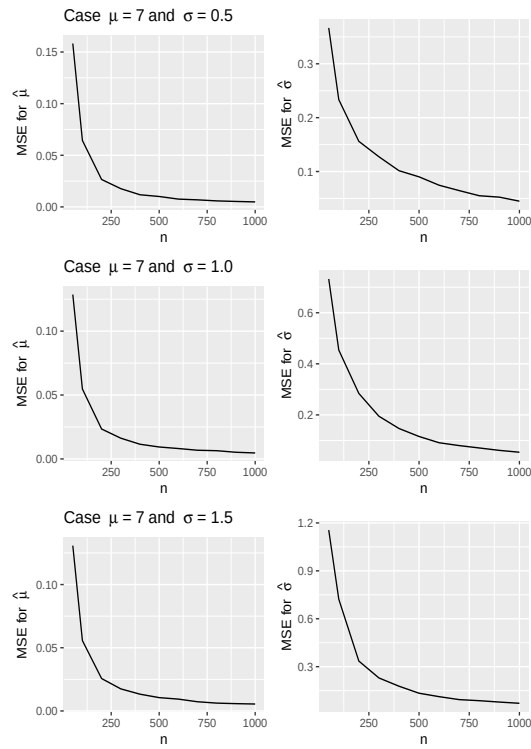


Figure 5: MSE for the estimated parameters $\hat{\mu}$ (on left) and $\hat{\sigma}$ (on right) versus n for $\mu = 7$ and $\sigma = 0.5, 1.0, 1.5$. Red horizontal lines correspond to the true values of μ and σ .

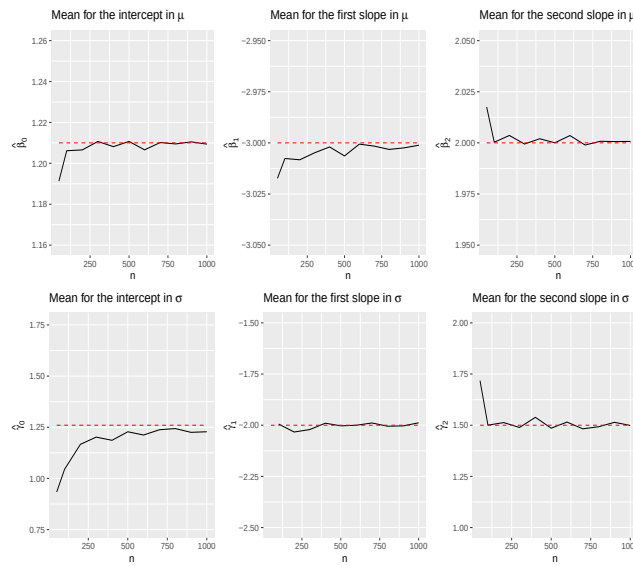


Figure 6: Mean for the estimated parameters $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, \hat{\gamma}_0, \hat{\gamma}_1$ and $\hat{\gamma}_2$ versus n . Red horizontal lines correspond to the objective value.

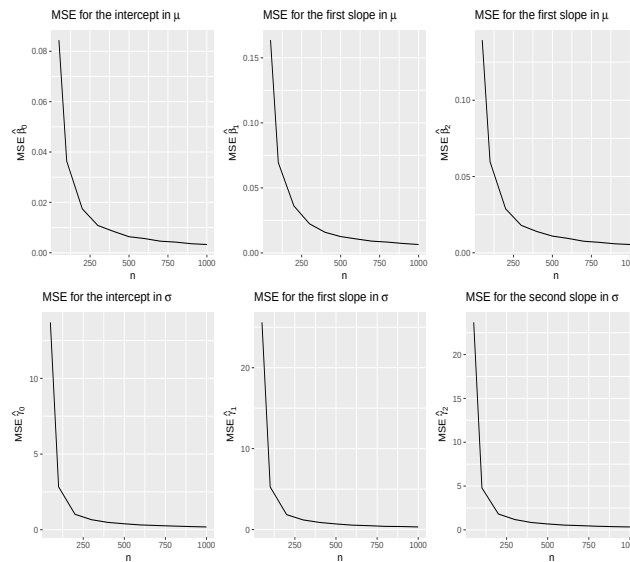


Figure 7: Mean Squared Error for the estimated parameters $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, \hat{\gamma}_0, \hat{\gamma}_1$ and $\hat{\gamma}_2$ versus n .

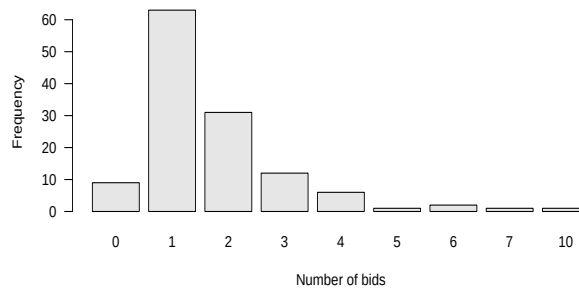


Figure 8: Histogram for the number of bids received by US firms.

6 Application

This section presents the proposed regression model for analyzing the number of bids of some US firms. We consider data from [Cameron and Johansson \(1997\)](#) and revisited by [Sáez-Castillo and Conde-Sánchez \(2013\)](#) regarding the number of bids received by 126 US firms that were targets of tender offers between 1978 and 1985. These firms were ultimately acquired within 52 weeks of the initial offer. The dependent variable of interest is the count of bids the target firm receives after the initial bid (numbids). The explanatory variables are:

- `leglrest`: equals 1 if legal defense by lawsuit.
- `rearest`: equals 1 if proposed changes in asset structure.
- `finrest`: equals 1 if proposed changes in ownership structure
- `whtknight`: equals 1 if management invitation for friendly third-party bid.
- `bidprem`: bid price divided by price 14 working days before bid.
- `insthold`: percentage of stock held by institutions.
- `size`: total book value of assets in billions of dollars.
- `regulatn`: equals 1 if intervention by the federal regulator.

The dataset Bids is available in the [Ecdat](#) R package by [Croissant and Graves \(2022\)](#). Figure 8 depicts a barplot with the frequency for the response variable number of bids received by US firms. The mean and variance for the number of bids are 1.74 and 2.05, respectively. As the variance is slightly greater than the mean, this means that the data present overdispersion.

In this illustration, we fitted the same model considered by [Sáez-Castillo and Conde-Sánchez \(2013\)](#) using the second parameterization for the hyper-Poisson distribution (HYPERPO2). The target model can be summarized as follows:

$$\begin{aligned}
\text{numbids} &\sim \text{HYPERPO2}(\mu, \sigma), \\
\log(\mu) &= \beta_0 + \beta_1 \text{leglrest} + \beta_2 \text{rearest} + \beta_3 \text{finrest} + \beta_4 \text{whtknight} \\
&\quad + \beta_5 \text{bidprem} + \beta_6 \text{insthold} + \beta_7 \text{size} + \beta_8 \text{size}^2 + \beta_9 \text{regulatn}, \\
\log(\sigma) &= \gamma_0 + \gamma_1 \text{rearest} + \gamma_2 \text{finrest} + \gamma_3 \text{bidprem} + \gamma_4 \text{regulatn}.
\end{aligned} \tag{17}$$

With the R code below, we can fit the model using the proposed functions in the [DiscreteDists](#) R package ([Hernandez-Barajas et al., 2025](#)).

```

library(Ecdat)
library(DiscreteDists)
library(gamlss)
mod3 <- gamlss(numbids ~ leglrest + rearest + finrest +
               whtknight + bidprem + insthold +
               size + I(size^2) + regulatn,
               sigma.fo= ~ rearest + finrest +
               bidprem + regulatn,
               family=HYPERPO2,
               data=Bids,
               control=gamlss.control(n.cyc=150, trace=TRUE))

```

After adjusting the model, we obtain the parameter estimates shown in Table 3. This model has a $BIC = 386.48$ and pseudo $R^2 = 0.47$, better values when compared to a classical Poisson model with $BIC = 418.25$ and pseudo $R^2 = 0.23$.

Estimated effects in modeling $\log(\mu)$				
	Estimate	Std. Error	t-value	$P(> t)$
Intercept	1.8625	0.1390	13.3965	0.0000
leglrest	0.2048	0.1215	1.6863	0.0946
rearest	-0.3489	0.1506	-2.3159	0.0224
finrest	0.4368	0.2098	2.0819	0.0397
whtknight	0.3617	0.1078	3.3573	0.0011
bidprem	-1.1770	0.0928	-12.6789	0.0000
insthold	-0.9665	0.1016	-9.5084	0.0000
size	0.1753	0.0344	5.0958	0.0000
size ²	-0.0094	0.0022	-4.2336	0.0000
regulatn	0.2551	0.1386	1.8400	0.0684
Estimated effects in modeling $\log(\sigma)$				
Intercept	31.1371	0.0944	329.9483	0.0000
rearest	4.1331	1.1658	3.5452	0.0006
finrest	9.9899	2.1817	4.5790	0.0000
bidprem	-29.2006	0.0908	-321.6851	0.0000
regulatn	6.3391	1.1863	5.3436	0.0000

Table 3: Estimated effects for $\log(\mu)$ and $\log(\sigma)$ to model the Number of Bids using the model (17).

Figure 9 shows the worm plot and the Randomized Quantile Residual (RQR) plot ([van Buuren and Fredriks, 2001](#)). From the worm plot, we can observe that the points and the red curve are inside the dashed curve lines, indicating that the fitted model explains the observed pattern for the response variable. Additionally, the RQR plot indicates no departure from the expected pattern from a Normal distribution for the Randomized Quantile Residuals.

7 Conclusions

The hyper-Poisson regression model, which can handle both under-dispersed and over-dispersed data, is integrated into the Generalized Additive Models for Location, Scale, and Shape (GAMLSS) framework using the [DiscreteDists](#) package. The new package addresses the shortcomings of the previously proposed [DGLMExtPois](#) package, which lacks functions for computing probabilities, quantiles, and generating random values for the hyper-Poisson distribution. The [DiscreteDists](#)

- A. Huang. Mean-parametrized Conway–Maxwell–Poisson regression models for dispersed counts. *Statistical Modelling*, 17(6):359–380, 2017. doi: 10.1177/1471082X176977. [p127]
- N. L. Johnson, A. W. Kemp, and S. Kotz. *Univariate discrete distributions*, volume 444. John Wiley & Sons, 2005. [p127]
- T. Kneib. Beyond mean regression. *Statistical Modelling*, 13(4):275–303, 2021. [p130]
- F. Marmolejo-Ramos, R. Ospina, and F. Hernández-Barajas. The place of probability distributions in statistical learning. *Australian & New Zealand Journal of Statistics*, 64(3):406–412, 2022. [p130]
- F. Marmolejo-Ramos, M. Tejo, M. Brabec, J. Kuzilek, S. Joksimovic, V. Kovanović, J. González, T. Kneib, P. Bühlmann, L. Kook, G. Briseño-Sánchez, and R. Ospina. Distributional regression modeling via generalized additive models for location, scale, and shape. *WIREs Data Mining and Knowledge Discovery*, 13(1):e1479, 2023. [p127]
- J. A. Nelder and R. W. M. Wedderburn. Generalized Linear Models. *Journal of the Royal Statistical Society. Series A*, 135(3):370–384, 1972. [p130]
- A. Peluso, V. Vinciotti, and K. Yu. Discrete Weibull generalized additive model: An application to count fertility data. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 68(3):565–583, 2019. doi: 10.1111/rssc.12311. [p127]
- T. G. Ramires, L. R. Nakamura, A. J. Righetto, R. J. Carvalho, L. A. Vieira, and C. A. B. Pereira. Comparison between Highly Complex Location Models and GAMLSS. *Entropy*, 23(4), 2021. [p130]
- E. E. Ribeiro Jr, W. M. Zeviani, W. H. Bonat, C. G. Demetrio, and J. Hinde. Reparametrization of COM–Poisson regression models with applications in the analysis of experimental data. *Statistical Modelling*, 20(5):443–466, 2020. [p129]
- R. A. Rigby and D. M. Stasinopoulos. Generalized Additive Models for Location, Scale and Shape. *Journal of the Royal Statistical Society*, 54(3):507–554, 2005. doi: 10.1111/j.1467-9876.2005.00510.x. [p130]
- R. A. Rigby, M. D. Stasinopoulos, G. Z. Heller, and F. De Bastiani. *Distributions for modeling location, scale, and shape: Using GAMLSS in R*. Chapman and Hall/CRC, 2019. [p129]
- D. Rügamer, C. Kolb, and N. Klein. Semi-structured distributional regression. *The American Statistician*, 78(1):88–99, 2024. doi: 10.1080/00031305.2022.2164054. [p131]
- A. Sáez-Castillo and A. Conde-Sánchez. A hyper-Poisson regression model for overdispersed and underdispersed count data. *Computational Statistics & Data Analysis*, 61:148–157, 2013. doi: 10.1016/j.csda.2012.12.009. [p127, 129, 139]
- A. J. Sáez-Castillo, A. Conde-Sánchez, and F. Martínez. DGLMExtPois: Advances in Dealing with Over and Under-dispersion in a Double GLM Framework. *The R Journal*, 14:121–140, 2023. doi: 10.32614/RJ-2023-002. [p127]
- K. F. Sellers, S. Borle, and G. Shmueli. The COM-Poisson model for count data: a survey of methods and applications. *Applied Stochastic Models in Business and Industry*, 28(2):104–116, 2012. doi: 10.1002/asmb.918. [p127]
- D. M. Stasinopoulos and R. A. Rigby. Generalized additive models for location scale and shape (GAMLSS) in R. *Journal of Statistical Software*, 23:1–46, 2008. doi: 10.18637/jss.v023.i07. [p127, 130]
- D. M. Stasinopoulos, A. R. Rigby, Z. G. Heller, V. Voudouris, and F. De Bastiani. Flexible Regression and Smoothing: Using GAMLSS in R. *Chapman and Hall/CRC*, 2017. [p131]
- M. D. Stasinopoulos, T. Kneib, N. Klein, A. Mayr, and G. Z. Heller. *Generalized Additive Models for Location, Scale and Shape: A Distributional Regression Approach, with Applications*. Cambridge University Press, 2024. [p131]
- N. Umlauf, N. Klein, T. Simon, and A. Zeileis. bamlss: A Lego Toolbox for Flexible Bayesian Regression. *Journal of Statistical Software*, 100(4):1–53, 2021. doi: 10.18637/jss.v100.i04. [p131]
- S. van Buuren and M. Fredriks. Worm plot: a simple diagnostic device for modelling growth reference curves. *Statistics in Medicine*, 20:1259–1277, 2001. doi: 10.1002/sim.746. [p140]
- R. Winkelmann and K. F. Zimmermann. A new approach for modeling economic count data. *Economics Letters*, 37(2):139–143, 1991. doi: 10.1016/0165-1765(91)90122-2. [p127]

Freddy Hernández-Barajas
Universidad Nacional de Colombia - sede Medellín
Carrera 65 Nro. 59A - 110
Medellín, Colombia
<https://fhernanb.github.io/>
ORCID: 0000-0001-7459-3329
fhernanb@unal.edu.co

Jamiu S. Olumoh
American University of Nigeria
Department of Mathematics and Statistics, Yola, 640001
Nigeria
ORCID: 0000-0002-7371-3920
jamiu.olumoh@aun.edu.ng

Osho O. Ajayi
American University of Nigeria
Department of Mathematics and Statistics, Yola, 640001
Nigeria
ORCID: 0000-0002-2436-3338
osho.ajayi@aun.edu.ng

Fernando Marmolejo-Ramos
Flinders University
Adelaide, 5042
Australia
ORCID: 0000-0003-4680-1287
fernando.marmolejoramos@flinders.edu.au

Estimation of the Log Gaussian Cox Process Model with Point and Aggregated Case Data: The `rts2` Package for R

by Samuel I. Watson

Abstract Log Gaussian Cox Process (LGCP) models are a key method for disease and other types of surveillance, and provide a means of predicting risk across an area of interest based on spatially-referenced and time-stamped case data. However, these models can be difficult to specify and computationally demanding to estimate. For many surveillance scenarios we require results in near real-time using routinely available data to guide and direct policy responses, or due to limited availability of computational resources. There are limited software implementations available for this real-time context with reliable predictions and quantification of uncertainty. The `rts2` package provides a range of modern Gaussian process approximations and model fitting methods to fit the LGCP, including estimation of covariance parameters, using both Bayesian and stochastic Maximum Likelihood methods. The package provides a suite of data manipulation tools. We also provide a new implementation to estimate the LGCP when case data are aggregated to an irregular grid such as census tract areas.

1 Introduction

The statistical analysis of geolocated and possibly time-stamped data representing cases of a phenomenon of interest can help identify areas with a high probability of high or growing risk. In this article, we focus primarily on disease surveillance, but other applications include social policy, crime, ecological, geological, or other point process phenomena. In the context of disease surveillance, prevalence mapping surveys are a rigorous method for measuring disease risk and conduct a 0/1 test on a sample from a population in a given area to predict the spatial distribution of risk (Diggle and Giorgi, 2016). Recent examples include the mapping of seroprevalence during the SARS-CoV-2 pandemic (e.g. Riley et al. (2020)). However, such studies require significant time and resources. In many contexts there is a need for more rapid analyses of case data to be able to quickly identify and respond to emerging disease outbreaks. Again, SARS-CoV-2 provides a good example (Watson et al., 2021), but in many low- and middle-income country settings identifying emerging clusters of diseases like cholera, malaria, or influenza can facilitate targeted intervention to prevent the development of an epidemic (e.g. Ratnayake et al. (2020b,a)). Beyond emergency applications, reliable quantification of disease risk across an area of interest using routine data provides a useful tool for public health surveillance and research. Obtaining predictions in a reasonable amount of time without using excessive computational resources can facilitate their adoption.

Many healthcare and public health systems collect finely spatially-resolved and time-stamped data that can be used to monitor disease epidemiology. These data may include daily hospital admissions, positive tests, or calls to public health telephone services (Diggle et al., 2005). Case data are often also provided as counts at a spatially aggregated level, such as census tracts. Geospatial statistical models provide a principled basis for generating predictions of disease epidemiology and its spatial and temporal distribution from these data.

The `rts2` package provides data manipulation and model fitting methods for when these data can be assumed to be a realisation from a spatio-temporal point process over the area of interest. A point, such as the residential address of a hospital admission on a particular date, is assumed to be more likely to be observed in areas of higher risk than lower risk. Analysis of these data can provide probabilistic predictions about the underlying disease risk and be used to quantify our uncertainty about the presence of “hotspots”.

A growing use case for spatio-temporal point process models is “real-time” disease surveillance that uses a stream of real-time data, such as hospital admissions, to provide rapid feedback to public health authorities. However, geospatial models can be computationally intensive and slow to run, and they can scale poorly with the size of the data. ‘Fast’ methods like kernel-based mapping require often difficult selection of bandwidth parameters and may have less reliable quantification of uncertainty. [rts2](#) provides a range of Gaussian process approximations and approximate inference, including both Bayesian and Frequentist approaches, to support faster and more scalable analyses for real-time scenarios. We first describe the statistical models and model fitting and other relevant packages for R, and then provide a discussion of the use of the package and generation and interpretation of outputs.

2 The log gaussian cox process

Our statistical model is a Log Gaussian Cox process (LGCP) ([Diggle et al., 2013](#)), whose realization is observed on the Cartesian area of interest $A \subset \mathbb{R}^2$ and time period of interest $T \subset \mathbb{N}$ (we assume discrete time periods). The resulting data are realisations of an inhomogeneous Poisson process with stochastic intensity function $\{\lambda(s, t) : s \in A, t \in T\}$. The number of cases occurring in any locally finite random set $S \subseteq A$ is Poisson distributed conditional on $\lambda(s, t)$:

$$Y(S, t) \sim \text{Poisson} \left(\int_S \lambda(s, t) ds \right) \quad (1)$$

The intensity is decomposed as the log-linear model:

$$\lambda(s, t) = r(s, t) \exp(X(s, t)\gamma + Z(s, t)) \quad (2)$$

where $r(s, t)$ is a spatially or spatio-temporally varying Poisson offset, typically the population density. $X(s, t)$ is a length Q row vector of spatially, temporally, or spatio-temporally varying covariates including an intercept and $\{Z(s, t) : s \in A, t \in T\}$ is a latent field.

We use an auto-regressive specification for the latent field:

$$\begin{aligned} Z(s, 1) &= (1 + \rho^2)^{-1} \mathcal{Z}_1(s) \\ Z(s, t) &= \rho Z(s, t-1) + \mathcal{Z}_t(s) \text{ for } t > 1 \end{aligned} \quad (3)$$

where ρ is the auto-regressive parameter. This specification, sometimes referred to as a “spatial innovation” model, facilitates computationally efficient model fitting as we describe in the subsequent section. We include both Bayesian and maximum likelihood model fitting approaches in the package and, depending on the paradigm, the spatial innovation term $\{\mathcal{Z}_t(s) : s \in A\}$ is given a Gaussian process prior distribution, or is itself a Gaussian process. In both cases, the realisations of $\mathcal{Z}_t(s)$, which we notate as $\mathbf{z}$ are multivariate Gaussian distributed with mean zero and covariance Σ . We use a minimally parameterised covariance function for the Gaussian process to define the elements of Σ :

$$\text{cov}(\mathcal{Z}_t(s), \mathcal{Z}_t(s')) = \sigma^2 h(\|s - s'\|; \phi) \quad (4)$$

where $\|\cdot\|$ is the L2-norm. The package includes, for example, the squared exponential covariance function

$$h(\|s - s'\|; \phi) = \exp \left(\frac{-(\|s - s'\|)^2}{\phi^2} \right) \quad (5)$$

and the exponential covariance function

$$h(\|s - s'\|; \phi) = \exp \left(\frac{-\|s - s'\|}{\phi} \right) \quad (6)$$

where ϕ is the spatial range (or length scale) parameter.

The LGCP is complex to estimate as the marginal likelihood does not have a closed form solution. We include methods and approximations where there do not currently exist implementations in other software packages, which are discussed in Section 7.

Model fitting in this package uses the standard practice of dividing the area of interest into a regular grid (Diggle et al., 2013; Taylor et al., 2013, 2015) with grid cells $S_i : i = 1, \dots, n$. The size of the grid cells should permit the assumption that the latent field Z is approximately constant within each cell. The counts in each of the n grid cells comprise our data $\mathcal{D} = \{Y(S_i, t) : i = 1, \dots, n; t = 1, \dots, T\}$ such that our model is:

$$\begin{aligned} Y(S_i, t) &\sim \text{Poisson}(\lambda(S_i, t)) \\ \lambda(S_i, t) &= r(S_i, t) \exp(X(S_i, t)\gamma + Z(S_i, t)) \end{aligned} \quad (7)$$

where Z is as specified in Equation (3). For the Bayesian approaches we also require specification of priors and hyperpriors for the remaining model parameters as we discuss below.

For a given data set $\mathcal{D}$, we write the nT -length vector of realisations of the Gaussian process model as $\mathbf{z}$. We can then write the likelihood of the model parameters as

$$L(\gamma, \theta | \mathcal{D}) = \int \prod_{i=1}^n \prod_{t=1}^T f_{Y|z}(Y(S_i, t) | \gamma, \mathbf{z}) dF_{z|\theta}(\mathbf{z} | \theta) \quad (8)$$

where $f_{Y|z}$ is the Poisson probability density function with intensity given in (7) and $f_{z|\theta}$ is the multivariate Gaussian density with parameters $\theta = [\sigma^2, \phi, \rho]$. The components of the log-likelihood are therefore:

$$\begin{aligned} \log(f_{Y|z}(Y(S_i, t) | \gamma, \mathbf{z})) &= Y(S_i, t)\eta(S_i, t) - \lambda(S_i, t) - \log(Y(S_i, t)!) \\ \log(f_{z|\theta}(\mathbf{z} | \theta)) &= -\frac{nT}{2} \log(2\pi) - \frac{1}{2} \log(|\Sigma|) - \frac{1}{2} \mathbf{z}^T \Sigma^{-1} \mathbf{z} \end{aligned} \quad (9)$$

where $\eta(S_i, t) = \log(r(S_i, t)) + X(S_i, t)\gamma + Z(S_i, t)$. The autoregressive specification of the latent field means we can partially reduce the complexity of the log likelihood. We can write:

$$\Sigma = P \otimes \Sigma_0$$

where Σ_0 is the covariance matrix for a single time period and

$$P = \begin{bmatrix} 1 & \rho & \rho^2 & \dots & \rho^T \\ \rho & 1 & \rho & \dots & \rho^{T-1} \\ \vdots & & \ddots & \vdots & \vdots \\ \rho^T & \rho^{T-1} & \rho^{T-2} & \dots & 1 \end{bmatrix}$$

Letting $\mathbf{z}_{[t]}$ refer to the elements of $\mathbf{z}$ for time period t and $\mathbf{v}$ be an $n \times T$ matrix formed by stacking the elements of $\mathbf{z}$ into columns with column t equal to $\mathbf{z}_{[t]}$, we can then write:

$$\begin{aligned} \mathbf{z}^T \Sigma^{-1} \mathbf{z} &= \mathbf{z}^T (P \otimes \Sigma_0)^{-1} \mathbf{z} \\ &= \mathbf{z}^T \text{Vec}(\Sigma_0^{-1} \mathbf{v} P^{-1}) \\ &= \sum_t \mathbf{z}_{[t]}^T \Sigma_0^{-1} \tilde{\mathbf{v}}_{[t]} \end{aligned}$$

where $\tilde{\mathbf{v}} = \mathbf{v} P^{-1}$. Similarly $\log(|\Sigma|) = n \log(|P|) + T \log(|\Sigma_0|)$. Computational time with this approach is therefore linear in the number of time periods. However, evaluation of Σ_0^{-1} is still computationally demanding and scales as $O(n^3)$, which may still be prohibitive for many real-time surveillance applications.

2.1 Model for spatially-aggregated case data

The models discussed so far have assumed that the precise location of each case is known. However, in many cases we will have instead case counts aggregated to census or administrative areas for reasons such as preserving anonymity. These areas are almost always irregularly shaped and have a large variability in area. As a result, we cannot necessarily assume that the underlying latent process is approximately constant within each area, as we do with the regular lattice, which can cause bias. We can instead extend the LGCP model described above to account for the irregular areas onto which case counts are aggregated.

Similar to Li et al. (2012a), we notate the count data for this ‘region’ model as $Y(R_j, t)$ where R_j is the j th region (i.e. irregular area) of r total regions. The intersection areas between the regions and the computational grid are $Q_{ij} = R_j \cap S_i$ where there are q non-empty intersection areas. Letting λ_s be the $n \times T$ vector of grid-level intensities in (7), and y_r be the $r \times T$ vector of region level outcome variables. We consider models of the form:

$$y_r \sim \text{Poisson}(W\lambda_s) \quad (10)$$

where W is a weight matrix with elements in the j th row and i th column of $W_{ji} = \frac{|Q_{ij}|}{|S_i|}$. This formulation of the spatially aggregated model only uses the information that the intensities in the regions are the sums of the intersecting grid cells. The intensity for cell i is:

$$\lambda_{s,i} = \exp(X_s(S_i, t)\gamma + Z(S_i, t)) \quad (11)$$

Population density information can be added from either the region or cell level by appropriately multiplying the rows or columns, respectively.

Li et al. (2012a), building on work in Li et al. (2012b), describe a data augmentation scheme for estimating the LGCP with aggregated case data. Their approach models the counts at the intersection level. As the locations of each case within a region are not known, and so it is not known which computational grid cell to add them to, they assign a case to a grid cell within the region with probability proportional to the $w_{ij} \exp(Z(S_i, t))$ on each iteration of the MCMC sampling and then model the intersection intensities. Despite proposing an INLA approach to modelling the random field, the method is computationally intensive given the number of intersections and the need for a data simulation step. In [rts2](#) we instead aim to model the counts at the region level directly and instead aggregate the intensities of the intersections, which is a more efficient method of sampling from or fitting an equivalent model. We assume that the population density and any covariates for the regions are constant throughout the area; the model is then amenable to the fitting strategies discussed in this article.

3 Maximum likelihood estimation

For maximum likelihood estimation we use the algorithm described in [Watson et al. \(2026\)](#). We provide a brief overview of the algorithm here. The method is a Markov Chain Monte Carlo Maximum Likelihood (MCML) algorithm, which is an expectation-maximisation type algorithm. The marginal log-likelihood of the model parameters includes an intractable integral averaging over the latent effect terms. To approximate this algorithm, MCML algorithms use Monte Carlo samples of the latent effects to generate estimates of the averages. MCML algorithms for mixed models are described by [McCulloch \(1997\)](#) and implemented for GLMMs in general in the [glmmrBase](#) package for R, which we build on for model fitting in this package. The MCML algorithm has three steps that are implemented on each iteration. On the k th iteration:

1. Sample m_k latent effects $\mathbf{z}^{(k)}$ from the distribution $\mathbf{z}|Y$ using importance sampling, with a Gaussian proposal distribution.

2. Update the estimates $\hat{\gamma}^{(k)}$ from the likelihood of Y conditional on $\mathbf{z}^{(k)}$ using a Newton-Raphson step averaging the gradient and Hessian over the latent effect samples.
3. Update the estimates of the covariance parameters $\hat{\theta}^{(k)}$ using a Newton-Raphson step averaging the gradient and Hessian over the latent effect samples.

The steps are then repeated until convergence, see below.

Stopping criteria

MCML algorithms are subject to Monte Carlo error so do not converge to a single point estimate, but reach a stationary, post-convergence phase where the log-likelihood does not improve, on average. Previously proposed stopping criteria for these algorithms, such as Caffo et al. (2005), consider a null hypothesis test that the running mean of the difference in log-likelihood values is zero. However, passing such a test may require a relatively large number of post-convergence iterations, which may be computationally burdensome for spatial models. Watson et al. (2026) propose a different stopping criterion for stochastic gradient descent algorithms for GLMMs. In particular, if the difference in successive likelihood values is:

$$\Delta\mathcal{L}(k|k-1) = \mathcal{L}^{(k)}(\hat{\gamma}^{(k)}, \hat{\theta}^{(k)}) - \mathcal{L}^{(k)}(\hat{\gamma}^{(k-1)}, \hat{\theta}^{(k-1)})$$

with μ_Δ the mean difference. Then the algorithm is halted based on the Bayes Factor:

$$BF = \frac{Pr(\mu_\Delta \leq 0 | \mathcal{D}_t)}{Pr(\mu_\Delta > 0 | \mathcal{D}_t)} = \frac{1 - p_t}{p_t} \frac{1 - \pi_t}{\pi_t}$$

where $\mathcal{D}_t$ is the values of the difference in log-likelihood values from the preceding recent iterations, p_t is the one-sided p-value from a null hypothesis test of $H_0 : \mu_\Delta > 0$ versus $H_1 : \mu_\Delta \leq 0$ and π_t , which is the prior probability $Pr(\mu_\Delta \leq 0)$ at iteration t . The prior probability is determined by a Weibull model with expected number of iterations until convergence of k_0 . Thus, the stopping criterion is specified according to the threshold for BF and values of k_0 and the history length.

Region model

Further details of the model fitting algorithm are provided in Watson et al. (2026). We provide a discussion of extensions to the algorithm for spatially aggregated data here. The first step is to update the proposal distribution of the latent effects, the posterior mean of which is updated using an iteratively re-weighted least squares procedure. The random effects are sampled on a transformed scale $\mathbf{v} \sim N(0, I)$. Letting L be the Cholesky decomposition of $\Sigma = LL^T$, $\lambda_r = W\lambda_s$, $\boldsymbol{\psi}$ be a vector with i th element $y_i / \lambda_{r,i}$, then the update to the posterior mean of $\mathbf{v}$, $\bar{\mathbf{v}}$ is:

$$\begin{aligned} C &= \text{diag}(\lambda_r^{-\frac{1}{2}}) W \text{diag}(\lambda_s) L \\ \mathbf{a} &= C^T C \bar{\mathbf{v}} + (W \text{diag}(\lambda_s) L)^T (\boldsymbol{\psi} - \mathbf{1}) \\ \bar{\mathbf{v}}_{new} &= \mathbf{a} - C^T (CC^T + I)^{-1} C \mathbf{a} \end{aligned}$$

which is repeated iteratively until convergence. The computational complexity of this step is $O((rT)^3)$. To generate a Monte Carlo sample from this distribution we draw $\mathbf{v}_a, \mathbf{v}_b \sim N(0, I)$, then generate $\mathbf{v}_c = \mathbf{v}_a + C^T \mathbf{v}_b$, and $\mathbf{v}_{sample} = \mathbf{v}_c - C^T (CC^T + I)^{-1} C \mathbf{v}_c + \bar{\mathbf{v}}$. Importance weights are calculated for each sample w_m for $m = 1, \dots, M$.

The updating step for β uses a Newton-Raphson step with estimated gradient:

$$\mathbf{g}_\beta = \sum_{m=1}^M w_m X^T \text{diag}(\lambda_s) W^T (\boldsymbol{\psi} - \mathbf{1})$$

and Hessian:

$$H_{\beta} = \sum_{m=1}^M w_m X^T \text{diag}(\lambda_s) W^T \text{diag}(\lambda_r^{-1}) W \text{diag}(\lambda_s) X \quad (12)$$

so that $\hat{\beta}_{new} = \hat{\beta} + H_{\beta}^{-1} g_{\beta}$. The updating step for the covariance parameters is the same as the standard GLMM algorithm.

Standard errors

We report the usual generalized least squares standard errors. For the region model, the fixed effect parameter standard errors are generated from the inverse of the negative of the matrix in (12).

4 Bayesian model fitting

There are several approaches to estimating the LGCP in a Bayesian framework. Letting $\Theta = [\beta, \theta, \sigma, Z]$ represent all the model parameters, the Bayesian approach aims to draw samples from the posterior

$$f_{\Theta|\mathcal{D}}(\Theta|\mathcal{D}) \propto f_{\mathcal{D}|\Theta}(\mathcal{D}|\Theta) f_{\Theta}(\Theta)$$

Markov Chain Monte Carlo (MCMC) methods are often used. In the context of the LGCP, there has been some discussion of the choice of the proposal density for the Metropolis-Hastings algorithm. Møller et al. (1998) propose a Langevin kernel, which is known as the Metropolis-adjusted Langevin algorithm (MALA). MALA is a special case of Hamiltonian Monte Carlo (HMC), which uses Hamiltonian dynamics to construct proposals for the Metropolis-Hastings algorithm (Carpenter et al., 2017). MALA was used by (Taylor et al., 2013) for fitting an LGCP. HMC is potentially more efficient than MALA though (Teng et al., 2017).

MCMC algorithms have the advantage of providing consistent estimation of various characteristics of the posterior distribution, including well calibrated probabilities. However, it is computationally intensive, and so may provide a further limit on the granularity of predictions in settings where time is a constraint. Several deterministic methods for approximate Bayesian inference have been proposed to try to overcome this limitation. One such method is Variational Bayes (VB) using an approximating density $r(\Theta; \phi)$ parameterised by ϕ . VB minimises the Kullback-Leibler (KL) divergence from the approximation density to the posterior:

$$\min_{\phi} \text{KL}(r(\Theta; \phi) || f_{\Theta|\mathcal{D}}(\Theta|\mathcal{D}))$$

For the LGCP, the KL divergence lacks an analytical form, so a proxy is used, the evidence lower bound:

$$\mathcal{L}(\phi) = E[\log f_{\Theta|\mathcal{D}}(\Theta, \mathcal{D})] - E[\log r(\Theta; \phi)]$$

where the expectations are with respect to the prior densities. The minimisation problem is then to find ϕ that maximises $\mathcal{L}(\phi)$ such that the approximation density is in the support of the posterior. One of the difficulties with VB is deriving an appropriate family of approximating densities and then determining the appropriate model-specific quantities for the VB problem. Teng et al. (2017) give a specific derivation for the full LGCP. However, an alternative approach is to use automatic differentiation methods that provide a general algorithm for probabilistic models, which enables use of the Gaussian process approximations discussed here.

```

Calculation of  $(I - A)^{-1}D^{1/2}$ 
Let  $\tilde{L}$  be an  $n \times n$  lower triangular matrix
for all  $i \in 1, \dots, n$  do
  for all  $k \in i, \dots, n$  do
    Let  $l = A_{\mathcal{N}_i, i}^T \tilde{L}_{\mathcal{N}_i, k}$ 
    if  $i = k$  then
       $\tilde{L}_{i, k} = (1 + l)\sqrt{D_{kk}}$ 
    else
       $\tilde{L}_{i, k} = l\sqrt{D_{kk}}$ 
    end if
  end for
end for

```

Figure 1: Calculation of $(I - A)^{-1}D^{1/2}$

5 Gaussian process approximations

Evaluating the likelihood is computationally expensive owing to the need to invert Σ . Computational complexity and memory requirements to evaluate the log likelihood in Equation (9) scale as $O(Tn^3)$ and $O(Tn^2)$, respectively. For the Bayesian model fitting methods, these models therefore quickly become infeasible with even moderately sized grids, which can limit their use. We offer an option to skip model fitting with “known” covariance parameter values, which can be seen as akin to the problem of choosing a bandwidth for kernel-based approaches. We provide a function to estimate the empirical semivariogram to support parameter choice. However, for Bayesian fitting procedures including the covariance parameters, there are two main classes of approximation to the likelihood of a Gaussian process we include.

Nearest neighbour gaussian process

The multivariate Gaussian likelihood can be rewritten as the product of the conditional densities:

$$f(z) = f(z_1) \prod_{j=2}^n f(z_j | z_1, \dots, z_{j-1})$$

where, for convenience, z here represents a spatial-only field. Vecchia (1988) proposed that one can approximate $f(z)$ by limiting the conditioning sets for each z_j to a maximum size of m . For geospatial applications, Datta et al. (2016a,b) proposed the nearest neighbour Gaussian process (NNGP), in which the conditioning sets are limited to the m ‘nearest neighbours’ of each observation.

Let $\mathcal{N}_j$ be the set of up to m nearest neighbours of j with index less than j . We use the notation $\Sigma_{\mathcal{N}, \mathcal{M}}$ to represent the submatrix of Σ with rows in $\mathcal{N}$ and columns in $\mathcal{M}$, where these sets could be of size one. The approximation is:

$$f(z) \approx f(z_1) \prod_{j=2}^n f(z_j | \mathcal{N}_j)$$

which leads to:

$$\begin{aligned} z_1 &= \eta_1 \\ z_j &= \sum_{i=1}^m a_{ji} z_{\mathcal{N}_{ji}} \end{aligned} \tag{13}$$

where $\mathcal{N}_{ji}$ is the i th nearest neighbour of j .

Equation (13) can be more compactly written as $z = Az + \eta$ where A is a sparse, strictly lower triangular matrix, $\eta \sim N(0, D)$ with D a diagonal matrix with entries $D_{11} = \text{Var}(z_1)$

and $D_{ii} = \text{Var}(w_i | \mathcal{N}_i)$ (Finley et al., 2019). The approximate covariance matrix can then be written as $\Sigma_0 \approx (I - A)^{-1} D (I - A)^{-T}$ such that the quadratic form is approximated as:

$$\sum_t \mathbf{z}_{[t]} (I - A)^T D^{-1} (I - A) \tilde{\mathbf{v}}_{[t]}$$

As $I - A$ is upper triangular, the quadratic form can be calculated cheaply using forward substitution, iterating only over the nearest neighbours for each row. The log determinant is:

$$\log(|\Sigma_0|) \approx \sum_{i=1}^m -\log(D_{ii})$$

Finley et al. (2019) provide algorithms for the calculation of the matrices A and D . The non-zero elements of A are given by the linear system $\Sigma_{0;\mathcal{N}_j,\mathcal{N}_j} A_{j,\mathcal{N}_j}^T = \Sigma_{0;\mathcal{N}_j,j}$, where the subscripts indicate the indices of the relevant submatrix, and the diagonal elements of matrix D are $D_{j,j} = \Sigma_{0;j,j} - A_{j,\mathcal{N}_j} \Sigma_{0;\mathcal{N}_j,j}$. The NNGP has computational complexity scaling with $O(Tnm^3)$ and so the log-likelihood is now linear in both time and grid size.

There are two main factors that can affect the accuracy of the NNGP approximation. The ordering of the grid cells in terms of their indices can affect performance of the approximation. Guinness (2018) compares the performance of several different ordering schemes. They suggest that a “minimax” scheme, in which the next observation in the order is the one which maximises the minimum distance to the previous observations, often performed best, although in several scenarios ordering in terms of the vertical coordinate, or at random, provided comparable performance. The number of neighbours did not appear to affect the relative performance of the orderings; they considered both 30 and 60, although Datta et al. (2016a) uses 15 nearest neighbours in their simulations.

Hilbert space gaussian process

Low, or reduced, rank approximations aim to approximate the matrix Σ with a matrix $\tilde{\Sigma}$ with rank $m < n$. The optimal low-rank approximation is $\tilde{\Sigma} = \Phi \Lambda \Phi^T$ where Λ is a diagonal matrix of the m leading eigenvalues of Σ and Φ the matrix of the corresponding eigenvectors. However, the computational complexity of generating the eigendecomposition scales the same as matrix inversion. Solin and Särkkä (2020) propose an efficient method to approximate the eigenvalues and eigenvectors using Hilbert space methods, so we refer to it as a Hilbert Space Gaussian Process (HSGP). Riutort-Mayol et al. (2023) provide further discussion of these methods.

Stationary covariance functions, including those in the Matern class like exponential and squared exponential, can be represented in terms of their spectral densities. For example, the spectral density function of the squared exponential function in d dimensions is:

$$Sp(\omega) = \sigma^2 (\sqrt{2\pi})^d \phi^d \exp(-\phi^2 \omega^2 / 2)$$

Consider first a unidimensional space ($d = 1$) with support on $[-c, c]$. The eigenvalues λ_j (which are the diagonal elements of Λ) and eigenvectors ϕ_j (which form the columns of Φ) of the Laplacian operator in this domain are:

$$\lambda_j = \left(\frac{j\pi}{2c} \right)^2$$

and

$$\phi_j(s) = \sqrt{\frac{1}{c}} \sin \left(\sqrt{\lambda_j} (x + c) \right)$$

Then the approximation in one dimension is

$$\mathcal{Z}(s) \approx \sum_{j=1}^m Sp\left(\sqrt{\lambda_j}\right)^{1/2} \phi_j(s) \beta_j$$

where $\beta_j \sim N(0, 1)$. This result can be generalised to multiple dimensions. The total number of eigenvalues and eigenfunctions in multiple dimensions is the combination of all univariate eigenvalues and eigenfunctions over all dimensions. For example, if there were two dimensions and $m = 2$ then there would be four multivariate eigenfunctions and eigenvalues equal to the four combinations over the two dimensions. The approximation is then as described above but with the multivariate equivalents.

The approximation can be used in the full model such that the intensity becomes

$$\lambda(S_i, t) = r(S_i, t) \exp\left(X(S_i, t)\gamma + (P^{\frac{1}{2}} \otimes \Phi \Lambda^{\frac{1}{2}})\beta\right) \quad (14)$$

and where $\beta \sim N(0, I_{m^2})$. The matrix Φ does not depend on the covariance parameters and can be pre-computed, so only the product $\Phi \Lambda^{\frac{1}{2}}$ needs to be re-calculated during model fitting, which scales as $O(Tnm^2)$.

The performance of both the HSGP and the NNGP depends on m , either the number of basis functions or the number of nearest neighbours, respectively, and several other parameters. The other key value for the HSGP is the boundary condition c . [Riutort-Mayol et al. \(2023\)](#) provide a detailed analysis of the HSGP for unidimensional linear Gaussian models, examining in particular how the choice of c and m affect performance of posterior inferences and model predictions. Here, we assume c is relative to the maximum and minimum coordinate in each dimension (i.e. the space is scaled to $[-1, 1]^2$). They find values of $m = 15$ and $c = 1.5$ to 2.5 provide a good approximation for the models they consider with squared exponential covariance function, but also show that for higher values of m we generally require larger values of c . For example, $m = 20$ and $c = 2$ provides similar performance to $m = 30$ and $c = 4$.

6 Related software and methods

Software packages for fitting an LGCP are relatively few. In the R ecosystem, we are aware of two dedicated packages. First, [lgcp](#) ([Taylor et al., 2013, 2015](#)) provides maximum likelihood and Bayesian MCMC estimation of the LGCP model. However, it uses only the full model and while there are some features to improve computational time, it suffers from long running times for all but the smallest applications. The temporal structure of the implemented model does not scale linearly with time either. Furthermore, it makes use of a no-longer supported package and standard for spatial data ([sp](#)) and does not make use of state-of-the-art Bayesian methods.

The R package [inlabru](#) ([Bachl et al., 2019](#)) provides a set of tools to approximate Bayesian posteriors for the LGCP using integrated nested Laplacian approximation (INLA), a popular and highly efficient Bayesian approximation. INLA (made available in R more generally through [R-INLA](#) ([Lindgren and Rue, 2015](#))) approximates the posterior means of model parameters, and in this case spatially-continuous Gaussian processes, using Gaussian Markov random fields (GMRFs). A comprehensive overview is provided in [Rue and Held \(2005\)](#) Chapter 5. We do not include INLA given the availability of [inlabru](#). The quality of INLA in terms of quantifying uncertainty beyond posterior means may also be less than needed for surveillance applications. [Taylor and Diggle \(2014\)](#) compare MCMC (specifically MALA) and INLA using the methods in [lgcp](#) and [inlabru](#) and conclude that while MCMC does take much longer, it offers much better predictive accuracy. However, the scale of the problems examined may be considered relatively small; for larger grids and multiple time periods, MCMC may be prohibitively slow. [Teng et al. \(2017\)](#) come to similar conclusions in a broader comparison of MCMC and other methods.

More recently, the package **stelfi** (Jones-Todd and van Helsdingen, 2024) has been released that provides maximum likelihood estimation and predictions from spatio-temporal LGCPs and Hawkes processes. The package provides novel functionality for temporal and spatiotemporal self-exciting point process models, which we do not currently include in **rts2**. Dovers et al. (2024) describes how generic generalised additive model fitting software, particularly **mgcv** in R, can be used to fit LGCPs as the software includes a basis function expansion to approximate the Gaussian random field. We include a low-rank approximation in this package.

Finally, we note two approaches in the literature to modelling spatially aggregated case data. (Johnson et al., 2019) describes an approach for fitting and predicting from an LGCP model with spatially aggregated data using an MCMC sampling scheme and assuming a piecewise constant latent field in each area. At the time of writing though, software to implement this approach was no longer available on CRAN. Li et al. (2012a) describe a Bayesian data augmentation approach to fitting these models, which can be implemented in general Bayesian software.

We are not aware of any packages specifically designed for LGCPs, or real-time surveillance more generally, for other software environments like Stata or SAS. However, one may also fit LGCPs using more general mixed model software. For example, provided one generate the appropriate grid data, the R package **glmmTMB** offers Laplace approximation model fitting for GLMMs including Poisson models with spatial covariance structure. One can extract the processed grid data generated by **rts2** for use in other packages. For full maximum likelihood model fitting one can use **glmmrBase**. For Bayesian MCMC the Stan probabilistic programming language can fit any of the models described here. The **rts2** package uses **glmmrBase** and **Stan** for model fitting. In Stata and SAS, one could theoretically also use the GLMM fitting functionality to fit these models; however, as with the other methods, they would be impractically slow for even moderately sized grids. We are not aware of any specific implementation of LGCP models in other software environments. We are also not aware of any other implementation of what we have termed the ‘region model’ or of NNGP or HSGP with LGCPs.

7 **rts2** package

We now describe the functioning of the package in R. The accompanying R and data files provide reproducible code for all these examples, including pre-estimated models. We provide some comparisons between the different methods. MCMC sampling uses Stan through **rstan** (Carpenter et al., 2017).

The main functionality of **rts2** is provided by the `grid` class, which uses the **R6** package’s object-orientated class system. An object-orientated method is preferred here, as much of the functionality of the package involves repeatedly manipulating the same data object. As we demonstrate below, use of an object-orientated class system simplifies the syntax relative to the more standard R functional methods in these cases. Geographic data uses the **sf** package. Initialisation of a new `grid` object can use either a single polygon describing the boundary of an area of interest, or a set of polygons in the case of the region data model described above.

7.1 Point data

We use simulated data for this first example, which are provided in the package, as real-world location-identifying data are typically not anonymised. These data, along with the relevant spatial data objects, are provided in the replication materials. The boundary is also provided and represents the city of Birmingham, UK along with a set of covariate data at the middle layer super output area (MSOA) level, which is the second smallest census tract area.

To instantiate a new `grid` object we require the boundary and the cell size of the grid:

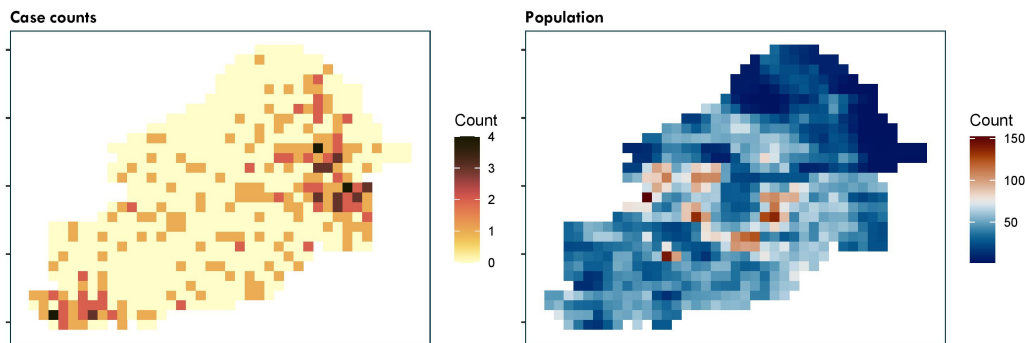


Figure 2: Example simulated data for the city of Birmingham, UK. Left panel: case counts of point data mapped to the computational grid. Right panel: population density mapped to the computational grid.

```
g1 <- grid$new(boundary, cellsize=0.008)
```

The next step is to map the point data to the grid to create the counts. We provide the convenience function `create_points()` to generate `sf` point data from an ordinary R data frame with location and possibly time data. Then, the member function `points_to_grid()` will aggregate the case counts, for example:

```
g1$points_to_grid(point_data = create_points(y,
                                             pos_vars = c("Y", "X"),
                                             t_var = c("t")),
                 t_win = "month",
                 laglength = 1)
```

or without a temporal component:

```
g1$points_to_grid(point_data = create_points(y, pos_vars = c("Y", "X")))
```

For spatio-temporal analysis, we could determine here what size time step we want (day, week, or month) and how many previous periods to include (`laglength`). The choice of lag length is similar to the choice of cell size; we want it to be long enough to provide the best predictions of current incidence, but as short as possible otherwise to minimise the required computational resources. The function adds columns to the grid object (labelled `t*` for spatio-temporal or `y` for spatial-only). Figure 2 top-left panel shows the aggregated case counts, which can be plotted using `g1$plot("y")`.

7.2 Overlaying covariate data

We can include covariates in our model. The aim of including covariates in surveillance applications is not typically to provide inference about the effect of any particular covariate but to improve predictions or to generate conditional, e.g. age-adjusted, predictions of relative risk. For example, by including day of the week as a covariate we can capture differences in hospital attendances and admissions resulting from individual behaviour not related to variations in disease incidence. At a minimum we would suggest including population density to enable population-standardised predictions. Covariates can be added to the grid data using the member function `add_covariates()`.

The covariate data are provided to the function in a form that can be manipulated into polygons, either `sf` or `raster` data. There are several approaches to mapping data onto the grid provided in the package. Primarily, we offer a straightforward simple ‘flat’ mapping of the covariate data. Let W_c be an $n_c \times n$ weight matrix, where n_c is the number of polygons in the covariate data, and the entries of W_c are equal to the areas of the intersections of each

grid cell and covariate polygon. The mapping is $t(W_c)\mathbf{x}_c/\mathbf{w}_c$ where $\mathbf{x}_c$ is the covariate data for each polygon, and $\mathbf{w}_c$ are the column sums of W_c .

Spatially-varying only covariates

Where the time step is short, such as days, it is unlikely there will be data that varies by time step over the area of interest. For example, population density data will be static over the area of interest. To add spatially-varying covariates to the grid data we require an `sf` object with polygons encoding the covariate data. The `add_covariates()` function then, for each grid cell, takes an average of the values in the overlapping polygons, weighted by either the size of the area of overlap or the population, if population density data are available. A good source of population density predictions is WorldPop (at www.worldpop.org), which will provide rasters at 100m or 1km resolution for all countries.

First, we add the population density:

```
gl$add_covariates(msoa,
                  zcols="pop",
                  weight_type="area")
```

where `msoa` is the MSOA data as an `sf` object. Figure 2 top right panel shows the population density.

Temporally-varying only covariates

The example includes only a single period, but in other cases with multiple time periods we may have both temporally and spatio-temporally varying covariates. Some covariates, like day of the week, vary over time but are the same for all grid cells in each time period. To add a temporally-varying covariate we need to produce a data frame with a column for time period (from 1 to `laglength`) and then columns with the values of the covariates. To obtain day of the week we can use the member function `get_dow()`, which will return a data frame mapping the date of each time period to a day of the week. For example,

```
df_dow <- grid_data$get_dow()
print(df_dow)
```

will return the data frame and then these columns can then be added to the main grid data:

```
grid_data$add_covariates(df_dow,
                        zcols=c("dayMon", "dayTue",
                               "dayWed", "dayThu", "dayFri",
                               "daySat", "daySun"))
```

The member function `add_time_indicators()` will generate fixed effect indicator variables for each time period in multi-period data. These will be labelled `time1i`, `time2i`, and so forth.

Spatially- and temporally-varying covariates

In some cases there may be covariates that vary over time and space. There are two ways of adding these into the main grid data. Both ways need to generate, for each covariate, as many columns as there are time periods. The column names will be the covariate name followed by the time period index.

If the covariate data are included in different data frames for each time period, then we can repeat the process above for each time period and add a label using the `t_label` argument:

```
grid_data$add_covariates(msoa_t1,
                        zcols=c("covA", "covB", ...),
                        t_label = 1)
```

Alternatively, if the value of the covariates are all included in different columns of the same data frame then they can be added at the same time, ensuring we keep to the same naming convention of covariate name and time index as a string:

```
grid_data$add_covariates(msoa,
  zcols=c("covA1", "covA2", "covA3", ...))
```

7.3 Bayesian priors

For the Bayesian models, one must consider the prior distributions of the parameters. Currently, these distributions are proscriptive and one must only select the location and scale. For the lengthscale ϕ and variance (σ^2) parameters, we specify half-normal distributions (i.e. a normal distribution truncated below zero). And for the parameters of the linear prediction γ , we specify normal distributions. The choice of mean and standard deviation values for these prior distributions should be provided to the `grid` object as a list:

```
grid_data$priors <- list(
  prior_lscale=c(0,0.5),
  prior_var=c(0,0.5),
  prior_linpred_mean=c(-5,rep(0,7)),
  prior_linpred_sd=c(3,rep(1,7))
)
```

7.4 Model fitting

The two model fitting functions are `lgcp_bayes()` and `lgcp_ml()` for Bayesian and maximum likelihood methods, respectively. The arguments to the two functions are highly similar. The functions described above add the case counts and covariates to the grid data. The model fitting functions require specification of which covariates to include, the model to use, and any approximation and fitting parameters. For example, to use the exponential covariance function with the NNGP approximation with ten nearest neighbours, we can use the following code. We first re-order the computational grid using the ‘minimax’ ordering.

```
g1$lgcp_ml(popdens = "pop")
```

The model fit can be returned from this call, but it is also stored internally and can be returned using `g1$model_fit()`, which will return an S3 object of class `rtsFit`.

Figure 3 shows the predicted relative risk from three different model fitting methods. We discuss extraction and plotting of model fits in the next section. Code to generate these model fits is provided in the replication materials. The predictions are qualitatively similar, although there are notable differences in the smoothness and magnitude of the relative risk surface, which result from different choices about approximation parameters and other aspects of the analysis.

7.5 Region data

To illustrate the use of the regional data model we use publicly available crime data from the UK. Monthly counts of different types of crime are available at the middle-layer super output area level from crime.uk. We extract the monthly counts for 2022 for the city of Birmingham, UK for burglary. These data are available in the package.

If a new `grid` class object is instantiated with data comprised of multiple polygons, it assumes a regional data model. The provided polygons are assumed to represent all the areas of interest. Generally, these should be a division of a contiguous area, although this is not a strict requirement. Since the context for a regional data model is that the data are only available in their aggregated format, the data used to initialise the new `grid` object should already contain these counts. Column names should be in the same format as generated for point data described above: if there is only a single time period then the counts should be in a column named `y`, otherwise the columns should be labelled `t1`, `t2`, `t3`, ... and so

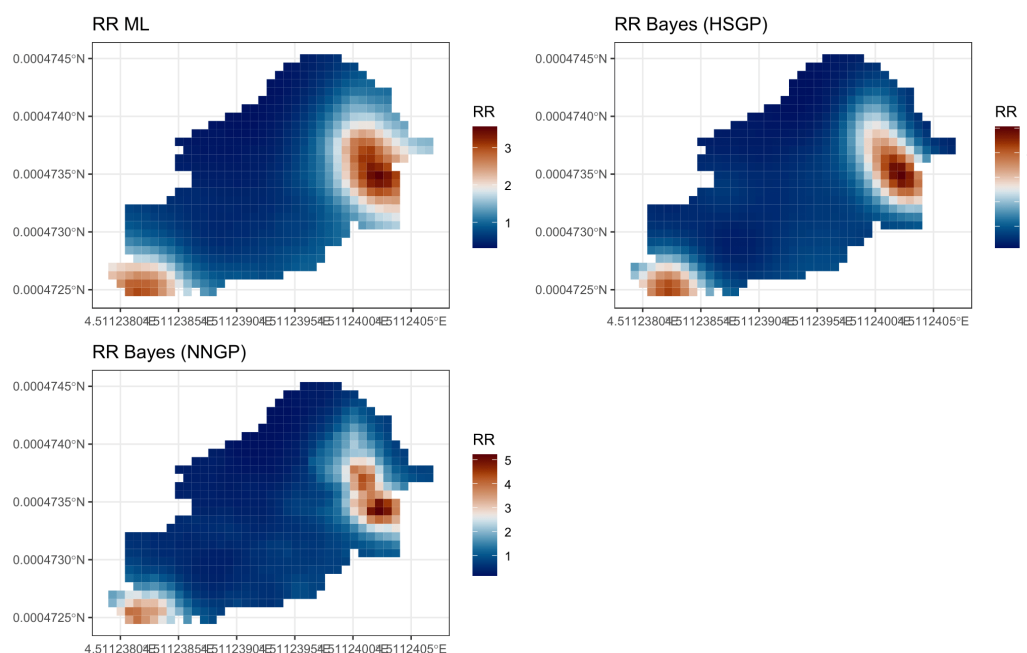


Figure 3: Example model fits using three methods: maximum likelihood (ML) and MCMC along with HSGP and NNGP approximations. The HSGP model fits used 15 basis function per dimension and the NNGP used 25 nearest neighbours.

forth in chronological order. As with the point data we must also provide a cell size for the computational grid. Covariates must also be included in the regional data object with the same naming conventions for multi-period models. One can add grid level covariates from another source using the `add_covariates()` function as described above.

```
g2 <- grid$new(msoa, cellsize = 0.008) # create new region grid
g2$add_time_indicators()
```

We do not include any additional covariates in this example beyond the time indicators. Model fitting uses the same commands as for the basic spatial data model described above. We fit the model with maximum likelihood and both NNGP and HSGP approximations:

```
g2$lgcp_ml("pop", # population density
           covs = paste0("time", 2:12, "i") # time period indicators)
```

7.6 Analysing the output

Figure 4 shows several outputs from the models fitted in the previous section. Here, we describe how to extract these outputs.

Predictions

The output from the model enables us to generate four main predictions across our area of interest:

1. Predicted total incidence.
2. Predicted incidence per head of population.
3. The relative risk, i.e. the relative difference between predicted and expected incidence. For example, a relative risk of 2 would imply that the predicted incidence is twice as high as would be expected given the area mean and local covariate values.

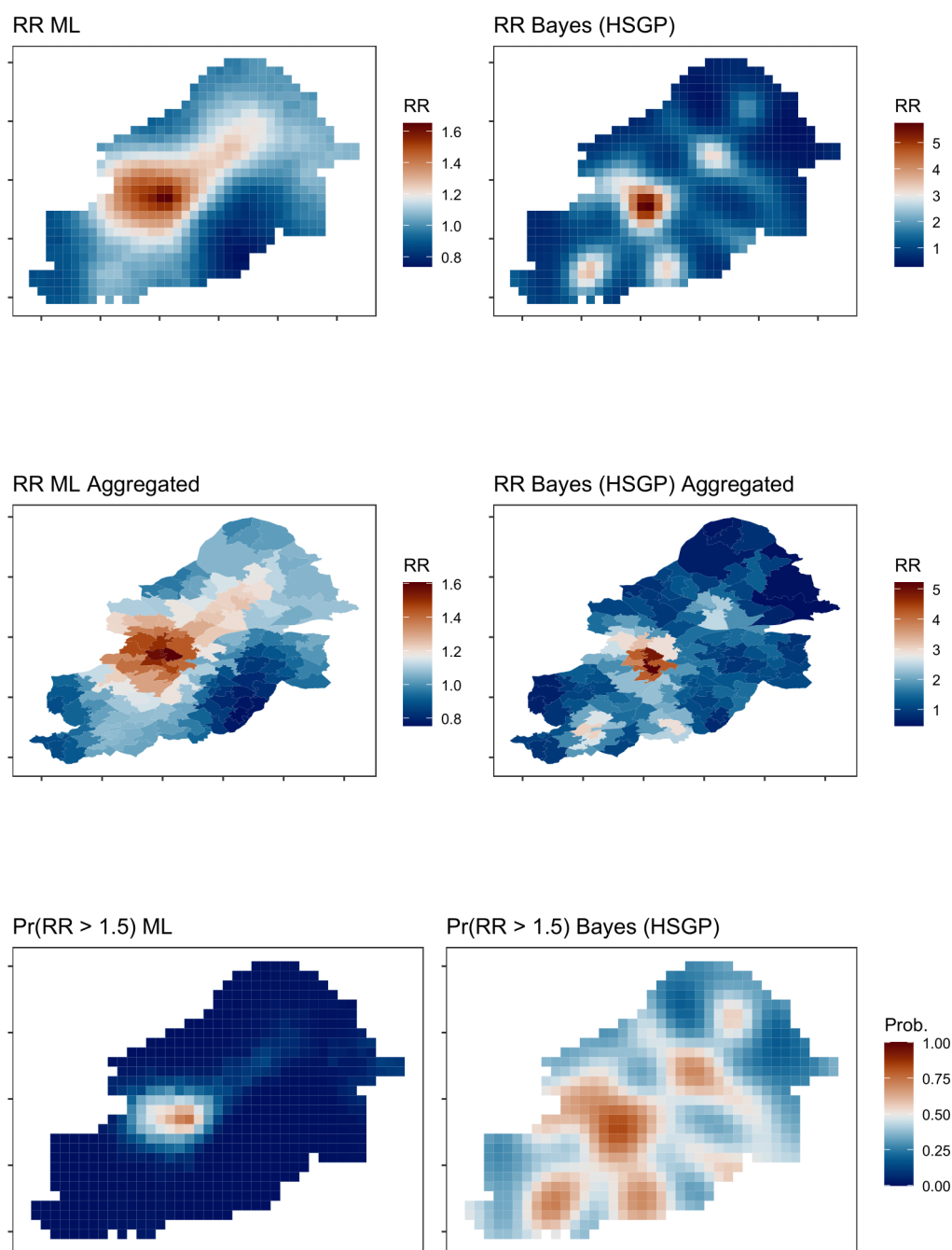


Figure 4: Model fitting outputs for the Birmingham crime example. The left column provides outputs from a maximum likelihood model fit; the right column is an HSGP Bayesian fit with 10 basis functions per dimension. The top row is the predicted relative risk, the middle row the aggregated relative risk, and the bottom row the probability that the relative risk is greater than 1.5.

4. The incidence rate ratio relative to some previous time point for spatio-temporal analyses. For example, we may be interested in comparing incidence to seven days prior; a value of 1.5 would imply that the incidence has risen 50% in a week.

We can extract any or all of these predictions using the function `extract_preds()`, which will extract the predictions from the last model fit. For this example, we will extract from the crime data analysis the relative risk and incidence rate ratio. For region data models incidence predictions and incidence rate ratio are added to the region data, with the relative risk added to the grid data.

```
g2$extract_preds(type = c("rr", "irr"),
                 irr.lag = 11,
                 popdens = "pop")
```

Hotspots

The concept of a “hotspot” is often poorly or vaguely defined and it can mean different things to different people (Lessler et al., 2017). Generally, we can define it as an area where there is a high probability that some epidemiological measure(s) exceed predefined threshold(s). Here, we can use any of the outputs of the model listed above (incidence, relative risk, or incidence rate ratio). The `hotspots()` member function allows us to define a hotspot based on any of these criteria. For example, we can calculate the probability that the relative risk exceeds 1.5 across the grid:

```
g2$hotspots(threshold = 1.5, stat = "rr", popdens = "pop")
```

will add a column to the grid data labelled `rr` with the probabilities that the cell has a relative risk greater than 1.5.

7.7 Aggregating to other geographies

It may be desirable to summarise the results of the analysis on a more familiar geography or to aggregate grid data to the region data. For example, we may wish to aggregate the results onto administrative areas that might be used to direct public health responses. In this example, we aggregate the relative risk results back onto the MSOA geography for the city. As with the covariate averaging we can either average with respect to the area or the population, here we’ll average with respect to the area.

```
g2$region_data <- g2$aggregate_output(g2$region_data, zcols=c("rr"))
```

The second row of Figure 4 shows an example.

7.8 Visualisation

As the output of the analysis is an `sf` object (stored in the slot `g2$grid_data`), there are many different tools we can use to plot and visualise the results. We can use the default plot function in the `grid` class, which calls `sf`’s plot function for a variable in the grid data. For example, for the relative risk:

```
grid_data$plot("rr")
```

Other plotting tools include **ggplot2** with the `geom_sf` function; **tmap** provides a range of tools for map plotting including allowing multiple panels to be plotted. The **mapview** package provides functionality to create interactive overlays of `sf` data on OpenStreetMap maps.

7.9 Model parameters

We may also be interested in the posterior samples of particular model parameters, which are contained in the output of the model fit. We note two important considerations for interpreting these parameters:

- For HSGP model fitting, the coordinates are scaled to $[-1, 1]$, so the length scale parameters will be equivalently scaled. Use the `scale_conversion_factor()` function to extract these factors.
- Spatially-varying parameters of the linear predictor (γ) will differ from a non-spatial model. The parameters on spatially varying covariates will give an indication of the direction of an effect, but their magnitude may have different interpretations (Hodges and Reich, 2010; Reich et al., 2006).

8 Examples comparing packages

There are few software alternatives for some of the functionality we have described in this article. Here, we provide examples to compare outputs and code between packages.

8.1 Spatial point data

We firstly illustrate the use of the package alongside **inlabru** (Bachl et al., 2019), which provides approximate Bayesian model fitting using INLA. We will use the ‘gorillas’ dataset from that package, which describes the nesting sites of gorillas in an area.

inlabru model fitting

The example data already includes the meshing and preparation of covariates required for model fitting. To load the data we specify

```
require(inlabru)
data <- gorillas_sf
data$gcov <- gorillas_sf_gcov()
elev <- data$gcov$elevation
data$gcov$elev <- (elev - mean(terra::values(elev), na.rm = TRUE))/sd(terra::
  values(elev), na.rm = TRUE)
```

The elevation data is mean standardised to improve the stability of model fitting. We set the SPDE prior as following the example provided in **inlabru**. We include an intercept and standardised elevation as a covariate in the model, in addition to the spatial field. To specify the formula, fit the model, and generate predictions we use:

```
ecom <- geometry ~ elev(f.elev(.data.), model = "linear") +
  mySmooth(geometry, model = matern) + Intercept(1)

efit <- lgcp(ecomp, data$neats, samplers = data$boundary, domain = list(geometry =
  data$mesh))
pred.df <- fm_pixels(data$mesh, mask = data$boundary)
e.pred <- predict(
  efit,
  pred.df,
  ~ list(
    int = exp(mySmooth + elev + Intercept),
    int.log = mySmooth + elev + Intercept,
    rr = exp(mySmooth)
  )
)
```

rts2 model fitting

To fit an equivalent LGCP model, using the HSGP approximation with Bayesian MCMC model fitting, in **rts2**, we follow the steps described above:

```
mod <- grid$new(
  data$boundary,
  cellsize = 0.025
)
```



```
mod$points_to_grid(data$neests)
mod$add_covariates(cov_data = data$gcov$elev, zcols = "elev")
fit3 <- mod$lgcp_ml(covs = c("elev"))
mod$extract_preds(c("pred", "rr"))
```

The code will generate a grid with 3,663 grid cells, and we have specified to use 15 basis functions per dimension.

Comparison of *inlabru* and *rts2* results

Figure 5 shows the predicted intensity surface and relative risk, and their log standard deviations, from *inlabru* and *rts2*. Qualitatively the intensity and relative risk surfaces are similar with *inlabru* showing a higher degree of smoothing. The intensity reported by *rts2* is per grid cell, whereas for *inlabru* the intensity is over the whole area. The range of the relative risk was larger with *inlabru*.

8.2 Spatially-aggregated case data

For spatially aggregated case data, a common model is a conditionally autoregressive model in which the counts in each area are independent of one another conditional on the counts in neighbouring areas. The Besag-York-Mollie (BYM) model is the key example of this approach. In R, the INLA package (van Niekerk et al., 2021) is often used for implementation of this model with spatially aggregated case data. Here, we provide a comparison of generating predictions from spatio-temporal count data using the LGCP model described above in *rts2*, with the version of the BYM model described by Simpson et al. (2017) implemented in INLA. We use the common exemplar dataset describing sudden infant death syndrome cases in North Carolina in the 1970s described in Cressie and Chan (1989) and elsewhere. These data are available in the *spData* package on CRAN and can be loaded using:

```
nc.sids <- sf::st_read(system.file("shapes/sids.gpkg", package="spData"))[1])
```

Complete code for this example is provided in the reproduction materials.

Code for INLA

We adapt code for these data described in Palmí-Perales et al. (2021). There are several approaches to specifying the statistical model, for example, one could include additional iid noise beyond the method we specify here. This example is therefore intended to be illustrative. The processing of the data is achieved using the following code:

```
# First time period
# Compute expected cases
r74 <- sum(nc.sids$SID74) / sum(nc.sids$BIR74)
nc.sids$EXP74 <- r74 * nc.sids$BIR74

# Second time period
# Compute expected cases
r79 <- sum(nc.sids$SID79) / sum(nc.sids$BIR79)
nc.sids$EXP79 <- r79 * nc.sids$BIR79

d <- data.frame(OBS = c(nc.sids$SID74, nc.sids$SID79),
                PERIOD = c(rep("74", 100), rep("79", 100)),
                EXP = c(nc.sids$EXP74, nc.sids$EXP79))

# County-period index
d$Sidarea <- rep(1:100, 2)

# prepare the graph
nb <- spdep::poly2nb(nc.sids)
spdep::nb2INLA("map.adj", nb)
```

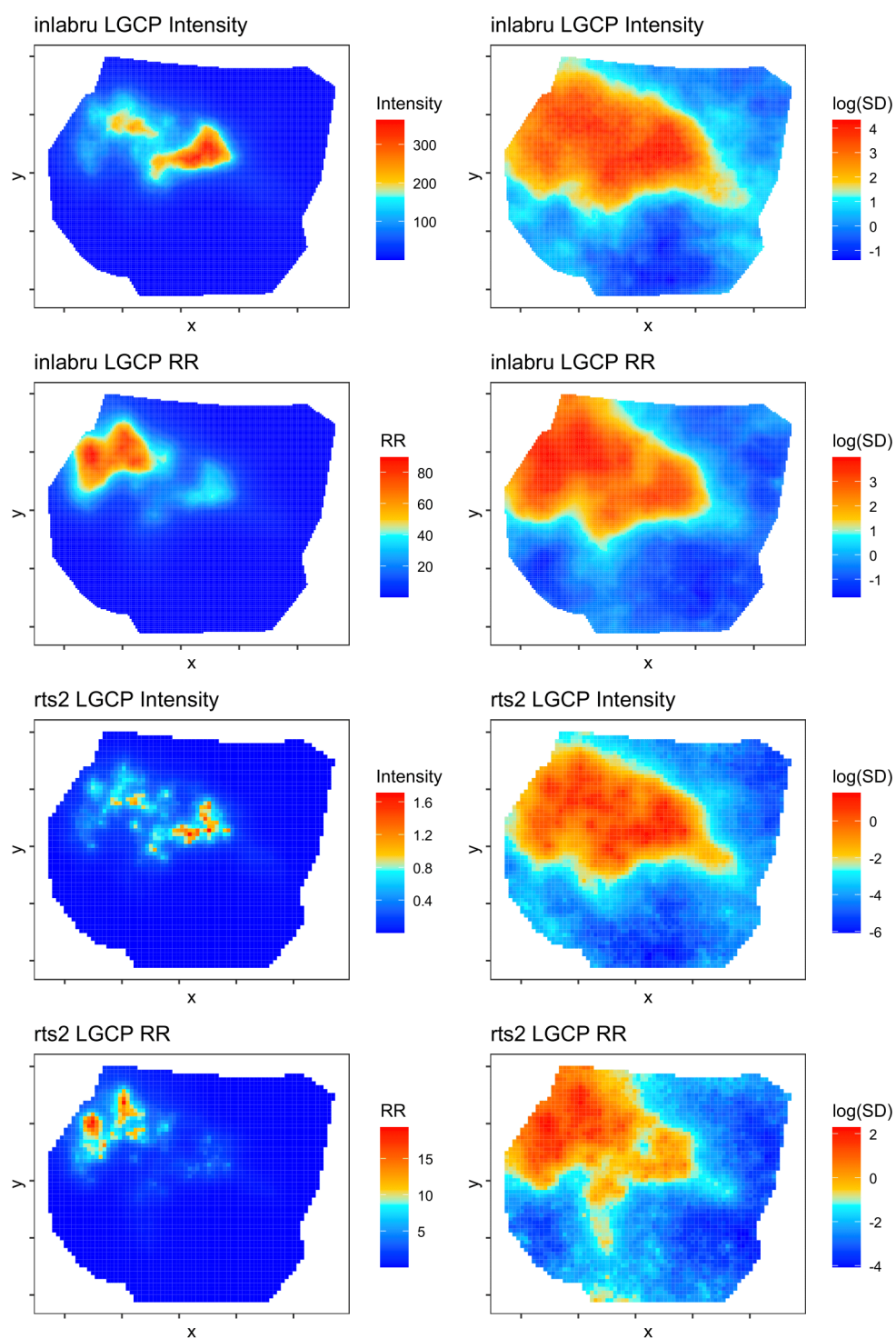


Figure 5: Predicted intensity, relative risk, and log standard deviations using the gorilla nesting sites data in *inlabru* package comparing *rts2* maximum likelihood and the SPDE approach in *inlabru*

```
require(INLA)

g <- inla.read.graph(filename = "map.adj")
```

The command in INLA to fit the model with an intercept and time period indicator, along with the BYM correlation structure is:

```
res <- inla(OBS ~ f(idarea, model = "bym2", graph = g) + PERIOD, family = "poisson", data = d, E = EXP, control.predictor = list(compute = TRUE))

nc.sids$rr_inla <- res$summary.random$idarea[, "mean"][101:200]
nc.sids$rr_lci_inla <- res$summary.random$idarea[, "0.025quant"][101:200]
nc.sids$rr_uci_inla <- res$summary.random$idarea[, "0.975quant"][101:200]
nc.sids$prob_inla <- 1 - pnorm(log(1.5), nc.sids$rr_inla, res$summary.random$idarea[, "sd"][101:200])
nc.sids$pred_inla <- res$summary.fitted.values[, "mean"][101:200]
```

where we have used default non-informative priors.

Code for rts2

For the rts2 package, we need to generate a new grid object using the data `nc.sids` described above, and rename the variables relative to time period.

```
require(rts2)

mod <- grid$new(nc.sids, cellsize = 0.25)
# rename the columns
colnames(mod$region_data)[c(10,13)] <- c("t1", "t2")
colnames(mod$region_data)[c(24,26)] <- c("exp1", "exp2")
# add a time indicator
mod$add_time_indicators()
```

Then the model fitting and extraction of predictions are achieved using the following code. In this example, we use a maximum likelihood approach to model fitting. We again use the HSGP with the default of ten basis functions per dimension.

```
# fit the model
res3 <- mod$lgcp_ml("exp", model = "fexp",
                   covs = c("time2i"),
                   iter_sampling = 250)

mod$extract_preds(c("pred", "rr"))
mod$hotspots(threshold = 1.5, stat = "rr")
# generate a fitted value net of offset
mod$grid_data$pred_fitted <- mod$grid_data$pred_mean_pp/mod$grid_data$exp2
# generate confidence intervals for prediction using the standard errors
mod$region_data$pred_lci <- mod$region_data$pred_mean_total -
  qnorm(0.975)*mod$region_data$pred_mean_total_se
mod$region_data$pred_lci <- ifelse(mod$region_data$pred_lci >= 0,
  mod$region_data$pred_lci, 0)
mod$region_data$pred_uci <- mod$region_data$pred_mean_total +
  qnorm(0.975)*mod$region_data$pred_mean_total_se

# attach to the original data file
nc.sids$pred <- mod$region_data$pred_mean_total
nc.sids$pred_lci <- mod$region_data$pred_lci
nc.sids$pred_uci <- mod$region_data$pred_uci
```

Comparison of outputs

Figure 6 shows the predicted relative risk and standardised mortality ratios at the county level using **INLA** and **rts2**. An evident difference is the degree of smoothing is lower in **rts2**. The magnitude of the relative risk is comparable between the two approaches, and the broad spatial patterns are comparable. However, the specific counties identified as high probability of an increased relative risk differ.

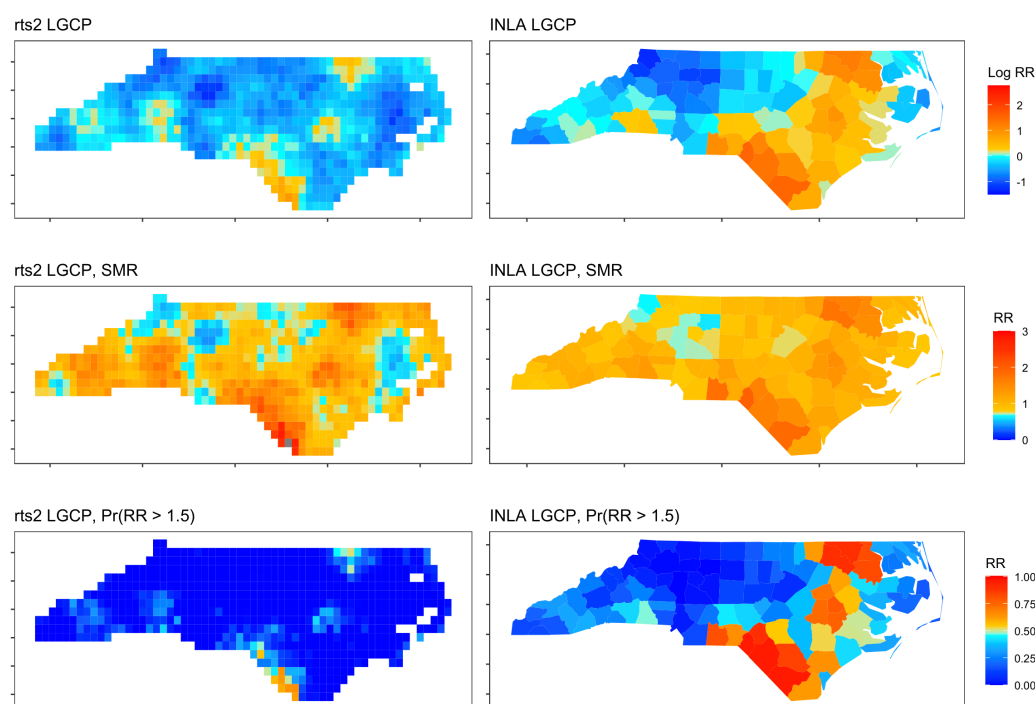


Figure 6: Predictions of the relative risk, standardised mortality ratio, and probability $RR > 1.5$ of SIDS in counties in North Carolina in 1979 using rts2 and INLA

9 Conclusions

The **rts2** package provides multiple different methods and approximations for fitting the LGCP. The motivation for this package was to support real-time disease surveillance efforts by providing predictions in a fraction of the time of a full LGCP model with no approximation, while maintaining the reliability and calibration of the quantification of uncertainty. In a set of comparisons, models that took days to run can now be run in minutes or hours, using either maximum likelihood or Bayesian methods with approximation. [Watson et al. \(2026\)](#) illustrate that significant reductions in computational time can also be achieved using modern GPU hardware, which is a future objective with these models. We also included Bayesian methods with approximations that can be ‘tuned’: at the extreme the approximations are equal to the full model. However, further research is required to establish the quality of the predictions and calibration of uncertainty, especially when compared with other ‘fast’ alternatives like INLA and kernel-based methods. We have also provided functionality for data manipulation and extraction from model fits to support adoption and use of these methods.

The package provides a variety of model fitting methods and approaches, including both Bayesian and maximum likelihood methods, and different Gaussian Process approximations. Given the variety of methods, and the existence of other alternatives including INLA, a full simulation-based comparison is warranted to determine the quality of predictions and parameter estimation. In terms of computational time, the HSGP method is the fastest and scales well with increasing number of basis functions. We have also found qualitatively that the method is generally stable and produces results comparable with similar methods, as demonstrated by our examples.

While our aim with this package is to support ‘real-time disease surveillance’ the methods are applicable to multiple areas where case data are used to identify ‘hotspots’ to support public and policy responses. We used an example with crime data. Other areas include geology, ecology, social policy, and many more.

References

- F. E. Bachl, F. Lindgren, D. L. Borchers, and J. B. Illian. inlabru: an R package for Bayesian spatial modelling from ecological survey data. *Methods in Ecology and Evolution*, 10: 760–766, 6 2019. ISSN 2041-210X. doi: 10.1111/2041-210X.13168. [p152, 160]
- B. S. Caffo, W. Jank, and G. L. Jones. Ascent-based Monte Carlo expectation–maximization. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67:235–251, 4 2005. ISSN 1369-7412. doi: 10.1111/j.1467-9868.2005.00499.x. [p148]
- B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. Stan : A probabilistic programming language. *Journal of Statistical Software*, 76, 2017. ISSN 1548-7660. doi: 10.18637/jss.v076.i01. [p149, 153]
- N. Cressie and N. H. Chan. Spatial modeling of regional variables. *Journal of the American Statistical Association*, 84:393–401, 6 1989. ISSN 0162-1459. doi: 10.1080/01621459.1989.10478783. [p161]
- A. Datta, S. Banerjee, A. O. Finley, and A. E. Gelfand. Hierarchical nearest-neighbor Gaussian process models for large geostatistical datasets. *Journal of the American Statistical Association*, 111:800–812, 4 2016a. ISSN 0162-1459. doi: 10.1080/01621459.2015.1044091. [p150, 151]
- A. Datta, S. Banerjee, A. O. Finley, and A. E. Gelfand. On nearest-neighbor Gaussian process models for massive spatial data. *WIREs Computational Statistics*, 8:162–171, 9 2016b. ISSN 1939-5108. doi: 10.1002/wics.1383. [p150]
- P. Diggle, B. Rowlingson, and T.-I. Su. Point process methodology for on-line spatio-temporal disease surveillance. *Environmetrics*, 16(5):423–434, aug 2005. ISSN 1180-4009. doi: 10.1002/env.712. URL <http://doi.wiley.com/10.1002/env.712>. [p144]
- P. J. Diggle and E. Giorgi. Model-Based Geostatistics for Prevalence Mapping in Low-Resource Settings. *Journal of the American Statistical Association*, 111(515):1096–1120, jul 2016. ISSN 0162-1459. doi: 10.1080/01621459.2015.1123158. URL <https://www.tandfonline.com/doi/full/10.1080/01621459.2015.1123158>. [p144]
- P. J. Diggle, P. Moraga, B. Rowlingson, and B. M. Taylor. Spatial and Spatio-Temporal Log-Gaussian Cox Processes: Extending the Geostatistical Paradigm. *Statistical Science*, 28(4):542–563, nov 2013. ISSN 0883-4237. doi: 10.1214/13-STS441. URL <http://projecteuclid.org/euclid.ss/1386078878>. [p145, 146]
- E. Dovers, J. Stoklosa, and D. I. Warton. Fitting log-gaussian Cox processes using generalized additive model software. *The American Statistician*, 78:418–425, 10 2024. ISSN 0003-1305. doi: 10.1080/00031305.2024.2316725. [p153]
- 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*, 28:401–414, 4 2019. ISSN 1061-8600. doi: 10.1080/10618600.2018.1537924. [p151]
- J. Guinness. Permutation and grouping methods for sharpening Gaussian process approximations. *Technometrics*, 60:415–429, 10 2018. ISSN 0040-1706. doi: 10.1080/00401706.2018.1437476. [p151]
- J. S. Hodges and B. J. Reich. Adding Spatially-Correlated Errors Can Mess Up the Fixed Effect You Love. *The American Statistician*, 64(4):325–334, nov 2010. ISSN 0003-1305. doi: 10.1198/tast.2010.10052. URL <http://www.tandfonline.com/doi/abs/10.1198/tast.2010.10052>. [p160]
- O. Johnson, P. Diggle, and E. Giorgi. A spatially discrete approximation to log-gaussian Cox processes for modelling aggregated disease count data. *Statistics in Medicine*, 38:4871–4887, 10 2019. ISSN 0277-6715. doi: 10.1002/sim.8339. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.8339>. [p153]

- C. M. Jones-Todd and A. B. M. van Helsdingen. stelfi: An R package for fitting hawkes and log-gaussian Cox point process models. *Ecology and Evolution*, 14, 2 2024. ISSN 2045-7758. doi: 10.1002/ece3.11005. [p153]
- J. Lessler, A. S. Azman, H. S. McKay, and S. M. Moore. What is a Hotspot Anyway? *The American Journal of Tropical Medicine and Hygiene*, 96(6):1270–1273, jun 2017. ISSN 0002-9637. doi: 10.4269/ajtmh.16-0427. URL <http://www.ajtmh.org/content/journals/10.4269/ajtmh.16-0427>. [p159]
- Y. Li, P. Brown, D. C. Gesink, and H. Rue. Log Gaussian Cox processes and spatially aggregated disease incidence data. *Statistical Methods in Medical Research*, 21:479–507, 10 2012a. ISSN 0962-2802. doi: 10.1177/0962280212446326. [p147, 153]
- Y. Li, P. Brown, H. Rue, M. al Maini, and P. Fortin. Spatial modelling of lupus incidence over 40 years with changes in census areas. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 61:99–115, 1 2012b. ISSN 0035-9254. doi: 10.1111/j.1467-9876.2011.01004.x. [p147]
- F. Lindgren and H. Rue. Bayesian spatial modelling with R - INLA. *Journal of Statistical Software*, 63, 2015. ISSN 1548-7660. doi: 10.18637/jss.v063.i19. [p152]
- C. E. McCulloch. Maximum likelihood algorithms for generalized linear mixed models. *Journal of the American Statistical Association*, 92:162, 3 1997. ISSN 01621459. doi: 10.2307/2291460. [p147]
- J. Møller, A. R. Syversveen, and R. P. Waagepetersen. Log Gaussian Cox processes. *Scandinavian Journal of Statistics*, 25:451–482, 9 1998. ISSN 0303-6898. doi: 10.1111/1467-9469.00115. [p149]
- F. Palmí-Perales, V. Gómez-Rubio, and M. A. Martínez-Beneito. Bayesian multivariate spatial models for lattice data with INLA. *Journal of Statistical Software*, 98, 2021. ISSN 1548-7660. doi: 10.18637/jss.v098.i02. [p161]
- R. Ratnayake, F. Finger, A. S. Azman, D. Lantagne, S. Funk, W. J. Edmunds, and F. Checchi. Highly targeted spatiotemporal interventions against cholera epidemics, 2000–19: a scoping review. *The Lancet Infectious Diseases*, oct 2020a. ISSN 14733099. doi: 10.1016/S1473-3099(20)30479-5. URL <https://linkinghub.elsevier.com/retrieve/pii/S1473309920304795>. [p144]
- R. Ratnayake, F. Finger, W. J. Edmunds, and F. Checchi. Early detection of cholera epidemics to support control in fragile states: estimation of delays and potential epidemic sizes. *BMC Medicine*, 18(1):397, dec 2020b. ISSN 1741-7015. doi: 10.1186/s12916-020-01865-7. URL <https://bmcmmedicine.biomedcentral.com/articles/10.1186/s12916-020-01865-7>. [p144]
- B. J. Reich, J. S. Hodges, and V. Zadnik. Effects of Residual Smoothing on the Posterior of the Fixed Effects in Disease-Mapping Models. *Biometrics*, 62(4):1197–1206, dec 2006. ISSN 0006341X. doi: 10.1111/j.1541-0420.2006.00617.x. URL <http://doi.wiley.com/10.1111/j.1541-0420.2006.00617.x>. [p160]
- S. Riley, K. Ainslie, O. Eales, C. Walters, H. Wang, C. Atchinson, C. Fronterre, P. J. Diggle, D. Ashby, C. Donnelly, G. Cooke, W. Barclay, H. Ward, A. Darzi, and P. Elliott. REACT-1 round 6 updated report: high prevalence of SARS-CoV-2 swab positivity with reduced rate of growth in England at the start of November 2020. 2020. URL <https://spiral.imperial.ac.uk/handle/10044/1/83912>. [p144]
- G. Riutort-Mayol, P.-C. Bürkner, M. R. Andersen, A. Solin, and A. Vehtari. Practical hilbert space approximate Bayesian Gaussian processes for probabilistic programming. *Statistics and Computing*, 33:17, 2 2023. ISSN 0960-3174. doi: 10.1007/s11222-022-10167-2. [p151, 152]

- H. Rue and L. Held. *Gaussian markov random fields: Theory and applications*. 2005. ISBN 9780203492024. doi: 10.1198/tech.2006.s352. [p152]
- D. Simpson, H. Rue, A. Riebler, T. G. Martins, and S. H. Sørbye. Penalising model component complexity: A principled, practical approach to constructing priors. *Statistical Science*, 32, 2 2017. ISSN 0883-4237. doi: 10.1214/16-STS576. [p161]
- A. Solin and S. Särkkä. Hilbert space methods for reduced-rank Gaussian process regression. *Statistics and Computing*, 30:419–446, 3 2020. ISSN 0960-3174. doi: 10.1007/s11222-019-09886-w. URL <http://link.springer.com/10.1007/s11222-019-09886-w>. [p151]
- B. M. Taylor and P. J. Diggle. INLA or MCMC? a tutorial and comparative evaluation for spatial prediction in log-gaussian Cox processes. *Journal of Statistical Computation and Simulation*, 84:2266–2284, 10 2014. ISSN 0094-9655. doi: 10.1080/00949655.2013.788653. URL <http://www.tandfonline.com/doi/abs/10.1080/00949655.2013.788653>. [p152]
- B. M. Taylor, T. M. Davies, B. S. Rowlingson, and P. J. Diggle. Lgcp: Inference with spatial and spatio-temporal log-gaussian cox processes in R. *Journal of Statistical Software*, 52:1–40, 2013. ISSN 15487660. [p146, 149, 152]
- B. M. Taylor, T. M. Davies, B. S. Rowlingson, and P. J. Diggle. Bayesian inference and data augmentation schemes for spatial, spatiotemporal and multivariate log-gaussian Cox processes in R. *Journal of Statistical Software*, 63, 2015. ISSN 1548-7660. doi: 10.18637/jss.v063.i07. URL <http://www.jstatsoft.org/v63/i07/>. [p146, 152]
- M. Teng, F. Nathoo, and T. D. Johnson. Bayesian computation for log-gaussian Cox processes: a comparative analysis of methods. *Journal of Statistical Computation and Simulation*, 87:2227–2252, 7 2017. ISSN 0094-9655. doi: 10.1080/00949655.2017.1326117. URL <https://www.tandfonline.com/doi/full/10.1080/00949655.2017.1326117>. [p149, 152]
- J. van Niekerk, H. Bakka, H. Rue, and O. Schenk. New frontiers in Bayesian modeling using the INLA package in R. *Journal of Statistical Software*, 100, 2021. ISSN 1548-7660. doi: 10.18637/jss.v100.i02. [p161]
- A. V. Vecchia. Estimation and model identification for continuous spatial processes. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50:297–312, 1 1988. ISSN 00359246. doi: 10.1111/j.2517-6161.1988.tb01729.x. [p150]
- S. I. Watson, P. J. Diggle, M. G. Chipeta, and R. J. Lilford. Spatiotemporal analysis of the first wave of COVID-19 hospitalisations in Birmingham, UK. *BMJ Open*, 11(10): e050574, oct 2021. ISSN 2044-6055. doi: 10.1136/bmjopen-2021-050574. URL <https://bmjopen.bmj.com/lookup/doi/10.1136/bmjopen-2021-050574>. [p144]
- S. I. Watson, Y. Wang, and E. Giorgi. A fast monte carlo newton-raphson algorithm to estimate generalized linear mixed models with dense covariance. 1 2026. [p147, 148, 164]

1 A minimal, reproducible example

In this section we provide code for a simple, reproducible example using simulated data.

```
b1 <- sf::st_sf(sf::st_sfc(sf::st_polygon(list(cbind(c(0,3,3,0,0),c(0,0,3,3,0))))))
)
g1 <- grid$new(b1,0.5)
dp <- data.frame(y=runif(10,0,3),x=runif(10,0,3),date=paste0("2021-01-",11:20))
dp <- create_points(dp,pos_vars = c('y','x'),t_var='date')
cov1 <- grid$new(b1,0.8)
cov1$grid_data$cov <- runif(nrow(cov1$grid_data))
g1$add_covariates(cov1$grid_data,
                 zcols="cov")
```



```
g1$points_to_grid(dp, laglength=5)
g1$lgcp_ml(popdens="cov")
fit <- g1$model_fit()
summary(fit)
g1$extract_preds("rr")
g1$hotspots(threshold = 1.2, stat = "rr")

# plot some of the outputs
g1$plot("rr")
g1$plot("hotspot_prob")
```

We can compare our simulated data with the predicted incidence and relative risk.

Asymptotic Benchmarking Using the *atime* package

Doris Amoakohene, Anirban Chetia, Igor Steinmacher, Toby Hocking

Abstract Time and memory efficiency are important features of statistical software. When multiple implementations of a given algorithm are available, benchmarking can be used to determine which one is most efficient for a specific task. Typical benchmarking tools provide support for comparing time/memory usage of different computations for a data set of a given size N . In this paper, we present the *atime* package, which provides asymptotic benchmarking, meaning that time and memory (as well as other quantities) can be easily measured for a sequence of increasing data sizes. Asymptotic measurement allows the estimation of complexity classes (big- O notation) and provides a robust new method for testing the performance of R packages. This paper provides a detailed comparison of *atime* with similar packages, and a discussion of how *atime* has been used to improve efficiency in base R and *data.table*.

1 Introduction

Computational efficiency, measured in terms of time and memory usage, is a critical attribute of statistical software. As the scale and complexity of data continue to grow, optimized code and algorithms become more important. For example, the task of writing a large CSV file can be accomplished by various functions, each with different performance characteristics. In R, the `base::write.csv()` function is a standard approach for writing CSV files ([R Core Team, 2022](#)). The R package *data.table* offers a highly efficient `fwrite()` function, optimized for speed and large datasets ([Dowle and Srinivasan, 2021](#)). The `readr::write_csv()` from the *readr* package is another CSV writing function ([Wickham and Hester, 2018](#)). In Python, there are other functions for CSV writing, such as `pandas.DataFrame.to_csv()` and `dask.dataframe.to_csv()` ([McKinney et al., 2010](#); [dask contributors, 2016](#)). To compare the performance of these functions, common practice is to use packages like *microbenchmark* ([Mersmann, 2024](#)) or *bench* ([Hester and Vaughan, 2024](#)), which can run each function on a dataset of fixed size N , and then report metrics such as execution time and memory usage.

However, running time and memory benchmarks using a single data size N can be misleading, because the chosen data size N may not be representative of typical use cases. Since most real-world use cases involve large data sizes N , we propose an asymptotic benchmarking system, where the same code is measured across varying data sizes. For example, instead of benchmarking functions only for writing a CSV file with $N = 100$ rows, we can also run them with $N = 1000$ rows, and so on. This type of analysis makes it possible to estimate asymptotic complexity classes (big- O notation), which tells us how the time or memory usage grows as a function of data size N . This paper discusses various complexity classes, such as linear, $O(N)$; quadratic, $O(N^2)$; etc. The big- O notation gives an upper bound on the growth rate of a function, which in the context of benchmarking is typically the time or memory usage, as a function of data size N . For a more complete introduction to big- O notation, the textbook of [Cormen et al. \(2009\)](#) is a good reference.

In this article, we focus on two kinds of analyses: comparative benchmarking and performance testing, which we define below.

Definition of comparative benchmarking. We use the term “comparative benchmarking” for efforts that aim to compare the performance (typically time and memory usage) of different functions that do the same computation (for example, different functions for writing CSV files). By comparing the performance of these different functions, the aim is to help users make informed choices about what software is most efficient for a particular data manipulation or analysis operation.

Definition of performance testing. We use the term “performance testing” for efforts that aim to ensure that the software stays efficient when the code is updated (for example, changing the source code of a function for writing CSV files). Note that performance testing is a special case of comparative benchmarking, where the different functions to compare are different versions (before and after updating the function’s code). The goal is for package developers to be able to see if their code updates affect performance (time or memory usage). We use the term “continuous performance testing” for the automated and ongoing assessment of performance (time and memory usage), as part of the development workflow. Another term is “continuous benchmarking,” a synonym used by *Bencher*—a web application that provides a graphical user interface for visualizing historical results (*Bencher web site authors*, 2024). We propose a system based on GitHub Actions, where performance testing is run for each commit of a Pull Request, and results can be used to avoid performance regressions before merging each Pull Request.

In this paper, we present a detailed comparison of *atime* with existing benchmarking tools, demonstrating how it can be used to improve performance in R packages like *data.table*.

2 Related work

In this section, we summarize several related software tools for comparative benchmarking and performance testing (Table 1).

2.1 Previous software for comparative benchmarking

Several software packages provide comparative benchmarking functionalities. Base R’s `system.time()` provides a quick way to measure the execution time of one piece of R code, for one data size (*R Core Team*, 2022). *rbenchmark* wraps `system.time()` to evaluate multiple expressions in a specified environment (*Kusnierczyk*, 2024). *microbenchmark* provides nanosecond-precision timing of multiple R expressions (*Mersmann*, 2024), with controls such as randomization of execution order (but without memory measurement). *bench* measures time and memory usage of several pieces of code (*Hester and Vaughan*, 2024), for a single data size (and this is what our proposed *atime* package uses internally). While `bench::press()` could be used to measure time/memory usage for different data sizes, the proposed *atime* package provides a similar method with two advantages. First, *atime* offers a faster approach, which stops measurement if the median time exceeds a certain threshold. Second, sometimes it is desirable to save the results of each expression (for example, to check if the results are consistent). In that case, `bench::mark(check=TRUE)` can be used to automatically stop with an error if any pair of results is not equal. In contrast, the proposed `atime(result=TRUE)` saves results but does not stop automatically, so the user can implement their own checks, which is more flexible.

There are a few examples of previous software for estimating asymptotic complexity classes (big-O notation) from empirical data. For the case when the asymptotic class is known, *Scutari and Malvestio* (2023) suggest using linear least squares to estimate the coefficients. This is similar to *testComplexity* (*Chetia*, 2025), which is an R package that uses linear models to predict a complexity class (constant, log, linear, log-linear, quadratic) for an empirical time or memory curve. A drawback of the linear modeling approach is that all data sizes are equally weighted in the estimation of coefficients, but for asymptotic complexity class estimation, only large data sizes are relevant (time and memory measurements for small data sizes are typically dominated by constant overhead, which does not depend on data size). Therefore, the proposed *atime* package estimates the best fit asymptotic complexity class by aligning reference curves to the two largest data sizes, as explained below in the section “Asymptotic complexity class estimation.”

2.2 Previous software for performance testing

Several other software packages provide functionality for performance testing (Table 1). **airspeed velocity** is a Python library for performance testing (Droettboom et al., 2024), which is used by the numpy and pandas projects (NumPy Developers, 2023; McKinney et al., 2010). Different tests are run for a given code version and data size, and then the results are saved to disk, creating a historical result. Performance is tested by comparing the results for the current code with the historical results. This approach can be useful if the data size is definitely relevant, and the results are always computed on the same computer. However, the usefulness of this approach can be limited because of two factors. First, different data sizes are relevant for testing on different computers, so some benchmarks could be irrelevant on a given computer (if the size is too small). Second, each version of the code to test is typically run on a different job under continuous integration services such as GitHub Actions. With such a setup, it is rarely possible to guarantee access to the same computer, which means that important variations in code versions can be mistaken for insignificant variations over multiple different servers. This type of performance testing system, therefore, results in a high level of false positives (the system says there is a performance regression, but there was just noise due to using different servers). Other similar tools include **conbench** (Conbench authors, 2024), and **bencher** (Bencher web site authors, 2024).

Bruynooghe and the pytest-benchmark Contributors (2024) proposed **pytest-benchmark**, which integrates **airspeed velocity** benchmarking into **pytest**, which is a framework for unit testing. Typically, unit tests are defined for small data sets, which are not relevant to performance testing, so the usefulness of this approach is limited.

In contrast, we propose a performance testing method that overcomes the drawbacks of the previous approaches. To overcome the first drawback (fixed data size), we propose to define each test as a piece of code that is a function of the data size, N . The code is run for increasing data sizes until the median time taken is greater than some threshold (default 0.01 seconds), which can be used to ensure that the largest data sizes are relevant for performance testing. To overcome the second drawback (historical runs of different versions on different servers), we propose to keep a historical database of known fast/slow versions. Instead of running each test using only the current version of the code (HEAD in git terms), we additionally propose to run each test using the known fast/slow versions, along with any other versions that are relevant in the context of a pull request (CRAN, base, merge-base). The advantage is that this approach eliminates most false positives, since all measurements are computed on a single computer, in the same R session. False positives are, of course, still possible, for example, because of “noisy neighbors” on shared virtual machines where the tests are run (but these issues also affect other testing software deployed on virtual machines). The drawback of our proposed approach is that it requires more computation time during each run to install and test several versions of the code (instead of just the most recent version). Our proposed method is similar to the “relative continuous benchmarking” concept on **bencher** (Bencher web site authors, 2024), which is limited to testing a single data size, using two code versions (main and feature branch); this concept is also implemented in **touchstone** (Walthert and Wujciak-Jens, 2024).

3 Example of comparative benchmarking using **atime**

In this section, we demonstrate how **atime** can be used for comparative benchmarking. We use the example of benchmarking R code for text processing using regular expressions (regex), using either PCRE (version 2) or TRE, which are two C libraries that provide similar regex functionality. Philip Hazel et al. (1997) introduced PCRE, which stands for Perl-Compatible Regular Expressions, and Ville Laurikari (2001) proposed TRE, which is an approximate regular expression matching library. With base R functions like `regexpr()`, PCRE is used when `perl=TRUE` and TRE is used when `perl=FALSE`. We can compare the performance of these two libraries using **atime**, and the first step is to define a vector of data sizes, as in the code below.

Table 1: Comparison of software for Comparative Benchmarking (Comp. Bench.) and Performance Testing (Perf. Test.).

	Language	Notable Users	Result Display	Comp. Bench.	Perf. Test.
atime (proposed)	R	data.table	PR comments	yes	yes
bench	R	no	no	yes	no
microbenchmark	R	no	no	yes	no
<code>system.time()</code>	R	no	no	yes	no
rbenchmark	R	no	no	yes	no
airspeed velocity	Python	numpy, pandas	web page	no	yes
conbench	any	arrow, velox	web page, PR comments	no	yes
touchstone	R	styler	PR comments	no	yes

```
> (subject.size.vec <- unique(as.integer(10^seq(0, 3, l=100))))
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
[16] 16 17 18 20 21 23 24 26 28 30 32 35 37 40 43
[31] 46 49 53 57 61 65 70 75 81 86 93 100 107 114 123
[46] 132 141 151 162 174 187 200 215 231 247 265 284 305 327 351
[61] 376 403 432 464 497 533 572 613 657 705 756 811 869 932 1000
```

In the code above, we created a sequence of integer values that will be used to define different data sizes to test with each library. Using a grid of values on the log scale (10^{seq}) is recommended for studying asymptotic time/memory usage. Using `as.integer()` converts each value in the sequence to an integer, and `unique()` ensures that there are no duplicates. For each of the data sizes above, we will create a subject and pattern using the function below.

```
> create_subject_pattern <- function(N) list(
+   subject = paste(rep("a", N), collapse = ""),
+   pattern = paste(rep(c("a?", "a"), each = N), collapse = ""))
> str(create_subject_pattern(3))
List of 2
 $ subject: chr "aaa"
 $ pattern: chr "a?a?a"
```

The `create_subject_pattern()` function above generates a list with two elements, `subject` and `pattern`, based on the input parameter `N`. The `subject` element is a string formed by repeating the letter "a", and `pattern` is constructed by repeating the regex patterns "a?" and "a". In the code below, we use this function in the `setup` argument to create the data required to run the comparative benchmark. We additionally provide the `PCRE` and `TRE` arguments, which are R expressions that will be evaluated for each data size defined in the `N` argument. Ten independent runs are used by default to estimate computation time, and this can be configured by providing the `times` argument.

```
> atime.list <- atime::atime(
+   N = subject.size.vec,
+   setup = {
+     N.list <- create_subject_pattern(N)
+   },
+   PCRE = with(N.list, regexpr(pattern, subject, perl = TRUE)),
+   TRE = with(N.list, regexpr(pattern, subject, perl = FALSE)))
> atime.list
```

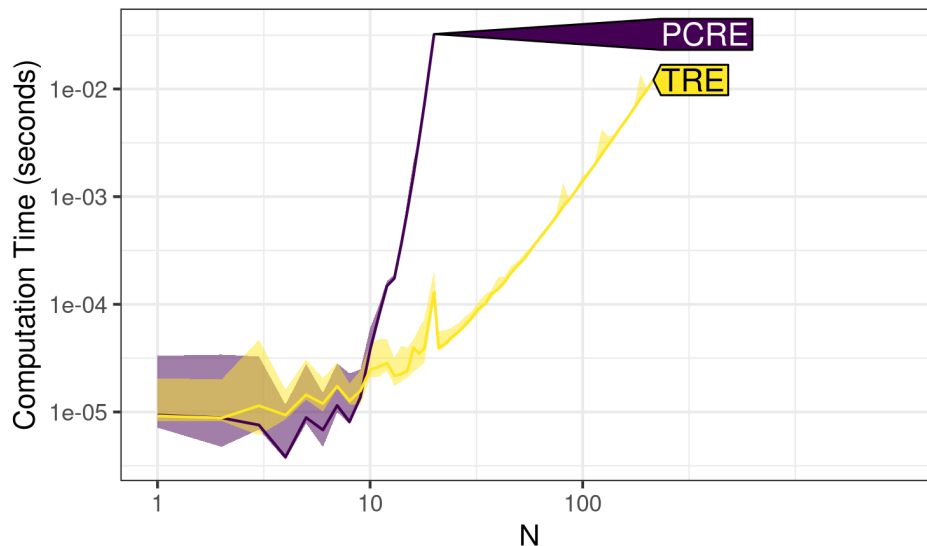


Figure 1: Comparing the computation time for using PCRE and TRE for regex matching with subjects and patterns of size N . Line shows the median, and the shaded band shows min/max, over 10 timings.

```
atime list with 61 measurements for
PCRE(N=1 to 17)
TRE(N=1 to 114)
```

The output above shows the min and max N values that were run for each of the expressions: PCRE went up to 17, and TRE went up to 114. These data sizes exceeded the time limit (default 0.01 seconds), so `atime()` does not run the corresponding expression for any larger data sizes. Therefore, `atime()` is almost always much faster than running a simple loop over data sizes. This is especially true when the different expressions have different asymptotic time complexity classes, as in this case (PCRE is much slower than TRE). In the code below, we use the plot method, which results in Figure 1.

```
> atime.list$unit.col.vec <- c(seconds = "median")
> plot(atime.list) + ggplot2::facet_null() +
+   ggplot2::scale_y_log10("Computation time (seconds)")
```

Note in the code above that the plot method returns a `ggplot` object, which we modify by adding null facets and a different y scale. By default, `atime` measures memory in kilobytes, as well as computation time in seconds. To simplify the discussion of this first example, we set the `unit.col.vec` element in the code above, which ensures that only the computation time is displayed (median line and min/max band over 10 timings). See Figure 4 for a more complicated example that also shows memory usage. Figure 1 shows the computation time for PCRE and TRE, making it easy to see the ranking of these libraries (TRE is faster than PCRE).

3.1 Asymptotic complexity class estimation

To estimate the asymptotic complexity class of each expression, we use the code below:

```
> (best.list <- atime::references_best(atime.list))

references_best list with 61 measurements, best fit complexity:
PCRE (2^N seconds)
TRE (N^3 seconds)
```

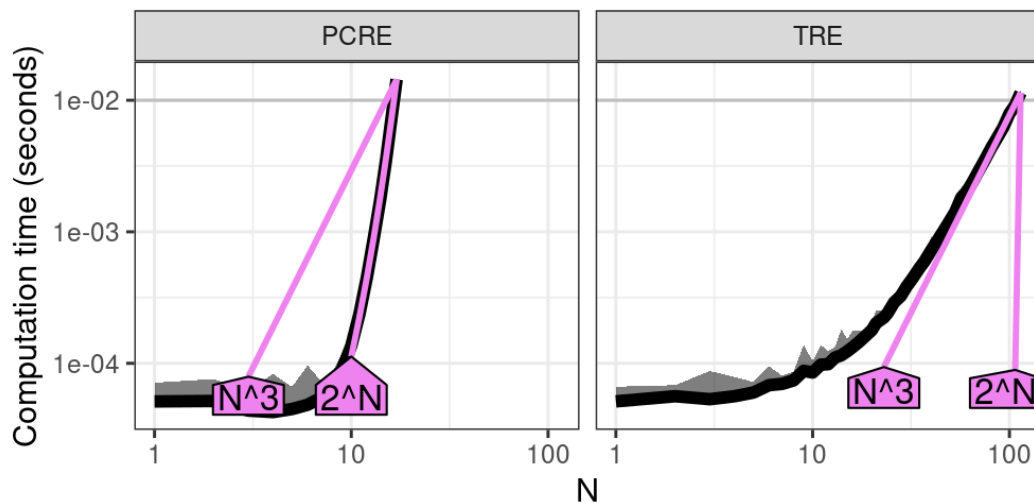


Figure 2: Computation time (seconds) as a function of input size N for PCRE and TRE. Empirical timings are shown in black (median line and min/max band over 10 timings), while violet curves show reference asymptotic growth rates: exponential $O(2^N)$ is the best fit for PCRE, and cubic $O(N^3)$ is the best fit for TRE.

The code above fits an asymptotic reference curve for each of several complexity classes to each empirical timing curve. The complexity classes that are implemented by default include linear $O(N)$, log $O(\log N)$, quadratic $O(N^2)$, exponential $O(2^N)$, etc. The user can define complexity classes to use if the defaults are not sufficient. An asymptotic reference curve is fit for each complexity class by aligning it with the empirical curve for the largest N value. The output is the best fit asymptotic reference curve for each empirical curve, which is defined as the reference curve that is closest to the empirical curve, for the second to largest N value. To visualize the results, we use the code below.

```
> plot(best.list) + ggplot2::facet_grid(. ~ expr.name) +
+   ggplot2::scale_y_log10("Computation time (seconds)")
```

Figure 2 shows the timings of each expression as a function of data size N (black), as well as two asymptotic reference curves (violet, closest reference curve that is smaller/larger, with text labels that can be interpreted in terms of big O notation). Since we have chosen N and the time limit appropriately, we are able to observe the following. For small values of N , the timings are dominated by the overhead, resulting in a nearly constant curve (especially for PCRE). As N increases, the PCRE curve becomes super-linear, indicating exponential complexity, as shown by the increasing slope in the log-log plot, and the almost perfect alignment between the black empirical curve and the violet exponential $O(2^N)$ reference curve. Note that this example is a worst case for the PCRE library, which is actually quite fast for other, more typical subjects and patterns. The worst case is due to a pathological pattern and subject that induces catastrophic backtracking. For TRE, two references are shown: cubic $O(N^3)$ appears to be the best fit, whereas exponential $O(2^N)$ is the closest reference that is slower. Note that all polynomial references, including cubic $O(N^3)$, appear as linear asymptotic trends on the log-log plot. Overall, Figure 2 shows the empirical timings, along with best fit asymptotic reference curves, which indicate that PCRE is exponential time, and TRE is polynomial time, as a function of the size of the subject/pattern N .

3.2 Comparing latency and throughput

When comparing algorithms in terms of computational resources, we can either compare the time/memory required for a given data size N (latency), or show the data size N possible for a given time/memory budget (throughput). To compare latency for a given data size, we can simply subset the data table of empirical measurements, as below:

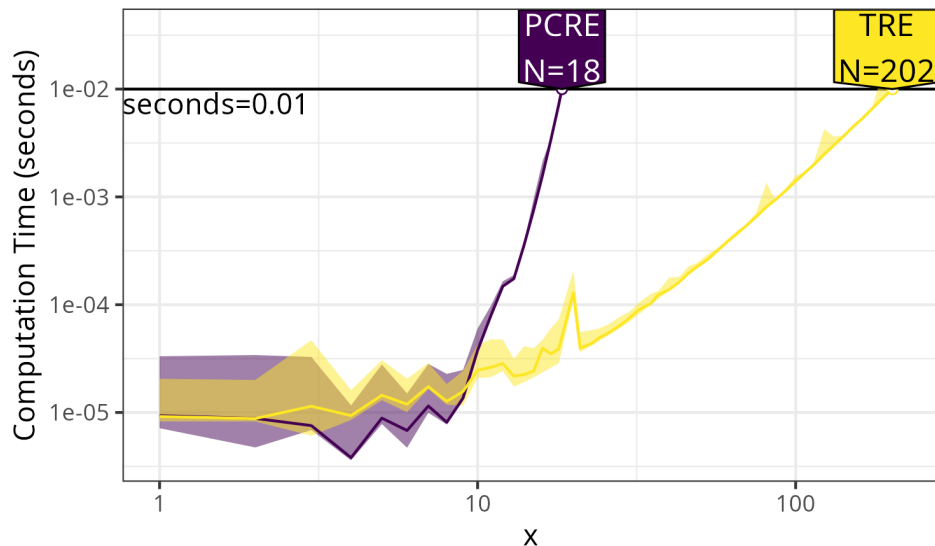


Figure 3: Computation time (seconds) versus data size (N = regex subject/pattern size), highlighting the largest data size N that can be processed within the default time limit of 0.01 seconds. Lines show median, and shaded bands show min/max, over 10 timings.

```
> atime.list$measurements[N==15, .(expr.name, seconds=median)]
```

	expr.name	seconds
1:	PCRE	0.0034745010
2:	TRE	0.0001235669

The output above shows that TRE is more than ten times faster than PCRE, at the given data size. To compare throughput, we use the code below.

```
> (pred.list <- predict(best.list))
```

	unit	expr.name	unit.value	N
1:	seconds	PCRE	0.01	16.4573
2:	seconds	TRE	0.01	109.7062

The output above shows a table with one row per expression. It shows that at the default time limit (0.01 seconds), TRE has a much larger estimated throughput than PCRE (N column). The throughput is estimated using linear interpolation on the log-log plot. The `atime_prediction` object has a plot method, which can be used to compare throughput, as in the code below.

```
> plot(pred.list) + ggplot2::facet_null() +
+   ggplot2::scale_y_log10("Computation time (seconds)", limits=c(NA, 5e-2))
```

Figure 3 shows the throughput, which is the data size N that can be handled in a given amount of time. It is clear that the TRE can handle about 10x larger N for the given time limit.

4 Analyzing different units as a function of data size

In this section, we explain how `atime` can be used to analyze asymptotic properties of other units, in addition to computation time. We consider the example of creating vectors and matrices of size N . In R, dense matrices are created using `matrix()`, whereas sparse matrices

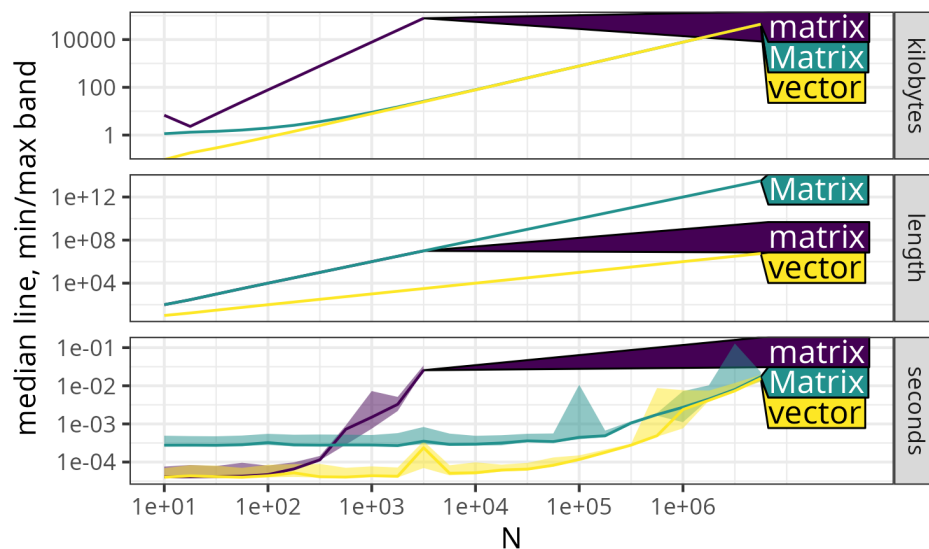


Figure 4: Asymptotic comparison of memory usage (kilobytes), object length, and execution time (seconds), for creating an N -vector, an $N \times N$ dense matrix, and an $N \times N$ sparse Matrix. The second panel shows median lines and min/max bands over 10 timings. Other panels show a line for the single measurement.

are created using `Matrix()`. The advantage of using a sparse matrix is that when the number of non-zeros is sub-quadratic, then the memory usage should also be sub-quadratic (whereas memory usage with a dense matrix would be quadratic). In the code below, we use `atime` to verify these properties, and we also compare with a one-dimensional `numeric()` vector (linear memory usage). Finally, the code below demonstrates usage of the `result` argument, which is a function applied to the result of each expression. Because this function returns a data frame with one row, the column in that data frame (`length`) is interpreted as another unit to analyze (in addition to default units, time in seconds and memory in kilobytes).

```
> library(Matrix)
> vec.mat.result <- atime::atime(
+   N = 10^seq(1, 7, by=0.25),
+   vector = numeric(N),
+   matrix = matrix(0, N, N),
+   Matrix = Matrix(0, N, N),
+   result = function(x) data.frame(length = length(x)))
> plot(vec.mat.result)
```

The code above creates Figure 4, in which there are three panels, each representing a different unit of measurement, as a function of data size N .

Kilobytes: This panel shows memory usage. The $N \times N$ sparse Matrix and N -vector both exhibit linear $O(N)$ asymptotic memory usage, while $N \times N$ dense matrix requires quadratic $O(N^2)$ memory, as shown by the greater slope in the log-log plot.

Length: This panel represents the output of the `length` function. Both matrix and Matrix structures have the same quadratic $O(N^2)$ length values, whereas vector has a smaller linear $O(N)$ length asymptotically, as shown by its smaller slope on the log-log plot.

Seconds: This panel displays execution time. For small values of N , Matrix is slower than both vector and matrix due to a small constant overhead. However, as N increases, Matrix and vector converge to the same linear $O(N)$ asymptotic time complexity, much faster than the quadratic $O(N^2)$ time matrix for large N .

As in the previous section, `references_best()` can be used to estimate asymptotic complexity classes, for each of the expressions (`matrix`, `Matrix`, `vector`), and each of the units

(kilobytes, length, and seconds). Additionally, the `predict()` method can be used to estimate the throughput of each expression, for a given limit of seconds, kilobytes, and/or length.

Finally, we note that Figure 4 is a great example of the benefits of asymptotic measurement (as a function of N). In this example, we obtain different conclusions for different values of N :

$N=10$ to 100 `Matrix` is significantly slower than `matrix`, which is about the same as `vector`.

But this observation is not indicative of performance in large data, because all methods are in the non-asymptotic regime (timings are dominated by constant overhead operations, which do not depend on the data). Also note the small spike for `matrix` at $N=10$, which may be attributed to garbage collection, and can be safely ignored since we are more concerned with performance for large N .

$N=1,000$ to $100,000$ `matrix` is significantly slower than `Matrix`, which is significantly slower than `vector`. Again, this observation is not indicative of performance in large data, because `Matrix` and `vector` are in the non-asymptotic regime (but `matrix` is).

$N=1,000,000$ and more `matrix` is significantly slower than `Matrix`, which is about the same as `vector`. This observation is indicative of performance in large data, because all methods are in the asymptotic regime (timings increase with data size).

The observations above highlight how the conclusions about which method is fastest depend on the data size N . Asymptotic analyses like Figure 4 are advantageous, because they allow the reader to immediately see if each method is in the asymptotic regime (which is desirable for comparing performance in large data).

5 Performance testing using `atime`

The `atime` package also provides functionality for performance testing R packages that are versioned using git. For performance testing an R package, we compute asymptotic time and memory usage for different git versions, and visualize the results to ensure that package code modifications do not result in performance regressions. In the context of our paper, a *performance regression* is a significant increase in execution time or memory usage for a particular test case. In this section, we explain how `atime` has been used to implement performance testing for `data.table`, an R package with users that depend on its impressive performance.

The `atime` package provides functions for prototyping test cases, defining test cases in an R package, and then running the test cases. The first step of performance testing is typically prototyping, which means experimenting with different test code, package versions, and data sizes until significant differences in performance can be observed. Prototyping is done using `atime_versions()`, which runs a given piece of R code, for several data sizes N , and for several package versions (defined using git SHA1 hashes). After prototyping, the arguments used with `atime_versions()` can be reused with `atime_test()`, which is used to define test cases in an R package. Finally, `atime_pkg()` can be used to run all test cases in an R package, or `atime_pkg_test_info()` can be used to extract test metadata and run one test at a time.

5.1 Prototyping performance tests using `atime_versions()`

The first step of performance testing is typically using `atime_versions()` for prototyping. Some arguments of `atime_versions()` are the same as `atime()`: N is a sequence of data sizes, and `setup` is an expression to create data for use in the test. The different package versions are specified as named arguments (`Fast` and `Slow` in the code below) whose values should be SHA1 hashes of the desired git versions. Additionally, `pkg.path` is the path to a local clone of a git repository containing the R package, and `expr` is an expression to time for each different package version. Note that this expression must contain a double or triple colon reference to the package name, such as `data.table:::shallow()` in the code

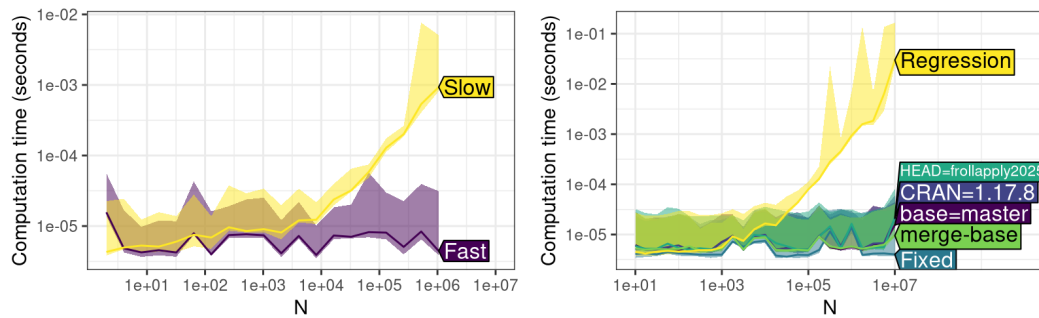


Figure 5: **Left:** `plot()` method for `atime_versions()` shows the asymptotic time taken for `shallow()`, which performs a shallow copy of a data table with N rows, an operation which is expected to be constant time (as in Fast version, but not Slow version). **Right:** the same test case ran in the context of a Pull Request on GitHub, resulting in three additional versions (CRAN, HEAD, base). It is clear that all versions other than Slow/Regression are about the same speed as Fast/Fixed, which indicates that HEAD is OK to merge into the base.

below. This is necessary because the package name is replaced with an edited package name. Testing the different versions is implemented by installing different versions of the package to the same R package library. The different package versions have different names, so they can be loaded and run in the same R session. The names of the different R packages are created by appending the SHA1 hash to the package name, for example `data.table.c4a2085e35689a108d67dacb2f8261e4964d7e12`. The `pkg.edit.fun` argument is a function to edit the package, so that it can be installed and loaded using the versioned package name. A default `pkg.edit.fun` is provided that works with most R packages, including packages that use the **Rcpp** interface of Eddelbuettel and François (2011). The default editing function finds instances of the package name in source code and metadata files (such as DESCRIPTION, NAMESPACE, RcppExports.cpp), and replaces the package name with the edited package name. Because `data.table` has some special configuration and build files, it requires creating a custom function, `edit.data.table()`, see GitHub repository for details, <https://github.com/Rdatatable/data.table/blob/master/.ci/atime/tests.R>. We consider the example usage in the code below.

```
atime::atime_versions(
  pkg.path = "~/data.table",
  pkg.edit.fun = edit.data.table,
  setup = {
    set.seed(1L)
    dt <- data.table(a = sample.int(N))
    setindexv(dt, "a")
  },
  expr = data.table::shallow(dt),
  Slow = "b1b1832b0d2d4032b46477d9fe6efb15006664f4",
  Fast = "9d3b9202fddb980345025a4f6ac451ed26a423be")
```

The code above runs a performance test on two different versions of the `data.table` package (Fast and Slow), involving computing a shallow copy of an indexed table with a variable number of rows N . It is expected that this operation should be constant $O(1)$ time, independent of the number of rows N . When we use the `plot` method (Figure 5, left), we see that the Slow version time is increasing with N , whereas the Fast version time is constant. Results like this, where significant differences can be observed between package versions, indicate that the code is a good candidate for a performance test. To summarize this section, `atime` provides the `atime_versions()` function, which can be used to compare the asymptotic performance of two git versions of an R package.

5.2 Defining performance tests in an R package

After having used `atime_versions()` to prototype a performance test, the next step is to move the test code into the R package. We propose defining performance test cases in the object named `test.list` defined in the `pkg/.ci/atime/tests.R` file, in a git repository `pkg` that contains an R package. Each element of the list should have an informative name for the test case, which will be used to display the results. Each value should be a list of named arguments to pass to `atime_versions()`, such as `setup`, `expr`, `Fast`, and `Slow`. Each test case is typically created based on a historical regression or performance improvement. For example, the R code below defines the test case involving `shallow()`, as discussed in the previous section.

```
test.list <- atime::atime_test_list(
  N = as.integer(10^seq(1, 7, by=0.25)),
  pkg.edit.fun = edit.data.table,
  "shallow speed improved in #4440" = atime::atime_test(
    setup = {
      set.seed(1L)
      dt <- data.table(a = sample.int(N))
      setindexv(dt, "a")
    },
    expr = data.table::shallow(dt),
    Slow = "b1b1832b0d2d4032b46477d9fe6efb15006664f4",
    Fast = "9d3b9202fddb980345025a4f6ac451ed26a423be"
  )
)
```

The code above uses the helper functions `atime_test_list()` and `atime_test()` to define the test case. Both functions are wrappers around `list()`, with non-standard evaluation for special arguments like `setup` and `expr` (un-evaluated expressions to pass to `atime_versions`). The arguments `N` and `pkg.edit.fun` are special names that are recognized by `atime_test_list`, and shared across all of the test cases in the list (only one test case shown above; see [data.table](#) repository for more). In addition to the code above, it is recommended to add names and/or comments that explain the origin of the test case. For example, the test case name "shallow speed improved in #4440" references the PR number, which resulted in the performance improvement. Also, for each SHA1 hash, we recommend a comment such as the following, which was used to document the origin of the Slow commit: Parent of the first commit (<https://github.com/Rdatatable/data.table/commit/0f0e7127b880df8459b0ed064dc841acd22f5b73>) in the PR (<https://github.com/Rdatatable/data.table/pull/4440/commits>) that improves speed. Such comments are useful because they can be used to double-check the validity of the SHA1 commit IDs. In this example, on the PR4440 web page, it can be seen that 0f0e712 is the first commit, and on that commit page, it can be seen that b1b1832 is indeed its parent (which is a good choice for a historical Slow reference because it occurred before the PR that introduced the performance improvement).

5.3 Running performance tests locally and on GitHub Actions

Running performance tests locally. After having defined test cases as above, all the test cases in the package can be run locally by using `atime_pkg("path/to/pkg")`. It runs `atime_versions()` using the arguments provided in each test case, and then saves the results (RDS file), along with summary PNG figure files, to the directory `pkg/.ci/atime`. Alternatively, to run and visualize the results for a single test case, the code below can be used.

```
pkg.info <- atime::atime_pkg_test_info("~/R/data.table")
one.call <- pkg.info$test.call[["shallow regression fixed in #4440"]]
```

Table 2: Version names recognized by atime performance testing.

Version name	Defined?	Version description
Fast	user	An efficient commit
Slow	user	An inefficient commit
Before	user	An efficient commit before a performance regression
Regression	user	An inefficient commit affected by a performance regression
Fixed	user	An efficient commit after fixing a performance regression
CRAN	local	Latest version on CRAN
HEAD	local	Most recent commit in current branch
base	GitHub	Target branch of PR (typically main or master), if current branch is involved in a PR
merge-base	GitHub	The common ancestor between base and HEAD

```
one.result <- eval(one.call)
plot(one.result)
```

The `atime_pkg_test_info()` function returns a list of information about the performance test cases, without actually running them yet. The `pkg.info$test.call` object is a list of unevaluated expressions, each of which calls `atime_versions()` for a given test case. The code `eval(one.call)` runs one performance test, and the `plot` method can be used to visualize the results (Figure 5, right). Running the test case locally adds three versions (Table 2): HEAD (current git version in local clone), base (GITHUB_BASE_REF or master), and CRAN (current release). The performance of these versions can be compared to the historical commits (Fast and Slow) to evaluate the performance of the current code.

Running performance tests on GitHub Actions. GitHub Actions is a continuous integration service that can run arbitrary code after every push to a git repository. A GitHub Action was created to facilitate performance testing of the changes that are introduced in a Pull Request (PR). The primary motivation behind this was to help ensure that `data.table` maintains its code efficiency as PRs are merged. The GitHub Action for performance testing can be enabled by creating a YAML file in `pkg/.github/workflows` (see atime web page for details). The GitHub Action first runs `atime_pkg()` after each push to a branch involved in a PR. The GitHub Action then creates a comment in the PR, with the summary figure that shows the result from the most recent performance test. The comment gets updated after each push to avoid cluttering the PR. The plot shows a column for each test case, with time and memory trends across different `data.table` versions (Table 2). In addition, the comment contains a link to download all the atime-generated result files (figure PNG files and RDS).

Several versions are provided automatically by atime, based on the context of the PR (base, HEAD, merge-base, CRAN). Other versions can be provided by the user based on what is relevant for each test case (Before, Regression, Fixed, Fast, Slow). For test cases that involve a regression, we recommend the version names Before, Regression, and Fixed, to indicate versions relative to the regression (Before and Fixed should be more efficient than Regression). For test cases that involve a performance improvement (or no known regression), we recommend using the version names Fast and Slow.

6 Comparisons between `textttatime` and other software

6.1 Features of `atime` and `touchstone` for performance testing

In terms of performance testing functionality, the most similar R package to `atime` is `touchstone`, which also provides functions for performance testing, but without asymptotic measurement as a function of several data sizes.

Both **touchstone** and **atime** use relative performance testing, meaning that each test case is run using different versions of the R package. Touchstone allows specifying two branches, corresponding to the HEAD of a PR and its base branch. In addition to those branches, **atime** supports other branches that are important in the context of a PR (merge-base and CRAN), as well as user-defined commits that represent historical Fast/Slow versions.

`touchstone::branch_install()` installs each branch to a separate library, and uses **callr** to run each branch in a separate R process. In contrast, `atime::atime_versions()` installs each package version to the same library (using a different package name for each version). Therefore, **atime** allows the different R package versions to be loaded into the same R session, for more direct comparison (reduced noise).

6.2 Comparative benchmarking using `atime::atime()` and `bench::press()`

In terms of comparative benchmarking functionality, the function most similar to `atime::atime()` is `bench::press()`, which allows the user to specify several parameters to vary (not only data size N). The main advantage of **atime** is that it provides convenient features for benchmarking code that scales with data size, N . For parameters other than N , the `atime_grid()` function can be used, as explained below.

Simple example with small data size N , for which **bench and **atime** work equally well.** The **bench** package can be used for asymptotic benchmarking, as long as the max data size N is chosen to yield a reasonable computation time for all expressions. For example, **bench** can be used to do a small-scale asymptotic comparison of PCRE and TRE, via the code below.

```
bench::press(
  N = 1:20,
  perl = c(TRUE, FALSE),
  with(create_subject_pattern(N), bench::mark(
    iterations = 10,
    regexpr(pattern, subject, perl = perl))))
```

The code above is an attempt to replicate Figure 1, but with smaller data sizes ($\text{max}=20$ instead of 1000, which would be too slow to compute using PCRE). In the code above, we use the `bench::press()` function to perform benchmarking across multiple parameter combinations (N and `perl`). For each combination, it uses `create_subject_pattern(N)` to generate a subject and pattern, and subsequently measures the performance of the `regexpr` function using `bench::mark()`. The `regexpr` function performs regular expression matching with the generated pattern on the subject, using the current value of `perl` to toggle Perl-compatible regex. The benchmarking process is repeated `iterations = 10` times for each parameter combination. Below we show the analogous **atime** code.

```
atime::atime(
  N = 1:20,
  setup = N.data <- create_subject_pattern(N),
  expr.list = atime::atime_grid(
    list(perl = c(TRUE, FALSE)),
    regexpr = regexpr(N.data$pattern, N.data$subject, perl = perl)))
```

The code above uses the `expr.list` argument, which is a list of expressions to be benchmarked, constructed using `atime::atime_grid`, which defines a grid of parameter combinations, similar to `bench::press()`. In this case, the grid includes one parameter, `perl`, with values TRUE or FALSE. Overall, we see from this example that the **atime** and **bench** code is very similar, for the case where the max data size N is small enough to be computable by all expressions.

More complex example showing the advantage of `atime`. An example that is more difficult to handle using `bench::press()` would be the analysis presented in Figure 4, which shows an extra unit (length, in addition to seconds and kilobytes), and involves very different time/memory scales (dense matrix is orders of magnitude less efficient than sparse Matrix). That analysis requires two useful features that are provided by `atime()`.

- To measure quantities other than seconds and kilobytes as a function of N (such as length in Figure 4), a custom for loop is required with `bench::press()`.
- If an expression takes longer than the time limit (default 0.01 seconds), then custom code is required for `bench::press()` to not be run for any larger N values. And in the case of Figure 4, it is also important to avoid running the large N values, to avoid running out of memory.

The 28 lines of code below can be used to achieve these two features using `bench::press()`, but it is complicated relative to the corresponding 6 lines of `atime` code that was used to create Figure 4 (see previous section).

```
seconds.limit <- 0.01
done.vec <- NULL
measure.vars <- c("seconds", "kilobytes", "length")
press_result <- bench::press(N = N_seq, {
  exprs <- function(...)as.list(match.call())[-1])
  elist <- exprs(
    vector=numeric(N),
    matrix=matrix(0, N, N),
    Matrix=Matrix(0, N, N))
  elist[names(done.vec)] <- NA #Don't run exprs which already exceeded limit.
  mark.args <- c(elist, list(iterations=10, check=FALSE))
  mark.result <- do.call(bench::mark, mark.args)
  desc.vec <- attr(mark.result$expression, "description")
  mark.result$description <- desc.vec
  mark.result$seconds <- as.numeric(mark.result$median)
  mark.result$kilobytes <- as.numeric(mark.result$mem_alloc/1024)
  mark.result$length <- NA
  for(desc.i in seq_along(desc.vec)){
    description <- desc.vec[[desc.i]]
    result <- eval(elist[[description]])
    mark.result$length[desc.i] <- length(result)
  }
  mark.result[desc.vec %in% names(done.vec), measure.vars] <- NA
  over.limit <- mark.result$seconds > seconds.limit
  over.desc <- desc.vec[is.finite(mark.result$seconds) & over.limit]
  done.vec[over.desc] <-< TRUE
  mark.result
})
```

The code above uses entries of `done.vec` to keep track of which expressions have already gone over the time limit. It also uses a for loop to evaluate each expression and save the length of the result to analyze as a function of N . Overall, the code is substantially more complex than the corresponding `atime` code, because it has to re-implement two features that `atime` provides.

To conclude this section, we have discussed the use of `bench` and `atime` packages for comparative benchmarking in R. Both methods can be used for comparative benchmarking across different parameter configurations. However, `atime::atime()` provides two key features which are not present in `bench::press()`: measurement of arbitrary units (other than time and memory), and stopping evaluation for expressions that go over a time limit.

package	seconds	PCRE data sizes	TRE data sizes
bench	0.42	19	19
bench	0.81	20	20
bench	1.69	21	21
atime	1.70	1–18	1–151
bench	3.40	22	22
bench	6.95	23	23

Table 3: Overall time required to compute regular expression benchmarks, as in Figure 1, using either **bench** (with various data sizes from 19 to 23) or **atime** (with the default time limit of 0.01 seconds, which results in a range of data sizes).

package	seconds	Matrix data sizes	matrix data sizes	vector data sizes
bench	0.66	5000	5000	5000
atime	2.21	1e+01–3e+06	1e+01–3e+03	1e+01–1e+07
bench	3.22	10000	10000	10000
bench	11.53	20000	20000	20000
bench	out of memory	1e+05	1e+05	1e+05

Table 4: Overall time required to compute sparse/dense matrix benchmarks, as in Figure 4, using either **bench** (with various data sizes from 5000 to 100,000) or **atime** (with the default time limit of 0.01 seconds, which results in a range of data sizes).

6.3 Comparing overall time using different benchmarking packages

In this section, we provide an empirical comparison between **atime** and **bench**, in terms of how much time it takes overall to compute benchmark results. Our goal is to demonstrate an advantage of the proposed asymptotic approach implemented in **atime**: we can gather a larger range of measurements in a comparable or smaller amount of time.

First, we considered the regular expression example in Figure 1. The overall time taken by **atime** to compute the data for this benchmark was 1.7 seconds, and it resulted in a range of measurements (N=1 to 18 for PCRE, and N=1 to 151 for TRE). For comparison, we ran **bench** with various data sizes from N=19 to N=23, and we show the timings in Table 3. The most comparable amount of time for **bench** was 1.69 seconds for data size N=21, which is slightly larger than the largest data size for PCRE with **atime** (N=18), but significantly smaller than the largest data size for TRE with **atime** (N=151). Smaller data sizes were faster (0.42 seconds for N=19), and larger data sizes were slower (6.95 seconds for N=23).

Second, we considered the sparse matrix example in Figure 4. The overall time taken by **atime** to compute the data for this benchmark was 2.21 seconds, and it resulted in a range of measurements (N=10 to 3,000,000 for Matrix, N=10 to 3,000 for matrix, N=10 to 10,000,000 for vector). For comparison, we ran **bench** with various data sizes from N=5,000 to N=100,000, and we show the timings in Table 4. The most comparable amount of time for **bench** was 3.22 seconds for data size N=10,000, which is slightly larger than the largest data size for the matrix with **atime** (N=3,000), but significantly smaller than the largest data size for the other **atime** measurements (millions for Matrix and vector). Smaller data sizes were faster (0.66 seconds for N=5,000), and larger data sizes were either slower (11.53 seconds for N=20,000) or did not run (not enough memory to run the matrix for N=100,000).

Overall, the comparisons in this section have shown the advantages of the **atime** approach, which computes measurements for a range of data sizes until the median time exceeds a limit (default 0.01 seconds). The **atime** approach can be faster overall, for more informative results (measurements for a range of data sizes, not just one data size).

6.4 Comparison with Python continuous benchmarking

Python continuous benchmarking software includes **airspeed velocity** and **conbench**. Whereas the proposed **atime** method keeps a historical database of commits which are

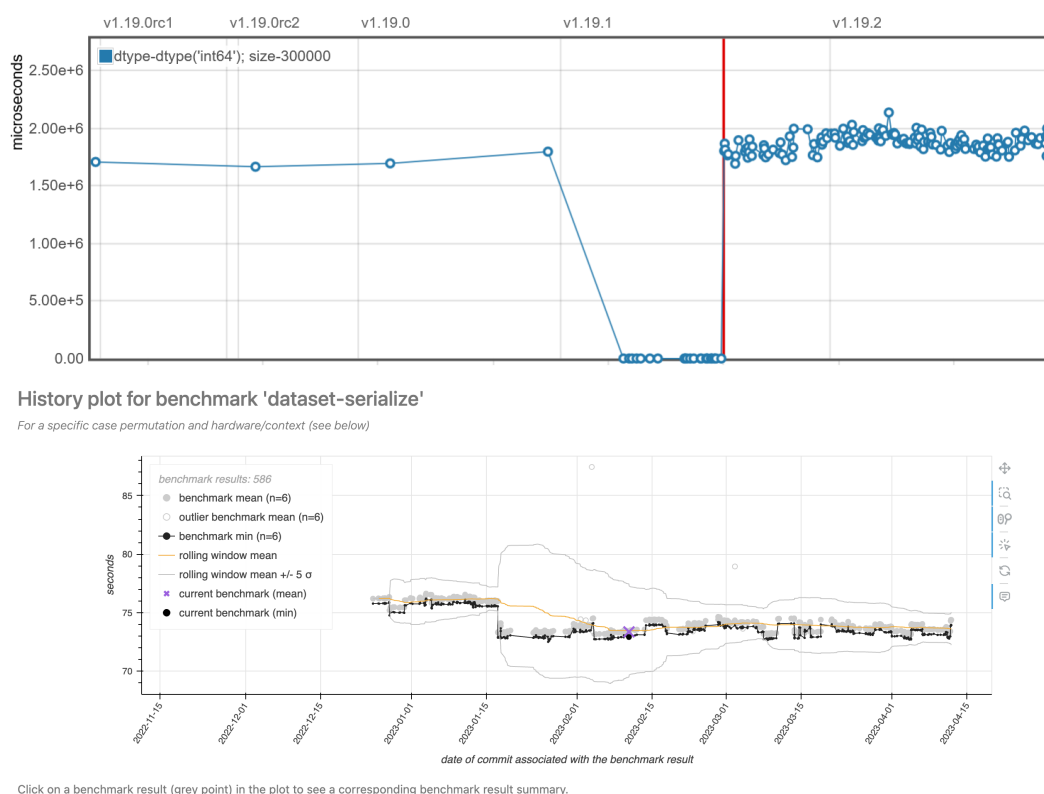


Figure 6: Comparison with other continuous benchmarking software. Air Speed Velocity (top) and Conbench (bottom) are measurements of computation time for a fixed data size (Y axis), shown as a function of the commit history (X axis). The main drawback of this method is that it results in false positive changes up and down, as can be seen in these figures.

known to be Fast or Slow, these packages keep a historical database of timings for all commits (each timing computed once for that commit and then saved to compare with timings computed for other commits). Another major difference is that each `atime` test results in several measurements for a set of increasing data sizes, whereas these other tools use a fixed data size (which may not be relevant on different computer hardware). Assuming a relevant data size is used, these alternative approaches could be very useful, as performance regressions could be visualized as abrupt increases in computation time over the history of commits. Examples of web pages that are generated by these packages are shown in Figure 6, which show results taken from https://pv.github.io/numpy-bench/#bench_trim_zeros.TrimZeros.time_trim_zeros and https://conbench.github.io/conbench/pages/lookback_zscore.html. The limitation of this approach is that it requires careful control of the underlying hardware that was used to compute the benchmark timings for each commit. If there are variations in the hardware over time, that can result in false positive changes in benchmark computation time, which makes it more difficult to detect real performance regressions due to changes in the code. `atime` does not suffer from this drawback, because it computes timings for each data size and code version at test time on the same machine (Table 2), so it does not need to keep a database of historical benchmarks. Instead of presenting the user a time series plot with different code versions on the X axis, `atime` presents data size on the X axis (Figure 5), with different code versions represented as different colored empirical timing curves. False positives are, of course, still possible with `atime`, for example, because of “noisy neighbors” on shared virtual machines, or random garbage collection events, but these issues also affect other testing software.

7 Discussion and conclusions

In this paper, we presented the **atime** package for comparative benchmarking of R code and performance testing in R packages. The unique feature of **atime** is asymptotic measurement of time and memory (as a function of data size), which makes it easy to see when the code performance is in the asymptotic regime (dependent on data size). The **atime** package enables users to compute and visualize asymptotic timings, including comparing empirical timings with theoretical references and estimating throughput. Several examples illustrated how **atime** can be applied for comparative benchmarking of different R functions that perform similar computations (but with different performance characteristics). Using **data.table** as an example, we explained how **atime** can be used for performance testing of R packages across multiple git versions.

Additionally, we compared **atime**'s syntax and functionality with other tools used for performance testing and benchmarking. Compared to other benchmarking tools (which support only a single data size), the unique feature of **atime** is support for sequences of data sizes. We showed that it is technically possible to use `bench::press()` for asymptotic analysis across a sequence of small data sizes, but **atime** is preferable because it supports convenience features such as stopping if a timing exceeds a time limit. Another similar package is **touchstone**, which provides a GitHub Action for performance testing based on comparing HEAD/base versions, but does not support asymptotic analysis (varying data sizes).

Limitations and future development. A limitation of **atime** comes from the fact that it uses `bench::mark()` to measure time/memory for each data size. This dependency means that **atime** inherits both the strengths and limitations of `bench::mark()`. Because this method uses `Rprofmem()`, it is limited to measuring R memory allocations (and can not measure other C memory allocations). Also, this method runs timings for each expression several times, without re-running the setup expression. This makes it difficult to run timings on functions that alter their inputs by reference. For example, `data.table::setkey(DT)` alters `DT` by reference (sort in place); the first execution for memory measurement will sort the data, and subsequent executions for time measurement will do a linear scan to verify the data is sorted (faster). Future versions of **atime** could overcome this limitation by removing the dependency on **bench** and implementing a time/memory measurement procedure that runs the setup expression before each measurement.

Current use cases. We note that **atime** is currently being used for continuous performance testing of **data.table**, with 16 test cases currently implemented, and running for every pull request on GitHub. The **atime** methodology has also been used to uncover and fix asymptotic performance issues in base R. For example, asymptotic analysis of `write.csv()` revealed that it was quadratic $O(N^2)$ in the number of columns N , whereas the expected complexity was linear $O(N)$. Sebastian Meyer realized this issue was related to another issue with replacing columns via `[, data.frame]`, which was observed to be super-linear, but expected to be linear. The fix for these two issues appeared in 2023 with R version 4.3.0. As mentioned in the introduction, **atime** has also been used for comparative benchmarking of CSV reading and writing functions (for details, see <https://tdhock.github.io/blog/2023/dt-atime-figures/> and <https://tdhock.github.io/blog/2024/pandas-dt/>). Another improvement to base R was in `substring()`, which was expected to be linear $O(N)$ time in the number of outputs N , and `gregexpr()`, which was expected to be linear $O(N)$ time in the number of characters N in the subject. Our asymptotic analysis revealed that both operations were quadratic $O(N^2)$ time, and Tomas Kalibera merged the proposed fixes into R version 3.6.0 (2019). Overall, we have shown how **atime** simplifies such asymptotic analysis, and we expect that it will be a useful tool for future analyses of other R functions.

Reproducible research statement: The following repositories contain all the code for reproducing the figures in this paper:

<https://github.com/tdhock/atime-article>
<https://doi.org/10.5281/zenodo.14796684>

References

- Bencher web site authors. What is continuous benchmarking? Available online at <https://bencher.dev/docs/explanation/continuous-benchmarking/>, 2024. [p171, 172]
- F. Bruynooghe and the pytest-benchmark Contributors. pytest-benchmark v5.1.0: pytest fixture for benchmarking code. <https://github.com/ionelmc/pytest-benchmark>, 2024. [p172]
- A. Chetia. *testComplexity: Asymptotic Complexity Testing Framework*, 2025. URL <https://github.com/Anirban166/testComplexity>. R package version 0.1.1. [p171]
- Conbench authors. conbench: Language-independent continuous benchmarking (CB) framework. <https://github.com/conbench/conbench> copyright Ursa Computing, 2024. [p172]
- T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to algorithms. *Journal of Algorithms*, 45(2):78–102, 2009. [p170]
- dask contributors. *dask: Parallel computing with task scheduling*, 2016. URL <https://dask.org>. Python package version 2022.10.0. [p170]
- M. Dowle and A. Srinivasan. *data.table: Extension of ‘data.frame’*, 2021. R package version 1.14.2. [p170]
- M. Droettboom, P. Virtanen, and asv Developers. airspeed velocity. A tool for benchmarking Python packages. Available at <https://asv.readthedocs.io/en/stable/>, 2024. [p172]
- D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. doi: 10.18637/jss.v040.i08. [p179]
- J. Hester and D. Vaughan. *bench: High Precision Timing of R Expressions*, 2024. URL <https://CRAN.R-project.org/package=bench>. R package version 1.1.3. [p170, 171]
- W. Kusnierczyk. *rbenchmark: Benchmarking routine for R*, 2024. URL <https://CRAN.R-project.org/package=rbenchmark>. R package version 1.0.0. [p171]
- W. McKinney et al. *pandas: Powerful data structures for data analysis in Python*, 2010. Python package version 1.5.0. [p170, 172]
- O. Mersmann. *microbenchmark: Accurate Timing Functions*, 2024. URL <https://CRAN.R-project.org/package=microbenchmark>. R package version 1.4.10. [p170, 171]
- NumPy Developers. *NumPy: The fundamental package for scientific computing with Python*, 2023. Available at <https://numpy.org/>. [p172]
- Philip Hazel, Z. Herczeg, and contributors. *PCRE2 - Perl Compatible Regular Expressions version 2*. University of Cambridge, 1997. Version 10.40, <https://www.pcre.org/>. [p172]
- R Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2022. URL <https://www.R-project.org/>. [p170, 171]
- M. Scutari and M. Malvestio. *The Pragmatic Programmer for Machine Learning: Engineering Analytics and Data Science Solutions*. Chapman and Hall/CRC, 2023. [p171]
- Ville Laurikari. *TRE - Approximate Regex Matching Library*. Department of Computer Science and Engineering, Helsinki University of Technology, 2001. Version 0.8.0, <https://laurikari.net/tre/>. [p172]

- L. Waltherth and J. Wujciak-Jens. *touchstone: Continuous Benchmarking with Statistical Confidence Based on 'Git' Branches*, 2024. URL <https://github.com/lorenzwalthert/touchstone>. R package version 0.0.0.9002, commit afa8d4284ed51aa99df59d6903fa5c5ad560b8dc. [p172]
- H. Wickham and J. Hester. *readr: Read Rectangular Text Data*, 2018. R package version 2.1.0. [p170]

StepReg: A Comprehensive and Intuitive R Package for Stepwise Regression Analysis

by Junhui Li, Kai Hu, Xiaohuan Lu, Cesar B. Sotelo, Sushmita Nayak, Michael A. Lodato, Wenxin Liu, and Lihua Julie Zhu

Abstract Stepwise regression remains widely used for model selection, yet the field lacks a comprehensive, well-documented tool that supports diverse model families and selection strategies, implements multiple information criteria, and addresses overfitting and post-selection inference. We present StepReg, an R package that unifies stepwise selection across linear, generalized linear (e.g., logistic, Poisson, Gamma, negative binomial), and Cox models, supporting forward, backward, bidirectional, and best-subset search under various information criteria. StepReg also supports multivariate multiple linear stepwise regression, enabling simultaneous modeling of multiple dependent variables. Users can explore multiple strategies and information criteria within a single function call, and optionally combine them for more efficient and flexible model selection. To enhance robustness, StepReg provides an optional randomized forward selection mode to mitigate overfitting and a data-splitting workflow to improve the reliability of post-selection inference. The package further provides logging and visualization of the selection path, along with exporting results in common formats. A companion R package, StepRegShiny, has also been developed to provide a Shiny-based GUI for point-and-click analysis. Accuracy was assessed on public datasets by cross checking results against SAS. Together, these features provide a transparent, extensible framework that streamlines stepwise regression while promoting best practices.

1 Introduction

Stepwise regression remains a practical, widely used technique for model building and predictor screening across many applied domains (Ho et al., 2024; Naei et al., 2024; Yang et al., 2024a,b, 2023; Zhu et al., 2023; Brandão-Dias et al., 2023; De Caires et al., 2022; Makepeace et al., 2022; Ebrahimi et al., 2024; Caires et al., 2023; Seter et al., 2023; Tadros et al., 2023), valued for its interpretability and ease of use when the candidate set is large. However, it carries well-recognized limitations, including a greedy, path-dependent search, susceptibility to overfitting, and challenges in conducting valid post-selection inference (Lewis-Beck, 1978; Flom and Cassell, 2007; Whittingham et al., 2006; Austin and Tu, 2004; Smith, 2018). These concerns, while important, can be mitigated through strategies such as principled stopping criteria (e.g., Akaike Information Criterion (AIC) (Akaike, 1973; Al-Subaihi, 2002; Hurvich and Tsai, 1989; Darlington, 1968; Judge et al., 1985) and Bayesian Information Criterion (BIC) (Judge et al., 1985; Sawa, 1978)), cross-validation, data splitting with hold-out refitting to support more reliable inference, penalized regression as a complementary check, bootstrap or post-selection inference adjustments, and randomized variants of forward selection to reduce overfitting and stabilize variable selection (Hastie et al., 2009; O'Hara and Sillanpää, 2009; George and McCulloch, 1993; Mitchell and Beauchamp, 1988; Stone, 1974; Efron and Tibshirani, 1993; Hastie et al., 2009; Tibshirani, 1996). With these safeguards in place, stepwise procedures can preserve their practical utility and interpretability while mitigating key limitations.

Stepwise regression encompasses a variety of methodologies (termed “regression families”); each regression family is applicable to different types of regression models with its own complexities and considerations. For instance, linear stepwise regression (Draper and Smith, 1998b; Burnham and Anderson, 2002) is used for predicting continuous outcomes, selecting predictors based on information criteria like AIC, BIC, and p-values. Logistic stepwise regression (Hosmer et al., 2013; Menard, 2002), on the other hand, is used for categorical outcomes, requiring adjustments to information criteria due to the non-linear nature of logistic models. The Cox proportional-hazards model (Cox, 1972; Therneau and Grambsch, 2000) is utilized for survival analysis, focusing on the time until an event occurs and requiring careful handling of both the proportional hazards assumption and censored data. Each of these methods necessitates careful considerations to balance model complexity, accuracy, and interpretability.

In terms of variable selection strategies, stepwise regression typically employs three widely used methods—forward selection, backward elimination, and bidirectional selection—along with the best-subset selection (see Supplementary Table 1). All strategies operate within a user-defined model scope and are influenced by predictor collinearity; they differ chiefly in path dependence and computational cost. Forward selection is a greedy, one-step-ahead procedure that can overfit if unchecked, yet it often

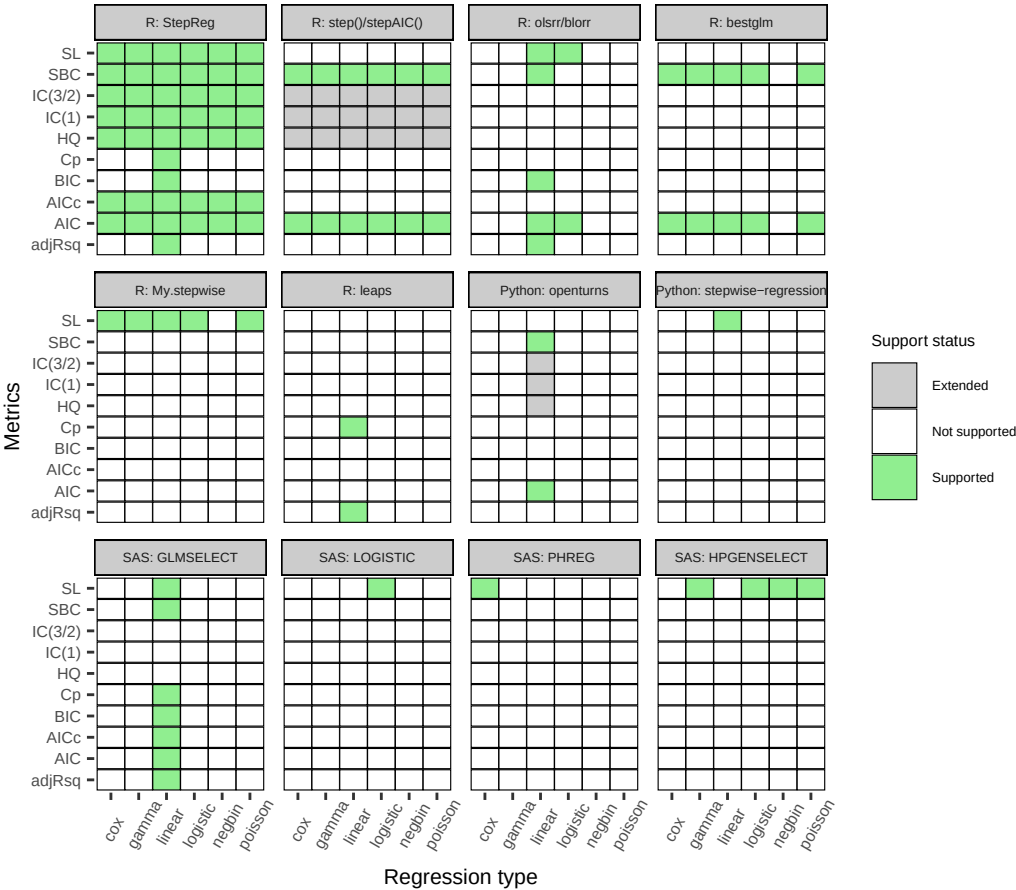


Figure 1: Information criteria/metrics supported by 14 stepwise regression tools for each regression family

attains comparable predictive performance against best-subset at substantially lower computational cost (Hastie et al., 2020). Backward elimination begins with a full model and is most reliable when the number of predictors (p) is modest relative to the number of observations (n). Bidirectional selection permits both additions and deletions, offering flexibility but remaining path dependent with the same sensitivities. Best subset selection exhaustively evaluates all subsets of the specified predictors under a chosen criterion. This approach is path-independent but computationally expensive (scaling roughly as 2^p) and susceptible to overfitting without proper validation. The inherent complexity of model selection motivates a unified framework that streamlines applying and comparing tailored or combined strategies across datasets of varying size and type.

To evaluate model performance, various information criteria have been proposed (see Supplementary Table 2 and Supplementary Table 3 for details). These include information criteria-based metrics, such as AIC and BIC, as well as significance level-based (SL) methods like the F-test or the Wald test. Each metric offers distinct advantages and limitations, with the penalty coefficient size influencing model complexity: too small a penalty can lead to overfitting, while too large a penalty can cause underfitting. Researchers often need to combine different variable selection strategies and information criteria to identify the most optimal model.

Currently, ten open-source tools and four SAS procedures implement stepwise regression or best-subset selection (see Figure 1 and Supplementary Table 4). Although each has its strengths, capabilities are uneven and often narrow. Notably, to our knowledge, no existing tool implements randomized forward selection, a procedure that evaluates a random subset of remaining predictors at each step to reduce overfitting (Mentch and Zhou, 2020; Chen et al., 2025), and most lack built-in data-splitting workflows to support more reliable post-selection inference. Coverage of regression families and information criteria is frequently limited. These constraints substantially restrict the model space that can be explored within any single tool, complicating comprehensive stepwise regression analyses. We summarize supported regression families, selection strategies, information criteria, and additional features by tool in Figure 1 and Supplementary Table 4.

Of the 14 existing stepwise/subset selection tools we surveyed, eight are implemented in R (R Core Team, 2025). The most widely used procedures—`stats::step()` (R Core Team, 2025) and

`MASS::stepAIC()` (Venables and Ripley, 2002)—provide forward, backward, and bidirectional search but are oriented primarily toward AIC-type criteria. Other R packages cover narrower scopes. For example, `leaps` (based on Fortran code by Alan Miller, 2024) implements best-subset selection for linear models with Mallows' Cp (Cp) (Hocking, 1976; Mallows, 1973), R-squared, and adjusted R-squared (Darlington, 1968; Judge et al., 1985). `bestglm::bestglm()` (McLeod et al., 2020) extends best-subset ideas to general linear model (GLM) families (e.g., logistic, Poisson, and Gamma), but remains focused on information criteria such as AIC, Schwarz Bayesian Information Criterion (SBC) (Al-Subaihi, 2002; Hurvich and Tsai, 1989; Judge et al., 1985; Schwarz, 1978) and related variants, along with mean square error evaluated via cross-validation (Shao, 1993, 1997; Davison and Hinkley, 1997; Hastie et al., 2009). `olsrr` (Hebbali, 2024) and `blorr` (Hebbali, 2020) offer visualization utilities for linear and logistic regression workflows, respectively, while supporting a restricted set of information criteria; in our test environment, their Shiny graphical user interface (GUI) did not launch due to unresolved dependencies. `My.stepwise` (Hu, 2017) prioritizes SL entry/removal and does not support alternative information criteria for GLM and Cox families.

Beyond R, two Python packages, `stepwise-regression` (Vijayakumar, 2019) and `openturns` (Baudin et al., 2016), implement stepwise procedures but are limited to linear regression. `stepwise-regression` provides forward and backward selection using SL thresholds and has relatively limited documentation. `openturns` offers broader functionalities and up-to-date documentation; for stepwise routines, its manual indicates native support primarily for AIC and SBC, with other criteria potentially emulated via the penalty parameter, but users need to explore this further. Experienced users can use functions or modules from `statsmodels` (Seabold and Perktold, 2010) or `sklearn` (Pedregosa et al., 2011) for stepwise regression. To our knowledge, no R or Python package currently implements randomized forward selection or offers built-in data-splitting workflows to mitigate overfitting and enable more reliable post-selection inference.

In SAS, multiple procedures support stepwise selection across various model families, variable selection strategies, and information criteria. Logistic and Cox regression are fit via PROC LOGISTIC (SAS Institute Inc., 2019c) and PROC PHREG (SAS Institute Inc., 2019a), respectively. PROC HPGENSELECT (SAS Institute Inc., 2019b) supports generalized linear models, including Poisson, Gamma, logistic, and negative binomial (NB) regression. Selection strategies typically include forward, backward, and stepwise variants; by default, these use SL entry/removal thresholds, with information criterion and resampling options available in some procedures. While data-splitting or cross-validation can be configured, we are not aware of randomized forward selection being available natively in SAS. SAS is commercial software. Although graphical interfaces such as SAS Studio/Enterprise Guide exist, stepwise selection capabilities are primarily accessed through programmatic procedures, which may pose a barrier for some users.

Unlike univariate regression, which focuses on one outcome, multivariate regression looks at how several outcomes interact with each other and with the same set of predictors (Izenman, 2013). This method can reveal complex interactions and correlations that are not apparent in univariate analyses, leading to more informed decision-making and better insights. This analysis is useful in fields like economics (Seyedaliakbar et al., 2019; Tintner, 1964), where multiple economic metrics are analyzed, or medicine (Matsui et al., 2007; Peng et al., 2010; Bonnini and Borghesi, 2022), where multiple health outcomes are studied. Multivariate regression uses techniques such as multivariate analysis of variance, canonical correlation analysis, and structural equation modeling to model and interpret these complex relationships. Notably, existing stepwise regression tools lack support for multivariate regression.

To address gaps in existing stepwise tools, we developed `StepReg`, a unified, versatile, and user-friendly R package for comprehensive stepwise regression analyses. `StepReg` supports six commonly used regression families—linear, logistic, Cox, Gamma, Poisson, and negative binomial regression—implements forward, backward, bidirectional, and best subset selection under various information criteria (e.g., AIC and BIC), and adds capabilities not found in other software tools, including multivariate regression (see Figure 1 and Supplementary Table 4). To enhance robustness, it provides optional randomized forward selection mode and a data-splitting workflow to mitigate overfitting and improve reliability of post-selection inferences. For purely predictive modeling, where the emphasis is on out-of-sample performance rather than formal inference, these inferential concerns are less central. We verify numerical correctness by cross-checking `StepReg`'s results against SAS on public datasets. To enhance accessibility, we provide an interactive Shiny-based GUI in the companion package `StepRegShiny`. Together, these features make `StepReg` a practical and comprehensive tool for stepwise regression analyses.

2 Materials and methods

2.1 Implementation

StepReg implements six commonly used stepwise regression families, including linear, logistic, Cox, Gamma, Poisson, and negative binomial regression in R. To streamline parameter acquisition for model comparison, **StepReg** utilizes the function `lm()` in stats for linear models, and function `glm()` in stats for logistic, Poisson, and Gamma regression models. For Cox proportional hazards models and negative binomial generalized linear models, **StepReg** employs function `coxph()` in **survival** (Therneau and Grambsch, 2000; Therneau, 2024) and function `glm.nb()` in **MASS** (Venables and Ripley, 2002), respectively. Additionally, **StepReg** supports multivariate multiple linear stepwise regression by utilizing function `lm()` in stats to fit multivariate regression models and function `anova()` in stats to approximate F-statistics for model selection. Visualization of the selection process and output generation are implemented using **ggplot2** (Wickham, 2016) and **flextable** (Gohel and Skintzos, 2024). The GUI is developed using the **shiny** (Chang et al., 2024) package and its extensions (Attali, 2022; Chang, 2021; Attali et al., 2024; Xie et al., 2024).

2.2 Selection strategies and information criterion formulas

StepReg implements four widely used variable selection strategies: forward selection, backward elimination, bidirectional selection, and the best subset selection (see Supplementary Table 1). Candidate models are evaluated using standard selection metrics. For information criteria such as AIC, BIC, and SBC, a lower value suggests a better model when adding or removing variables, whereas a higher adjusted R-squared reflects stronger model fit. Alternatively, for the SL, a predictor is included if its p-value falls below the entry threshold (`s1e`) and removed if it exceeds the stay threshold (`s1s`). Because definitions of some information criteria—particularly for linear models and small sample corrections—differ across sources and software (Darlington, 1968; Judge et al., 1985; Hurvich et al., 1998; Hurvich and Tsai, 1989), we documented the exact formulas implemented in **StepReg** in Supplementary Table 3 to ensure transparency and reproducibility.

2.3 Benchmark and datasets

We benchmarked **StepReg** against SAS using publicly available datasets: the `mtcars` dataset from the stats R package (R Core Team, 2025), and four datasets bundled with the **StepReg** R package—remission (Lee, 1974), lung (Therneau and Grambsch, 2000; Therneau, 2024), affairs (Kleiber and Zeileis, 2008), and tobacco (Draper and Smith, 1998a). To compare the outputs of multivariate regression, we utilized the SAS/IML script from a previous study (Al-Subaihi, 2002) and modified the information criterion formulas. Specifically, for the SBC metric, we adjusted the penalty coefficients by multiplying them by the number of dependent variables to mitigate overfitting issues, as suggested by previous work (Anderson, 2003). For other metrics, we made adjustments by either adding or multiplying a constant to align with the definition used in **StepReg**. The parameters used in the comparison are listed in Supplementary Table 5, together with the corresponding benchmark R script (2.3_Benchmark_StepReg.R) and SAS script (2.3_Benchmark_SAS.SAS) available in the Supplemental Materials.

3 Results

3.1 Workflow overview

StepReg provides four core functions, `stepwise()`, `plot()`, `performance()`, and `report()`, while the companion package **StepRegShiny** offers a Shiny-based GUI launcher, `StepRegGUI()`. Figure 2 summarizes the workflow. Analyses begin with `stepwise()`, which runs the selection procedure and returns a **StepReg** object recording the selection path and final model(s). This object can then be passed to `plot()` to visualize the path, to `performance()` to evaluate model performance on training and test data, and to `report()` to export results in common formats. Detailed arguments and default settings for each function are provided in Section 3.2 (Core functions and parameter definitions).

3.2 Core functions and parameter definitions

Below is a concise reference for the main arguments of the four core functions. See Table 1 for guidance on multiple dependent variables, survival outcomes, and predictor specification.

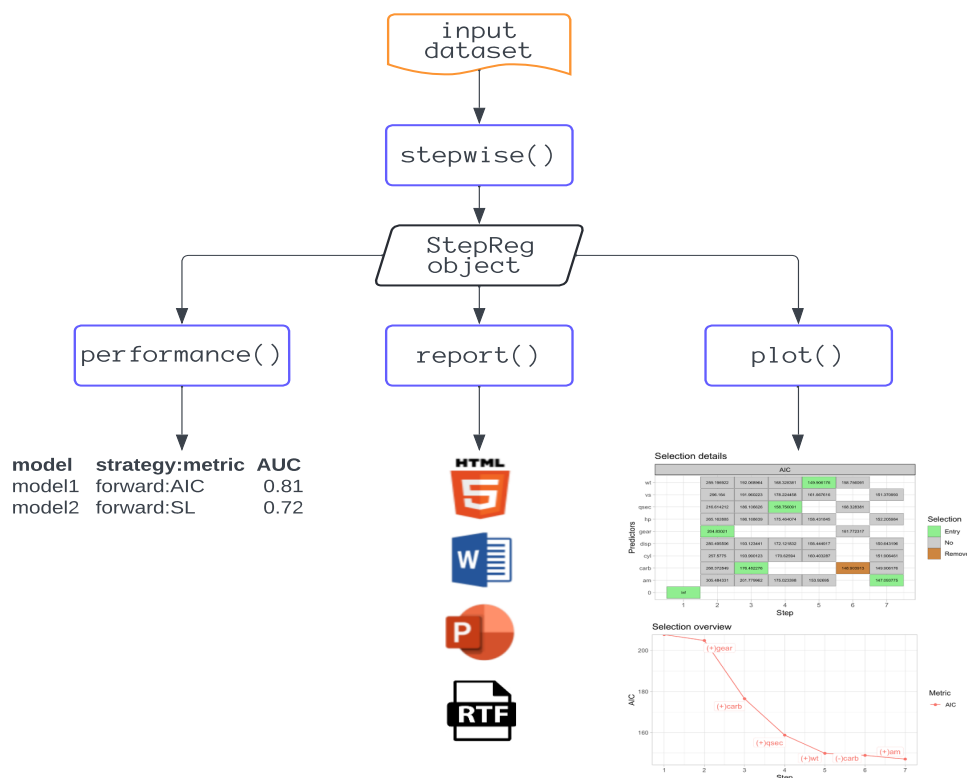


Figure 2: StepReg workflow overview

3.2.1 stepwise()

- Purpose

Runs the stepwise selection procedure and returns a "StepReg" object containing the selection path and final model(s).

- Key arguments

Model and data specification

formula: Model formula specifying a single response, multiple dependent variables, or survival outcomes on the left and predictors on the right (see Table 1).

data: data.frame containing the variables referenced in formula.

type (linear): Regression family. Options include "linear" (linear/Gaussian), "logit" (logistic/binomial), "poisson" (Poisson), "cox" (Cox proportional hazards), "gamma" (Gamma), and "negbin" (negative binomial).

Variable selection strategy

strategy: Variable selection strategy. Options include "forward" (forward selection), "backward" (backward elimination), "bidirection" (bidirectional selection), and "subset" (best-subset selection).

Selection criterion

metric: Model selection criterion. Supported values are "AIC", "AICc", "BIC", "SBC", "CP", "HQ", "adjRsqr", "SL", "IC(3/2)", and "IC(1)". Note: Definitions can differ across sources; the exact formulas used in StepReg are enumerated in Supplementary Table 3.

Entry/stay controls (for SL-based metric)

sle (0.15): Significance level threshold used for forward and bidirectional selection. A predictor is added to the model if its p-value is below this cutoff.

sls (0.15): Significance level threshold used for backward and bidirectional selection. A predictor remains in the model if its p-value is below this cutoff.

Hypothesis test options

Table 1: Example formulas implemented in StepReg

Formulas	Descriptions
$y \sim x1 + x2$	Specify multiple predictors x1 and x2
$y \sim .$	Specify all predictors
$y \sim . - x1$	Specify all predictors except x1
$y \sim x1 * x2$	Specify main effects x1, x2, and their interaction
$y \sim x1:x2$	Specify the continuous-nested-within-class effects x1:x2
$\text{cbind}(y1, y2) \sim .$	Specify multiple response variables y1 and y2
$y \sim . + 0$ or $y \sim . - 1$	Fits model without an intercept (The Cox model does not use an explicit intercept, adding + 0, - 1, or + 1 to the formula will not change the results)
$\text{Surv}(\text{time}, \text{status}) \sim . + \text{strata}(x1)$	Cox regression with stratification variable x1

`test_method_linear`: For multivariate linear models: "Pillai" (default), "Wilks", "Hotelling-Lawley", "Roy". For univariate linear regression, the F statistic is used automatically.

`test_method_glm`: For GLMs: "Rao" (default) or "LRT". (Only "Rao" is available with the "subset" strategy.)

`test_method_cox`: Ties handling for Cox: "efron" (default), "breslow", "exact".

Constraints and robustness

`include` (NULL): Character vector of predictors that must be included in all models, namely, forced-in terms.

`tolerance` (1e-07): Threshold for multicollinearity checks; smaller values impose stricter screening.

`weight` (NULL): Optional numeric vector of observation weights ranging from 0 to 1.

Randomized forward selection, data splitting, and reproducibility

`features_ratio` (1): Proportion of remaining predictors sampled at random at each forward step (randomized forward selection). Set to 1 for standard forward selection.

`test_ratio` (0): Proportion of data reserved as a test set (e.g., 0.30). When > 0, `performance()` reports training/test performance metrics to support out-of-sample evaluation.

`seed` (123): Random seed used when `test_ratio` > 0 and/or `features_ratio` < 1.

Output control

`best_n` (3): Maximum number of top models displayed at each model size in best-subset search.

`num_digits` (6): Number of decimal places for printed results.

- Return value of `stepwise()`

`stepwise()` returns an S3 object of class "StepReg", a named list that records the analysis specification and results. The key components are:

argument

A `data.frame` summarizing all user-supplied settings (formula, type, strategy, metric, seeds, etc.).

variable

A `data.frame` describing all variables referenced in the model, including names, storage types, and roles (Dependent/Independent).

performance

A `data.frame` containing model-level performance metrics for each strategy-by-criterion. The model evaluation metrics vary by regression type:

For linear, Poisson, Gamma, and negative binomial regression models, the key performance indicators for both training and test data sets include adjusted R-squared (`adjR-squared_train` / `adjR-squared_test` applicable only to linear regression), mean squared error (`mse_train` / `mse_test`), and mean absolute error (`mae_train` / `mae_test`). A higher adjusted R-squared and lower MSE or MAE generally indicate a better model fit, while a large discrepancy between training and test metrics often signals overfitting, suggesting the model may not generalize well to unseen data. Note: If the adjusted R-squared for the test set exceeds 1, this is typically due to the small size of the test sample, which can produce unstable or uninterpretable values for this statistic.

For logistic regression models, the key performance indicators for both training and test data sets include accuracy (`accuracy_train / accuracy_test`), area under the receiver operating characteristic (ROC) curve (AUC) (`auc_train / auc_test`), and log loss (`log_loss_train / log_loss_test`). Higher accuracy and AUC values, combined with lower log loss, generally indicate stronger predictive performance. As with other models, substantial gaps between training and testing results suggest potential overfitting and warrant further investigation.

For Cox proportional hazards regression models, the primary performance indicators are concordance index (`c_index_train / c_index_test`) and time-dependent area under the curve (`auc_sh`). Higher values indicate stronger discrimination between risk groups. The c-index ranges from 0.5 (random prediction) to 1.0 (perfect prediction). Values below 0.5 typically suggest reversed risk direction or miscoding of events. In such cases, verify event coding and ensure that the scoring system aligns with the interpretation that higher scores correspond to higher risk.

overview

A nested list (by strategy and metric) giving the step-by-step selection path: which variables entered/left at each step, along with the corresponding criterion values (e.g., AIC, BIC, SBC).

detail

A nested list (by strategy and metric) with candidate step diagnostics: variables evaluated at each step, test statistics, p values, and selection decisions.

fitted models (within the strategy-metric hierarchy)

For each combination of selection strategy (e.g., "forward", "backward", "bidirection", "subset") and metric (e.g., "AIC", "BIC", "SBC"), the object stores the corresponding fitted model. These can be analyzed with standard S3 generics, `summary()`, `anova()`, `plot()`, or `coef()`, which dispatch to the underlying model class (e.g., `coxph()`, `lm()`, `glm()`). Users can also access components directly, e.g., `result$forward$AIC$coefficients`. Note: "coxph" objects in the [survival](#) package include detailed summaries (hazard ratios, standard errors, test statistics), whereas `lm()` / `glm()` display only basic coefficients by default. Use `summary()` to obtain standard errors, t/z statistics, p values, and R-squared.

- Printing behavior and downstream use

By default, a "StepReg" object prints a concise summary (e.g., selected model(s), criterion values, path length) while suppressing verbose components—argument, variable, performance, overview, and detail—from the console. These elements remain in the object for programmatic access and are utilized by the downstream functions such as `plot()`, `performance()`, and `report()`.

3.2.2 plot()

- Purpose

Creates visualizations of the variable selection process from a "StepReg" object.

- Key arguments

x: A "StepReg" object.

strategy: Which strategy to display (e.g., "forward", "backward", "bidirection", "subset")

process: Selection view, one of "detail" (step-by-step path) or "overview" (summary of selected variables).

Outputs clear graphics for either the detailed path or a concise overview of included terms.

3.2.3 performance()

- Purpose

Produces a summary table of model performance across strategy-by-criterion combinations.

- Key arguments

x: A "StepReg" object.

If a hold-out split was specified, both training and test metrics are reported.

3.2.4 report()

- Purpose

Generates a formatted report from a "StepReg" object.

- Key arguments

x: A "StepReg" object.

name: Base file name for the report.

format: One or more output formats: "html", "docx", "pptx", and "rtf".

Reports include selection summaries, performance tables, and (optionally) figures.

3.2.5 StepRegGUI() in the companion R package StepRegShiny

It launches the Shiny GUI for point and click use. An online version is available at: <https://junhuili1017.shinyapps.io/StepRegShiny/>.

Together, these interfaces provide a concise print-view for readability and full programmatic access for analysis, visualization, benchmarking, and reporting. Comprehensive documentation, including vignettes and a user manual, is available with the package on CRAN: <https://CRAN.R-project.org/package=StepReg>.

3.3 Example usage

3.3.1 Data and preprocessing

This example used the lung dataset from the **StepReg** R package. We analyzed complete cases (n = 167) from patients with advanced lung cancer. The variables are:

- time: Overall survival time (days).
- status: Event status (1 = censored, 2 = death).
- age: Age (years).
- sex: Biological gender (1 = male, 2 = female).
- ph.ecog: ECOG performance score (0 = asymptomatic; 1 = symptomatic but completely ambulatory; 2 = in bed <50% of the day; 3 = in bed > 50% of the day but not bedbound; 4 = bedbound).
- ph.karno: Karnofsky performance score rated by physician (0 = worst, 100 = best).
- pat.karno: Karnofsky performance score rated by patient (0 = worst, 100 = best).
- meal.cal: Calories consumed at meals (kcal).
- wt.loss: Weight loss in last six months (pounds).
- inst: Institution code.

Preprocessing: We converted sex to a factor with levels "male" and "female" and excluded observations with missing data.

```
library(StepReg)
data(lung)
lung$sex <- factor(lung$sex, levels = c(1, 2), labels = c("male", "female"))
lung <- na.omit(lung)
str(lung)

#> 'data.frame':   167 obs. of  10 variables:
#> $ inst      : num  3 5 12 7 11 1 7 6 12 22 ...
#> $ time      : num  455 210 1022 310 361 ...
#> $ status    : num  2 2 1 2 2 2 2 2 2 ...
#> $ age       : num  68 57 74 68 71 53 61 57 57 70 ...
#> $ sex       : Factor w/ 2 levels "male","female": 1 1 1 2 2 1 1 1 1 1 ...
#> $ ph.ecog   : num  0 1 1 2 2 1 2 1 1 1 ...
#> $ ph.karno  : num  90 90 50 70 60 70 70 80 80 90 ...
#> $ pat.karno : num  90 60 80 60 80 80 70 80 70 100 ...
```

```
#> $ meal.cal : num 1225 1150 513 384 538 ...
#> $ wt.loss : num 15 11 0 10 1 16 34 27 60 -5 ...
#> - attr(*, "na.action")= 'omit' Named int [1:61] 1 3 5 12 13 14 16 20 23 25 ...
#> ..- attr(*, "names")= chr [1:61] "1" "3" "5" "12" ...
```

3.3.2 Model specification and selection settings

We fit a Cox model using forward selection under two information criteria, AICc and SL with entry threshold $sle = 0.05$. Data are randomly split into 70% training / 30% test ($test\_ratio = 0.3$). We constructed the survival response as `Surv(time, status)` and treated sex as a labeled factor; all remaining variables were considered as candidate predictors. See Table 1 for formula guidance and Section 3.2 (Core functions and parameter definitions) for argument details (e.g., `strategy`, `metric`, `features_ratio` for randomized forward selection, and `test_ratio` for train–test splitting).

3.3.3 Procedure

Following the [StepReg](#) workflow in Figure 2, we called `stepwise()` to run forward selection and obtain a "StepReg" object, employed `plot()` to visualize the selection path, used `performance()` to compute training/test metrics for the 0.7/0.3 split, and optionally generated a formatted report with `report()`.

```
result <- stepwise(formula = Surv(time, status) ~ .,
                  data = lung,
                  type = "cox",
                  strategy = "forward",
                  metric = c("AICc", "SL"),
                  sle = 0.05,
                  test_ratio = 0.3)

result

#> $forward
#> $forward$AICc
#> Call:
#> coxph(formula = Surv(time, status) ~ ph.ecog + sex + wt.loss +
#>       inst + ph.karno, data = data, weights = NULL, method = "efron")
#>
#>              coef exp(coef) se(coef)      z      p
#> ph.ecog      1.024028  2.784387  0.287734  3.559 0.000372
#> sexfemale -0.589911  0.554377  0.239437 -2.464 0.013749
#> wt.loss     -0.017024  0.983120  0.008828 -1.928 0.053799
#> inst        -0.030890  0.969582  0.015778 -1.958 0.050256
#> ph.karno     0.022345  1.022597  0.014420  1.550 0.121227
#>
#> Likelihood ratio test=23.03 on 5 df, p=0.0003326
#> n= 117, number of events= 86
#>
#> $forward$SL
#> Call:
#> coxph(formula = Surv(time, status) ~ ph.ecog + sex, data = data,
#>       weights = NULL, method = "efron")
#>
#>              coef exp(coef) se(coef)      z      p
#> ph.ecog      0.4463    1.5625  0.1490  2.996 0.00274
#> sexfemale -0.5212    0.5938  0.2354 -2.214 0.02681
#>
#> Likelihood ratio test=13.81 on 2 df, p=0.001001
#> n= 117, number of events= 86
```

3.3.4 Outputs

The "StepReg" object consists of a fitted Cox model for each strategy-by-criterion combination. With sex coded as a factor, treatment contrasts yield a dummy variable (e.g., `sexfemale`) representing the

effect of the female level relative to the male reference. For each selected model, stored results include coefficient estimates (`coef`), hazard ratios (`exp(coef)`), standard errors, z-statistics, p-values, and likelihood-ratio tests, enabling step-by-step review of the selection path and the final model summary. Standard S3 generics (e.g., `summary()`, `coef()`, `anova()`) work on the fitted objects. Individual components are also accessible via the `$` operator.

```
coef(result$forward$SL)

#>   ph.ecog  sexfemale
#> 0.4462914 -0.5212152

result$forward$AICc$coefficients

#>   ph.ecog  sexfemale  wt.loss      inst  ph.karno
#> 1.02402770 -0.58991071 -0.01702409 -0.03088999 0.02234546
```

3.3.5 Performance

`performance()` produces a comparison table across strategy–criterion pairs. For Cox models, it reports training and test concordance (c-index) and time-dependent AUC, enabling side-by-side evaluation.

```
performance(x=result)

#>                                     model
#> 1 Surv(time, status) ~ ph.ecog + sex + wt.loss + inst + ph.karno
#> 2                               Surv(time, status) ~ ph.ecog + sex
#>   strategy:metric c-index_train c-index_test auc_sh
#> 1   forward:AICc      0.6429      0.4355 0.6676
#> 2   forward:SL       0.6384      0.4113 0.6126
```

Practical notes: As a quality check, we verified event coding (death = 1, censored = 0) and confirmed that discrimination used the Cox linear predictor (higher = higher hazard); reversing either can spuriously drive the test-set C-index below 0.5. This example is for illustrative purposes only—we do not interpret test-set metrics further. In real analyses, if the test set C-index ≈ 0.5 (no discrimination) or < 0.5 , we recommend rechecking event coding and risk-score direction; quantifying uncertainty with a bootstrap confidence interval (CI); assessing covariate and event-time distribution shift between training and test sets; running k-fold cross-validation or bootstrap to evaluate stability/overfitting; reviewing proportional-hazards and calibration diagnostics; and, under heavy censoring, also examining time-dependent AUC or the censoring-adjusted C-statistic (Uno et al., 2011). If issues persist, consider model simplification or penalization.

3.3.6 Visualization

To illustrate the selection process, we used the `StepReg::plot()` and arranged panels with `cowplot::plot_grid()` to produce two views (Figure 3): (i) a step-by-step path highlighting the top candidate at each step under both AICc and SL (selected entries shaded in green), and (ii) an overview summarizing the variables ultimately retained, with AICc and p-values (SL) tracked across steps. In the forward strategy, at each step the remaining predictors are evaluated, and the one that yields the lowest AICc or the smallest p-value for SL is considered for inclusion. A variable is accepted only if the AICc value decreases relative to the previous step or if the p-value for SL is below the entry threshold `sle`. If no candidate meets these conditions, the procedure terminates and the current model is retained.

```
library(cowplot)
plot_list <- setNames(
  lapply(c("forward"), function(i){
    setNames(
      lapply(c("detail", "overview"), function(j){
        plot(result, strategy=i, process=j)
      }),
      c("detail", "overview")
    )
  })
)
```

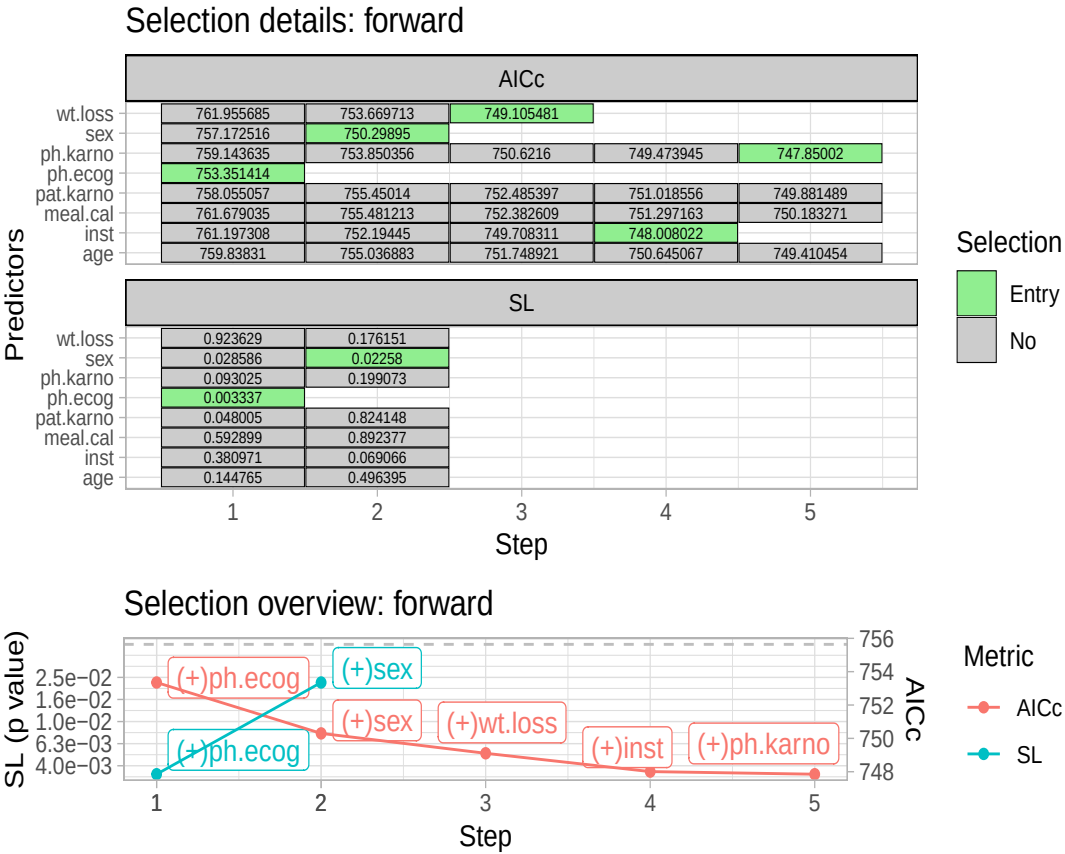


Figure 3: Visualization of variable selection: detailed and overview perspectives

```
  }),  
  c("forward")  
)  
cowplot::plot_grid(plotlist = plot_list$forward, ncol = 1, rel_heights = c(2, 1))
```

3.3.7 Reporting

The report() function can export results in multiple formats such as "rtf", "docx", "html", and "pptx". For example,

```
report(x=result, name = "results", format = c("html", "docx"))
```

creates results.html and results.docx (file extensions are added automatically). Replace "results" with any base name as needed.

3.4 Benchmark using SAS

We evaluated StepReg’s performance by comparing its outputs with those of SAS using multiple public datasets. For most cases, the information criterion values calculated at each step and the final models generated by StepReg were consistent with those from SAS across different strategies and metrics. However, it is important to note that p-value calculations for variable inclusion in Cox models may differ slightly between StepReg and SAS for variable entry. This discrepancy arises because survival::coxph() utilizes the likelihood ratio test (LRT), whereas SAS employs the score chi-square method (SAS Institute Inc., 2019c). Despite this difference, the final models remained consistent between StepReg and SAS across the test datasets (see Supplementary Table 5). These results confirm StepReg’s accuracy and reliability as a tool for stepwise regression analysis.

4 Discussion

4.1 StepReg enables comprehensive stepwise regression analyses

StepReg provides a unified and flexible framework for stepwise regression. It supports six commonly used regression families (type), four variable selection strategies (forward, backward, bidirectional, best-subset), and a broad set of information criteria (metrics). Unlike tools that run a single strategy–criterion combination at a time, **StepReg** lets users specify multiple strategies and criteria in one call, enabling efficient, side-by-side comparisons and reducing boilerplate code. To improve robustness, it offers data splitting via `test_ratio` for out-of-sample evaluation and more reliable post-selection summaries and an optional randomized forward selection mode via `features_ratio` to mitigate overfitting and stabilize choices.

For interpretability and diagnostics, **StepReg** performs basic multicollinearity checks (tolerance) and records the full selection path. The `plot()` function visualizes the path and selected variables; the `performance()` function summarizes train/test performance across all strategy-by-criterion combinations; and the `report()` function produces HTML/DOCX/RTF/PPTX reports for easy sharing.

To enhance accessibility, an interactive Shiny-based GUI is accessible through the companion package **StepRegShiny**, enabling point-and-click analyses for non-programmers.

In summary, by integrating broad model-family coverage, multiple strategies and criteria within a unified workflow, along with features such as data splitting, randomized forward selection, visualization, and automated reporting, **StepReg** makes stepwise regression more efficient, transparent, and accessible than existing tools.

4.2 Best practices and considerations for using StepReg

4.2.1 Scope of use

StepReg provides a flexible framework for stepwise regression suitable for both exploratory analysis and predictive modeling. Effective use depends on appropriate parameterization and careful interpretation.

4.2.2 Parameterization

- Variable selection strategy. Users should choose the strategy to match data and goals. Forward selection is efficient when starting from many candidate predictors; backward elimination is appropriate when a well-justified, estimable full model is available and the number of predictors is modest relative to the number of observations; bidirectional selection mixes additions and deletions but remains path-dependent; best-subset evaluates all subsets within the specified scope and is computationally intensive (roughly 2^p).
- Information criteria. Information criteria (e.g., AIC, BIC, and SBC) generally balance fit and complexity for prediction; SL rules align with hypothesis-driven screening but warrant caution in interpretation. **StepReg** allows multiple strategies and criteria in a single call (strategy, metric) to enable side-by-side comparisons.
- A priori inclusion. Use `include` to retain domain-critical predictors irrespective of selection statistics.

4.2.3 Limitations and mitigation

Models obtained via stepwise procedures are not suitable for classical post-selection inference without adjustment; the search can induce overfitting, spurious associations, and biased estimates. **StepReg** provides two built-in mitigations: data splitting (`test_ratio`) to select on a training subset and refit/evaluate on a hold-out set, and randomized forward selection (`features_ratio`) to sample a subset of candidates at each step, which can stabilize choices and reduce overfitting. When formal inference is the goal, complementary approaches—penalized regression, bootstrap or post-selection inference methods, and cross-validated regularization—should be considered.

4.2.4 Performance and scalability

StepReg runs efficiently on small to moderate problems; wall time increases with the number of candidate predictors p , the complexity of the model family (e.g., Cox, negative binomial), and the

number of strategy-by-criterion combinations evaluated. We omit parallel execution to avoid extra dependencies and variability, though it could reduce wall time in high-throughput settings. Tree-based and boosting methods are out of scope for this release; future versions may expose them through connectors to higher-level ecosystems (e.g., [tidymodels](#)).

4.2.5 Summary

[StepReg](#) is well suited for exploratory analysis, enabling rapid screening and ranking of candidate predictors, as well as for predictive modeling, where the priority is out-of-sample performance rather than coefficient interpretation. In the latter scenario, it provides a practical, unified workflow for selecting and comparing high-performing models. With thoughtful choices of strategy and criterion, principled inclusion of a priori predictors, and use of mitigations such as data splitting and randomized forward selection, [StepReg](#) can be applied effectively and responsibly across diverse analytic contexts. Although very large or highly complex problems may be computationally demanding, the current design balances functionality and usability while remaining readily extensible. While the current release standardizes stepwise selection for six likelihood-based families, future development will emphasize interoperability—exploring a [tidymodels](#) integration to expose alternative model engines while retaining [StepReg](#)'s standardized interface (formula syntax, information criteria, visualization, reporting).

4.3 Adoption and impact

[StepReg](#) has been publicly available on CRAN since 2019 and has been cited by over 80 research studies across diverse fields. It has been especially valuable in medicine and health sciences, where Cox and logistic regression are widely used for prognostic or predictive modeling and biomarker discovery. For example, [Flammia et al. \(2024\)](#) used [StepReg](#) to develop a Cox model predicting chronic kidney disease upstaging after robot-assisted partial nephrectomy, and [Pigeyre et al. \(2023\)](#) applied forward-selection logistic regression to identify 25 serum biomarkers for subtyping type 2 diabetes. Beyond clinical applications, [StepReg](#) has been used in other domains. [Yoon et al. \(2022\)](#) employed stepwise linear regression to assess how landscape factors reduce PM2.5 levels. Overall, [StepReg](#)'s comprehensive regression toolkit supports a wide range of scientific inquiries, making it a practical resource for advanced analyses across disciplines.

5 Acknowledgments

We thank the Department of Molecular, Cell, and Cancer Biology (MCCB) at UMass Chan Medical School for internal funding and Dr. Haibo Liu for his helpful suggestions during manuscript revision.

References

- A. A. Al-Subaihi. Variable selection in multivariable regression using SAS/IML. *Journal of Statistical Software*, 7(12):1–20, 2002. doi: 10.18637/jss.v007.i12. [p188, 190, 191]
- T. W. Anderson. *An Introduction to Multivariate Statistical Analysis*. John Wiley & Sons, 3rd edition, 2003. ISBN 978-0-471-36091-9. URL <https://www.wiley.com/en-us/An+Introduction+to+Multivariate+Statistical+Analysis%2C+3rd+Edition-p-9780471360919>. [p191]
- D. Attali. *shinyjs: Easily Improve the User Experience of Your shiny Apps in Seconds*, 2022. URL <https://CRAN.R-project.org/package=shinyjs>. R package version 2.1.0. [p191]
- D. Attali et al. *shinycssloaders: Add Loading Animations to a 'shiny' Output While It's Recalculating*, 2024. URL <https://CRAN.R-project.org/package=shinycssloaders>. R package version 1.1.0. [p191]
- P. C. Austin and J. V. Tu. Automated variable selection methods for logistic regression produced unstable models for predicting acute myocardial infarction mortality. *Journal of Clinical Epidemiology*, 57(11):1138–1146, 2004. doi: 10.1016/j.jclinepi.2004.04.003. [p188]
- T. L. based on Fortran code by Alan Miller. *leaps: Regression Subset Selection*, 2024. URL <https://CRAN.R-project.org/package=leaps>. R package version 3.2. [p190]
- M. Baudin et al. *OpenTURNS: An Industrial Software for Uncertainty Quantification in Simulation*, book section 1–38. Springer-Verlag, Cham, 2016. ISBN 978-3-319-11259-6. doi: 10.1007/978-3-319-11259-6_64-1. [p190]

- S. Bonnini and M. Borghesi. Relationship between mental health and socio-economic, demographic and environmental factors in the COVID-19 lockdown period—a multivariate regression analysis. *Mathematics*, 10(18):3237, 2022. doi: 10.3390/math10183237. [p190]
- P. F. P. Brandão-Dias et al. Suspended materials affect particle size distribution and removal of environmental DNA in flowing waters. *Environmental Science & Technology*, 57(35):13161–13171, 2023. doi: 10.1021/acs.est.3c02638. URL <https://www.ncbi.nlm.nih.gov/pubmed/37610829>. [p188]
- K. P. Burnham and D. R. Anderson. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Springer, 2nd edition, 2002. doi: 10.1007/b97636. [p188]
- S. A. D. Caires et al. Predicting soil depth in a humid tropical watershed: A comparative analysis of best-fit regression and geospatial models. *Catena*, 222(1), 2023. doi: 10.1016/j.catena.2022.106843. [p188]
- W. Chang. *shinythemes: Themes for shiny*, 2021. URL <https://CRAN.R-project.org/package=shinythemes>. R package version 1.2.0. [p191]
- W. Chang et al. *shiny: Web Application Framework for R*, 2024. URL <https://shiny.posit.co/>. R package version 1.9.1.9000. [p191]
- X. Chen, J. M. Klusowski, Y. S. Tan, and C. Yu. Revisiting randomization in greedy model search. <https://arxiv.org/abs/2506.15643>, 2025. arXiv preprint arXiv:2506.15643. [p189]
- D. R. Cox. Regression models and life-tables (with discussion). *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(2):187–220, 1972. doi: 10.1111/j.2517-6161.1972.tb00899.x. [p188]
- R. B. Darlington. Multiple regression in psychological research and practice. *Psychological Bulletin*, 69(3):161–182, 1968. doi: 10.1037/h0025471. [p188, 190, 191]
- A. C. Davison and D. V. Hinkley. *Bootstrap Methods and their Application*. Cambridge University Press, 1997. doi: 10.1017/CBO9780511802843. [p190]
- S. A. De Caires et al. Multivariate geospatial analysis for predicting soil variability along a toposequence of a watershed in the humid tropics. *CATENA*, 210:105919, 2022. doi: 10.1016/j.catena.2021.105919. [p188]
- N. R. Draper and H. Smith. *Applied Regression Analysis*. John Wiley & Sons, New York, 3rd edition, 1998a. doi: 10.1002/9781444316650. [p191]
- N. R. Draper and H. Smith. *Applied Regression Analysis*. Wiley, 3rd edition, 1998b. doi: 10.1002/9781118625590. [p188]
- P. Ebrahimi et al. A field-based thickness measurement dataset of fallout pyroclastic deposits in the peri-volcanic areas of Campania region (Italy): Statistical combination of different predictions for spatial thickness estimation. *Earth System Science Data*, 16(9):4161–4188, 2024. doi: 10.5194/essd-16-4161-2024. URL <https://essd.copernicus.org/articles/16/4161/2024/>. [p188]
- B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC, 1993. doi: 10.1201/9780429246593. [p188]
- R. S. Flammia et al. Development and internal validation of a nomogram predicting 3-year chronic kidney disease upstaging following robot-assisted partial nephrectomy. *International Urology and Nephrology*, 56(3):913–921, 2024. doi: 10.1007/s11255-023-03832-6. URL <https://www.ncbi.nlm.nih.gov/pubmed/37848745>. [p200]
- L. Flom and D. L. Cassell. Stopping stepwise: Why stepwise and similar selection methods are bad, and what you should use. *NorthEast SAS Users Group Inc 20th Annual Conference*, 11, 2007. URL <https://www.lexjansen.com/nesug/nesug07/sa/sa07.pdf>. [p188]
- E. I. George and R. E. McCulloch. Variable selection via Gibbs sampling. *Journal of the American Statistical Association*, 88(423):881–889, 1993. doi: 10.2307/2290777. [p188]
- D. Gohel and P. Skintzos. *flextable: Functions for Tabular Reporting*, 2024. URL <https://CRAN.R-project.org/package=flextable>. R package version 0.9.6. [p191]
- T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009. doi: 10.1007/978-0-387-84858-7. [p188, 190]

- T. Hastie, R. Tibshirani, and R. J. Tibshirani. Best subset, forward stepwise or lasso? analysis and recommendations based on extensive comparisons. *Statistical Science*, 35(4):579–592, 2020. doi: 10.1214/19-STS733. [p189]
- A. Hebbali. *blorr: Tools for Developing Binary Logistic Regression Models*, 2020. URL <https://CRAN.R-project.org/package=blorr>. R package version 0.3.0.9000. [p190]
- A. Hebbali. *olsrr: Tools for Building OLS Regression Models*, 2024. URL <https://CRAN.R-project.org/package=olsrr>. R package version 0.6.0.9000. [p190]
- W.-M. Ho et al. A longer time to relapse is associated with a larger increase in differences between paired primary and recurrent IDH wild-type glioblastomas at both the transcriptomic and genomic levels. *Acta Neuropathologica Communications*, 12(1):77, 2024. doi: 10.1186/s40478-024-01790-3. URL <https://www.ncbi.nlm.nih.gov/pubmed/38762464>. [p188]
- R. R. Hocking. A biometrics invited paper. the analysis and selection of variables in linear regression. *Biometrics*, 32(1):1–49, 1976. doi: 10.2307/2529336. [p190]
- D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant. *Applied Logistic Regression*. Wiley, 3rd edition, 2013. doi: 10.1002/9781118548387. [p188]
- F.-C. Hu. *My.stepwise: Stepwise Variable Selection Procedures for Regression Analysis*, 2017. URL <https://CRAN.R-project.org/package=My.stepwise>. R package version 0.1.0. [p190]
- C. M. Hurvich and C.-L. Tsai. Regression and time series model selection in small samples. *Biometrika*, 76:297–307, 1989. doi: 10.1093/biomet/76.2.297. [p188, 190, 191]
- C. M. Hurvich et al. Smoothing parameter selection in nonparametric regression using an improved Akaike information criterion. *Journal of the Royal Statistical Society Series B*, 60:271–293, 1998. doi: 10.1111/1467-9868.00125. [p191]
- A. J. Izenman. *Multivariate Regression*, book section Multivariate Regression. Springer-Verlag, New York, 2013. ISBN 978-0-387-78188-4. doi: 10.1007/978-0-387-78189-1_6. [p190]
- G. G. Judge et al. *The Theory and Practice of Econometrics, 2nd Edition*. John Wiley & Sons, 1985. ISBN 978-0-471-89530-5. doi: 10.1007/978-0-387-21706-2. URL <https://www.wiley.com/en-us/The+Theory+and+Practice+of+Econometrics%2C+2nd+Edition-p-9780471895305>. [p188, 190, 191]
- C. Kleiber and A. Zeileis. *Applied Econometrics with R*. Springer-Verlag, New York, 2008. doi: 10.1007/978-0-387-77318-6. URL <https://CRAN.R-project.org/package=AER>. [p191]
- E. T. Lee. A computer program for linear logistic regression analysis. *Computer Programs in Biomedicine*, 4:80–92, 1974. doi: 10.1016/0010-468X(74)90011-7. [p191]
- M. S. Lewis-Beck. Stepwise regression: A caution. *Political Methodology*, 5(2):213–240, 1978. doi: 10.1111/j.1540-5907.1978.tb00001.x. [p188]
- C. Makepeace et al. Evaluation of nonhazardous lime requirement estimation methods for Oregon soils. *Soil Science Society of America Journal*, 86(5):1354–1367, 2022. doi: 10.1002/saj2.20466. [p188]
- C. L. Mallows. Some comments on CP. *Technometrics*, 15(4):661–675, 1973. doi: 10.2307/1267380. [p190]
- S. Matsui et al. Genomic characterization of multiple clinical phenotypes of cancer using multivariate linear regression models. *Bioinformatics*, 23(6):732–738, 2007. doi: 10.1093/bioinformatics/btl663. URL <https://www.ncbi.nlm.nih.gov/pubmed/17237045>. [p190]
- A. McLeod et al. *bestglm: Best Subset GLM and Regression Utilities*, 2020. URL <https://CRAN.R-project.org/package=bestglm>. R package version 0.37.3. [p190]
- S. Menard. *Applied Logistic Regression Analysis*. Sage Publications, 2nd edition, 2002. doi: 10.4135/9781412983433. [p188]
- L. Mentch and S. Zhou. Randomization as regularization: A degrees of freedom explanation for random forest success. *Journal of Machine Learning Research*, 21(171):1–36, 2020. doi: 10.48550/arXiv.1911.00190. URL <http://jmlr.org/papers/v21/19-905.html>. [p189]
- T. J. Mitchell and J. J. Beauchamp. Bayesian variable selection in linear regression. *Journal of the American Statistical Association*, 83(404):1023–1032, 1988. doi: 10.2307/2290129. [p188]

- V. Y. Naei et al. Spatial proteomic profiling of tumor and stromal compartments in non-small-cell lung cancer identifies signatures associated with overall survival. *Clinical & Translational Immunology*, 13(7):e1522, 2024. doi: 10.1002/cti2.1522. URL <https://www.ncbi.nlm.nih.gov/pubmed/39026528>. [p188]
- R. B. O'Hara and M. J. Sillanpää. A review of Bayesian variable selection methods: What, how, and which. *Bayesian Analysis*, 4(1):85–118, 2009. doi: 10.1214/09-BA403. [p188]
- F. Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12 (85):2825–2830, 2011. URL <http://jmlr.org/papers/v12/pedregosa11a.html>. [p190]
- J. Peng et al. Regularized multivariate regression for identifying master predictors with application to integrative genomics study of breast cancer. *The Annals of Applied Statistics*, 4(1):53–77, 2010. doi: 10.1214/09-AOAS271SUPP. URL <https://www.ncbi.nlm.nih.gov/pubmed/24489618>. [p190]
- M. Pigeyre et al. Identifying blood biomarkers for type 2 diabetes subtyping: A report from the ORIGIN trial. *Diabetologia*, 66(6):1045–1051, 2023. doi: 10.1007/s00125-023-05887-7. URL <https://www.ncbi.nlm.nih.gov/pubmed/36854916>. [p200]
- 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/>. [p189, 191]
- SAS Institute Inc. *The PHREG Procedure*. Cary, NC, 2019a. URL https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.3/statug/statug_phreg_toc.htm. [p190]
- SAS Institute Inc. *The HPGENSELECT Procedure*. Cary, NC, 2019b. URL https://documentation.sas.com/doc/en/statcdc/14.3/statug/statug_hpgenselect_overview01.htm. [p190]
- SAS Institute Inc. *The LOGISTIC Procedure*. Cary, NC, 2019c. URL https://documentation.sas.com/doc/en/statug/14.3/statug_logistic_toc.htm. [p190, 198]
- T. Sawa. Information criteria for discriminating among alternative regression models. *Econometrica*, 46 (6):1273–1291, 1978. doi: 10.2307/1913828. [p188]
- G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. doi: 10.1214/aos/1176344136. [p190]
- S. Seabold and J. Perktold. *statsmodels: Econometric and statistical modeling with Python*. 2010. doi: 10.25080/Majora-92bf1922-011. [p190]
- H. Seter et al. How do drivers' attitudes to low emission zones change after experiencing it? a pilot study in Norway. *Transportation Research Interdisciplinary Perspectives*, 22:100934, 2023. doi: 10.1016/j.trip.2023.100934. [p188]
- Seyedaliakbar et al. Impact of energy consumption and economic growth on CO2 emission using multivariate regression. *Energy Strategy Reviews*, 26:100428, 2019. doi: 10.1016/j.esr.2019.100428. URL <https://www.sciencedirect.com/science/article/pii/S2211467X19301208>. [p190]
- J. Shao. Linear model selection by cross-validation. *Journal of the American Statistical Association*, 88: 486–494, 1993. doi: 10.2307/2290328. [p190]
- J. Shao. An asymptotic theory for linear models selection. *Statistica Sinica*, 7:221–264, 1997. doi: 10.5705/ss.202011.0055. [p190]
- G. Smith. Step away from stepwise. *Journal of Big Data*, 5(1):1–12, 2018. doi: 10.1186/s40537-018-0143-6. [p188]
- M. Stone. Cross-validated choice and assessment of statistical predictions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(2):111–147, 1974. doi: 10.1111/j.2517-6161.1974.tb00994.x. [p188]
- H. J. Tadros et al. Assessment of parental decision making in congenital heart disease, cardiomyopathy, and heart transplantation: An observational study analyzing decisional characteristics and preferences. *Archives of Disease in Childhood*, 108(8):641–646, 2023. doi: 10.1136/archdischild-2022-324373. URL <https://www.ncbi.nlm.nih.gov/pubmed/36732035>. [p188]
- T. M. Therneau. *A Package for Survival Analysis in R*, 2024. URL <https://CRAN.R-project.org/package=survival>. R package version 3.7-0. [p191]
- T. M. Therneau and P. M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer-Verlag, New York, 2000. ISBN 0-387-98784-3. doi: 10.1007/978-1-4757-3294-8. [p188, 191]

- R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996. ISSN 00359246. URL <http://www.jstor.org/stable/2346178>. [p188]
- G. Tintner. Some applications of multivariate analysis to economic data. *Journal of the American Statistical Association*, 41(236):472–500, 1964. doi: 10.1080/01621459.1946.10501892. [p190]
- H. Uno, T. Cai, M. J. Pencina, R. B. D’Agostino, and L. J. Wei. On the C-statistics for evaluating overall adequacy of risk prediction procedures with censored survival data. *Statistics in Medicine*, 30:1105–1117, 2011. doi: 10.1002/sim.4154. [p197]
- W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Fourth edition, 2002. ISBN 0-387-95457-0. URL <https://www.stats.ox.ac.uk/pub/MASS4/>. [p190, 191]
- A. R. Vijayakumar. Stepwise regression GitHub repository, 2019. URL <https://github.com/AakkashVijayakumar/stepwise-regression>. Accessed: 2024-10-07. [p190]
- M. J. Whittingham et al. Why do we still use stepwise modelling in ecology and behaviour? *Journal of Animal Ecology*, 75(5):1182–1189, 2006. doi: 10.1111/j.1365-2656.2006.01141.x. [p188]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, Verlag, New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p191]
- Y. Xie et al. *DT: A Wrapper of the JavaScript Library DataTables*, 2024. URL <https://CRAN.R-project.org/package=DT>. R package version 0.33. [p191]
- B. Yang et al. Identification of ferroptosis-related gene signature for tuberculosis diagnosis and therapy efficacy. *iScience*, 27(7):110182, 2024a. doi: 10.1016/j.isci.2024.110182. URL <https://www.ncbi.nlm.nih.gov/pubmed/38989455>. [p188]
- W.-N. Yang et al. Gustative Roussy immune score is a predictor for major pathological response in rectal cancer: A result from the preoperative intraarterial chemoembolization combined with radiotherapy (PCAR) study. *Cancer Investigation*, 42(6):527–537, 2024b. doi: 10.1080/07357907.2024.2366912. URL <https://www.ncbi.nlm.nih.gov/pubmed/38965994>. [p188]
- Y. Yang et al. A glycosylation-related gene signature predicts prognosis, immune microenvironment infiltration, and drug sensitivity in glioma. *Front Pharmacol*, 14:1259051, 2023. doi: 10.3389/fphar.2023.1259051. URL <https://www.ncbi.nlm.nih.gov/pubmed/38293671>. [p188]
- S. Yoon et al. Effects of landscape patterns on the concentration and recovery time of PM2.5 in South Korea. *Land*, 11(12):2176, 2022. doi: 10.3390/land11122176. [p200]
- X. Zhu et al. High prevalence of respiratory co-infections and risk factors in COVID-19 patients at hospital admission during an epidemic peak in China. *Infection and Drug Resistance*, pages 6781–6793, 2023. doi: 10.2147/IDR.S435143. [p188]

Junhui Li

University of Massachusetts Chan Medical School
Department of Molecular, Cell, and Cancer Biology
University of Massachusetts Chan Medical School
Worcester, MA, USA, 01605
ORCID: 0000-0003-3973-1700
junhui.li11@umassmed.edu

Kai Hu

University of Massachusetts Chan Medical School
Department of Molecular, Cell, and Cancer Biology
University of Massachusetts Chan Medical School
Worcester, MA, USA, 01605

Xiaohuan Lu

Clark University
Gustaf H. Carlson School of Chemistry and Biochemistry
Clark University
Worcester, MA, USA, 01610

Cesar B. Sotelo

University of Massachusetts Chan Medical School
Department of Molecular, Cell, and Cancer Biology
University of Massachusetts Chan Medical School
Worcester, MA, USA, 01605

Sushmita Nayak

University of Massachusetts Chan Medical School
Department of Molecular, Cell, and Cancer Biology
University of Massachusetts Chan Medical School
Worcester, MA, USA, 01605

Michael A. Lodato

University of Massachusetts Chan Medical School
Department of Molecular, Cell, and Cancer Biology
University of Massachusetts Chan Medical School
Worcester, MA, USA, 01605

Wenxin Liu

China Agricultural University
Key Laboratory of Crop Heterosis and Utilization
The Ministry of Education/Key Laboratory of Crop Genetic Improvement
Beijing Municipality/National Maize Improvement Center
College of Agronomy and Biotechnology
China Agricultural University
Beijing, China, 100193

Lihua Julie Zhu

University of Massachusetts Chan Medical School
Department of Molecular, Cell, and Cancer Biology
Program in Molecular Medicine
Department of Genomics and Computational Biology
University of Massachusetts Chan Medical School
Worcester, MA, USA, 01605
ORCID: [0000-0001-7416-0590](https://orcid.org/0000-0001-7416-0590)
julie.zhu@umassmed.edu

onewaytests 3: One-Way Tests Independent Groups Designs - R Package and Its Web-Based Tool

by Merve Kasikci, Osman Dag, H. Erkin Sulekli, Malwane M. A. Ananda and Samaradasa Weerandi

Abstract One-way tests in independent groups designs are widely used statistical techniques in various fields, including but not limited to medical sciences, engineering, social sciences, economics, psychology, and biology. These techniques are commonly applied in scientific experiments to thoroughly investigate and understand the impact of the independent variable on the dependent variable. In this paper, we introduce the latest version of **onewaytests**, which offers a wide range of tools conducting analysis for one-way independent groups layout. In this study, we make significant improvements to the onewaytests package by adding 15 new one-way tests to the existing seven tests available in the work done by Dag et al. (2018). With this update, users now have a total of 22 tests for comprehensive assessments. We completely rewrite the function of the graphical approaches from scratch to improve the functionality and customization options. By doing so, we incorporate new features and enhancements while preserving the core functionality of the function. We update the web tool of R package, making it compatible with mobile devices. Users can now conveniently access and utilize the web tool on their smartphones or tablets, making statistical analysis more accessible and convenient. In addition to the aforementioned features, the package offers the capability to perform exploratory data analysis, such as generating descriptive statistics, investigating the homogeneity of variances and normality of data in each group through various tests and plots. Moreover, it provides functionality for conducting two sample tests and performing pairwise comparison methods, enabling users to delve deeper into their data analysis and draw more accurate conclusions. The package is readily available for download on the Comprehensive R Archive Network (CRAN). Additionally, users can access the associated web-tool at <http://www.softmed.hacettepe.edu.tr/onewaytests>.

1 Introduction

One-way independent groups designs are fundamental and widely used designs in observational and experimental research. One-way fixed effect analysis of variance (ANOVA) is the most common test used in these designs. One-way ANOVA is a parametric method based on comparing the means of independent groups. The assumptions of normality in each independent group and homogeneity of variance between groups must be satisfied to perform one-way ANOVA. However, it is often not possible to find datasets that meet both of these assumptions in practice. Therefore, there are a great number of alternative tests to one-way ANOVA for analyzing one-way independent groups designs. Among these tests, Welch's heteroscedastic F test (Welch, 1951), Welch's heteroscedastic F test with trimmed means and Winsorized variances (Welch, 1951), Alexander-Govern test (Alexander and Govern), James second order test (James, 1951), Brown-Forsythe test (Brown and Forsythe, 1974a,b) and Kruskal-Wallis test (Kruskal and Wallis, 1952) are provided in **onewaytests** package version 1.5 (Dag et al., 2017). The **onewaytests** package is compact and comprehensive since it provides descriptive statistics, assessment of the assumptions, and summarizes results using various graphics in addition to performing statistical analysis. This helps researchers evaluate their data from multiple angles.

In this paper, we present the upgraded version of **onewaytests** package, **onewaytests 3**. This version includes Alvandi's F test (Sadooghi-Alvandi et al., 2012), Alvandi's generalized p -value test (Sadooghi-Alvandi et al., 2012), approximate F test (Asiribo and Gurland, 1990), adjusted Welch's heteroscedastic F test (Welch, 1951), B square test (Özdemir and Kurt, 2006), Box F test (Box, 1954), Cochran F test (Cochran, 1937), generalized test equivalent to

parametric bootstrap test (Weerahandi and Krishnamoorthy, 2019), generalized test equivalent to fiducial test (Weerahandi and Krishnamoorthy, 2019), Johansen F test (Johansen, 1980), modified Brown-Forsythe test (Mehrotra, 1997), permutation F test (Berry et al., 2002), Scott-Smith F test (Scott and Smith, 1971), Welch-Aspin test (Aspin, 1948), and Weerahandi's generalized F test (Weerahandi, 1995) in addition to the tests in version 1.5. Along with the newly added 15 tests, there are a total of 22 tests for one-way independent groups designs in this extended version. Additionally, two independent samples tests including Student's t -test, Welch's t -test, and Mann-Whitney U test are added to the package.

The graphical approaches in the package are restructured from scratch to enhance the functionality and customization options of the graphical approaches. We add new features and enhancements providing even more flexibility in visualizing data while maintaining the core functionality. With these updates and improvements, the new version of the package offers a more comprehensive and versatile set of tools for data visualization, empowering users to create visually compelling and informative graphics effortlessly.

There is a user-friendly web application based on **onewaytests** package. This web-tool is updated with the addition of 15 one-way independent groups tests, along with graphical improvements. Also, web-tool is made compatible with mobile devices. The web-tool is freely available at <http://www.softmed.hacettepe.edu.tr/onewaytests>.

The paper is organized as follows. First, one-way tests in independent groups designs with theoretical backgrounds are presented. Second, an extensive application of **onewaytests** 3 package is demonstrated by using a real dataset. Third, the upgraded web application is introduced. At last, general conclusions and further research are provided.

2 One-way tests in independent groups designs

In this section, the statistical tests that are used to test the equality of several populations in the one-way independent groups designs are explained. Among these tests, Alvandi's F test (Sadooghi-Alvandi et al., 2012), Alexander-Govern test (Alexander and Govern), Alvandi's generalized p -value test (Sadooghi-Alvandi et al., 2012), ANOVA, approximate F test (Asiribo and Gurland, 1990), adjusted Welch's heteroscedastic F test (Welch, 1951), B square test (Özdemir and Kurt, 2006), Brown-Forsythe test (Brown and Forsythe, 1974a,b), Box F test (Lix and Keselman, 1998), Cochran F test (Cochran, 1937), Weerahandi's generalized F test (Weerahandi, 1995), James second order test (James, 1951, 1954), Johansen F test (Johansen, 1980), Modified Brown-Forsythe test (Mehrotra, 1997), Permutation F test (Berry et al., 2002), Scott Smith F test (Scott and Smith, 1971), Welch-Aspin test (Aspin, 1948), Welch's heteroscedastic F test (Welch, 1951), generalized test equivalent to parametric bootstrap test (Weerahandi and Krishnamoorthy, 2019), and generalized test equivalent to fiducial test (Weerahandi and Krishnamoorthy, 2019) test the null hypothesis $H_0 : \mu_1 = \mu_2 = \dots = \mu_k$ versus alternative $H_1 : \text{at least one } \mu_j (j = 1, 2, \dots, k) \text{ is different}$. All of these tests are parametric, with the exception of Permutation F test. Permutation F test evaluates mean differences without depending on the theoretical F distribution. Rather, it relies on an empirical F distribution produced using permutations. Therefore, this test does not require the assumption of normality (Winkler et al., 2014). Welch's heteroscedastic F test with trimmed means and Winsorized variances (Welch, 1951) tests the equality of population trimmed means, $H_0 : \mu_{t1} = \mu_{t2} = \dots = \mu_{tk}$ versus alternative $H_1 : \text{at least one } \mu_{tj} \text{ is different}$, where μ_{tj} represents the trimmed mean of the j th population ($j = 1, 2, \dots, k$). Kruskal-Wallis test (Kruskal and Wallis, 1952) tests the null hypothesis $H_0 : \theta_1 = \theta_2 = \dots = \theta_k$ versus alternative $H_1 : \text{at least one } \theta_j (j = 1, 2, \dots, k) \text{ is different}$. For Kruskal-Wallis test, θ_j represents sum of the ranks of the j th population. One-way independent tests that are used in this study are shown in Table 1.

Table 1: One-way independent tests. Method is the argument available in `onewaytests()` function to specify the desired one-way test.[†] Parametric tests are indicated as *Param*, and nonparametric tests as *NonParam*.

Existing Methods in the Dag et al. (2018)			
Name	Function	Method*	Test Type [†]
One-way ANOVA	<code>aov.test</code>	<code>aov</code>	<i>Param</i>
Welch's heteroscedastic <i>F</i> test	<code>welch.test</code>	<code>welch</code>	<i>Param</i>
Welch's heteroscedastic <i>F</i> test with trimmed and Winsorized Variances	<code>welch.test</code>	<code>welch_tw</code>	<i>Param</i>
Brown-Forsythe test	<code>bf.test</code>	<code>bf</code>	<i>Param</i>
Alexander-Govern test	<code>ag.test</code>	<code>ag</code>	<i>Param</i>
James Second Order test	<code>james.test</code>	<code>james</code>	<i>Param</i>
Kruskal-Wallis test	<code>kw.test</code>	<code>kw</code>	<i>NonParam</i>
Methods Formulated in the Extension			
Name	Function	Method*	Test Type [†]
Alvandi's <i>F</i> test	<code>af.test</code>	<code>af</code>	<i>Param</i>
Alvandi's generalized p-value test	<code>agp.test</code>	<code>agp</code>	<i>Param</i>
Approximate <i>F</i> test	<code>ap.test</code>	<code>ap</code>	<i>Param</i>
Adjusted Welch's heteroscedastic <i>F</i> test	<code>aw.test</code>	<code>aw</code>	<i>Param</i>
B square test	<code>b2.test</code>	<code>b2</code>	<i>Param</i>
Box <i>F</i> test	<code>box.test</code>	<code>box</code>	<i>Param</i>
Cochran <i>F</i> test	<code>cochran.test</code>	<code>cochran</code>	<i>Param</i>
Generalized test equivalent to Param boot-strap test	<code>gp.test</code>	<code>gtb</code>	<i>Param</i>
Generalized test equivalent to fiducial test	<code>gp.test</code>	<code>gtf</code>	<i>Param</i>
Johansen <i>F</i> Test	<code>johansen.test</code>	<code>johansen</code>	<i>Param</i>
Modified Brown-Forsythe test	<code>mbf.test</code>	<code>mbf</code>	<i>Param</i>
Permutation <i>F</i> test (PF)	<code>pf.test</code>	<code>pf</code>	<i>NonParam</i>
Scott-Smith test	<code>ss.test</code>	<code>ss</code>	<i>Param</i>
Welch-Aspin test	<code>wa.test</code>	<code>wa</code>	<i>Param</i>
Weerahandi's generalized <i>F</i> test	<code>wgf.test</code>	<code>wgf</code>	<i>Param</i>

2.1 Alvandi's *F* test

[Sadooghi-Alvandi et al. \(2012\)](#) presented a practical test that can be used when the group variances are heterogeneous. The test statistic for Alvandi's *F* test is

$$F_{AF} = \sum_{j=1}^k (n_j / S_j^2) (\bar{X}_j - \bar{X}_w)^2, \quad (1)$$

where

$$\bar{X}_w = \sum_{j=1}^k (n_j / S_j^2) \bar{X}_j / \sum_{j=1}^k (n_j / S_j^2). \quad (2)$$

The weights for each group are $w_j = (n_j / S_j^2) / \sum_{j=1}^k (n_j / S_j^2)$. The null hypothesis is rejected when $F_{AF} \geq (k-1)F_{k-1,r}$ where r is obtained with $r = 2m_T / (m_T - (k-1))$ and $m_T = \sum_{j=1}^k (1 - w_j)(n_j - 1) / (n_j - 3)$.

2.2 Alexander-Govern test

Alexander and Govern proposed the following test statistic as an alternative to ANOVA when group variances are not homogeneous:

$$\chi_{AG}^2 = \sum_{j=1}^k z_j^2. \quad (3)$$

In Equation (3),

$$z_j = c + \frac{(c^3 + 3c)}{b} - \frac{(4c^7 + 33c^5 + 240c^3 + 855c)}{(10b^2 + 8bc^4 + 1000b)},$$

where $c = [\alpha \times \ln(1 + t_j^2/v_j)]^{1/2}$, $b = 48\alpha^2$, $\alpha = v_j - 0.5$ and $v_j = (n_j - 1)$. The t statistic for each group is

$$t_j = \frac{\bar{X}_j - X^+}{S'_j}. \quad (4)$$

The variance-weighted mean is $X^+ = \sum_{j=1}^k w_j \bar{X}_j$, the standard error for the j th group is $S'_j = \left[\sum_{i=1}^{n_j} (X_{ij} - \bar{X}_j)^2 / n_j(n_j - 1) \right]^{1/2}$ and weights for each group are

$$w_j = (1/S_j'^2) / \sum_{j=1}^k (1/S_j'^2). \quad (5)$$

The null hypothesis is rejected when $\chi_{AG}^2 \geq \chi_{k-1}^2$.

2.3 Alvandi's generalized p-value test

Sadooghi-Alvandi et al. (2012) presented a generalized p-value test. Alvandi's generalized p-value can be computed by simulation according to the following two steps (Cavus et al., 2021). The first step is computing T_w value (observed value of Cochran's statistic)

$$T_w = \sum_{j=1}^k (n_j/S_j^2)(\bar{X}_j - \bar{X}_w)^2, \quad (6)$$

where $\bar{X}_w$ is calculated as Equation 2. Second step starts with generating independent variables x_j from $N(0,1)$ and U_j from $\chi_{n_j-1}^2$ ($j=1,2,\dots,k$). Then T value is calculated as follows:

$$T = \sum_{j=1}^k \frac{n_j - 1}{U_j} [X_j - q_j \tilde{X}]^2. \quad (7)$$

$\tilde{X}$ in the equation can be computed as $\sum_{j=1}^k q_j x_j$ where

$$q_j = \sqrt{(n_j/S_j^2) / \sum_{j=1}^k (n_j/S_j^2)}. \quad (8)$$

The second step repeats a large number of times (M). Let $M(t)$ be the number of times that $T \geq T_w$, then Alvandi's generalized p-value is obtained as $M(t)/M$.

2.4 ANOVA

The one-way fixed effects analysis of variance (ANOVA) is a powerful test if the assumptions of normality and variance homogeneity are met. The test statistic for ANOVA is

$$F = \frac{\sum_{j=1}^k (n_j)(\bar{X}_j - \bar{X}_{..})^2 / (k-1)}{\sum_{i=1}^n \sum_{j=1}^k (X_{ij} - \bar{X}_j)^2 / (n-k)}. \quad (9)$$

In Equation 9, n is the total number of observations and $\bar{X}_{..}$ is the overall mean, where $\bar{X}_{..} = \sum_j n_j(\bar{X}_j) / n$. The null hypothesis is rejected when $F \geq F_{k-1, n-k}$.

2.5 Approximate F test

Asiribo and Gurland (1990) proposed the following test statistic that has approximately F distribution with f_1 and f_2 degrees of freedom:

$$U_0 = \frac{n-k}{k-1} \frac{\sum_{j=1}^k n_j(\bar{X}_j - \bar{X}_{..})^2}{\sum_{j=1}^k f_j S_j^2}, \quad (10)$$

$$f_1 = \frac{(\sum_{j=1}^k 1 - \frac{n_j}{n})\sigma_j^2)^2}{\frac{1}{n^2}(\sum_{j=1}^k n_j \sigma_j^2)^2 + \sum_{j=1}^k (1 - \frac{2n_j}{n})\sigma_j^4}, \quad f_2 = \frac{(\sum_{j=1}^k f_j \sigma_j^2)^2}{\sum_{j=1}^k f_j \sigma_j^4}.$$

The null hypothesis is rejected when $U_0 > \hat{b}F_{f_1, f_2}$ where $b = \frac{n-k}{n(k-1)} (\sum_{j=1}^k (n - n_j)\sigma_j^2) / (\sum_{j=1}^k f_j \sigma_j^2)$.

2.6 Adjusted Welch's heteroscedastic F test

Welch (1951) proposed a heteroscedastic alternative to ANOVA that is robust to the violation of variance homogeneity assumption. The test statistic for adjusted Welch's heteroscedastic F test is calculated as follows Cavus et al. (2021):

$$F_w = \frac{\sum_{j=1}^k w_j(\bar{X}_j - \sum_{j=1}^k h_j \bar{X}_j)^2}{(k-1) + 2(k-2)(1/(k+1)) \sum_{j=1}^k (1/n_j - 1)(1 - h_j)^2}. \quad (11)$$

In Equation (26), $w_j = n_j / ((n_j - 1) / (n_j - 3)s_j^2)$, and $h_j = w_j / \sum_{j=1}^k w_j$. The null hypothesis is rejected when $F_w > F_{k-1, 1/\nu}$ where ν is obtained with $\nu = \frac{(k^2-1)/3}{\sum_{j=1}^k (1 - h_j)^2 / (n_j - 1)}$.

2.7 B square test

Özdemir and Kurt (2006) developed a new and simple approximation procedure which intends to create an easy and applicable alternative to ANOVA. The test statistic is

$$B^2 = \sum_{j=1}^k [c \{ \log(1 + \frac{(\frac{\bar{X}_j - X^+}{S_{\bar{X}_j}})^2}{v_j}) \}^{\frac{1}{2}}]^2, \quad (12)$$

where $c = (4v_j^2 + (5(2z_c^2 + 3))/24)/(4v_j^2 + v_j + (4z_c^2 + 9)/12)v_j^{\frac{1}{2}}$, $X^+ = \sum_{j=1}^k w_j \bar{X}_j$, the weights are $w_j = (1/S_{\bar{X}_j}^2)/(\sum_{j=1}^k (1/S_{\bar{X}_j}^2))$ and $v_j = n_j - 1$. The null hypothesis is rejected when $B^2 > \chi_{k-1}^2$.

2.8 Brown-Forsythe test

Brown and Forsythe (1974a,b) proposed the following test statistic as a modification of ANOVA:

$$F_{BF} = \frac{\sum_{j=1}^k n_j (\bar{X}_j - \bar{X}_{..})^2}{\sum_{j=1}^k (1 - n_j/n) S_j^2}. \quad (13)$$

The null hypothesis is rejected when $F_{BF} > F_{k-1, f}$. The degrees of freedom for denominator is calculated as $\left(\sum_{j=1}^k c_j^2 / (n_j - 1) \right)^{-1}$ where $c_j = ((1 - n_j/n) S_j^2) / \left[\sum_{j=1}^k (1 - n_j/n) S_j^2 \right]$.

2.9 Box F test

Box F test statistic was presented by Box (1954) as follows:

$$F_{BOX} = \frac{\sum_{j=1}^k n_j (\bar{X}_j - \bar{X})^2}{\sum_{j=1}^k (1 - \frac{n_j}{n}) S_j^2}. \quad (14)$$

The null hypothesis is rejected when $F_{BOX} > F_{v_1, v_2}$ where $v_1 = [\sum_{j=1}^k (n - n_j) S_j^2] / (\sum_{j=1}^k n_j S_j^2)^2 + n \sum_{j=1}^k (n - 2n_j) S_j^4$ and $v_2 = [\sum_{j=1}^k (n_j - 1) S_j^2]^2 / \sum_{j=1}^k (n_j - 1) S_j^4$.

2.10 Cochran F test

Cochran (1937) proposed the following test statistic:

$$T_{Cochran} = \sum_{j=1}^k (n_j / S_j^2) (\bar{X}_j - \bar{X}_w)^2, \quad (15)$$

where $\bar{X}_w$ is calculated as Equation 2. The null hypothesis is rejected when $T_{Cochran} > \chi_{k-1}^2$.

2.11 Generalized test equivalent to parametric bootstrap test

Weerahandi and Krishnamoorthy (2019) showed that the parametric bootstrap test presented by Krishnamoorthy et al. (2007) is a regular generalized test. The test is carried out as follows. The null hypothesis is rejected when the generalized p-value given by the following formula is less than the desired nominal level, say 0.05:

$$p = Pr \left(\sum_{j=1}^k \frac{r_j Z_j^2}{U_j} - \frac{\sum_{j=1}^k \frac{n_j r_j}{s_j^2 U_j} \left(\frac{s_j Z_j}{\sqrt{n_j}} \right)}{\sum_{j=1}^k \frac{n_j r_j}{s_j^2 U_j}} \geq \sum_{j=1}^k \frac{n_j (\bar{x}_j - \bar{x}_s)^2}{s_j^2} \right), \quad (16)$$

where $Z_i \sim N(0,1)$, $U_i \sim \chi_{r_i}^2$, $r_j = n_j - 1$, $\bar{x}_j$ and s_j^2 are the observed sample mean and the observed sample variance from the j^{th} population, and $\bar{x}_s$ is the overall weighted sample mean.

2.12 Generalized test equivalent to fiducial test

Weerahandi and Krishnamoorthy (2019) also showed that the fiducial test developed by Li et al. (2011) is also a regular generalized test. The null hypothesis is rejected when the generalized p-value given by the formula below is less than the desired nominal level:

$$p = Pr \left(\sum_{j=1}^k T_j^2 - \frac{(\sum_{j=1}^k T_j \sqrt{n_j} / s_j)^2}{\sum_{j=1}^k n_j / s_j^2} \geq \sum_{j=1}^k \frac{n_j (\bar{x}_j - \bar{x}_s)^2}{s_j^2} \right), \quad (17)$$

where $T_j \sim t_{r_j}$.

2.13 James second order test

James (1951) presented an alternative test to ANOVA. This test statistic is

$$J = \sum_j t_j^2, \quad (18)$$

where t_j is given in Equation (4). The test statistic, J , is compared to a critical value, $h(\alpha)$, where

$$\begin{aligned} h(\alpha) = & r + \frac{1}{2}(3\chi_4 + \chi_2)T + \frac{1}{16}(3\chi_4 + \chi_2)^2 \left(1 - \frac{k-3}{r} \right) T^2 \\ & + \frac{1}{2}(3\chi_4 + \chi_2)(8R_{23} - 10R_{22} + 4R_{21} - 6R_{12}^2 + 8R_{12}R_{11} - 4R_{11}^2) \\ & + (2R_{23} - 4R_{22} + 2R_{21} - 2R_{12}^2 + 4R_{12}R_{11} - 2R_{11}^2)(\chi_2 - 1) \\ & + \frac{1}{4}(-R_{12}^2 + 4R_{12}R_{11} - 2R_{12}R_{10} - 4R_{11}^2 + 4R_{11}R_{10} - R_{10}^2)(3\chi_4 - 2\chi_2 - 1) \\ & + (R_{23} - 3R_{22} + 3R_{21} - R_{20})(5\chi_6 + 2\chi_4 + \chi_2) \\ & + \frac{3}{16}(R_{12}^2 - 4R_{23} + 6R_{22} - 4R_{21} + R_{20})(35\chi_8 + 15\chi_6 + 9\chi_4 + 5\chi_2) \\ & + \frac{1}{16}(-2R_{22} + 4R_{21} - R_{20} + 2R_{12}R_{10} - 4R_{11}R_{10} + R_{10}^2)(9\chi_8 - 3\chi_6 - 5\chi_4 - \chi_2) \\ & + \frac{1}{4}(-R_{22} + R_{11}^2)(27\chi_8 + 3\chi_6 + \chi_4 + \chi_2) + \frac{1}{4}(R_{23} - R_{12}R_{11})(45\chi_8 + 9\chi_6 + 7\chi_4 + 3\chi_2). \end{aligned}$$

For any integers s and t , $R_{st} = \sum_j (n_j - 1)^{-s} w_j^t$, $\chi_{2s} = r^s / [(k-1)(k+1) \dots (k+2s-3)]$ where r is the $(1 - \alpha)$ centile of a χ^2 distribution with $k - 1$ degrees of freedom, $T =$

$\sum_j (1 - w_j)^2 / (n_j - 1)$ and w_j is defined in Equation 5. The null hypothesis is rejected when $J \geq h(\alpha)$.

2.14 Johansen F test

Johansen (1980) proposed the following test statistics:

$$T_{JF} = \frac{\sum_{j=1}^k (w_j (\bar{x}_j - \sum_{j=1}^k (h_j \bar{x}_j))^2)}{c}, \quad (19)$$

where $w_j = n_j / \sigma_j$, $h_j = w_j / \sum_{j=1}^k (w_j)$, $c = (k - 1) + 2A - 6A / (k + 1)$ and $A = \sum_{j=1}^k ((1 - h_j)^2 / (n_j - 1))$. The null hypothesis is rejected when $T_{JF} \geq F_{v_1, v_2}$ where $v_1 = k - 1$ and $v_2 = k - 1(k + 1) / 3A$.

2.15 Kruskal-Wallis test

Kruskal and Wallis (1952) presented the nonparametric alternative to ANOVA. The test statistics is

$$\chi_{KW}^2 = \frac{1}{S^2} \left(\sum_{j=1}^k \frac{R_j^2}{n_j} - \frac{n(n+1)^2}{4} \right), \quad (20)$$

where r_{ij} denote the rank of X_{ij} when $n = n_1 + \dots + n_k$ observations are ranked from smallest to largest. $R_j = \sum_{i=1}^{n_j} r_{ij}$ is the sum of ranks assigned to the observations in the j th group and $\bar{R}_j = R_j / n_j$ is the average rank for these observations and

$$S^2 = \frac{1}{n-1} \left(\sum_{j=1}^k \sum_{i=1}^{n_j} r_{ij}^2 - \frac{n(n+1)^2}{4} \right).$$

The null hypothesis is rejected when $\chi_{KW}^2 \geq \chi_{k-1}^2$.

2.16 Modified Brown-Forsythe test

Mehrotra (1997) proposed a modification of Brown-Forsythe test to overcome the problem of higher than acceptable rate of false positives. The test statistics is

$$F_{MBF} = \frac{\sum_{j=1}^k n_j (\bar{X}_j - \bar{X}_{..})^2}{\sum_{j=1}^k (1 - \frac{n_j}{n}) s_j^2}. \quad (21)$$

The null hypothesis is rejected when $F_{MBF} > F_{f_1, f_2}$.

$$f_1 = \frac{(\sum_{j=1}^k \sigma_j^2 - \sum_{j=1}^k n_j \sigma_j^2 / n)^2}{\sum_{j=1}^k \sigma_j^4 + (\sum_{j=1}^k n_j \sigma_j^2 / n)^2 - 2 \sum_{j=1}^k n_j \sigma_j^4 / n}, \quad f_2 = \frac{[\sum_{j=1}^k (1 - n_j / n) \sigma_j^2]^2}{\sum_{j=1}^k (1 - n_j / n)^2 \sigma_j^4 / (n_j - 1)}. \quad (22)$$

The degrees of freedom of the test statistic can be calculated by using the Equation 22.

2.17 Permutation F test

Berry et al. (2002) presented a permutational test as an alternative to ANOVA. The test statistics is

$$F_{PF} = \frac{T - n\bar{x}^2/(k-1)}{V - T/(n-k)}, \quad (23)$$

where $T = \sum_{j=1}^k n_j \bar{x}_j^2$, $V = \sum_{j=1}^k \sum_{i=1}^{n_j} x_{ij}^2$ and $\bar{x} = \frac{1}{n} \sum_{j=1}^k n_j \bar{x}_j$. The null hypothesis is rejected when $F_{PF} > F_{k-1, n-k}$.

2.18 Scott Smith F test

Scott and Smith (1971) proposed the following test statistic where group variances are not homogeneous:

$$F_{SS} = \sum_{j=1}^k \frac{n_j(\bar{X}_j - \bar{X})^2}{S_j^{*2}}, \quad (24)$$

where $S_j^{*2} = (n_j - 1)/(n_j - 3)S_j^2$. The null hypothesis is rejected when $F_{SS} > \chi_k$.

2.19 Welch-Aspin test

Aspin (1948) presented a modification of Welch test. The test statistic is

$$F_{WA} = \frac{\sum_{j=1}^k (\bar{X}_j - X^+)^2 / S_j^2}{(k-1)[1 + \frac{2k-2}{k^2-1}\Lambda]}, \quad (25)$$

where $\Lambda = \sum [(1 - w_j)^2 / v_j]$, $w_j = 1/S_j^2 / \sum_{j=1}^k 1/S_j^2$ and X^+ is the overall mean. The null hypothesis is rejected when $F_{WA} > F_{v_1, v_2}$ where $v_1 = k - 1$ and $v_2 = (k^2 - 1)/(3\Lambda)$.

2.20 Welch's heteroscedastic F test

Welch (1951) proposed a robust test that can be used when homogeneity of variance is not met. The test statistic is

$$F_w = \frac{\sum_j w_j (\bar{X}_{.j} - X'_{..})^2 / (k-1)}{[1 + \frac{2}{3}((k-2)\nu)]}, \quad (26)$$

where $w_j = n_j / S_j^2$, $S_j^2 = \sum_i (X_{ij} - \bar{X}_{.j})^2 / (n_j - 1)$, $X'_{..} = \sum_j w_j \bar{X}_{.j} / \sum_j w_j$, and

$$\nu = \frac{3 \sum_j \left[\left(1 - \frac{w_j}{\sum_j w_j} \right)^2 / (n_j - 1) \right]}{k^2 - 1}.$$

The test statistics follow an F distribution with degrees of freedom $k - 1$ and $1/\nu$.

2.21 Welch's heteroscedastic F test with trimmed means and Winsorized variances

Welch (1951) presented a robust procedure for independent groups design by replacing the usual means and variances with trimmed means and Winsorized variances. Let $X_{(1)j} \leq$

$X_{(2)j} \leq \dots \leq X_{(n_j)j}$ be the ordered observations in the j th group and $g_j = \|\epsilon n_j\|$, ϵ is the proportion to be trimmed in each tail of the distribution. After trimming, the effective sample size for the j th group becomes $h_j = n_j - 2g_j$. Then j th sample trimmed mean is

$$\bar{X}_{tj} = \frac{1}{h_j} \sum_{i=g_j+1}^{n_j-g_j} X_{(i)j},$$

and j th sample Winsorized mean is

$$\bar{X}_{wj} = \frac{1}{n_j} \sum_{i=1}^{n_j} Y_{ij},$$

where

$$Y_{ij} = \begin{cases} X_{(g_j+1)j} & \text{if } X_{ij} \leq X_{(g_j+1)j}, \\ X_{ij} & \text{if } X_{(g_j+1)j} < X_{ij} < X_{(n_j-g_j)j}, \\ X_{(n_j-g_j)j} & \text{if } X_{ij} \geq X_{(n_j-g_j)j}. \end{cases}$$

The sample Winsorized variance is

$$s_{wj}^2 = \frac{1}{(n_j - 1)} \sum_{i=1}^{n_j} (Y_{ij} - \bar{X}_{wj})^2,$$

where $q_j = (n_j - 1)s_{wj}^2/h_j(h_j - 1)$, $w_j = \frac{1}{q_j}$, $U = \sum_j w_j$, $\tilde{X} = \frac{1}{U} \sum_j w_j \bar{X}_{tj}$. The test statistic is

$$F_{WT} = \frac{A}{B + 1}, \quad (27)$$

where $A = \frac{1}{k-1} \sum_j w_j (\bar{X}_{tj} - \tilde{X})^2$, $B = \frac{2(k-2)}{k^2-1} \sum_j \frac{(1-w_j/U)^2}{h_j-1}$. The null hypothesis is rejected when $F_{WT} > F_{k-1, \nu'}$ where $\nu' = \left(\frac{3}{k^2-1} \sum_j \frac{(1-w_j/U)^2}{h_j-1} \right)^{-1}$.

2.22 Weerahandi's generalized F test

Weerahandi (1995) provided an extension of the classical F-test for the case of unequal variances. The generalized p value of that test is obtained by using the following formula involving an expected value of the form $EG_{k'}()$, where $G_{k'}$ is the cdf of the chi-squared distribution with k' degrees of freedom, and $k' = k - 1$.

$$p = 1 - EG_{k'}(s_B \left(\frac{n_1 s_1^2}{Y_1}, \frac{n_2 s_2^2}{Y_2}, \dots, \frac{n_k s_k^2}{Y_k} \right)) \quad (28)$$

where $Y_j \sim \chi_{r_j}^2$ and

$$s_B(w_1, \dots, w_k) = \sum_{j=1}^k w_j (\bar{x}_j - \bar{x}_B)^2$$

is the weighted between group sum of squares, w_j is the j^{th} weight, and $\bar{x}_B$ is the overall weighted sample mean based on the same weights.

3 Implementation of R package

The **onewaytests** package is a comprehensive collection of functions specifically designed for analyzing data from one-way independent groups designs. This versatile package provides a wide range of tools to facilitate statistical analysis and exploration of such

designs. The package includes 22 one-way tests for independent groups designs. In addition to the one-way tests, the package offers functions for generating descriptive statistics, conducting goodness-of-fit tests to check the fundamental assumptions, performing pairwise comparisons, and employing graphical approaches to enhance data interpretation.

In this section, we work on penguins data set, originally published in the work done by Gorman et al. (2014), available in the `palmerpenguins` package (Horst et al., 2020). For implementation of the package, we utilize body mass as a response variable and species as a grouping variable.

After successfully installing and loading the `onewaytests` package, researchers have access to a wide range of functions specifically designed for analyzing one-way independent groups designs. These functions are incredibly useful when researchers compare k populations with respect to continuous outcomes. The one-way tests provided by this package not only calculate the test statistics and p -values, but they also provide the pairwise comparisons for valuable insights by evaluating the hypothesis of the statistical process. With the `onewaytests` package, researchers can easily conduct thorough and in-depth analyses of their one-way independent groups designs, gaining valuable insights and making well-informed conclusions.

In this part, we use the functions, originally developed in our previously published work (Dag et al., 2018) to describe data and assess the main assumptions, variance homogeneity and normality.

The `describe()` functions return the output involving sample size, mean, standard deviation, median, minimum value, maximum value, 25th percentile, 75th percentile, skewness, kurtosis and number of missing values (NA) for one-way layout.

```
R> library(onewaytests)
R> library(palmerpenguins)
R> describe(body_mass_g ~ species, data = penguins)
```

	n	Mean	Std.Dev	Median	Min	Max	25th	75th	Skewness	Kurtosis	NA
Adelie	151	3700.662	458.5661	3700	2850	4775	3350.0	4000	0.28249381	2.405611	0
Chinstrap	68	3733.088	384.3351	3700	2700	4800	3487.5	3950	0.24194125	3.463681	0
Gentoo	123	5076.016	504.1162	5000	3950	6300	4700.0	5500	0.06878276	2.257871	0

Researchers perform variance homogeneity tests with the `homog.test()` function. It offers three variance homogeneity tests; Levene, Bartlett, and Fligner-Killeen tests.

```
R> homog.test(body_mass_g ~ species, data = penguins, method = "Levene")
```

```
Levene's Homogeneity Test (alpha = 0.05)
-----
dependent var : body_mass_g
grouping var  : species

F          = 5.335495
num df     = 2
denom df   = 339
p-value    = 0.005230535

Result : Variances are not homogeneous.
-----
```

Levene's homogeneity test results reveal that the variances between penguin species are not homogeneous ($F = 5.335495$, p -value = 0.005230535).

Researchers can assess normality using the `nor.test()` function. This function offers six normality tests: Shapiro-Wilk, Shapiro-Francia, Lilliefors (also known as Kolmogorov-Smirnov test), Anderson-Darling, Cramer-von Mises, and Pearson Chi-square tests.

```
R> nor.test(body_mass_g ~ species, data = penguins, method = "SW")
```

```
Shapiro-Wilk Normality Test (alpha = 0.05)
```

```
-----
data : body_mass_g and species

      Level Statistic    p.value    Normality
1   Adelie 0.9807079 0.03239702      Reject
2 Chinstrap 0.9844938 0.56050824    Not reject
3   Gentoo 0.9859276 0.23361649    Not reject
-----
```

Shapiro-Wilk normality test results state that body masses of chinstrap and gentoo species are normally distributed since the associated p-values (0.56050824 and 0.23361649, respectively) are greater than 0.05. On the other hand, there is enough evidence to reject the normality of body mass for adelie group (p-value = 0.03239702).

In the [onewaytests](#) package, the first argument is specified as a formula where the left-hand side represents sample values and the right-hand side represents groups. The formula must have one variable on each side, with the left side being numeric and the right side being a factor. Otherwise, an error message is returned by each function.

3.1 One-way tests in independent groups designs: onewaytests(...)

In our previously published paper ([Dag et al., 2018](#)), we introduced this package that included seven one-way tests. In this latest version of the package, we have expanded its capabilities by adding 15 new functions specifically designed for one-way independent groups designs. These additional functions provide users with a wider range of options for conducting their analyses. The complete list of all the one-way tests, their corresponding functions, and the method name of the `onewaytests()` function can be found in Table 1.

In this section, we introduce a generic function called `onewaytests()` that encompasses a total of 22 one-way tests. By utilizing this function, researchers can conveniently apply any of the available one-way tests. To specify the desired method, researchers simply need to provide the method argument within the `onewaytests()` function. The list of all methods is provided under Method in Table 1.

All one-way tests can be utilized with separate functions, offering flexibility and versatility. To facilitate ease of use and accessibility, we have meticulously gathered all methods for researchers' convenience. The `onewaytests()` function provides four different types of output. We present each of four kinds to introduce the outputs. As an example, we have incorporated Alvandi's *F* test, Scott-Smith test, Weerahandi's generalized *F* test, and James second order test, which can also be applied using `af.test()`, `ss.test()`, `wgf.test()`, `james.test()` functions, respectively, to demonstrate the practical usage of the `onewaytests()` function. The test results can be displayed by setting the `verbose` argument to `TRUE`. The results are returned as an object of class `hstest`.

```
R> out <- onewaytests(body_mass_g ~ species, data = penguins, method = "af")
```

```
Alvandi's F Test
```

```
data: body_mass_g and species
```

```
F = 318.69, num df = 2.000, denom df = 97.832, p-value < 2.2e-16
```

For more comprehensive results, the `summary()` function can be used, as demonstrated in the example below.

```
R> summary(out)
```

Alvandi's F Test (alpha = 0.05)

 dependent var : body_mass_g
 grouping var : species

F = 318.6896
 num df = 2
 denom df = 97.83241
 p-value < 2.2e-16

Result : Difference is statistically significant.

 * statistic is distributed as F distribution.

Here, Alvandi's test statistic is distributed as F with the degrees of freedom for numerator (num df) and denominator (denom df). Also, p-value is the significance value of the test statistic. Since the p-value is lower than 0.05, there is enough evidence to conclude that the difference between the penguin species in terms of body mass is statistically significant ($F = 318.6896$, $p\text{-value} < 2.2 \times 10^{-16}$).

```
R> out <- onewaytests(body_mass_g ~ species, data = penguins, method = "ss",
  verbose = FALSE)
```

```
R> summary(out)
```

Scott-Smith Test (alpha = 0.05)

 dependent var : body_mass_g
 grouping var : species

X-squared = 639.8695
 df = 3
 p-value < 2.2e-16

Result : Difference is statistically significant.

 * statistic is distributed as chi-squared distribution.

In the output, Scott-Smith test statistic is distributed as χ^2 with the degrees of freedom (df). Also, p-value is the significance value of the test statistic. Since the p-value is lower than 0.05, one can conclude that the difference between the species is statistically significant ($\chi^2_{AG} = 639.8695$, $p < 2.2 \times 10^{-16}$).

```
R> out <- onewaytests(body_mass_g ~ species, data = penguins, method = "wgf",
  verbose = FALSE)
```

```
R> summary(out)
```

Weerahandi's Generalized F Test (alpha = 0.05)

 dependent var : body_mass_g
 grouping var : species

p-value < 2.2e-16

Result : Difference is statistically significant.

 * p-value is obtained using Monte Carlo simulation.

Here, p-value is the significance value of the test statistic. Since the p-value is lower than 0.05, there is sufficient evidence to conclude that the difference between the penguin species in terms of body mass is statistically significant.

```
R> out <- onewaytests(body_mass_g ~ species, data = penguins, method = "james",
                      verbose = FALSE)
```

```
R> summary(out)
```

```
James Second Order Test (alpha = 0.05)
```

```
-----
dependent var : body_mass_g
grouping var  : species
```

```
Jtest          = 637.3792
CriticalValue = 6.108713
```

```
Result : Difference is statistically significant.
```

```
-----
* test statistic is sum of the squared standardized differences and compared to a
critical value.
```

In the output, Jtest is the James second order test statistic (J), criticalValue is the critical value ($h(\alpha)$) associated to the significance level (α). The null hypothesis is rejected when J exceeds $h(\alpha)$. We conclude that there is a statistically significant difference between the penguin species since $J = 637.3792$ is larger than $h(\alpha) = 6.108713$.

The **onewaytests** package provides the capability to perform pairwise comparisons when a statistically significant difference is observed. It offers several options for controlling the type I error rate, including bonferroni, holm (Holm, 1979), hochberg (Hochberg, 1988), hommel (Hommel, 1988), BH (Benjamini and Hochberg, 1995), BY (Benjamini and Yekutieli, 2001), and none in paircomp() function. The default is set to "bonferroni". The development of paircomp() function is extensively placed in the article work done by Dag et al. (2018). The paircomp() function is capable of recognizing the output of the onewaytests() function. In this part, we include the pairwise comparison following to Alvandi's F test to protect the compactness of the paper.

```
R> out <- onewaytests(body_mass_g ~ species, data = penguins, method = "af",
                      verbose = FALSE)
```

```
R> paircomp(out, adjust.method = "bonferroni")
```

```
Bonferroni Correction (alpha = 0.05)
```

```
-----
Level (a) Level (b)      p.value  No difference
1   Adelie Chinstrap 1.000000e+00    Not reject
2   Adelie   Gentoo 2.765295e-48      Reject
3 Chinstrap   Gentoo 1.639844e-34      Reject
-----
```

According to the result obtained by adjusting with Bonferroni method, Gentoo group has statistically greater body mass than Adelie and Chinstrap groups.

3.2 Graphical approaches based on ggplot2 package

In this part, we rewrite the gplot function from scratch. The gplot function offers different graphic types, box-and-whisker plot, violin plot, box-and-whisker plot with violin lines and error bars. These options can be selected in the type argument with "boxplot", "violin", "boxplot-violin", and "errorbar", respectively. There exists width argument involving three

numeric values. These numeric values specify the widths for the boxes of box-and-whisker plots (defaults to 0.3), violin plots (defaults to 1.0), and the little lines at the tops and bottoms of the error bars (defaults to 0.2), respectively. Also, there exists the dots argument to draw observations on the plots. The binwidth argument offers to change the size of dots. Users can change the colors of the plots with color_manual argument. Moreover, researchers can set color palette using wes_palette function available in [wesanderson](#) package (FantasticFox1 palette is used as a default). Additionally, the theme of plot can be changed with theme argument. The list of all theme is available with ggtheme function available in [ggplot2](#) package ([Wickham, 2016](#)). We use the classic dark-on-light ggplot2 theme as a default theme by setting theme_bw() to theme argument. Researchers can use standard deviation or standard error with option argument while drawing error bars. Also, researchers can add mean bar to error bars with add argument. Additionally, the labels of x and y axes and title can be changed with xlab, ylab, and title arguments, respectively.

In this part, we draw some graphic types to illustrate the usage of gplot function. These graphics involve box-and-whisker plot with violin line and dots (Figure 1a), box-and-whisker plot with violin line (Figure 1b), box-and-whisker plot (Figure 1c), violin plot (Figure 1d), mean $\pm$ standard deviation graph (Figure 1e), and mean $\pm$ standard deviation graph with mean bar (Figure 1f). These graphics can be obtained via following codes:

```
# Box-and-whisker plot with violin lines and dots
R> gplot(body_mass_g ~ species, data = penguins, type = "boxplot-violin",
width = c(0.4, 0.95, NA), binwidth = 50, ylab = "body mass (grams)")

# Box-and-whisker plot with violin lines
R> gplot(body_mass_g ~ species, data = penguins, type = "boxplot-violin",
width = c(0.4, 0.95, NA), dots = FALSE, ylab = "body mass (grams)")

# Box-and-whisker plot
R> gplot(body_mass_g ~ species, data = penguins, type = "boxplot",
width = c(0.4, NA, NA), dots = FALSE, ylab = "body mass (grams)")

# Violin plot
R> gplot(body_mass_g ~ species, data = penguins, type = "violin",
width = c(NA, 0.95, NA), dots = FALSE, ylab = "body mass (grams)")

# Error bar (mean +- standard deviation)
R> gplot(body_mass_g ~ species, data = penguins, type = "errorbar",
width = c(NA, NA, 0.3), binwidth = 50, ylab = "body mass (grams)",
option = "sd")

# Error bar (mean +- standard deviation) with mean bar
R> gplot(body_mass_g ~ species, data = penguins, type = "errorbar",
width = c(NA, NA, 0.3), binwidth = 50, ylab = "body mass (grams)",
option = "sd", bar = TRUE)
```

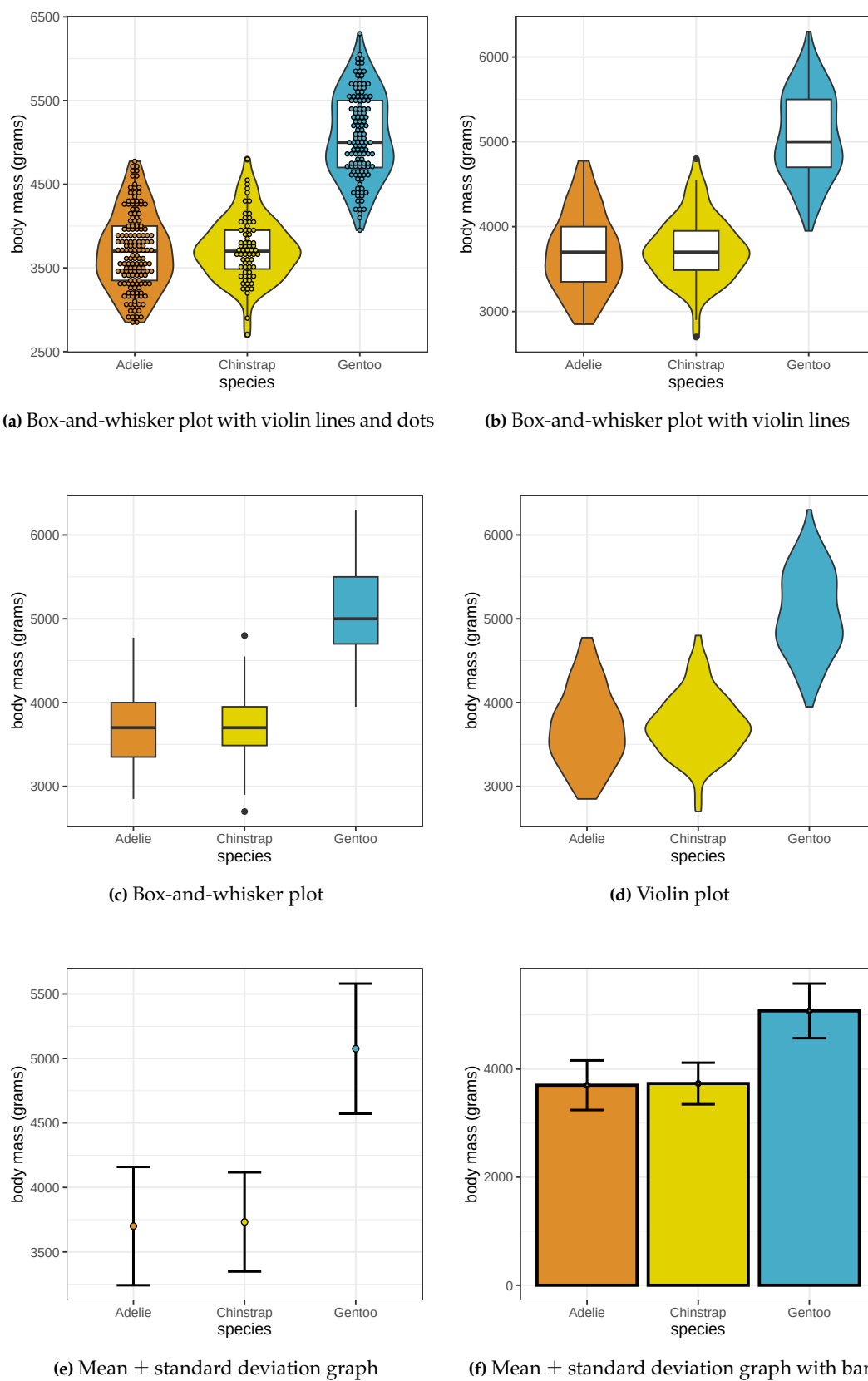


Figure 1: Graphical approaches for one-way layout

4 Web-based tool

There is an easy-to-use web application available for users, which is built upon the **onewaytests** package, providing a convenient and accessible way to perform statistical analysis. This web-tool has recently undergone significant updates and enhancements. One of the major improvements is the addition of 15 one-way independent groups tests, allowing users to conduct a wide range of statistical comparisons. Furthermore, the web-tool now features enhanced graphics, providing users with clear and visually appealing representations of their data. Moreover, we have made the web-tool compatible with a mobile phone application due to the growing popularity of mobile devices. This means that users can conveniently access and utilize the web-tool on their smartphones or tablets, making statistical analysis more accessible and convenient than ever before. Researchers can freely access the web application by visiting the following link, <http://www.softmed.hacettepe.edu.tr/onewaytests>.

We utilize the **shiny** package (Chang et al., 2017) to create the web-interface of the **onewaytests** package. This tool is specifically designed to cater to the needs of new R users who may not have extensive programming experience, as well as applied researchers who require a straightforward interface for conducting their analyses. Our web tool aims to simplify the process of performing one-way tests and empower users to efficiently explore and interpret their data by harnessing the power of the **shiny** package. There is an example well-known data set called iris data collected by Anderson (1935) placed in the web-tool, also available in R, to help users learn the usage of the tool.

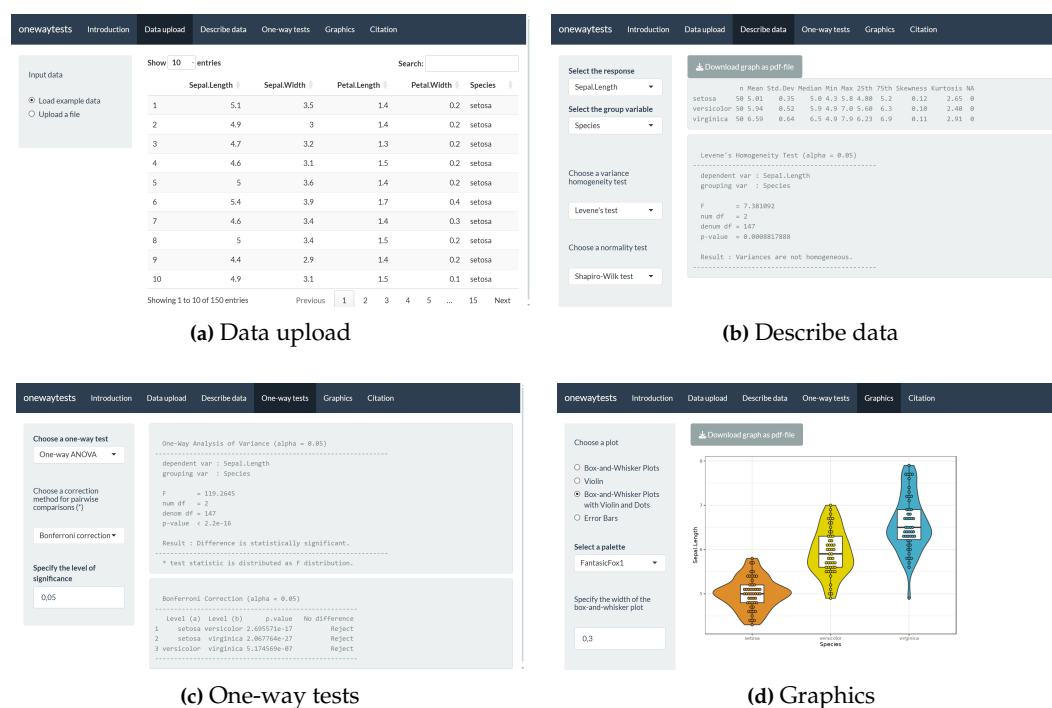


Figure 2: Data analysis interfaces of onewaytests web-tool

Researchers can obtain descriptive statistics using the "Describe data" tab (Figure 2b). In this tab, the assumptions of variance homogeneity and normality can be assessed. This tab provides Levene's test, Bartlett's test, and Fligner-Killeen test to check for variance homogeneity. It also offers various normality tests (Shapiro-Wilk, Cramer-von Mises, Lilliefors (Kolmogorov-Smirnov), Shapiro-Francia, Anderson-Darling, Pearson Chi-Square tests) and plots (Q-Q plot and Histogram with normal curve) to assess the normality of data in each subgroup.

Users can perform 22 one-way tests (Scott-Smith test, Box F test, Johansen F test, Generalized tests equivalent to Parametric Bootstrap and Fiducial tests, Alvandi's F test, Alvandi's generalized p -value, approximate F test, B square test, Cochran test, Weerahandi's

generalized F test, modified Brown-Forsythe test, adjusted Welch's heteroscedastic F test, Welch-Aspin test, Permutation F test, one-way analysis of variance, Welch's heteroscedastic F test, Welch's heteroscedastic F test with trimmed means and Winsorized variances, Brown-Forsythe test, Alexander-Govern test, James second order test, Kruskal-Wallis test) using the "One-way tests" tab (Figure 2c). This tab also allows users to make pairwise comparisons (Bonferroni, Holm, Hochberg, Hommel, Benjamini-Hochberg, Benjamini-Yekutieli, no corrections) to investigate the group(s) leading to the differences.

Additionally, researchers can summarize their research findings using well-designed **ggplot2** graphics (box-and-whisker plot, violin plot, and error bars) in the "Graphics" tab (Figure 2d). The web tool is freely available at <http://www.softmed.hacettepe.edu.tr/onewaytests>.

5 Conclusion

One-way tests in independent groups designs are highly popular and extensively utilized statistical techniques in a multitude of fields. These fields encompass medical sciences, engineering, social sciences, economics, psychology, and biology. These techniques allow the researchers for the identification of potential differences among groups, which can further enhance the scientific knowledge and contribute to the advancement of various disciplines.

In this paper, we present the latest version of **onewaytests**, which provides a variety of tools for one-way independent groups designs. We make significant enhancements to the package by introducing 15 new one-way tests (Scott-Smith test, Box F test, Johansen F test, Generalized tests equivalent to Parametric Bootstrap and Fiducial tests, Alvandi's F test, Alvandi's generalized p -value, approximate F test, B square test, Cochran test, Weerahandi's generalized F test, modified Brown-Forsythe test, adjusted Welch's heteroscedastic F test, Welch-Aspin test, Permutation F test) in addition to the existing seven tests (one-way analysis of variance, Welch's heteroscedastic F test, Welch's heteroscedastic F test with trimmed means and Winsorized variances, Brown-Forsythe test, Alexander-Govern test, James second order test, Kruskal-Wallis test) in the work done by Dag et al. (2018). Users now have a total of 22 tests available for comprehensive assessments.

In order to enhance the functionality and customization options of the graphical approaches, we undertake a complete overhaul of their function. Through this process, we not only add new features and enhancements, but also ensure that the core functionality of the function remains intact. This comprehensive revision allows us to fully optimize the graphical approaches, resulting in a more versatile tool for the users.

In this study, we improve the web tool of **onewaytests** package, ensuring its compatibility with a wide range of mobile devices. This significant update allows users to effortlessly access and utilize the web tool on their smartphones or tablets. This mobile compatibility opens up a world of possibilities, enabling researchers and statisticians to seamlessly integrate statistical analysis into their daily routines, regardless of their location or device.

The **onewaytests** package also provides additional capabilities to enhance your data analysis which involve generating descriptive statistics, examining the homogeneity of variances and normality of data in each group using a variety of tests and plots. Also, it has capability to perform two sample tests. Furthermore, the package offers functionality for conducting pairwise comparison methods after the statistically significant difference among groups is obtained. These features allow users to explore their data from multiple angles in a compact manner and draw more accurate conclusions.

Currently, the **onewaytests** package provides a comprehensive set of tools for conducting analysis on one-way independent groups designs. These tools include conducting one-way tests, performing pairwise comparisons, obtaining descriptive statistics, conducting two-sample tests, utilizing graphical approaches, and assessing variance homogeneity and normality through the use of tests and plots. Additionally, it is worth noting that the

package and its associated web-tool will be regularly updated to ensure that users have access to the latest features and improvements.

References

- R. A. Alexander and D. M. Govern. A new and simpler approximation for ANOVA under variance heterogeneity. *Journal of Educational and Behavioral Statistics*, (2):91–101. [p207, 208, 210]
- E. Anderson. The irises of the Gaspé Peninsula. *Bulletin of the American Iris society*, 59:2–5, 1935. [p223]
- O. Asiribo and J. Gurland. Coping with variance heterogeneity. *Communications in Statistics-Theory and Methods*, 19(11):4029–4048, 1990. URL <https://doi.org/10.1080/03610929008830427>. [p207, 208, 211]
- A. A. Aspin. An examination and further development of a formula arising in the problem of comparing two mean values. *Biometrika*, 35(1/2):88–96, 1948. URL <https://doi.org/10.2307/2332631>. [p208, 215]
- Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 289–300, 1995. URL <https://doi.org/10.1111/j.2517-6161.1995.tb02031.x>. [p220]
- Y. Benjamini and D. Yekutieli. The control of the false discovery rate in multiple testing under dependency. *Annals of statistics*, pages 1165–1188, 2001. doi: 10.1214/aos/1013699998. URL <https://www.jstor.org/stable/2674075>. [p220]
- K. J. Berry, P. W. Mielke Jr, and H. W. Mielke. The Fisher-Pitman permutation test: an attractive alternative to the F test. *Psychological reports*, 90(2):495–502, 2002. URL <https://doi.org/10.2466/pr0.2002.90.2.495>. [p208, 215]
- G. E. Box. Some theorems on quadratic forms applied in the study of analysis of variance problems, I. Effect of inequality of variance in the one-way classification. *The annals of mathematical statistics*, pages 290–302, 1954. URL <https://www.jstor.org/stable/2236731>. [p207, 212]
- M. B. Brown and A. B. Forsythe. The small sample behavior of some statistics which test the equality of several means. *Technometrics*, 16(1):129–132, 1974a. URL <https://doi.org/10.1080/00401706.1974.10489158>. [p207, 208, 212]
- M. B. Brown and A. B. Forsythe. Robust tests for the equality of variances. *Journal of the American Statistical Association*, 69(346):364–367, 1974b. URL <https://doi.org/10.1080/01621459.1974.10482955>. [p207, 208, 212]
- M. Cavus et al. Testing the equality of normal distributed and independent groups' means under unequal variances by doex package. *The R Journal*, 12(2):134–154, 2021. URL <https://doi.org/10.32614/RJ-2021-008>. [p210, 211]
- W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *shiny: web application framework for R*, 2017. R package version 1.0.1. [p223]
- W. G. Cochran. Problems arising in the analysis of a series of similar experiments. *Supplement to the Journal of the Royal Statistical Society*, 4(1):102–118, 1937. URL <https://www.jstor.org/stable/2984123>. [p207, 208, 212]
- O. Dag, A. Dolgun, and N. M. Konar. *onewaytests: one-way tests in independent groups designs*, 2017. R package version 1.5. [p207]

- O. Dag, A. Dolgun, and N. M. Konar. onewaytests: An R package for one-way tests in independent groups designs. *R Journal*, 10(1):175–199, 2018. URL <https://doi.org/10.32614/RJ-2018-022>. [p207, 209, 217, 218, 220, 224]
- K. B. Gorman, T. D. Williams, and W. R. Fraser. Ecological sexual dimorphism and environmental variability within a community of Antarctic penguins (genus *Pygoscelis*). *PloS one*, 9(3):e90081, 2014. URL <https://doi.org/10.1371/journal.pone.0090081>. [p217]
- Y. Hochberg. A sharper Bonferroni procedure for multiple tests of significance. *Biometrika*, 75(4):800–802, 1988. URL <https://doi.org/10.1093/biomet/75.4.800>. [p220]
- S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, pages 65–70, 1979. URL <https://www.jstor.org/stable/4615733>. [p220]
- G. Hommel. A stagewise rejective multiple test procedure based on a modified Bonferroni test. *Biometrika*, 75(2):383–386, 1988. URL <https://doi.org/10.1093/biomet/75.2.383>. [p220]
- A. M. Horst, A. P. Hill, and K. B. Gorman. *palmerpenguins: Palmer Archipelago (Antarctica) penguin data*, 2020. URL <https://allisonhorst.github.io/palmerpenguins/>. R package version 0.1.0. [p217]
- G. S. James. The comparison of several groups of observations when the ratios of the population variances are unknown. *Biometrika*, 38(3-4):324–329, 1951. URL <https://www.jstor.org/stable/2332578>. [p207, 208, 213]
- G. S. James. Tests of linear hypotheses in univariate and multivariate analysis when the ratios of the population variances are unknown. *Biometrika*, 41(1-2):19–43, 1954. URL <https://www.jstor.org/stable/2333003>. [p208]
- S. Johansen. The Welch-James approximation to the distribution of the residual sum of squares in a weighted linear regression. *Biometrika*, 67(1):85–92, 1980. URL <https://doi.org/10.1093/biomet/67.1.85>. [p208, 214]
- K. Krishnamoorthy, F. Lu, and T. Mathew. A parametric bootstrap approach for ANOVA with unequal variances: Fixed and random models. *Computational Statistics & Data Analysis*, 51(12):5731–5742, 2007. URL <https://doi.org/10.1016/j.csda.2006.09.039>. [p213]
- W. H. Kruskal and W. A. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621, 1952. URL <https://doi.org/10.1080/01621459.1952.10483441>. [p207, 208, 214]
- X. Li, J. Wang, and H. Liang. Comparison of several means: A fiducial based approach. *Computational statistics & data analysis*, 55(5):1993–2002, 2011. URL <https://doi.org/10.1016/j.csda.2010.12.009>. [p213]
- L. M. Lix and H. Keselman. To trim or not to trim: Tests of location equality under heteroscedasticity and nonnormality. *Educational and Psychological Measurement*, 58(3):409–429, 1998. URL <https://doi.org/10.1177/0013164498058003004>. [p208]
- D. V. Mehrotra. Improving the Brown-Forsythe solution to the generalized Behrens-Fisher problem. *Communications in Statistics-Simulation and Computation*, 26(3):1139–1145, 1997. URL <https://doi.org/10.1080/03610919708813431>. [p208, 214]
- A. F. Özdemir and S. Kurt. One way fixed effect analysis of variance under variance heterogeneity and a solution proposal. *Selcuk Journal of Applied Mathematics*, 7(2):81–90, 2006. [p207, 208, 212]
- S. Sadooghi-Alvandi, A. Jafari, and H. Mardani-Fard. One-way ANOVA with unequal variances. *Communications in Statistics-Theory and Methods*, 41(22):4200–4221, 2012. URL <https://doi.org/10.1080/03610926.2011.573160>. [p207, 208, 209, 210]

- A. Scott and T. Smith. Interval estimates for linear combinations of means. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 20(3):276–285, 1971. URL <https://doi.org/10.2307/2346757>. [p208, 215]
- S. Weerahandi. ANOVA under unequal error variances. *Biometrics*, pages 589–599, 1995. URL <https://www.jstor.org/stable/2532947>. [p208, 216]
- S. Weerahandi and K. Krishnamoorthy. A note reconciling ANOVA tests under unequal error variances. *Communications in Statistics-Theory and Methods*, 48(3):689–693, 2019. URL <https://doi.org/10.1080/03610926.2017.1419264>. [p208, 213]
- B. L. Welch. On the comparison of several mean values - an alternative approach. *Biometrika*, 38(3-4):330–336, 1951. URL <https://www.jstor.org/stable/2332579>. [p207, 208, 211, 215]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p221]
- A. M. Winkler, G. R. Ridgway, M. A. Webster, S. M. Smith, and T. E. Nichols. Permutation inference for the general linear model. *Neuroimage*, 92:381–397, 2014. URL <https://doi.org/10.1016/j.neuroimage.2014.01.060>. [p208]

Merve Kasikci
Department of Biostatistics
Hacettepe University Faculty of Medicine, Sıhhiye, Ankara
Turkey
mervekasikci@outlook.com

Osman Dag
Department of Biostatistics
Hacettepe University Faculty of Medicine, Sıhhiye, Ankara
Turkey
osman.dag@outlook.com

H. Erkin Sulekli
Department of Biostatistics
Hacettepe University Faculty of Medicine, Sıhhiye, Ankara
Turkey
erkin.sulekli@gmail.com

Malwane M. A. Ananda
Department of Mathematical Statistics
University of Nevada, Las Vegas, NV
USA
malwane.ananda@unlv.edu

Samaradasa Weerahandi
X-Techniques
Edison, NJ
USA
weerahandi@aol.com

ggvfields: Vector Field Visualization in R

by Dusty Turner, Rodney X. Sturdivant, and David Kahle

Abstract The `ggvfields` R package extends the functionality of `ggplot2` by enabling more intuitive and effective visualizations of 2D vector fields. It introduces several new geoms designed to simplify traditional and cumbersome methods for visualizing vector fields and related objects in R in a principled way. These geoms improve upon existing tools by offering a more aesthetically pleasing and mathematically rigorous way to represent vector fields.

1 Introduction

Vectors are fundamental objects central to the modern scientific description of natural phenomena. They occur in the mathematical representation of virtually every area of inquiry, either directly in terms of the phenomenon of interest (e.g. fluid flow) or indirectly in terms of objects used to investigate or analyze those phenomena (e.g. gradients). However, the visualization of vector fields, vector-valued functions, is surprisingly under supported in the R language. In this section we introduce the basic problem of visualizing vector fields and R's current support for visualizing them. We begin with some definitions and notation.

Depending on the context, vectors can be defined in many different ways. In `ggvfields`, all vectors are two dimensional, and thus typically represented in Cartesian coordinates as pairs of real numbers. This can itself take several forms:

$$\mathbf{u} = (x, y) = x \begin{bmatrix} 1 \\ 0 \end{bmatrix} + y \begin{bmatrix} 0 \\ 1 \end{bmatrix} = x\mathbf{e}_1 + y\mathbf{e}_2$$

where x and y are the components of the vector along the x - and y -axes, respectively, and $\mathbf{e}_1$ and $\mathbf{e}_2$ represent the standard basis vectors of $\mathbb{R}^2$. Vectors can also be expressed in polar coordinates (r, θ) , where the magnitude of the vector $\mathbf{u}$, denoted $\|\mathbf{u}\| = r$, is its (Euclidean) length and the direction is the angle θ the vector forms with the positive x -axis, here ranging from $(-\pi, \pi]$. The conventional transformation to polar form and back is

$$\begin{aligned} r = r(x, y) &= \sqrt{x^2 + y^2} = \|\mathbf{u}\| & x = x(r, \theta) &= r \cos \theta \\ \theta = \theta(x, y) &= \text{see below} & y = y(r, \theta) &= r \sin \theta, \end{aligned}$$

with

$$\theta(x, y) = \text{atan2}(y, x) = \begin{cases} \tan^{-1}\left(\frac{y}{x}\right) & x > 0 \\ \tan^{-1}\left(\frac{y}{x}\right) + \pi & x < 0, y \geq 0 \\ \tan^{-1}\left(\frac{y}{x}\right) - \pi & x < 0, y < 0 \\ \frac{\pi}{2} & x = 0, y > 0 \\ -\frac{\pi}{2} & x = 0, y < 0 \\ \text{undefined} & x = 0, y = 0. \end{cases}$$

As the notation suggests, this function is implemented in R as `atan2(y, x)` (R Core Team, 2024). Similarly, it is often helpful to adhere to a notational convention for vectors. As used above, when we adopt the perspective of an object being a vector we typeset it with an upright bold font, e.g. $\mathbf{u}$ or $\mathbf{e}_1$.

Vectors and functions are often found in the same contexts. The standard mathematical notation for a function is $f: \mathcal{U} \rightarrow \mathcal{V}$, where $\mathcal{U}$ is the domain and $\mathcal{V}$ is the codomain, and the notation communicates that every element $u \in \mathcal{U}$ is assigned by f exactly one corresponding element of $\mathcal{V}$, which is typically denoted $f(u) = v \in \mathcal{V}$. Alternatively, if we wish to focus on the association of u to v by f , we write $u \xrightarrow{f} v = f(u)$ and say that u is mapped to v by f . The graph of f is the set $\mathcal{G}(f) = \{(u, v) \in \mathcal{U} \times \mathcal{V} : u \in \mathcal{U} \text{ and } v = f(u)\}$.

If $\mathcal{U}$ is a collection of vectors and $\mathcal{V}$ is $\mathbb{R}$, we say that f is a scalar field. It is a scalar-valued function of a vector argument, so we may write $f(\mathbf{u})$. For example, if $\mathcal{U} = \mathbb{R}^2$, f assigns to every point in the plane a number, and we may visualize this assignment by coloring the points of the xy -plane, e.g. a raster plot. This covers the three dimensional nature of the function: the two spatial dimensions (the x - and y -axes) visually communicate $\mathbf{u} = (x, y)$, and the color aesthetic visually communicates the value assigned to $\mathbf{u}$ by the function f . The color aesthetic is provided with a legend so that the viewer can relate the color observed to the numeric value of $f(\mathbf{u}) = f(x, y)$. This is an alternative way to visualize the graph of f $\mathcal{G}(f) = \{(x, y, z) \in \mathbb{R}^3 : x, y \in \mathbb{R}^2 \text{ and } z = f(x, y)\}$ that avoids three spatial dimensions.¹

If $\mathcal{U}$ and $\mathcal{V}$ are collections of vectors, we say that f is a vector field; it is a vector-valued function of a vector argument. We often use vector notation to emphasize this point, e.g. $\mathbf{f}(\mathbf{u})$. Of course, if $\mathcal{U} = \mathbb{R}^2$ and $\mathcal{V} = \mathbb{R}^2$ as in this work, we may write this in myriad ways:

$$\mathbf{f}(\mathbf{u}) = \begin{bmatrix} f_x(\mathbf{u}) \\ f_y(\mathbf{u}) \end{bmatrix} = \begin{bmatrix} f_x(x, y) \\ f_y(x, y) \end{bmatrix} = \mathbf{f}(x, y),$$

and we take this liberty as convenient for the setting at hand. Here f_x and f_y are called the component functions of the vector field $\mathbf{f}$. In particular, they are not the partial derivatives of a scalar field f , although the same notation is commonly used for that purpose. Any of the above formulations may be preferable in different contexts to highlight different perspectives, but the overall point is the same: vector fields assign to each point in the plane a vector.

Unlike scalar fields, which may be visualized via their graph in three dimensions using a tool such as **plotly** (Sievert, 2020) or colored raster as with **base** or **ggplot2**, vector fields are more challenging to visualize. It is the purpose of **ggvfields** to help users visualize such functions in R.

2 Visualizing vectors and vector fields in base R

Visualizing vector fields is a common task in various scientific and engineering disciplines, but it is surprisingly under-supported in the R ecosystem. After some initial reflection, visualizing vector fields is rather easy in concept but complex in details. It can also be error-prone and time-consuming when coding from scratch. In this section we review the standard strategy to do it in R using base graphics and defer a discussion about other plotting systems in a later section. We start with visualizing vector data.

2.1 Plotting vector data

As a typical applied scenario, consider wind vectors observed in various locations $\mathbf{u}_i = (x_i, y_i)$ across a spatial extent, where the wind measurements are taken in polar coordinates of speed and direction. Here is a synthetic example of such a dataset:

```
set.seed(1234)
n <- 10

# generate wind data in polar coordinates
data <- data.frame(
  "x" = rnorm(n),
  "y" = rnorm(n),
  "dir" = runif(n, -pi, pi), # angle, in radians
  "spd" = rchisq(n, df = 2) # speed
```

¹Notice that here we overload the notation f to identify $\mathbf{u}$ with x and y so that $f(\mathbf{u}) = f(x, y)$. This is relatively immaterial (even a bit pedantic) as a notational convention on paper; however, it is important in implementations as the relevant functions $\mathbf{f}$ have different arity: $\mathbf{f}(\mathbf{u})$ vs $\mathbf{f}(x, y)$. This is a common detail in R, which conventionally adopts the $\mathbf{f}(\mathbf{u})$ notation that assumes the function receives a vector argument. For example, `optim()` accepts functions that would be called as $\mathbf{f}(\mathbf{u})$, not $\mathbf{f}(x, y)$.

```
)

data |> round(digits = 2) |> head()
```

```
#>      x      y  dir  spd
#> 1 -1.21 -0.48  0.34  3.55
#> 2  0.28 -1.00  0.92  2.19
#> 3  1.08 -0.78 -1.18  2.99
#> 4 -2.35  0.06  0.77 10.81
#> 5  0.43  0.96 -1.07  3.45
#> 6  0.51 -0.11  0.01  3.91
```

A common way to visualize this kind of data places translated arrows at each evaluation point, with their tails at each of the observed locations $\mathbf{u}_i = (x_i, y_i)$ and heads at the points $\mathbf{u}_i + \mathbf{f}(\mathbf{u}_i) = (x_i + f_x(x_i, y_i), y_i + f_y(x_i, y_i))$. However, since the Cartesian coordinate system is used in the representation of $\mathbf{f}$ —as it is also in most plotting systems—we first need to convert the polar coordinates into Cartesian coordinates before we manually plot each vector. This is illustrated in Figure 1.

```
# transform into Cartesian coordinates for plotting
data <- data |> transform( "fx" = spd*cos(dir), "fy" = spd*sin(dir) )

data |> round(digits = 2) |> head()

#>      x      y  dir  spd  fx  fy
#> 1 -1.21 -0.48  0.34  3.55  3.35  1.17
#> 2  0.28 -1.00  0.92  2.19  1.33  1.74
#> 3  1.08 -0.78 -1.18  2.99  1.13 -2.76
#> 4 -2.35  0.06  0.77 10.81  7.79  7.49
#> 5  0.43  0.96 -1.07  3.45  1.66 -3.02
#> 6  0.51 -0.11  0.01  3.91  3.91  0.05

# plot data
with(data, {
  plot( NULL, xlab = "x", ylab = "y", xlim = range(c(x, x + fx)), ylim = range(c(y, y + fy)) )
  arrows(x, y, x + fx, y + fy, length = .10)
})
```

Note that `plot()` is used above to initialize the plot over a reasonable spatial extent. Using `arrows()` alone results in the (simplified) error `Error in arrows(...): plot.new has not been called yet`, a minor but not insignificant stumbling block to the uninitiated. More, simply calling `plot.new()` before `arrows()` doesn't resolve the problem, as it cannot anticipate the spatial extent needed for the arrows.

As shown in the creation of Figure 1, plotting a collection of vectors in base R^2 requires multiple steps: converting coordinates, manually defining plot limits to accommodate the entire arrow, and drawing each vector with `arrows()`. The approach is cumbersome and easy to mess up, and the result leaves a lot to be desired. These drawbacks are exacerbated when visualizing vector fields, as we see next.

2.2 Plotting a vector field

In other cases, we need to visualize a vector field generated by a function $\mathbf{f}$. Myriad good examples exist: wind patterns over a geographic region, the flow of bodies of water, electric fields, etc. One example is provided by the simple rotational field $\mathbf{f}(x, y) = (-y, x)$.

²We talk about other R packages in a later [section](#) that plot vectors and vector fields but all struggle with similar problems.

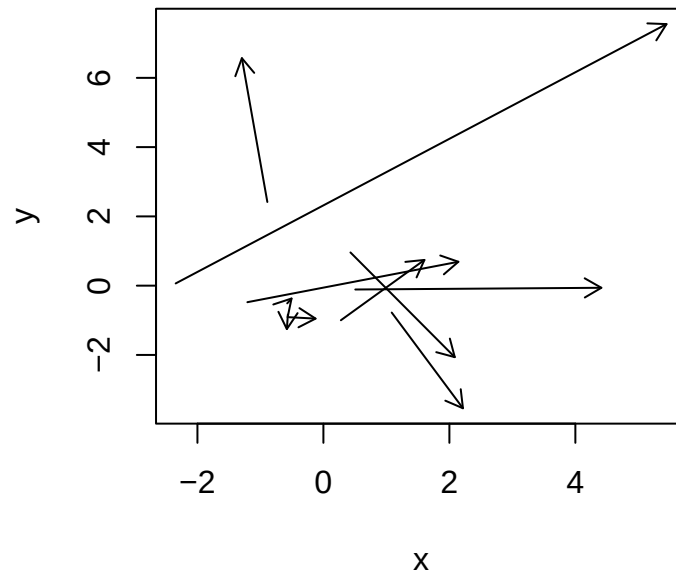


Figure 1: Plotting (raw) vector data in base R.

Visualizing a vector field f is similar in many respects to plotting individual vectors; however, there are some important differences that suggest how one might want to modify the graphic previously described to make it better. To begin, and unlike the data case, obviously we can't possibly visualize f at all points $\mathbf{u} = (x, y)$, so we have to choose which ones we would like to use to represent the whole of f . The domain is usually sampled in some way to generate values $\mathbf{u}_i = (x_i, y_i)$ that can be visualized in the same way vector data is. A regular rectangular mesh over some spatial extent $c(xmin, xmax)$ and $c(ymin, ymax)$ is typical, and its values are stored in a data frame. Figure 2 is an example that illustrates the rotational field above.

```
# define vector field of single vector argument
f <- function(u) {
  x <- u[1]; y <- u[2]
  c(-y, x)
}

# make data frame grid with space for (fx(x,y), fy(x,y))
N <- 11
df <- expand.grid( "x" = seq(-1, 1, len = N), "y" = seq(-1, 1, len = N) )
df$fy <- df$fx <- NA # make placeholders for fx(x,y) and fy(x,y) values

# evaluate f(u) at each point and look at result
for (i in 1:nrow(df)) df[i,3:4] <- f( as.numeric(df[i,1:2]) )

head(df)

#>      x  y fx  fy
#> 1 -1.0 -1  1 -1.0
#> 2 -0.8 -1  1 -0.8
#> 3 -0.6 -1  1 -0.6
#> 4 -0.4 -1  1 -0.4
#> 5 -0.2 -1  1 -0.2
#> 6  0.0 -1  1  0.0

# visualize using base R
with(na.omit(df), {
```

```
plot( NULL, xlab = "x", ylab = "y", xlim = range(x, x + fx), ylim = range(y, y + fy) )
  arrows(x, y, x + fx, y + fy, length = .05)
})
```

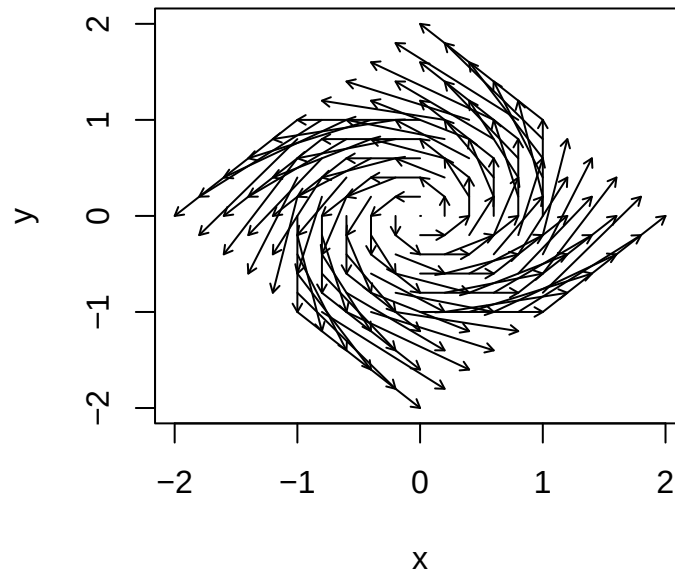


Figure 2: Vector fields can be visualized by sampling their domain, evaluating the vector field, and plotting the resulting vectors as translated arrows. Here the rotational field $f(x, y) = (-y, x)$ is visualized using base R graphics using the same strategy as Figure 1.

While Figure 2 is accurate in some technical sense, the overplotting of arrows and varying magnitudes of the vectors cause significant visual clutter and make it difficult to actually interpret the function. The overlapping of arrows obscures the finer details of the vector field, especially in regions of significant overplotting, and this is a mild example. The varying magnitudes of the vectors demand the axes accommodate the largest vectors, which can easily cause visual swamping of the smaller ones. Although it doesn't happen in this example, it is in fact the more typical case. This one-two punch of overplotting and spatial extent swamping renders most of these graphics essentially unusable as a data visualization tool.

Visualizing vector data or vector fields is challenging in base R. The manual steps required to convert data, calculate vectors, and plot them are burdensome and error-prone; their remedies are, too. These challenges highlight the need for a more efficient and streamlined approach to vector field visualization. This is exactly what **ggvfields** provides: a more intuitive, efficient, flexible, attractive, and error-resistant solution for creating high-quality vector field visualizations within the **ggplot2** framework.

3 Visualizing vectors and vector fields using **ggvfields**

3.1 Plotting vector fields with **ggvfields**

Let's revisit the previous plots using **ggvfields** functions, in reverse. When presented with a vector field f , **ggvfields** plots the function directly, performing all the computations automatically behind the scenes. This is illustrated in Figure 3.

```
library("ggvfields")

ggplot() + geom_vector_field(fun = f, xlim = c(-1, 1), ylim = c(-1, 1))
```

Most **ggplot2** users will notice that `geom_vector_field()` mimics **ggplot2**'s built-in function `geom_function()`. This is a general design choice of **ggvfields**: where there is

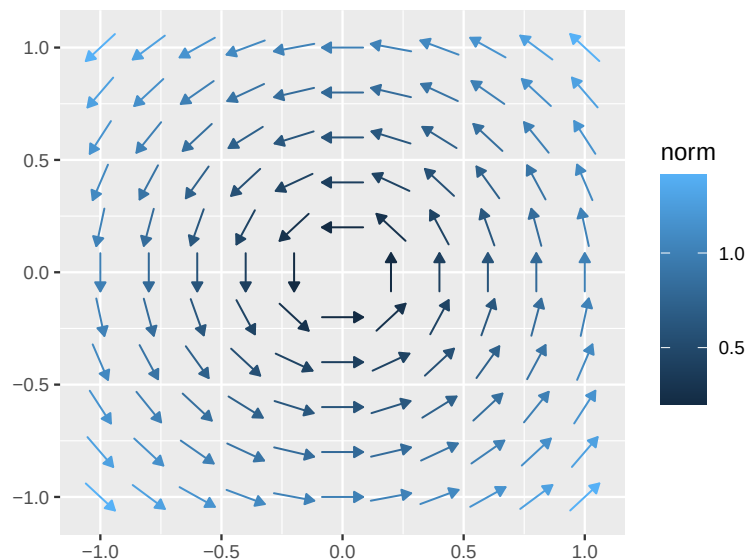


Figure 3: `ggvectorfield`'s `geom_vector_field()` visualizes vector fields. This plot is analogous to Figure 2, but generated using `ggvectorfield`.

an analogy to be made with pre-existing functions in `ggplot2`, `ggvectorfield` functions lend significant deference to that implementation in order to be more intuitive to `ggplot2` users. In that sense, `ggvectorfield` is a member of the growing suite of `ggplot2` extension packages, such as `ggdensity`, `ggdist`, and `patchwork` (Otto and Kahle, 2023; Kay, 2023; Pedersen, 2024). However, among extensions it is more on the infrastructural end, being intended for general purposes and not specific use cases, and contains tools that feel a bit more like what someone one might expect in `ggplot2` itself.³

This relationship to `ggplot2` can easily be seen through concrete examples. Plotting the function $f(x) = \sin(x)$, for example, uses the syntax `ggplot() + geom_function(fun = sin, xlim = c(0, 2*pi))`. The typical usage of `geom_function()` requires two arguments: `fun` and `xlim`. Internally, `ggplot2` creates a uniform sequence of points $x_1, \dots, x_n$ from `xlim[1]` to `xlim[2]`, evaluates `fun` at those points to obtain ordered pairs $(x_1, f(x_1)), \dots, (x_n, f(x_n))$, and draws small line segments between the points to visually simulate a smooth curve. This sequence of connected line segments is called a *polyline*. So conceptually `geom_function()` is simply a wrapper around `geom_path()`, the `ggplot2` function used to plot polylines via `grid`'s `polylineGrob()`. n is set by the default argument in `geom_function()`⁴, $n = 101$.

Conceptually `geom_vector_field()` works just like `geom_function()` but with a `ylim` argument supporting the second input dimension. Thus, it accepts `fun`, `xlim`, and `ylim`. It then internally creates a regular rectangular grid of points as the Cartesian product of 2 uniform meshes on `xlim` and `ylim`, each of size n (defaulted to 11), evaluates `fun` on that grid to obtain vectors, and plots the vectors by wrapping the more basic `geom_stream()`, which is a `ggvectorfield` function we describe shortly. `xlim` and `ylim` default to `c(-1, 1)`, which allows us to drop them in the following examples to focus on other aspects of the code. `geom_vector_field()` also facilitates plotting vector fields over regular hexagonal meshes, which can be created manually with the `grid_hex()` function and passed into `geom_vector_field()` using the `grid` argument, or simply by stating `grid = "hex"`. This is illustrated in Figure 4. In our experience, either of the grid types may be preferable for a given field. Users may also specify their own grids with a data frame of domain points using `grid`.

In contrast to our made-from-scratch base vector field plot, `geom_vector_field()` doesn't overplot. It achieves this by drawing arrows not from $\mathbf{u}_i$ to $\mathbf{u}_i + \mathbf{f}(\mathbf{u}_i)$ but from $\mathbf{u}_i$ to $\mathbf{u}_i + L \text{ sign}(\mathbf{f}(\mathbf{u}_i))$, where L is a positive constant depending on `xlim`, `ylim`, n , and the grid

³At the time of this writing, `ggplot2` is more or less a closed canon, as development and maintenance resources are scarce.

⁴Actually `stat_function()`, but the difference is not material.

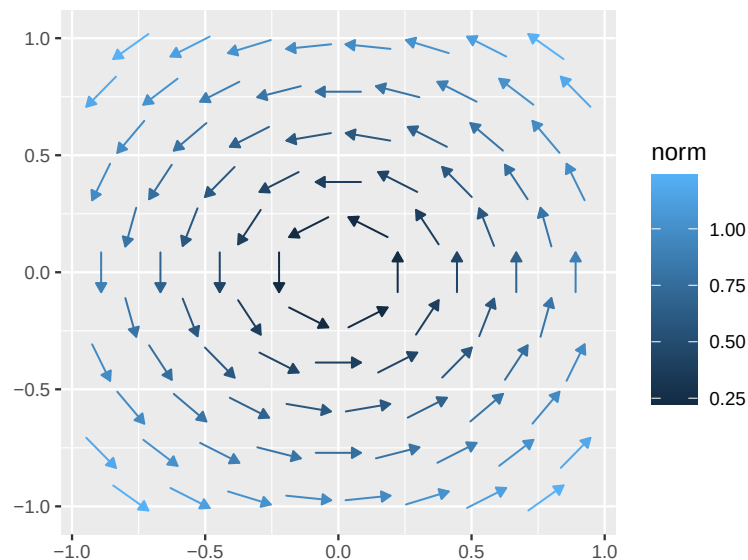


Figure 4: `geom_vector_field()` supports hexagonal grids by specifying `grid = "hex"`.

type that is automatically chosen to avoid overplotting. Mathematically, L is the length of the arrows in plot, since the sign function normalizes $\mathbf{f}(\mathbf{u}_i)$, and these lengths are with reference to *the domain's data space*. Setting `normalize = FALSE` and removing color can be used to create a plot similar to the one we observed with base graphics, see Figure 5.

```
ggplot() + geom_vector_field(fun = f, normalize = FALSE, color = "black")
```

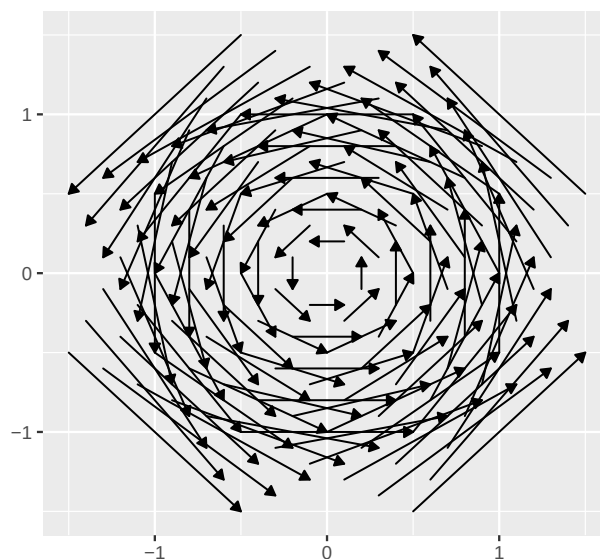


Figure 5: Setting `normalize = FALSE` and `color = "black"` results in a plot similar to the vector fields created with `base`'s `arrow()` in Figure 2.

Besides being a mess, Figure 5 reveals another feature of the way `ggvfields` draws vector fields: that the arrows in `ggvfields` don't run from $\mathbf{u}_i$ to $\mathbf{u}_i + L \text{sign}(\mathbf{f}(\mathbf{u}_i))$, but rather from $\mathbf{u}_i - \frac{L}{2} \text{sign}(\mathbf{f}(\mathbf{u}_i))$ to $\mathbf{u}_i + \frac{L}{2} \text{sign}(\mathbf{f}(\mathbf{u}_i))$. They are *centered* at the evaluation point $\mathbf{u}_i$. Disabled by setting `center = FALSE`, this is a default design choice that avoids drawing the eye away from the evaluation point $\mathbf{u}_i$, which also can create visual artifacts such as the oblique look created when the normalized vectors are based at the evaluated points. This effect is illustrated in Figure 6.

```
library("patchwork")
```

```
p1 <- ggplot() + geom_vector_field(fun = f
                                   ) + ggtitle("center = TRUE")
p2 <- ggplot() + geom_vector_field(fun = f, center = FALSE) + ggtitle("center = FALSE")
p1 + p2 + plot_layout(guides = "collect")
```

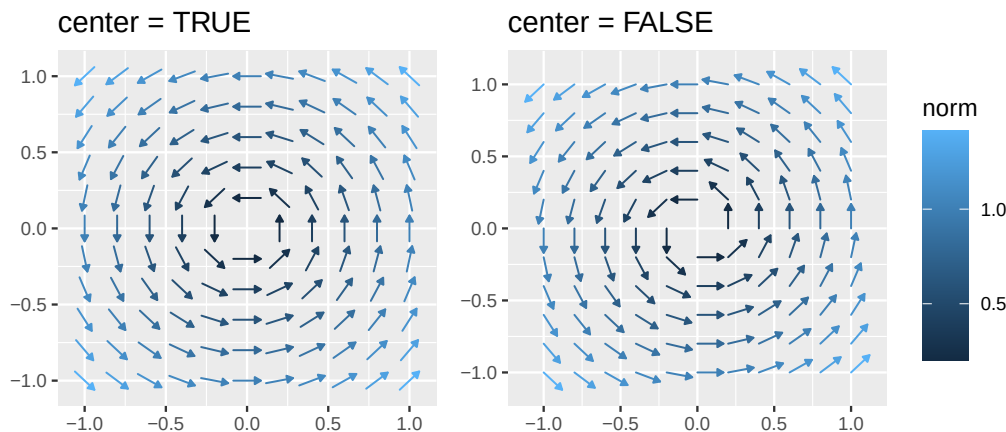


Figure 6: By default, vector glyphs (e.g. arrows) are centered over the point for which they are evaluated.

The reason that the (non-zero) vectors can be normalized despite them varying in magnitude is that their magnitudes are mapped to the color aesthetic and a legend is provided, consistent with the grammar of graphics (Wilkinson, 2011; Wickham, 2010). In other words, the move from arrows $\mathbf{u}_i \pm \frac{1}{2}\mathbf{f}(\mathbf{u}_i)$ to arrows $\mathbf{u}_i \pm \frac{1}{2}\text{sign}(\mathbf{f}(\mathbf{u}_i))$ is permissible because the other piece of information in $\mathbf{f}(\mathbf{u}_i)$, namely its magnitude $\|\mathbf{f}(\mathbf{u}_i)\|$, is contained in the legend. This provides an elegant albeit imperfect solution to the overplotting problem: instead of using position to communicate $\|\mathbf{f}(\mathbf{u}_i)\|$, use color. This may feel strange, because, all else being equal, differences in position tend to be easier to perceive than differences in color, but in this graphic, magnitudes are communicated by color rather than by length (Cleveland and McGill, 1984; Franconeri et al., 2021). This has advantages and disadvantages. One advantage is that overplotting is dramatically reduced; another is that vector direction can still be seen for small vectors, i.e. those that would be swamped by the big scale differences in previous graphics. A disadvantage is that small changes in magnitude (color) are potentially more difficult to perceive, especially with a simple monochrome scale such as the default `ggplot2` scale. This is one situation where binned scales or highly polychromatic scales such as the spectral scale, which is visually richer but generally frowned upon in the data visualization community for various reasons, may in fact be helpful (Liu and Heer, 2018). Since `ggvfields` is built on top of `ggplot2`, these are easily manipulable with `ggplot2`'s scale functions; these are shown in Figure 7.

```
p1 <- ggplot() + geom_vector_field(fun = f)

p2 <- p1 + scale_color_viridis_b(n.breaks = 3)
p3 <- p1 + scale_color_gradientn(colors = rainbow(10), n.breaks = 6)

p2 + p3 & coord_equal()
```

In cases where users want to avoid the color aesthetic, e.g. for black and white printing, `ggvfields` provides a second option: `geom_vector_field2()`. Instead of the arrows provided by `geom_vector_field()` that range $\mathbf{u}_i \pm \frac{1}{2}\text{sign}(\mathbf{f}(\mathbf{u}_i))$, `geom_vector_field2()` plots line segments from $\mathbf{u}_i$ to $\mathbf{u}_i + \alpha_i L \text{sign}(\mathbf{f}(\mathbf{u}_i))$, where $\alpha_i = \|\mathbf{f}(\mathbf{u}_i)\| / \max_{1 \leq i \leq n} \|\mathbf{f}(\mathbf{u}_i)\|$ ranges from 0 to 1. We refer to the resulting vectors as being normalized and *scaled*. The biggest vector is mapped to the previous length L , the length that avoids overplotting, and other vectors are mapped to a fraction of this length depending on their relative magnitudes. Direction is communicated by a dot at the tail point $\mathbf{u}_i$ instead of an arrow, and the segments are not centered. This is illustrated in Figure 8.

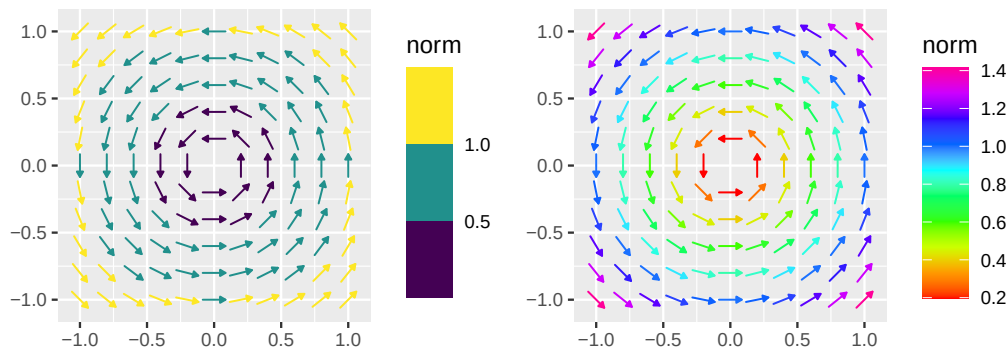


Figure 7: Polychromatic scales (right) can help visually discern relative magnitudes of the vector fields, as can using a binned scale (left). These are alternatives to Figure 6 (left).

```
ggplot() + geom_vector_field2(fun = f)
```

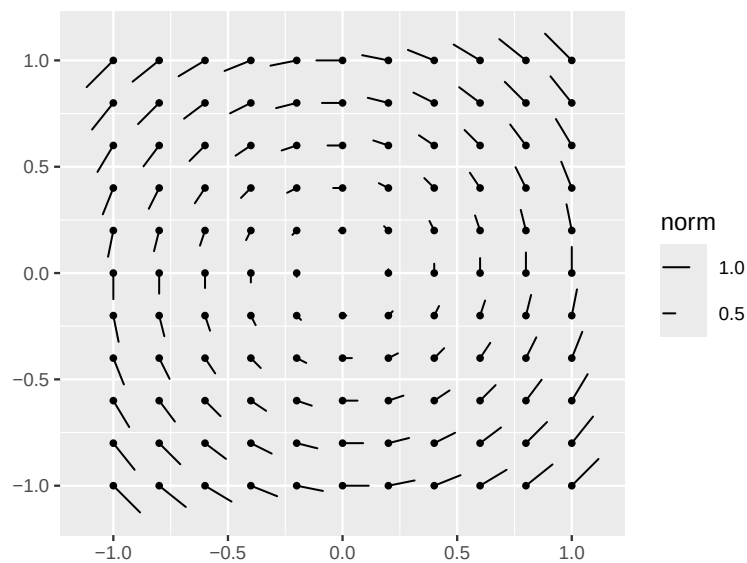


Figure 8: `geom_vector_field2()` provides a more print friendly alternative to `geom_vector_field()`.

Viewers associate the scaled lengths to norm values $\|f(u_i)\|$ via a legend. This is a bit tricky since length is not a `ggplot2` aesthetic, so the lengths need to be computed at the `grid` level so that the segments are independent of the device size (or more generally aspect ratio). In other words, the absolute length of the line segments displayed in the legend should correspond exactly, not proportionately, to the line segments displayed on the graphic, irrespective of the size of the device. It is easy to verify that this is the case for `geom_vector_field2()`, this is illustrated in Figure 9.

```
# make the plot to be resized
p <- ggplot() + geom_vector_field2(fun = f, n = 5)

# draw the plot in two different sizes
(p + p & theme(plot.background = element_rect(color = "gray75"))) +
  plot_layout(design = c(area(1,1,1,1), area(2,2,3,3)))
```

These plots look nice but carry with them similar perceptual problems to the color version `geom_vector_field()`. While the difficulty of `geom_vector_field()` is associating shades of color to a point on a color bar, the difficulty of `geom_vector_field2()` is associating the lengths of small line segments in the plot to line segments in the legend, mentally interpolating or even extrapolating for line segments whose lengths are not explicitly in

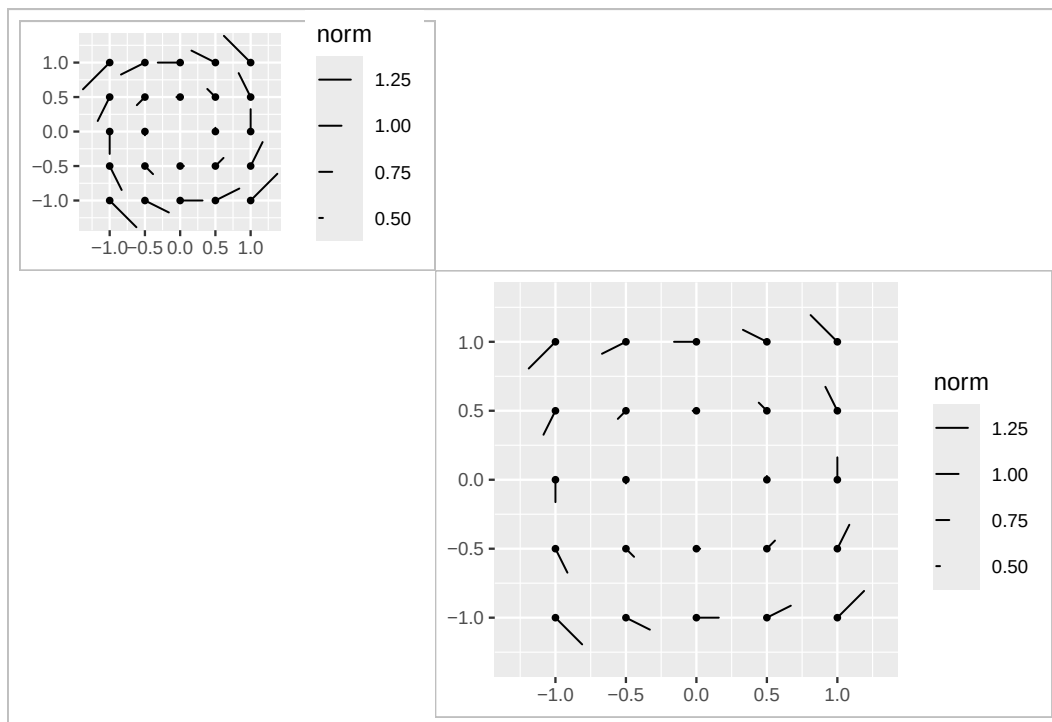


Figure 9: The length aesthetic of `geom_vector_field2()` is independent of aspect ratio, which defaults to the size of the device.

the legend, which is of course challenging. This is further complicated by the requirement that the lengths of the line segments in the graphic need to correspond exactly to those in the legend for reasons described above, which limits how long the line segments can be in the legend and forces them to be smaller than might be preferred. Setting them longer in the legend may make differences in the graphic easier to perceive, but it also has the consequence of making the legend enormous. This is illustrated in Figure 10.

```
ggplot() +
  geom_vector_field2(fun = f, grid = "hex") +
  scale_length_continuous(max_range = 1.5) # max_range = L
```

Obviously, no strategy is perfect and comes without disadvantages. One of the objectives of **ggvfields** is to provide the user with a range of tools so they can decide what is appropriate for their particular situation.

3.2 Plotting vector data with **ggvfields**

`geom_vector_field()` and `geom_vector_field2()` have data analogues `geom_vector()` and `geom_vector2()` that can be used when analyzing vector data, and those functions follow the same general conventions with the same arguments. Instead of a function argument `fun` like `geom_vector_field()`, `geom_vector()` requires an `x` and `y` aesthetic representing $\mathbf{u} = (x, y)$ and a corresponding vector value $\mathbf{f}(\mathbf{u}) = (f_x(x, y), f_y(x, y))$ specified by aesthetics `fx` and `fy`, as in Figure 11.

```
p1 <- ggplot(data) + geom_vector(aes(x, y, fx = fx, fy = fy))
p2 <- ggplot(data) + geom_vector2(aes(x, y, fx = fx, fy = fy))
p1 + p2
```

Like their vector field counterparts, `geom_vector()` normalizes and centers by default and uses color and arrow heads, whereas `geom_vector2()` normalizes and scales with

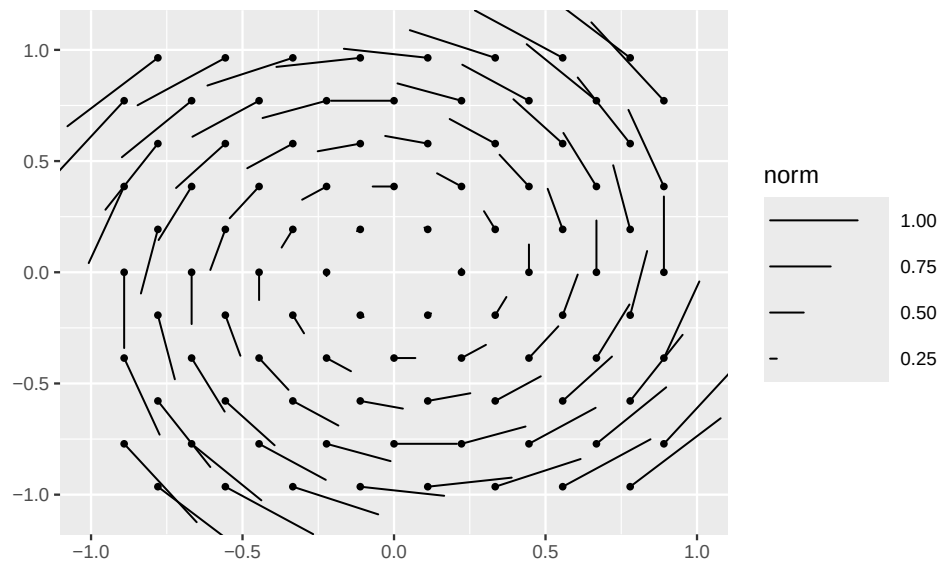


Figure 10: The length aesthetic can be modified with the new scale function `scale_length_continuous()`.

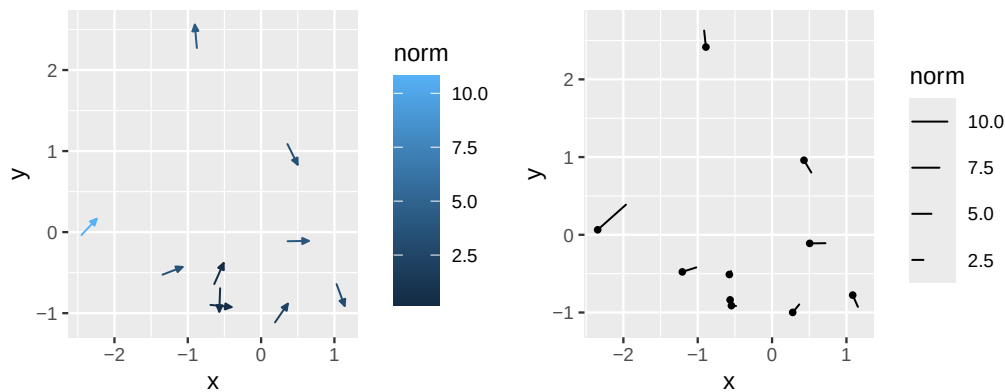


Figure 11: `geom_vector()` (left) and `geom_vector2()` (right) are analogues of `geom_vector_field()` and `geom_vector_field2()` that are used with data. Compare these graphics to Figure 1.

the length aesthetic, doesn't center, and uses a tail point. As so much vector data is parameterized by angle/direction and magnitude/speed/distance/norm, the data values can also be specified in that parameterization with new aesthetics `angle` and `norm`. Thus, the following code produces precisely the same plot, see Figure 12. The calculations used to perform the transformation are precisely those discussed in the first section of this article.

```
p1 <- ggplot(data) + geom_vector(aes(x, y, angle = dir, distance = spd))
p2 <- ggplot(data) + geom_vector2(aes(x, y, angle = dir, distance = spd))
p1 + p2
```

Angles are assumed to be in radians ($(-\pi, \pi]$) by default; however, if the user has angles in degrees they can use the simple helper aesthetic `angle_deg`.

We now turn to other strategies used to visualize vector fields, in particular, the use of streams.

4 Vector fields, calculus, and stream plots

Calculus is the field of mathematics that studies continuous change, typically represented by ordinary or partial differential equations, and vector fields naturally arise in these contexts. In this section we present these two additional perspectives, dynamical systems

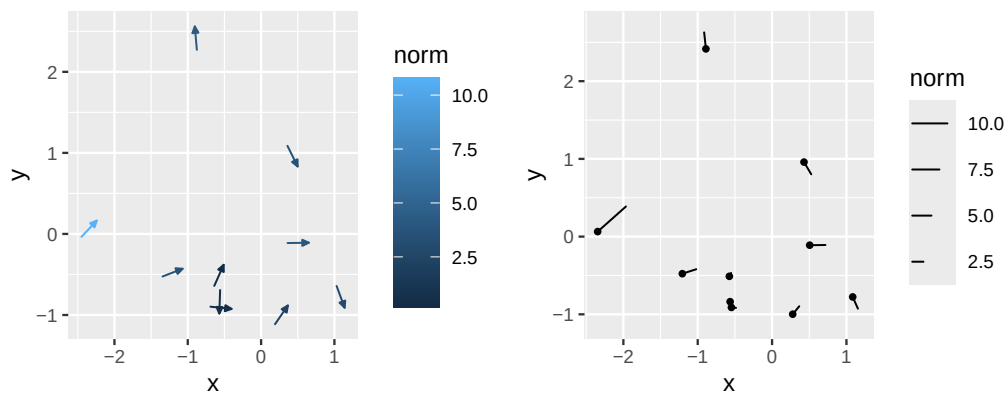


Figure 12: `geom_vector()` (left) and `geom_vector2()` (right) accept vectors in polar coordinates.

and conservative vector fields, for which `ggvfields` provides additional tools. While common in popular media and in some textbooks ranging from multivariate calculus to engineering to environmental science, these graphics are surprisingly rarely seen in statistical data science, probably because, like vector fields, the core object—the stream—is not directly observed as data. Consequently, there appears to be no base R implementation for many of these kinds of graphics.

4.1 Dynamical systems

One natural way of visualizing vector fields is to bend the arrows presented [above](#) as a kind of flow. Figure 13 provides an example of such a visualization enabled by `ggvfields` using `geom_stream_field()`.

```
p1 <- ggplot() + geom_vector_field(fun = f)
p2 <- ggplot() + geom_stream_field(fun = f)
p1 + p2 & coord_equal()
```

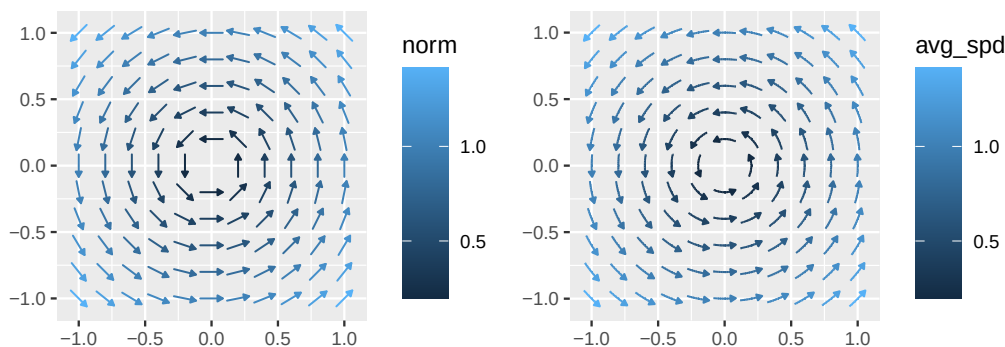


Figure 13: `geom_stream_field()` (right) creates stream line plots, a smoothly flowing alternative to `geom_vector_field()` (left).

A more formal mathematical description of this perspective, which we call the dynamical systems perspective, is helpful. Up until this point, we have simply viewed the vector field $\mathbf{f} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ as a function that accepts and returns a 2 dimensional vector. In the dynamic systems view, the component functions of the vector field $\mathbf{f}(x, y) = (f_x(x, y), f_y(x, y))$ are understood to be the time derivatives of the x - and y -coordinates of the position of a particle at time t , i.e. $(x(t), y(t))$. Like the presentation in the introduction, this can be written in many ways. One common way is $x' = f_x(x, y)$ and $y' = f_y(x, y)$, where x is shorthand for $x(t)$ and similarly for y . More explicitly, we could write

$$\begin{aligned} x'(t) &= f_x(x(t), y(t)) \\ y'(t) &= f_y(x(t), y(t)) \end{aligned}$$

In this view, the vector field is a function on a phase space, the phase plane often studied in a first semester differential equations course. The vector field can be thought of as the velocity field of a particle moving in time, and the trajectory of the particle, a directed path we call a stream, can be visualized as the image of a function $\mathbf{u}(t) = (x(t), y(t))$ over a period of time $[0, T]$ time by solving the system of differential equations from an initial point $\mathbf{u}(0) = \mathbf{u}_0 = (x_0, y_0) = (x(0), y(0))$.⁵ In other words, streams can be computed by solving initial value problems (IVPs), which can be done with any number of numerical integrators in R.

As an example, Figure 14 plots the stream of a point originating at $\mathbf{u}_0 = (x_0, y_0) = (-\frac{1}{2}, \frac{1}{2})$ through our system $\mathbf{f}(x, y) = (-y, x)$ up to time $T = 6$, computed numerically using `deSolve` (Soetaert et al., 2010, 2012). Clearly, the particle is moving in a circle. This is because the IVP, $x' = -y$ and $y' = x$ with $x(0) = x_0$ and $y(0) = y_0$, is solved when $x(t) = x_0 \cos(t) - y_0 \sin(t)$ and $y(t) = y_0 \cos(t) + x_0 \sin(t)$, and so starting at any point $\mathbf{u}_0 = (x_0, y_0)$, the particle moves counter-clockwise in perfect circles with radius equal to the norm $\|\mathbf{u}_0\| = \sqrt{x_0^2 + y_0^2}$, since $x(t)^2 + y(t)^2 = x_0^2 + y_0^2$. Unfortunately, solutions cannot usually be obtained via nice closed form solutions, as is typically the case with applied problems.

```
library("deSolve")

# create a wrapper function for deSolve::ode(), scale = -1 runs system in reverse
f_wrapper <- function(scale = 1) {
  function(t, y, parms, ...) list(scale*f(y))
}

# define a time scale over which to solve the system and the initial position
T <- 6
ts <- seq(0, T, .01)
u0 <- c(-1/2, 1/2)

# solve the ODE system and convert into a data frame, suppressing ode() messages
invisible(capture.output( ode_out <- ode(u0, ts, f_wrapper()) ))
df_ode <- ode_out |> as.data.frame() |> `names<-`( c("t", "x", "y") )

# superimpose the solution on the vector field
ggplot() +
  geom_vector_field2(fun = f, xlim = c(-1,1), ylim = c(-1,1), alpha = .15) +
  geom_stream(aes(x, y, t = t, color = t), data = df_ode)
```

In `ggvfields` streams are plotted as finely sampled polylines with `geom_stream()`, an analogue of `geom_vector()` requiring (typically) aesthetics x , y , and t , a new aesthetic.⁶ These curves $\mathbf{u}(t)$ nominally range from $t = 0$ to $t = T$ starting from some initial point $\mathbf{u}_0$. Of course, such curves exist originating from each point $\mathbf{u} \in \mathcal{U}$ in the domain provided $\mathbf{f}(\mathbf{u}) \neq \mathbf{0}_2$, so in the same way that visualizing a vector field using arrows involves restricting one's view to a sampling of the domain, visualizing vector fields with streams is done by visualizing the trajectories of points initialized at different points on a sampling of the domain, and so the grid argument is available for stream plots as well. Since the system is autonomous,⁷ if the vector field is smooth⁸ the streams are guaranteed not to cross so long

⁵There are some subtle differences between the meanings of "streamlines," "streaklines," and "pathlines" in the fluid dynamics literature, and our choice to call these "streams" is perhaps unfortunate, as they are more accurately "paths." Nevertheless, we retain the current nomenclature because (1) "path" is already taken in the `ggplot2` ecosystem, (2) other packages (e.g. `metR`) use the same nomenclature, and (3) for time-independent fluid flows (i.e. autonomous systems, like the framework above), the two are the same.

⁶In fact, the implementation of `geom_vector()` is a simple wrapper of `geom_stream()`.

⁷A system of ODEs $\mathbf{u}'(t) = \mathbf{F}(t, \mathbf{u})$ is autonomous if $\mathbf{F}(t, \mathbf{u})$ is independent of t , i.e. the system doesn't involve any t 's on the right hand side that aren't in $\mathbf{u}(t)$.

⁸Or more generally Lipschitz continuous in (x, y) , which would be the case if the functions have continuous first partial derivatives, for example.

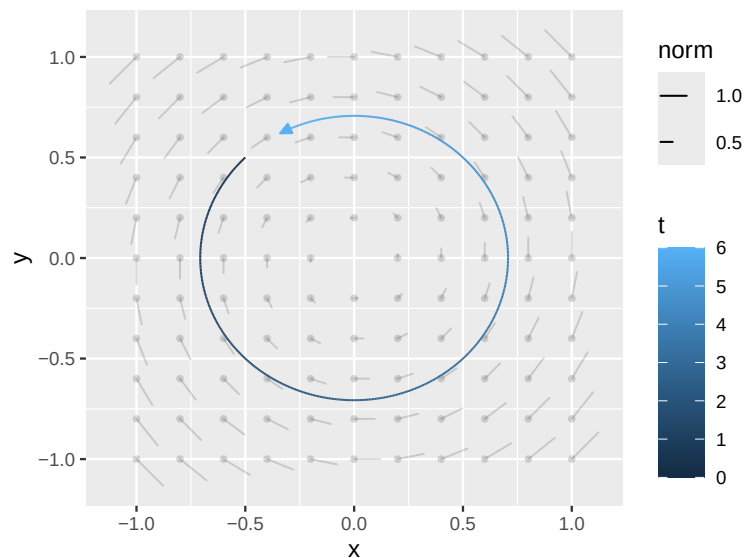


Figure 14: The streams in stream line plots are computed by solving IVPs initialized at points on a grid. Here a single stream is computed manually using `deSolve`'s `ode()`.

as they don't cross singular points where the vector field vanishes; this is a consequence of the Picard–Lindelöf existence and uniqueness theorem (Arnold, 1973). So while solutions starting from different grid points $\mathbf{u}_i$ may merge if they are on the same solution path, overplotting due to crossing solutions typically cannot occur. Solutions can, of course, become close.

While visually appealing, creating stream plots presents myriad challenges. Most of these are related in some way to solving the IVPs. In `ggvfields` `deSolve`'s `ode()` is used for this purpose, since it uses a sophisticated general purpose solver appropriate for both non-stiff and stiff ODEs, and so abstracts away the problem of solving the IVP $\mathbf{u}'(t) = \mathbf{f}(\mathbf{u})$ with $\mathbf{u}(0) = \mathbf{u}_0$ from $t = 0$ to T . The question then is: how do we select T ? The answer is by analogy with arrow-based graphics.

Viewed properly, the dynamical systems perspective can in fact generalize the arrow-based graphics made with `geom_vector_field()` in interesting ways related to specifying T . First, the concept of normalization applies to stream plots in a way very similar to vector plots. With `geom_vector_field()`, setting `normalize = TRUE` scales the length of each vector to L (in the data space) and then maps the actual lengths (norms) to the color aesthetic. The analogue for streams is to track the movement of the particles from their initial point $\mathbf{u}_i$ to a point T_i for which the length of the stream is L . This is tricky for two reasons. First, the system of ODEs is parameterized—and more importantly solved with `deSolve`—by reference to T , not the arc length L . Second, there is no guarantee that streams originating from every point can go length L even if we let t grow to infinity. For example, if for some point in the stream $\mathbf{f}(x, y) = \mathbf{0}_2$, called a sink, the trajectory will necessarily stop.

The first problem can be resolved in the following way. `deSolve::ode()` is generally designed to solve IVPs given $\mathbf{u}_0$ and T ; however, it is a flexible function and our IVP can be reformulated into a system that solves what we want. For any smooth curve $\mathbf{u}(t) = (x(t), y(t))$, a basic fact from multivariate calculus is that the arc length of the curve is $l(t) = \int_0^t \sqrt{x'(s)^2 + y'(s)^2} ds$. By the fundamental theorem of calculus, this means that $l'(t) = \sqrt{x'(t)^2 + y'(t)^2}$, but since these are the component functions of the vector field, it is equivalent to expanding the system

$$x' = f_x(x, y), \quad y' = f_y(x, y), \quad \text{and} \quad l' = \sqrt{f_x(x, y)^2 + f_y(x, y)^2}.$$

`deSolve::ode()` accepts a function `rootfun` that automatically terminates computing a solution when a certain condition is met, namely the root (zero) of a function is achieved.

If this function is set to $r(t, (x, y, l)) = l - L$, the solver stops when $l = L$, i.e. the arc length of the stream is L , a pre-specified length. In practice, L is automatically set in `geom_stream_field()` much like it is in `geom_vector_field()`, to avoid overplotting based on `xlim`, `ylim`, and `n`, or the grid in general.

In the arrows case, the norm of the vector was mapped to color. In the streams case, assuming all streams do grow to length L , a similar quantity can be computed with the time T_i it takes for the stream to grow to length L . Specifically, we can divide each stream L by the time T_i it took for that stream to reach L from the starting point $\mathbf{u}_i$. The result is interpretable as the average speed the particle moved over the stream and is why the legend is labeled `avg_spd` in Figure 13. As a consequence, for constant vector fields, the arrow and stream versions of visualizing vector fields are in fact the same, as can be seen in Figure 15. For the second problem, if L is not achieved, we can simply take the length achieved and divide it by the time it took to reach that length (a stopping criteria similar to the root function will stop the stream if it doesn't move). These streams will be shorter than the others.

```
constant_field <- function(u) c(3,4)
p1 <- ggplot() + geom_vector_field(fun = constant_field)
p2 <- ggplot() + geom_stream_field(fun = constant_field)
p1 + p2 & coord_equal()
```

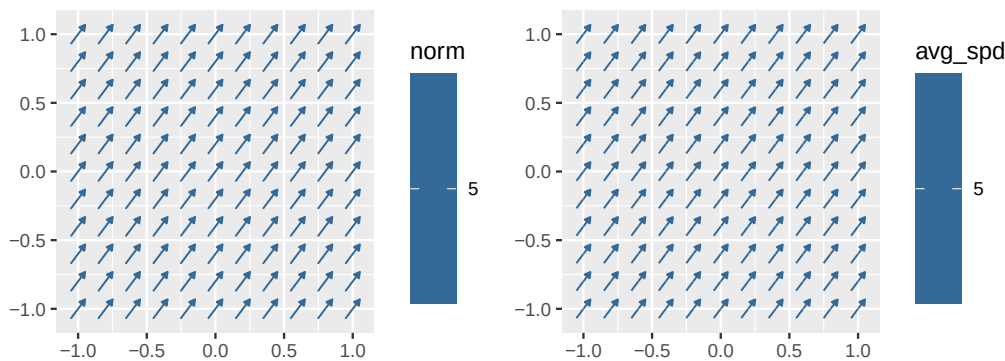


Figure 15: `geom_stream_field()` and `geom_vector_field()` produce the same plot for constant vector fields, although they are computed differently.

Taken together, the considerations above solve the normalization problem: by solving the system up to a pre-specified arc length L . This same strategy can be used to center the streams. In the case of vectors, `center = TRUE` translated the arrow half way towards its tail, which centered the arrow on the evaluation point. This cannot be done with streams, since the stream would indicate a trajectory through which the center point doesn't in fact move. However, a similar effect can be achieved by computing the solution up to length $L/2$ in forward time and backwards up to $L/2$ in reverse time, and then stitching the two streams together. To run the stream in reverse time, we simply run the negative of the original system $-\mathbf{f}(x, y)$ in forward time. The result is an arc length L stream whose arc length center passes through $\mathbf{u}_i$.

Creating an analogue to `geom_vector_field2()` is a bit trickier. The key feature of `geom_vector_field2()` is that it normalizes *and* scales each of the resulting arrows. A similar effect can be simulated with streams by (1) automatically determining an L as usual based on the grid; (2) computing all streams up to length L ; (3) identifying the smallest time required to achieve L across all the streams (i.e. the time associated with the fastest stream); and (4) cropping all streams back to their positions at this smallest time. The result is `geom_stream_field2()`, displayed in Figure 16. These look a bit different than those created with `geom_vector_field2()`, which did not use color. This is because the length aesthetic does not apply to `geom_stream_field2()`, and so without color the user has no way of knowing the magnitudes of the average speeds. However, an effect similar to `geom_vector_field2()` can be achieved by setting the color to "black". While this plot

allows the determination of relative magnitudes of the field across the spatial extent, it does not communicate their absolute magnitudes.

```
p1 <- ggplot() + geom_vector_field2(fun = f)
p2 <- ggplot() + geom_stream_field2(fun = f)
p1 + p2 & coord_equal()
```

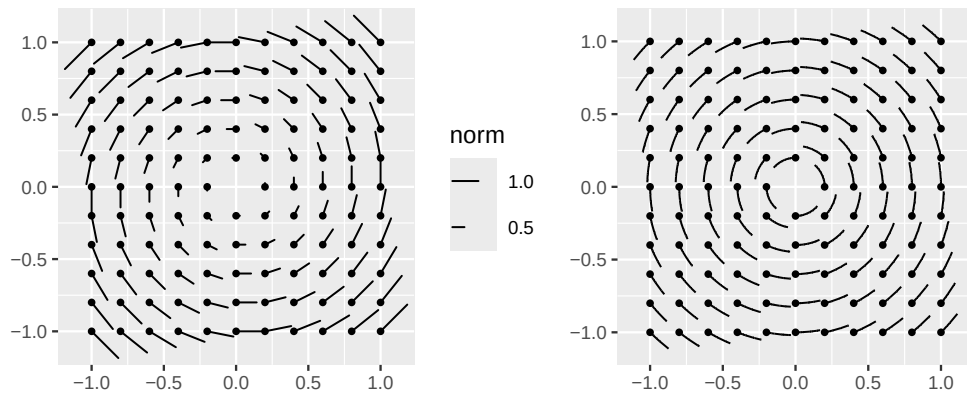


Figure 16: A stream version of `geom_vector_field2()` (left) is available as `geom_stream_field2()` (right).

The dynamical systems perspective admits another really nice graphic, which we informally refer to as comet plots. These are achieved by mapping l , the internally computed length variable l , to the alpha aesthetic. Various modifications can be made, e.g. by modifying the rate of the fading by transforming l , however the basic graphic is quite appealing. We illustrate this in Figure 17.

```
ggplot() +
  geom_stream_field(
    fun = f, aes(alpha = after_stat(l)), arrow = NULL, L = .5,
    grid = grid_hex(c(-1,1), c(-1,1), d = .2) |> purrr::modify(jitter, amount = .05)
  ) +
  scale_alpha(range = c(0,1), guide = "none") +
  coord_equal()
```

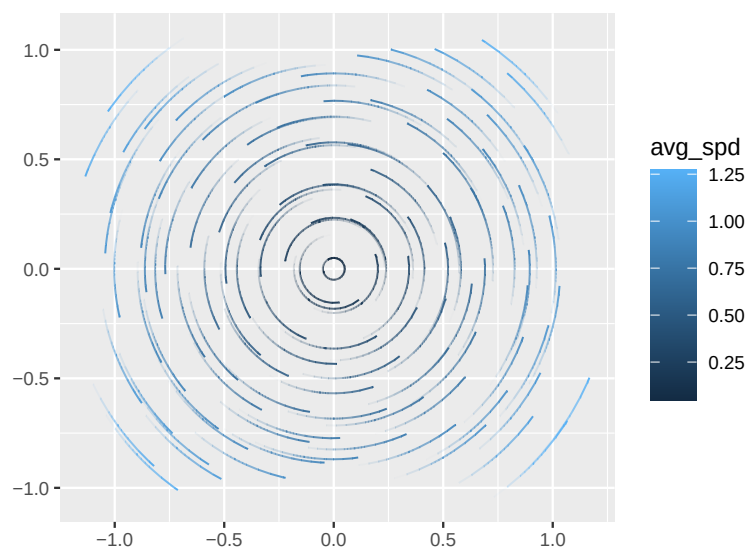


Figure 17: Stream plots can be faded by mapping the computed l aesthetic to alpha, which can be manipulated to accentuate the movement effect.

While the dynamical systems viewpoint provides valuable insight into vector fields as evolving particles in time, calculus offers another perspective: interpreting vector fields as gradients of scalar functions. We now turn our attention to that approach.

4.2 Gradient Fields

A second major lens through which vector fields often arise is through gradients of scalar fields. Specifically, if $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ is a scalar function, then its gradient is the vector field given by $\nabla f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, $(x, y) \mapsto (\partial_x f(x, y), \partial_y f(x, y))$. $\nabla f(x, y)$ points in the direction of steepest increase of f at the point (x, y) , and its magnitude $\|\nabla f(x, y)\|$ indicates how quickly f is changing there. This perspective is especially important in many applications, including physics and gradient-based optimization.

In **ggvfields**, gradient fields of scalar fields can be visualized directly using `geom_gradient_field()`. This function internally computes partial derivatives $\partial_x f$ and $\partial_y f$ of the scalar function f numerically using **numDeriv** and then plots those derivatives as vectors—just like `geom_vector_field()` (Gilbert and Varadhan, 2019). Streams can be plotted by setting `type = "stream"`. These are illustrated on the scalar field $f(x, y) = x^4 - 6x^2y^2 + y^4$ in Figure 18. Like other functions, `geom_gradient_field()` defaults to `xlim = c(-1, 1)` and `ylim = c(-1, 1)`, and we use this fact in our examples to be highlight more salient features.

```
scalar_field <- function(u){
  x <- u[1]; y <- u[2]
  x^4 - 6*x^2*y^2 + y^4
}

p1 <- ggplot() + geom_gradient_field(fun = scalar_field)
p2 <- ggplot() + geom_gradient_field(fun = scalar_field, type = "stream")

p1 + p2 + coord_equal()
```

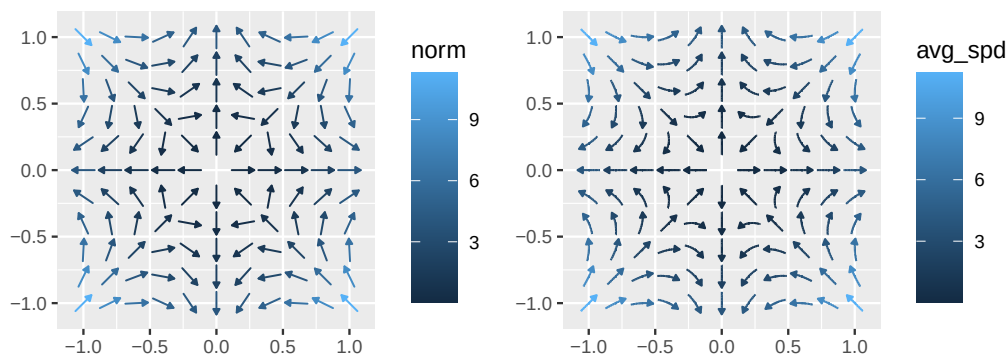


Figure 18: `geom_gradient_field()` can plot the gradient vector field of an input scalar function, computing it using **numDeriv**. Setting the `type` argument toggles between arrows and streams.

To help visually confirm the accuracy of the plot, it would be helpful to be able to superimpose the gradient field on top of a surface plot of its scalar field. This brings us to the concept of potentials.

4.3 Conservative vector fields and their potentials

A vector field that can be written as the gradient of a scalar field is called a conservative field. Such fields arise often in applications and derive from an underlying potential function $\phi(x, y)$. Building on our discussion of gradients, we now show how **ggvfields** can visualize not only the gradient field of a scalar field, but the potential of a vector field.

Of course, not every vector field $\mathbf{f}$ is conservative, so not every vector field has a corresponding potential. It is a standard fact of the calculus of two variables that if the coordinate functions $f_x(x, y)$ and $f_y(x, y)$ have continuous first partial derivatives, then the field is conservative precisely when $\partial_x f_y = \partial_y f_x$ (Larson and Edwards, 2014). Geometrically, this corresponds to the field having zero curl, which measures how much the field rotates around a point. A concept naturally belonging to three dimensions, the curl of a two-dimensional vector field is defined by embedding it into $\mathbb{R}^3$ as $\mathbf{f}^\uparrow(x, y, z) = (f_x(x, y), f_y(x, y), 0)$ and then looking at the corresponding field's curl: $\text{curl}(\mathbf{f}^\uparrow) = \nabla \times \mathbf{f}^\uparrow = (0, 0, \partial_x f_y - \partial_y f_x)$. The curl of $\mathbf{f}$ is then taken to be the scalar field $\text{curl}(\mathbf{f}) = \partial_x f_y - \partial_y f_x$. A reasonable approach to verifying this, therefore, is to check it by computing the relevant quantities numerically at points in the domain; this is the approach we take in `geom_potential()` when `verify_conservative = TRUE`. It defaults to `FALSE` to conserve time.

`geom_potential()` accepts a vector field `fun` and then plots the corresponding potential as a raster plot described in the introduction. This is illustrated in Figure 19 using the gradient field corresponding to the previous graphic, here coded in closed form.

```
gradient_field <- function(u){
  x <- u[1]; y <- u[2]
  c(4*x^3 - 12*x*y^2, y^4 - 12*x^2*y)
}

ggplot() +
  geom_potential(fun = gradient_field, xlim = c(-1,1), ylim = c(-1,1)) +
  geom_gradient_field(fun = scalar_field, type = "stream") +
  scale_fill_viridis_c() +
  scale_color_viridis_c() +
  coord_equal()
```

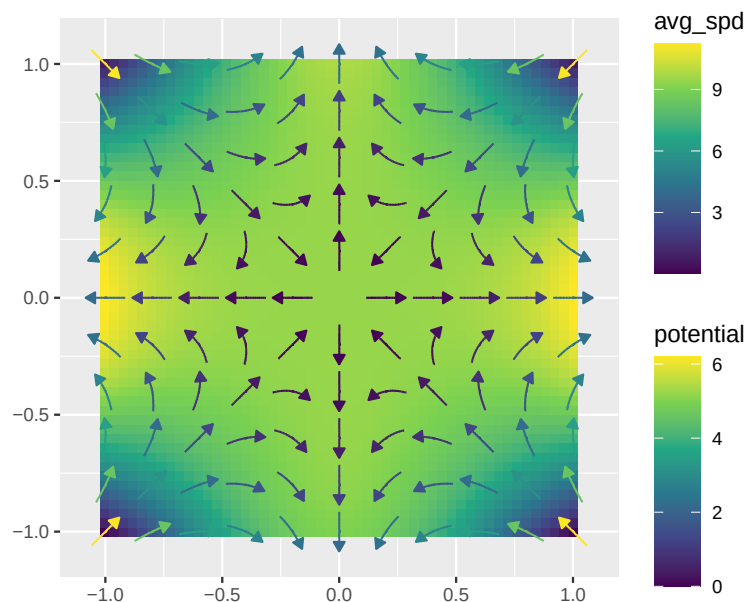


Figure 19: `geom_potential()` is the opposite of `geom_gradient_field()`: where the latter accepts a scalar field and plots a vector field, the former accepts a vector (gradient) field and plots the corresponding potential (scalar field).

A few more implementation notes may be helpful. Since R is not a symbolic language, `geom_potential()` finds the potential of a vector field by numerical methods rather than analytical ones. `geom_potential()` always works over a rectangular grid in the domain. To check whether the field is conservative, `geom_potential()` uses functions from `numDeriv` to approximate the Jacobian matrix of $\mathbf{f}$ at each grid point. By comparing the off diagonal

entries of this Jacobian, it checks whether $\partial f_y / \partial x \approx \partial f_x / \partial y$ within a given tolerance. If the difference exceeds that tolerance anywhere in the grid, `geom_potential()` concludes that the curl is not negligible and warns that the field may be non-conservative. When the curl check passes, `geom_potential()` runs a crude but efficient trapezoid quadrature rule from a chosen reference corner $(x_{\min}, y_{\min})$ to each grid coordinate, thereby estimating a scalar value for the potential. This value is finally mapped to the fill aesthetic of a raster layer, producing a heatmap of $\phi(x, y)$ across the specified domain.

In practice, many real-world vector fields may only be approximately conservative. If `geom_potential(verbose = TRUE)` detects any significant deviation from $\partial f_y / \partial x = \partial f_x / \partial y$, it warns that the field may not be conservative, and thus the resulting potential surface should be interpreted with caution.

As a quick contrast, recall our rotational field $\mathbf{f}(x, y) = (-y, x)$ from our first [example](#). Because its curl is nonzero, `geom_potential()` produces a warning and the resulting “potential” surface is not meaningful physically, as shown here:

```
ggplot() +
  geom_potential(fun = f, verify_conservative = TRUE) +
  geom_vector_field(fun = f) +
  scale_fill_viridis_c() +
  scale_color_viridis_c() +
  coord_equal()
```

```
#> Warning: ! The provided vector field does not have a potential function everywhere
#> within the specified domain.
#> > Ensure that the vector field satisfies the necessary conditions for a
#> potential function.
```

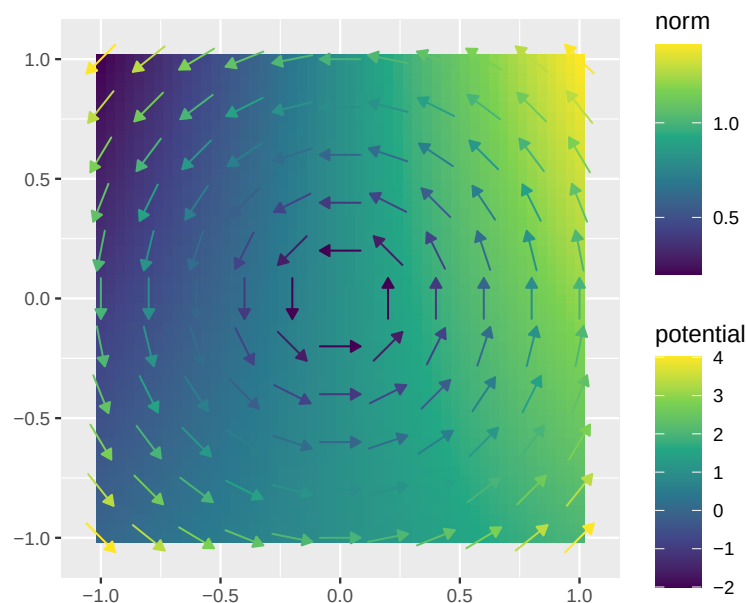


Figure 20: `geom_potential` checks if $f(x, y)$ is conservative by evaluating $\partial f_y / \partial x - \partial f_x / \partial y$. A warning is issued if significant deviations occur.

We now turn our attention to further examples that illustrate the flexibility and power of `ggvfields` in handling more challenging vector field problems.

5 Further examples

We conclude with a few examples that illustrate the flexibility and power of **ggvfields** in handling more challenging vector field problems. These examples extend the methods described above to applications in differential equations and gradient-based optimization.

Slope field of a differential equation

One common example of vector fields in teaching differential equations is slope fields. For a simple differential equation $y'(x) = f(x)$, the resulting vector field is independent of y , and solution curves pass along the presented slopes. In this example, we visualize the slope field associated with the differential equation $\frac{dy}{dx} = x^2 - x - 2$. By plotting the vector field corresponding to the slope at each point and overlaying solution curves, we can gain insight into the behavior different solutions. This is illustrated in Figure 21.

```
f <- function(u) {
  x <- u[1]; y <- u[2]
  c(1, x^2 - x - 2)
}

f_soln <- function(c = 0) function(x) x^3/3 - x^2/2 - 2*x + c

ggplot() +
  geom_vector_field(fun = f, xlim = c(-10,10), ylim = c(-10, 10), arrow = NULL, n = 31, color = "gray70") +
  geom_function(fun = f_soln( 4), aes(color = "+4"), xlim = c(-10, 10)) +
  geom_function(fun = f_soln( 0), aes(color = " 0"), xlim = c(-10, 10)) +
  geom_function(fun = f_soln(-4), aes(color = "-4"), xlim = c(-10, 10)) +
  scale_color_manual("c", breaks = c("+4", " 0", "-4"), values = c("red", "steelblue", "springgreen3")) +
  theme_minimal() + theme(panel.grid = element_blank(), axis.title = element_blank()) +
  coord_cartesian(xlim = c(-5,5), ylim = c(-10,10))
```

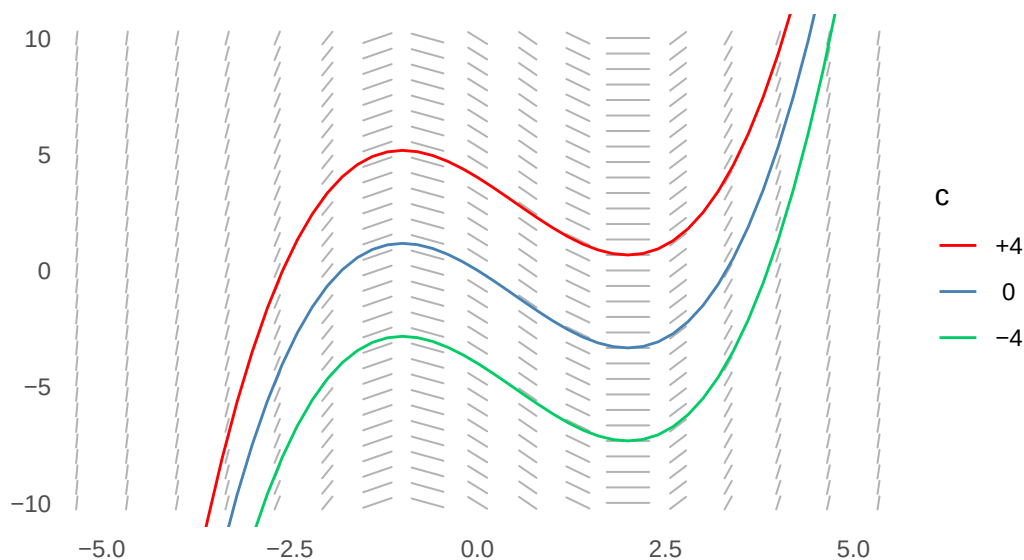


Figure 21: Slope field for the differential equation $\frac{dy}{dx} = x^2 - x - 2$ with overlaid solution curves illustrating different constant integration values.

Gradient-based optimization algorithms

Figure 22 provides an example of how **ggvfields** can be used to visualize and teach optimization. In the example we construct a log-likelihood function for normally distributed data and compute its gradient field. Visualizing the resulting gradient flows using

`geom_stream_field()` provides an intuitive view of how gradient-based optimization algorithms, such as gradient descent or Newton's method, might navigate the parameter space by following the direction of steepest descent.

```
set.seed(1234)
library("scales")

# set parameters
mu <- 20.5; si <- 8.3

# generate data
n <- 200
x <- rnorm(n, mean = mu, sd = si)

# create log-likelihood function and estimate MLE
logL <- function(th) sum(dnorm(x, mean = th[1], sd = th[2], log = TRUE))
mle <- optim(c(0,1), logL, control = list("fnscale" = -1))$par

# visualize
ggplot() +
  geom_gradient_field(fun = logL, xlim = c(15, 25), ylim = c(5, 15), type = "stream") +
  scale_color_viridis_c(trans = "log", labels = number_format(accuracy = .1)) +
  annotate("point", x = mu, y = si, color = "red", size = 3, shape = 17) +
  annotate("point", x = mle[1], y = mle[2], color = "red", size = 3) +
  labs("x" = expression(mu), "y" = expression(sigma), color = "Learning\\nrate") +
  coord_equal(xlim = c(15, 25), ylim = c(5, 15)) +
  theme_minimal() + theme(legend.title = element_text(size = 10))
```

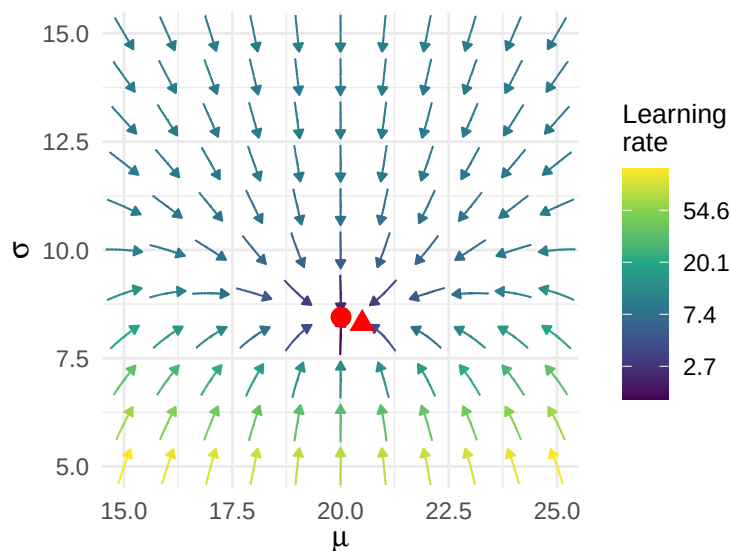


Figure 22: Gradient field of the log-likelihood function for normally distributed data of size $n = 200$. The true value of the unknown parameter is illustrated with the red triangle; the MLE with the red circle. The learning rate corresponds to what was previously labeled average speed.

Another example can be provided by Newton's method of optimization. Newton's method depends on local linear approximants to the gradient field, which can be solved to provide update directions. Specifically, if $f(\mathbf{x})$ is a scalar field to be optimized, Newton proposes solving linear approximants to the gradient field $\nabla f(\mathbf{x})$, i.e. $\nabla f(\mathbf{x}) \approx \nabla f(\mathbf{x}_0) + \mathbf{H}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) =: \mathbf{L}_{\mathbf{x}_0}(\mathbf{x})$, where $\mathbf{H}(\mathbf{x}_0)$ is the Hessian of $f(\mathbf{x})$, i.e. the Jacobian of its gradient. We can use `ggvfields` to visualize these two fields: the gradient field $\nabla f(\mathbf{x})$, which is nonlinear in $\mathbf{x}$, and the first order linear approximant $\mathbf{L}_{\mathbf{x}_0}(\mathbf{x})$ by superimposing

them with different color scales. This is illustrated in Figure 23. In this example we use `geom_stream_field2()`, suppressing speed information, in an attempt to visually simplify the graphic and focus on the direction of the field.

```
grad_field <- function(u) numDeriv::grad(logL, u)

Jf <- function(u) numDeriv::jacobian(grad_field, u)

# define linearization at x0
x0 <- c(17,12)
L <- function(u) grad_field(x0) + as.numeric( Jf(x0) %*% (u - x0) )

# compute newton direction
normalize <- function(x) x / sqrt(sum(x^2))
h <- solve( Jf(x0), -grad_field(x0) ) |> normalize() # newton direction

# make plot superimposing nonlinear vector field with linear approximant
ggplot() +
  annotate("segment", x = x0[1], y = x0[2], xend = (x0+h)[1], yend = (x0+h)[2],
    color = "red", arrow = arrow(type = "closed", length = unit(.02, "npc"))) +
  annotate("point", x = x0[1], y = x0[2], size = 2, color = "red") +
  geom_stream_field2(fun = grad_field, aes(color = "gf"), xlim = c(15, 25), ylim = c(5, 15)) +
  geom_stream_field2(fun = L, aes(color = "lf"), xlim = c(15, 25), ylim = c(5, 15)) +
  scale_color_manual(values = c("chocolate", "steelblue"),
    labels = c(
      expression(nabla * log * L * (mu * , ' * sigma)),
      expression(bold('L')[mu[0] * , ' * sigma[0]](mu * , ' * sigma))
    )
  ) +
  labs("x" = expression(mu), "y" = expression(sigma)) +
  coord_equal(xlim = c(15, 25), ylim = c(5, 15)) +
  theme_minimal() + theme(panel.grid = element_blank(), legend.title = element_blank())
```

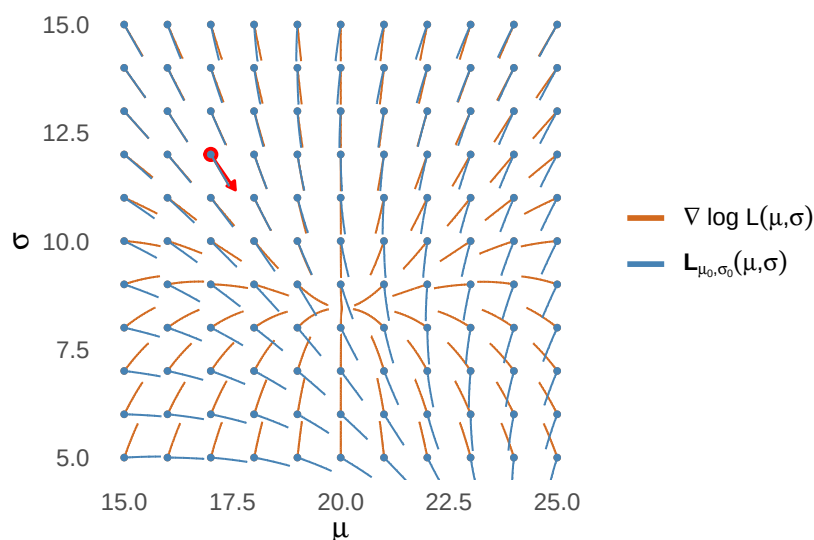


Figure 23: The linear approximant to a vector field aligns with the vector field it approximates near the approximating point, shown in red. Here the two vector fields of the gradient of the log-likelihood (L ; blue) and its linearization (orange) are superimposed. Overlapping indicates the vector field and its approximant are pointing in the same direction. The Newton update direction is illustrated with a red arrow. Note: minor numerical warnings issued by `deSolve::ode()` have been suppressed.

5.1 Related works

We have intentionally deferred discussing related works to the end of this manuscript in order to frame the problems and solutions in a way consistent with how we think about them; however, any user interested in **ggvfields** should be aware of a few other great tools that can be used for some of the kinds of things **ggvfields** can. This section considers five notable packages: **metR**, **ggquiver**, **ggarchery**, **ggfields**, and **rasterVis**—and contrasts their offerings with those of **ggvfields**.

1. **metR**. The **metR** package is designed for meteorological applications. It provides tools for geospatial and atmospheric data visualization. Its `geom_streamline()` function creates streamlines by first taking a regular rectangular grid of vector components (dx , dy) as input and using linear interpolation to estimate the vector field across the input data. The function then traces streamlines through the estimated field using forward Euler integration. While highly effective within its domain, its reliance on precomputed grids and structured data limits its flexibility for other types of visuals, e.g. stream normalizing and centering (Campitelli, 2021).

2. **ggquiver**. The **ggquiver** package offers `geom_quiver()` for visualizing vector fields as arrows. Like **metR**, it relies on a grid-based data input with defined vector components (dx , dy). While it produces visually clear and appealing quiver plots, the package's functionality is limited to static vector visualizations. Its simplicity and clarity make it a practical choice for quick, grid-based visualizations in contexts like engineering and physics (O'Hara-Wild, 2023).

3. **ggarchery**. The **ggarchery** package introduces `geom_arrowsegment()` for creating customizable arrow segments. This feature enhances the aesthetics of vector plots by allowing users to control arrow placement, size, and style. Customization options, including flexible arrowheads and segment styles, make it suitable for crafting visually compelling arrows, but not plots per se (Hall, 2024). It may provide a good base for expansion of **ggvfields** in the future.

4. **ggfields**. The **ggfields** package provides tools to plot vectors on **sf** layers and also allows vector arrows to be added to plots with x and y data. It focuses on offering methods for overlaying vector field information onto spatial data visualizations within the **ggplot2** framework. The package's narrow focus limits its broad application to vector fields that are available in **ggvfields**, but its attention to details of transformations may be an improvement over **ggvfields** (de Vries, 2024).

5. **rasterVis**. The **rasterVis** package in R also provides tools for visualizing vector fields through `vectorplot()` and `streamplot()` functions (Oscar Perpiñán and Robert Hijmans, 2023). These functions display stunning vector fields as arrows or streamlines, respectively, and are specifically designed to work with raster data in the **lattice** graphics system (Sarkar, 2008). While **rasterVis** effectively handles spatial vector data, perhaps its biggest difference from **ggvfields** is its reliance on **lattice**. This makes it less intuitive for users accustomed to **ggplot2** workflows and simply a different tool.

6. **Other Programming Environments**. In addition to R packages, other programming languages provide tools for visualizing vector fields. Mathematica's `StreamPlot[]` function dynamically generates streamlines for vector fields defined by user-defined functions (Wolfram Research, 2023), and we were interested to find a kind of convergent evolution between **ggvfields** and some of these functions. For example, earlier versions of Mathematica seem to have defaulted to rectangular grids, whereas now hexagonal grids seem to be the default. Similarly, Python's Matplotlib library includes the `streamplot` function which visualizes 2D vector fields by integrating vector components over a grid; however, its computations in general are very different than those described here (Matplotlib Development Team, 2023). Both tools share similarities with **ggvfields** in their ability to handle function-defined vector fields. These similarities from other coding languages highlight the need for more robust vector visualization tools in **ggplot2** and a kind of cross-language learning.

Taken together, these packages demonstrate the wide interest in vector field visualization across domains—from meteorology and geospatial analysis to mathematical modeling. However, most are specialized for a particular context or input structure, and many are limited to a single type of visual representation.

The **ggvfields** package was designed to unify and extend these capabilities in a general-purpose tool, built entirely within the familiar grammar of graphics framework that **ggplot2** provides. It supports a wide range of use cases by allowing users to define vector fields directly as functions or use precomputed grid data. The package offers both arrow-based and stream-based visualizations through a single consistent interface with multiple styles of glyphs available for use. Built-in features for normalization, centering, and scaling enhance communication, while flexible grid geometry options (including rectangular and hexagonal) expand control. Integration with **ggplot2** enables layering, faceting, theming, and scale tuning in line with modern R workflows. This modular design makes **ggvfields** especially well suited for all domains.

By combining these capabilities in one package, **ggvfields** offers a unique and more comprehensive solution for vector field visualization in R, without the need to stitch together specialized tools or workflows.

6 Conclusion and future directions

The **ggvfields** package extends the capabilities of **ggplot2** by providing a set of specialized geoms for visualizing vectors, vector fields, streamplots, and flowplots. Compared to other R tools, **ggvfields** offers a cohesive and customizable approach that easily integrates with the **ggplot2** ecosystem. It supports a wide range of vector field types which include both Cartesian and polar encodings and provides built-in functionality for calculating field properties.

One of the planned enhancements is the development of `geom_vector_smooth()` which will bring the familiar functionality of `geom_smooth()` to vector data. This new geom will allow users to fit a smooth vector field to a set of observed vectors, similar to how `geom_smooth()` fits regression lines to scatter plots. The addition of `geom_vector_smooth()` will provide a powerful tool for users to explore trends and patterns in vector data, making it easier to represent and interpret the underlying structure of complex vector fields.

With these planned additions, **ggvfields** aims to become an even more versatile tool for vector field visualization and analysis in R. By continuing to expand its capabilities, the package will help users more effectively explore and communicate insights from complex vector data in a clear and intuitive manner.

References

- V. I. Arnold. *Ordinary Differential Equations*. MIT Press, Cambridge, MA, first edition edition, 1973. ISBN 978-0-262-51018-2. Translated from Russian by Richard A. Silverman. [p240]
- E. Campitelli. *metR: Tools for Easier Analysis of Meteorological Fields*, 2021. URL <https://eliocamp.github.io/metR/>. R package version 0.17.0. [p249]
- W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American statistical association*, 79(387):531–554, 1984. [p234]
- P. de Vries. *ggfields: Add Vector Field Layers to Ggplots*, 2024. URL <https://CRAN.R-project.org/package=ggfields>. R package version 0.0.6. [p249]
- S. L. Franconeri, L. M. Padilla, P. Shah, J. M. Zacks, and J. Hullman. The science of visual data communication: What works. *Psychological Science in the public interest*, 22(3):110–161, 2021. [p234]

- P. Gilbert and R. Varadhan. *numDeriv: Accurate Numerical Derivatives*, 2019. URL <https://CRAN.R-project.org/package=numDeriv>. R package version 2016.8-1.1. [p243]
- M. Hall. *ggarchery: Flexible Segment Geoms with Arrows for 'ggplot2'*, 2024. URL <https://CRAN.R-project.org/package=ggarchery>. R package version 0.4.3. [p249]
- M. Kay. *ggdist: Visualizations of Distributions and Uncertainty in the Grammar of Graphics*. *IEEE Transactions on Visualization and Computer Graphics*, 2023. [p232]
- R. Larson and B. Edwards. *Calculus: Early Transcendental Functions*. Cengage Learning, 2014. ISBN 9781285774770. [p244]
- Y. Liu and J. Heer. Somewhere over the rainbow: An empirical assessment of quantitative colormaps. In *Proceedings of the 2018 CHI conference on human factors in computing systems*, pages 1–12, 2018. [p234]
- Matplotlib Development Team. *Matplotlib Documentation: streamplot*, 2023. URL https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.streamplot.html. [p249]
- M. O'Hara-Wild. *ggquiver: Quiver Plots for 'ggplot2'*, 2023. URL <https://CRAN.R-project.org/package=ggquiver>. R package version 0.3.3. [p249]
- Oscar Perpiñán and Robert Hijmans. *rasterVis*, 2023. URL <https://oscarperpinan.github.io/rastervis/>. R package version 0.51.6. [p249]
- J. Otto and D. J. Kahle. *ggdensity: Improved Bivariate Density Visualization in R*. *The R Journal*, 15(2):220–236, 2023. [p232]
- T. L. Pedersen. *patchwork: The Composer of Plots*, 2024. URL <https://CRAN.R-project.org/package=patchwork>. R package version 1.3.0. [p232]
- R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2024. URL <https://www.R-project.org/>. [p227]
- D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer, New York, 2008. ISBN 978-0-387-75968-5. URL <http://lmdvr.r-forge.r-project.org>. [p249]
- C. Sievert. *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman and Hall/CRC, 2020. ISBN 9781138331457. URL <https://plotly-r.com>. [p228]
- K. Soetaert, T. Petzoldt, and R. W. Setzer. Solving differential equations in R: Package deSolve. *Journal of Statistical Software*, 33(9):1–25, 2010. doi: 10.18637/jss.v033.i09. [p239]
- K. Soetaert, J. Cash, F. Mazzia, K. Soetaert, J. Cash, and F. Mazzia. *Solving ordinary differential equations in R*. Springer, 2012. [p239]
- H. Wickham. A layered grammar of graphics. *Journal of computational and graphical statistics*, 19(1):3–28, 2010. [p234]
- L. Wilkinson. The grammar of graphics. In *Handbook of computational statistics: Concepts and methods*, pages 375–414. Springer, 2011. [p234]
- Wolfram Research. *StreamPlot: Visualizing Streamlines in Mathematica*, 2023. URL <https://reference.wolfram.com/language/ref/StreamPlot.html>. Version 13.2. [p249]

Dusty Turner
 Baylor University
 One Bear Place #97140
 Waco, Texas, 76706
<https://dustysturner.com>
 ORCID: 0000-0003-2998-8799
dusty.s.turner@gmail.com

Rodney X. Sturdivant
Baylor University
One Bear Place #97140
Waco, Texas, 76706
ORCID: 0000-0001-9316-7085
rodney_sturdivant@baylor.edu

David Kahle
Baylor University
One Bear Place #97140
Waco, Texas, 76706
<https://www.kahle.io/>
ORCID: 0000-0002-9999-1558
david_kahle@baylor.edu

neuralGAM: An R Package for Fitting Generalized Additive Neural Networks

by Ines Ortega-Fernandez and Marta Sestelo

Abstract Nowadays, neural networks are considered one of the most effective methods for various tasks such as anomaly detection, computer-aided disease detection, or natural language processing. However, these networks suffer from the “black-box” problem which makes it difficult to understand how they make decisions. In order to solve this issue, an R package called **neuralGAM** is introduced. This package implements a Neural Network topology based on Generalized Additive Models, allowing users to fit an independent Neural Network to estimate the contribution of each feature to the output variable, yielding a highly accurate and interpretable deep learning model. The **neuralGAM** package provides a flexible framework for training Generalized Additive Neural Networks, which does not impose any restrictions on the per-term neural network architecture, while the overall model remains additive. We illustrate the use of the **neuralGAM** package in both synthetic and real data examples.

1 Introduction

Neural networks (NN) are currently among the most widely used predictive modeling techniques, demonstrating superior performance across a broad range of tasks. Despite their success, a well-known shortcoming of neural networks is that they often function as “black-box” models. In most practical applications, it can be challenging to understand precisely *how* the network processes information and reaches a particular prediction (Szegedy et al., 2013). In recent years, there has been a growing research focus on enhancing trust in AI systems by improving their interpretability. Broadly, interpretability methods can be categorized into *post-hoc* and *ante-hoc* approaches (Došilović et al., 2018). *Post-hoc* methods aim to explain a trained black-box model (like a typical Neural Network) using an external, intrinsically interpretable (“white-box”) model to approximate or interpret the black-box decisions. In contrast, *ante-hoc* methods aim to build and train inherently interpretable (“white-box”) models that balance high predictive accuracy with transparency.

Following this *ante-hoc* approach, several approaches in the literature have been proposed to combine the interpretability of GAMs with the flexibility of neural networks, yielding the family of Generalized Additive Neural Networks (GANNs). Early approaches, such as the model by Potts (1999), used univariate multi-layer perceptrons without backpropagation and required manual tuning. Subsequent approaches, like that of Brás-Geraldes et al. (2019), improved flexibility with parametric link functions and bootstrap-based confidence intervals but remained constrained to shallow architectures. More recent methods implemented in Python take advantage of deep learning techniques: Neural Additive Models (NAMs) (Agarwal et al., 2021) train feature-specific subnetworks jointly with backpropagation but can suffer from overfitting due to sharp activation functions. GAMI-Net (Yang et al., 2021) extends this by allowing pairwise interactions and incorporating sparsity and marginal clarity constraints, but with an increased computational cost. Finally, IGANN (Kraus et al., 2023) introduces a GANN based on gradient boosting techniques with sparsified shallow networks, assuming initially linear effects and avoiding deep architectures.

The development of the **neuralGAM** package has been motivated by recent contributions in the development of GANNs, in particular, the method proposed by Ortega-Fernandez et al. (2024) to train a GANN using independent neural networks using the local scoring (LS) and backfitting (BF) algorithms.

Specifically, **neuralGAM** trains an ensemble of independent neural networks (one per feature) to learn each feature’s contribution to the response. By enforcing an additive decomposition via backfitting and local scoring, it prioritizes interpretability over the

flexibility of unconstrained deep networks: the aim is not to match their full expressive power but to strike a practical balance, delivering competitive predictive performance while maintaining the GAM-style additivity that supports clear feature-level interpretation.

Regarding other R packages that combine additive models with neural networks, the **deepregression** package (Rügamer et al., 2023) implements distributional regression models for mean, scale, and shape parameters using deep neural networks, extending the GAM framework by embedding spline terms directly into a neural network architecture. Smooth functions are pre-processed using **mgcv** (Wood, 2011), which generates the spline basis and penalty matrices; these are then represented as structured layers within a deep neural network model and regularized through a TensorFlow's loss function.

The **bamlss** package (Umlauf et al., 2018), in contrast, provides a more general Bayesian framework for additive models that can incorporate neural network terms. Estimation in this context is performed through Bayesian sampling or variational inference, which naturally provides posterior uncertainty quantification at the expense of higher computational costs. These packages illustrate complementary methodological directions: **deepregression** extends GAMs toward distributional regression with deep learning components, **bamlss** offers a Bayesian approach with uncertainty quantification, and **neuralGAM** provides a dedicated white-box GANN implementation through independent deep neural networks. In this regard, **neuralGAM** complements existing R software and, to the best of our knowledge, provides the only available GANN implementation in R that leverages fully independent deep Neural Networks per feature and estimates the additive model through local scoring and backfitting. This design offers both interpretability and the characteristic deep learning flexibility within the R ecosystem.

At last, beyond GANN-specific packages, several R packages provide model-agnostic tools for explainability. The **NeuralNetTools** package (Beck, 2018), for instance, offers visualizations and sensitivity analyses tailored to neural networks. More general frameworks such as **iml** (Molnar et al., 2018) and **DALEX** (Biecek, 2018) implement techniques such as feature importance, partial dependence, and Shapley values, supporting both local and global interpretability. While **iml** and **DALEX** are model-agnostic, **NeuralNetTools** is specifically designed for neural networks.

In this paper, we demonstrate how neural networks can be used to fit Generalized Additive Models using the **neuralGAM** package. This is illustrated through a simulated scenario with a Gaussian response, as well as a real-life application related to flight delay prediction based on weather and flight conditions' data.

The remainder of the paper is structured as follows: in Section 2.2 we briefly review the estimation procedures and explain the use of the main functions and methods of **neuralGAM**; Section 2.3 gives an illustration of the practical application of the package using simulated and real data; and finally, Section 2.4 concludes with a discussion and possible future extensions of the package.

2 Models and software

2.1 An overview of the methodology

The most common way to model the relationship between the response variable and the covariates is by using the multiple linear regression model, where the response variable (Y) is assumed to be normally distributed, and the covariates ($X_j, j = 1, \dots, p$) are assumed to have a linear effect on the response. However, the response variable may not be normally distributed and, in such cases, Generalized Linear Models (Nelder and Wedderburn, 1972) allow the use of other distribution families, e.g. Binomial, Poisson, etc. Additionally, in some cases, the assumption of linearity in the effects of covariates can be too restrictive and not supported by the available data. In this setting, nonparametric regression techniques emerge, allowing us to model this dependence between the response and the covariates without specifying in advance the function that links them. This leads to the Generalized

Additive Models (Hastie and Tibshirani, 1990) defined by

$$E[Y | \mathbf{X}] = m(\mathbf{X}) = h^{-1}\left(\alpha + \sum_{j=1}^p f_j(X_j)\right), \quad (1)$$

where $h(\cdot)$ is a monotonic known function (the link function) and $f_1, \dots, f_p$ are smooth and unknown functions. To ensure the identifiability of the model (the ability to uniquely determine the parameters of the model based on the observed data), a constant denoted by α is introduced in the model and the partial functions must satisfy the condition $E[f_j(X_j)] = 0$ where $j = 1, \dots, p$. This implies that $E[Y] = \alpha$, a crucial condition to ensure that the model's predictions remain unchanged even if we add a constant value to f_1 while subtracting the same constant value from f_2 (Hastie and Tibshirani, 1990).

As we mentioned, GAMs are widely used for their ability to capture complex relationships between covariates and response variables without requiring the shape of these relationships to be specified in advance. This flexibility makes GAMs a powerful and effective class of regression models. They also offer the advantage of being fully nonparametric while allowing certain covariates—such as categorical features—to enter the model linearly, resulting in a semi-parametric framework. Thanks to this versatility, GAMs are particularly well-suited for tackling real-world problems.

Distribution	Link	Z_i	W_i	$DEV_i(Y_i, \hat{\mu}_i)$
Gaussian	identity	Y_i	1	$(Y_i - \hat{\mu}_i)^2$
Binomial (s, μ)	logit	$\eta_i + (Y_i - \mu_i)s\mu_i(1 - \mu_i)$	$s\mu_i(1 - \mu_i)$	$-2(Y_i \log \hat{\mu}_i + (1 - Y_i) \log(1 - \hat{\mu}_i))$
Poisson	log	$\eta_i + (Y_i - \mu_i)/\mu_i$	μ_i	$Y_i \log \frac{Y_i}{\hat{\mu}_i} - (Y_i - \hat{\mu}_i)$

Table 1: Adjusted dependent variable Z_i , weights W_i , and deviance $DEV_i(Y_i, \hat{\mu}_i)$, at the local scoring algorithm for Gaussian, binomial and Poisson distributions (Hastie and Tibshirani, 1990).

To fit the previous model in (1), from an independent random sample $\{\mathbf{X}_i, Y_i\}_{i=1}^n$, where each $\mathbf{X}_i = (X_{i1}, \dots, X_{ip})$ is a realization of the p -dimensional random vector $\mathbf{X} = (X_1, \dots, X_p)$, we use a combination of the local scoring algorithm (see Algorithm 1) and the backfitting algorithm (see Algorithm 2).

In practice, some covariates may enter the model linearly while others are represented by smooth functions. In this case, for the model in (1), each f_j may either be a smooth, nonparametric function fitted using a neural network or a parametric linear effect of the form $f_j(x_j) = x_j^\top \beta_j$, leading to a semiparametric model.

Particularly, the estimation of the additive predictor $\eta = \alpha + \sum_{j=1}^p f_j(\cdot)$ is obtained by fitting a weighted additive model where each f_j is either a smooth function (estimated by a neural network) or a linear effect, yielding a Generalized Additive Neural Network. For each smooth f_j , the backfitting algorithm iteratively estimates the effect of each covariate $X_1, \dots, X_p$, with weights W_i , on the adjusted dependent variable Z_i , updated at each step of the local scoring algorithm.

Note that if the link function is the identity and the error distribution is Gaussian, then $Z_i = Y_i$ and the weights do not change, thus the procedure is simply an additive fit. In any other case, the dependent variable Z_i and the weights W_i are updated at each iteration l of the local scoring algorithm. This process is repeated until the convergence criterion is satisfied (see last step of Algorithm 1). Note that we use the deviance because it is an appropriate measure of the discrepancy between observed and fitted values.

Particularly, given the fitted mean response $\hat{\mu}_i = \hat{E}[Y_i | \mathbf{X}_i]$, the deviance is defined as

$$DEV = \frac{\sum_{i=1}^n DEV_i(Y_i, \hat{\mu}_i^{l-1}) - DEV_i(Y_i, \hat{\mu}_i^l)}{\sum_{i=1}^n DEV_i(Y_i, \hat{\mu}_i^{l-1})},$$

Algorithm 1: Local scoring algorithm

Initialise $\hat{\alpha} = h(\bar{Y}_i)$, $\hat{f}_j^0(\cdot) = 0 \forall j$, $l = 0$

for $l \leftarrow l + 1$ **do**

Construct an adjusted dependent variable Z_i according to Table 1 with

$$\hat{\eta}_i^{l-1} = \hat{\alpha} + \sum_{j=1}^p \hat{f}_j^{l-1}(X_{ij}), \quad \hat{\mu}_i^{l-1} = h^{-1}(\hat{\eta}_i^{l-1}).$$

Form the weights W_i according to Table 1.

Update the linear component (if any): fit a weighted least squares model of Z_i on the linear covariates. Update $\hat{\alpha}$ to be the fitted intercept from the parametric component, and update $\hat{f}_j^l$ for the linear terms accordingly.

Update the non-linear component: fit a weighted additive model to Z_i using the backfitting algorithm (Algorithm 2) with weights W_i to obtain the estimated functions $\hat{f}_j^l$, additive predictor $\hat{\eta}_i^l$, and fitted values $\hat{\mu}_i^l$.

end

Compute the convergence criterion $\text{DEV} < \delta$, with DEV as in Table 1.

Until the convergence criterion is satisfied

Algorithm 2: Backfitting Algorithm with neural networks

Initialize $\hat{f}_j^0(\cdot) = 0 \forall j$, $m = 0$

Iterate $m \leftarrow m + 1$

for $j \leftarrow 1, p$ **do**

Compute partial residuals $R_{ij} = Z_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k^{m-1}(X_{ik})$

For each smooth term, fit a Neural Network with R_{ij} and X_{ij} to obtain $\hat{f}_j^m$

Replace $\hat{f}_j^m$ with its centered version $\hat{f}_j^m(\cdot) - \frac{\sum_{i=1}^n \hat{f}_j^m(X_{ij})}{n}$

end

Until

$$\frac{\sum_{j=1}^p \sum_{i=1}^n (\hat{f}_j^m(X_{ij}) - \hat{f}_j^{m-1}(X_{ij}))^2}{\sum_{j=1}^p \sum_{i=1}^n (\hat{f}_j^{m-1}(X_{ij}))^2} < \epsilon$$

with DEV_i depending on the link (see Table 1). Several approaches have been described in the literature to estimate the regression model in (1), such as Bayesian approaches (Lang and Brezger, 2004), local polynomial kernel smoothers (Wand and Jones, 1994; Copeland, 1997) or regression splines (De Boor et al., 1978). Unlike these quite common techniques, we propose to use independent neural networks—which are universal function estimators (Hornik et al., 1989)—to learn the contribution of each covariate to the dependent variable, i.e., at each iteration of the backfitting algorithm, we train a feed-forward Neural Network for each covariate X_j with the entire set of training data for one epoch. The fits are improved at each epoch as the learned adjusted dependent variable Z_i approaches Y_i at each iteration of the local scoring algorithm. Algorithms 1 and 2 summarize the core idea.

Uncertainty estimation

To quantify epistemic uncertainty, we leverage Monte Carlo (MC) Dropout (Gal and Ghahramani, 2016), a widely used technique for estimating model uncertainty in neural networks by interpreting dropout as a Bayesian approximation. In **neuralGAM**, dropout layers are inserted into the neural networks used to estimate the smooth functions f_j . During training, dropout layers randomly “drop” (set to zero) a fraction of the units in each layer of the network, acting as a regularizer to prevent overfitting. At inference time, rather than being deactivated as in standard practice, dropout remains active to generate stochasticity. The network is then evaluated repeatedly through B stochastic forward passes, producing a distribution of predictions that can be used to quantify epistemic uncertainty for each individual term and for the additive predictor in the link and response scales.

Per-term uncertainty. For each smooth term f_j , the use of MC Dropout during inference yields B draws

$$\hat{f}_j^{(b)}(x_j)_{b=1}^B.$$

To guarantee the identifiability of the model, these smooth dropout predictions are re-centered by subtracting each term’s training-time centering constant, so that for every j

$$\frac{1}{n} \sum_{i=1}^n \hat{f}_j^{(b)}(x_{ij}) = 0.$$

The epistemic variance of each nonparametric component is therefore defined as

$$\widehat{\text{Var}}_{\text{np}}[f_j(x_j)] = \widehat{\text{Var}}_b(\hat{f}_j^{(b)}(x_j)),$$

where $\hat{f}_j^{(b)}(x_j)$ denotes the prediction of the j -th component in the b -th simulation or dropout pass, and $\widehat{\text{Var}}_b$ indicates the empirical variance over these B realizations.

For models including a linear (parametric) component $f_j(x_j) = x_j^\top \beta_j$, the uncertainty is obtained from the fitted linear model (using weighted least squares) via the standard errors of the predicted values, $\widehat{SE}_p(f_j(x_j))$. In practice, we use the standard errors reported by the `stats::predict(..., se.fit=TRUE)` function in R, which are based on the variance-covariance matrix of the estimated coefficients $\hat{\beta}$. These quantify the sampling variability of the estimated coefficients $\hat{\beta}$ under the classical linear model assumptions.

The corresponding epistemic variance for the parametric component is then defined as:

$$\widehat{\text{Var}}_p(f_j(x_j)) = \left[\widehat{SE}_p(\hat{f}_j(x_j)) \right]^2$$

To provide a unified notation, we define the epistemic variance of any model component $f_j(x_j)$ as

$$\widehat{\text{Var}}_{\text{epi}}[f_j(x_j)] = \begin{cases} \widehat{\text{Var}}_{\text{np}}[f_j(x_j)], & \text{if } f_j \text{ is a smooth term,} \\ \widehat{\text{Var}}_p[f_j(x_j)], & \text{if } f_j \text{ is a linear term.} \end{cases}$$

Since both the dropout-based and the standard error-based variances quantify epistemic uncertainty in their respective components, per-term confidence intervals at level $1 - \alpha$ can be expressed in a unified form as

$$\hat{f}_j(x_j) \pm z_{1-\alpha/2} \sqrt{\widehat{\text{Var}}_{\text{epi}}[f_j(x_j)]}.$$

Uncertainty for the additive predictor on the link scale. Moving from individual terms to the full model, to obtain uncertainty bands for the full additive predictor $\eta(\mathbf{x})$ on the link scale, we combine the uncertainty from the nonparametric and parametric components.

For covariates fitted using a neural network, nonparametric epistemic variability is captured by evaluating all smooth networks jointly under dropout. At each dropout pass b , define the smooth-only predictor

$$\eta_{np}^{(b)}(\mathbf{x}) = \sum_{j \in \mathcal{S}} \hat{f}_j^{(b)}(x_j),$$

where $\mathcal{S}$ denotes the set of smooth terms, so that the dropout variance across the B passes captures epistemic uncertainty and cross-term covariance among smooths:

$$\widehat{\text{Var}}_{np}[\eta(\mathbf{x})] = \text{Var}\{\eta_{np}^{(b)}(\mathbf{x})\},$$

where $\widehat{\text{Var}}_{np}$ denotes the empirical variance computed over the B dropout realizations.

When the model includes a parametric linear component $x_j^\top \hat{\beta}$, its variance is obtained from the standard errors of the linear predictor:

$$\widehat{\text{Var}}_p[\eta(\mathbf{x})] = \left[\widehat{SE}_p(\hat{f}(\mathbf{x})) \right]^2$$

Assuming independence between the linear and smooth parts, the total epistemic variance on the link scale is then

$$\widehat{\text{Var}}_{\text{epi}}\{\eta(\mathbf{x})\} = \widehat{\text{Var}}_{np}\{\eta(\mathbf{x})\} + \widehat{\text{Var}}_p\{\eta(\mathbf{x})\},$$

and the $(1 - \alpha)$ confidence interval on the link scale is

$$\hat{\eta}(\mathbf{x}) \pm z_{1-\alpha/2} \sqrt{\widehat{\text{Var}}_{\text{epi}}[\eta(\mathbf{x})]}.$$

Uncertainty for the additive predictor on the response scale. Finally, uncertainty on the response scale is obtained by applying the delta method. The fitted mean is related to the additive predictor through the inverse link function, $\mu = h^{-1}(\eta)$. Since uncertainty is first quantified on the link scale, we need to translate it into the scale of μ . The delta method provides a simple rule: the variance of μ can be approximated by multiplying the variance of η by the squared slope of the transformation. Formally,

$$\widehat{\text{Var}}\{\mu(\mathbf{x})\} \approx ((h^{-1})'(\hat{\eta}(\mathbf{x})))^2 \widehat{\text{Var}}_{\text{epi}}\{\eta(\mathbf{x})\},$$

where $(h^{-1})'(\eta) = \frac{d}{d\eta} h^{-1}(\eta)$ is the derivative of the inverse link. The corresponding standard error of the fitted mean on the response scale is therefore

$$\widehat{SE}_\mu(\mathbf{x}) = |(h^{-1})'(\hat{\eta}(\mathbf{x}))| \sqrt{\widehat{\text{Var}}_{\text{epi}}\{\eta(\mathbf{x})\}},$$

and the $(1 - \alpha)$ confidence interval can be obtained as

$$\hat{\mu}(\mathbf{x}) \pm z_{1-\alpha/2} \widehat{SE}_\mu(\mathbf{x}).$$

This procedure ensures that uncertainty bands respect the chosen link function: with the identity link (Gaussian models) the derivative is 1 and the intervals are unchanged, while with non-identity links (such as the logit or log links), the intervals are automatically rescaled according to the applied transformation.

2.2 Package structure and functionality

The **neuralGAM** package introduces a new methodology for fitting Generalized Additive Models, with Gaussian, binary, or Poisson responses, based on independent neural networks. It is composed of several functions that enable users to fit the models with the methods described above. **neuralGAM** relies on **keras** (Chollet et al., 2015) and **TensorFlow** (Abadi et al., 2015), high-level APIs for the implementation of neural networks well-known for their simplicity, flexibility, and ease of use. **neuralGAM** also relies on other R packages for visualization (**ggplot2**) and numerical utilities.

Function	Description
<code>neuralGAM</code>	Main function to fit a neuralGAM model. Fits per-feature neural networks with user-specified architectures, and combines them using backfitting and local scoring. Supports uncertainty quantification through Monte Carlo dropout.
<code>summary.neuralGAM</code>	Method for the generic summary function, providing numerical summaries of fitted models.
<code>print.neuralGAM</code>	Method for the generic print function, showing key model components.
<code>plot.neuralGAM</code>	Visualization of smooth and linear terms using base R graphics.
<code>autoplot.neuralGAM</code>	Visualization of fitted components and prediction/confidence intervals using ggplot2 (Wickham, 2016), returning a ggplot object, with support for confidence and prediction intervals.
<code>predict.neuralGAM</code>	Prediction method with options for link, response, or term-level contributions. Supports standard errors and confidence intervals.
<code>diagnose</code>	Diagnostic plots for model evaluation, including residuals and Quantile-Quantile plots. For Gaussian models, these plots diagnose symmetry, tail behavior, and mean/variance misfit. For discrete data (binomial, Poisson), randomized Quantile residuals are available, which often yield smoother QQ behavior.
<code>plot_history</code>	Visualization of the training and validation loss history for each term-specific network, across backfitting iterations.
<code>install_neuralGAM</code>	Creates a conda environment (installing miniconda if required) and sets up the Python requirements to run neuralGAM (TensorFlow and Keras, version 2.15).

Table 2: Summary of functions in the **neuralGAM** package.

The package is designed along lines similar to those of other R regression packages. The functions within **neuralGAM** are briefly described in Table 2. The main function of the package is `neuralGAM`, which fits a Generalized Additive Neural Network to estimate the contribution of each smooth function to the response. The arguments of this function are shown in Table 3. Note that through the argument `formula` users can decide to fit a model with neural networks($s(x)$), linear, or factor terms (x), and using the argument `family` it is possible to select the conditional distribution of the response variable. So far, the user can select between Gaussian, binomial, and Poisson.

Beyond specifying network architecture (`num_units`, `activation`, `initializers`, `regularizers`, `losses`, etc.), **neuralGAM** implements uncertainty estimation via the `uncertainty_method` argument, allowing the computation of epistemic uncertainty. Additional arguments such as `alpha`, `forward_passes`, `dropout_rate`, provide fine control over uncertainty estimation.

Cross-validation support during neural network training is implemented via the `validation_split` argument. Users can also visualize how the training and validation loss evolves after each backfitting iteration using the `plot_history` function.

Numerical and graphical summaries of the fitted object can be obtained by using the `print`, `summary`, `plot`, `autoplot`, `plot_history` and `diagnose` methods implemented for `neuralGAM` objects. Another of these methods is available for the `predict` function, which takes a fitted model of the `neuralGAM` class and, given a new data set of values of the

covariates by means of the argument `newdata`, produces predictions, including uncertainty estimates in the form of confidence intervals. Diagnostic checks can be performed with the `diagnose` function, which produces residual plots and QQ-plots under different reference distributions. At last, we provide a helper function `install_neuralGAM()` to assist the user in installing the required Python dependencies in a custom conda environment. Once the dependencies are installed using `install_neuralGAM()`, the user must reload the library again using `library(neuralGAM)` for the changes to take effect. Note that the first time the package is used after installing and setting up the dependencies, the process of loading the required Python packages might take some time.

	neuralGAM() arguments
<code>formula</code>	An object of class "formula": a description of the model to be fitted. Smooth neural network terms can be included using <code>s()</code> while plain variables are linear/factor terms.
<code>data</code>	A data frame containing the model response variable and covariates required by the formula. Additional terms not present in the formula will be ignored.
<code>family</code>	A string specifying the distribution and link to use for fitting. By default, "gaussian", also "binomial" and "poisson" for logistic and poisson regression.
<code>num_units</code>	Defines the number of units at each Dense layer of a Neural Network. If a scalar value is provided, a single hidden layer is used. If a vector or list is provided, a deep architecture is fitted with corresponding hidden units.
<code>learning_rate</code>	Learning rate for the optimizer.
<code>activation</code>	Activation function to use on every layer of the Neural Network. Defaults to "relu".
<code>kernel_initializer</code>	Kernel initializer for the Dense layers. Defaults to Xavier initializer "glorot_normal" (Glorot and Bengio, 2010).
<code>kernel_regularizer</code>	Optional regularizer applied to the kernel weights.
<code>bias_regularizer</code>	Optional regularizer applied to the bias vector.
<code>bias_initializer</code>	Optional initializer for the bias vector (default "zeros").
<code>activity_regularizer</code>	Optional regularizer applied to the output of the layer.
<code>loss</code>	Loss function for training. Defaults to "mse", and can be any Keras built-in (e.g., mse, mae, logcosh, etc.) or a custom function. See 2.2.2 for more details on how to define custom loss functions.
<code>uncertainty_method</code>	Type of predictive inference: one of "none" or "epistemic". Enables uncertainty quantification with MC-dropout (see Section 2.2.2).
<code>alpha</code>	Significance level for interval estimation (default 0.05).
<code>forward_passes</code>	Number of Monte Carlo forward passes when computing epistemic uncertainty. Default 100.
<code>dropout_rate</code>	Dropout probability for MC-dropout estimation. Default 0.1.
<code>validation_split</code>	Fraction of data used as validation set during training. By default NULL.
<code>w_train</code>	Optional sample weights.
<code>bf_threshold</code>	Convergence criterion of the backfitting algorithm. Defaults to 0.001.
<code>ls_threshold</code>	Convergence criterion of the local scoring algorithm. Defaults to 0.1.
<code>max_iter_backfitting</code>	Maximum number of iterations of the backfitting algorithm. Defaults to 10.
<code>max_iter_ls</code>	Maximum number of iterations of the local scoring algorithm. Defaults to 10.
<code>seed</code>	Random number generator seed for reproducibility.
<code>verbose</code>	Verbosity mode (0 = silent, 1 = print messages). Defaults to 1.
<code>...</code>	Other arguments to pass on to the Adam optimizer (Kingma and Ba, 2014).

Table 3: Arguments of `neuralGAM()` function.

Installing and setting up the package

Since **neuralGAM** relies on **keras** for fitting the underlying neural network models, the package requires a working installation of Python (version 3.10) with **keras** and its TensorFlow backend (version 2.15). The setup can be performed directly from within **neuralGAM**, ensuring that all necessary dependencies are installed before using the package.

To ease this process, we provide the `install_neuralGAM()` to automate the Python setup required to run **neuralGAM**, which also creates a custom conda environment to isolate the package requirements for **neuralGAM**. This ensures no conflict with previous installations or dependencies from other R packages.

From the user's perspective, installation of all Python dependencies requires only a single call:

```
library(neuralGAM)
# one-time setup of Python dependencies
install_neuralGAM()
# reload neuralGAM to activate environment
library(neuralGAM)
```

After running the function (and restarting the R session), the user can immediately fit neural additive models with **neuralGAM** without needing to manually install Python, TensorFlow, or Keras.

Basic functionality

An example of the use of **neuralGAM**, illustrating its basic features, is given in the following example of code. **neuralGAM** is configured to fit a Deep Neural Network with 64 units on each of its three deep layers (`num_units`). In this example, the linear predictor is composed of a linear term in x and neural network fitted terms for z and w , and the response is assumed to follow a Gaussian distribution (`family`):

```
neuralGAM(y ~ x + s(z) + s(w), data = data, num_units = c(64, 64, 64),
  family = "gaussian", learning_rate = 0.001, activation = "relu",
  kernel_initializer = "glorot_normal", loss = "mse",
  kernel_regularizer = NULL, bias_regularizer = NULL,
  uncertainty_method = "none",
  bias_initializer = 'zeros', activity_regularizer = NULL,
  bf_threshold = 0.001, ls_threshold = 0.1, max_iter_backfitting = 10,
  max_iter_ls = 10, seed = NULL, verbose = 0, ...)
```

Other arguments of **neuralGAM** will be familiar to **keras** users: `activation` defines the activation function for the Dense layers of each fitted Neural Network (defaults to `relu`). **neuralGAM** uses Adam (Kingma and Ba, 2014) as an optimizer for stochastic gradient descent, whose behaviour can be customized using the `loss`, `activation`, `learning_rate`, `kernel_initializer`, `kernel_regularizer`, `bias_regularizer`, `bias_initializer`, and `activity_regularizer` arguments.

The backfitting and local scoring algorithms used to fit the GAM model can be configured using the `bf_threshold` and `ls_threshold` arguments which adjust the convergence criterion of both algorithms, while `max_iter_backfitting` and `max_iter_ls` adjust the maximum number of iterations of each algorithm. Finally, `seed` specifies an optional random number generator seed for algorithms dependent on randomization, `verbose` sets the verbosity of the printed outputs, while the `...` argument allows setting other arguments available for the Adam implementation in **keras** such as the exponential decay rate for the 1st and 2nd-moment estimates.

The summary method returns a summary of the fit where it is possible to observe the model architecture for both the neural networks and the linear terms (if working in a

semi-parametric setting) and the per-term network configuration (units at each layer, activation function, and training parameters and settings) and network layer configuration (such as layer type or regularization and initialization configurations). When using `uncertainty_method = "epistemic"`, dropout layers are included in the model architecture. These dropout layers are not only used during training for regularization, but also at prediction time (through Monte Carlo dropout) to quantify epistemic uncertainty and construct confidence intervals. At last, the summary function provides a summary of the model training history, including information about the evolution of the model fit for each covariate, allowing the monitoring of both the training loss (in this case, the Mean Squared Error `mse`), the deviance explained by the model, and the elapsed time on each training epoch. The training loss per-term and backfitting iteration can also be visualized using the `plot_history()` function.

Creating efficient and appropriate data visualizations becomes increasingly important as the complexity of the data increases. **neuralGAM** provides two different methods for plotting data: one based on R's standard plotting function (`plot.default`) in which the plot function inherits all arguments from graphics package and can be set as usual, and one based on `ggplot2` (Wickham, 2016). With this latter plot method, the `autoplot` function creates a `ggplot` object that can be modified later by the user just using the functionality provided in the `ggplot2` package. The following excerpt of code shows analogous plots using the two methods implemented:

```
plot(ngam, main = "My plot")
library(ggplot2)
autoplot(ngam, which = "terms", term = "z") + ggtitle("My plot")
```

The `predict` method for a fitted **neuralGAM** produces predictions on the link, response, or per-term scales via the `type` argument (default `"link"`). If `newdata` is omitted, the cached training data are used:

```
pred_link <- predict(ngam, test, type = "link")
```

We can also obtain the prediction on the response scale:

```
pred_resp <- predict(ngam, test, type = "response")
```

At last, a subset of per-term contributions can be selected with the `terms` argument when `type = "terms"` (the intercept is never included):

```
terms <- predict(ngam, newdata = test, type = "terms")
terms <- predict(ngam, newdata = test, type = "terms", terms = c("z", "w"))
```

Uncertainty estimation and visualization

Uncertainty estimates can be obtained directly through the `predict()` and `autoplot()` methods. For example, by setting `uncertainty_method = "epistemic"` during the fitting of the model, the user can request confidence intervals (CIs) for the conditional mean based on MC-Dropout. As noted in Table 3, `alpha` is the significance level, `dropout_rate` controls the Dropout probability, and `forward_passes` defines the number of stochastic passes to be run:

```
ngam <- neuralGAM(
  y ~ x + s(z) + s(w),
  data = data,
  num_units = c(64, 64, 64),
  uncertainty_method = "epistemic",
  alpha = 0.05, dropout_rate = 0.01,
  forward_passes = 500
)
```

To obtain the uncertainty in predictions, the argument `se.fit = TRUE` enables the computation of standard errors for the fitted mean. These errors capture only the *epistemic* uncertainty, estimated through the MC-Dropout procedure. When `type = "response"` is specified, the standard errors are mapped from the link scale to the response scale using the delta method. In addition to standard errors, the `predict` method also returns the lower and upper CIs at level $1 - \alpha$:

```
# obtain predictions with confidence intervals
pred <- predict(ngam,
  which = "response",
  se.fit = TRUE,
  interval = "confidence",
  alpha = 0.05)

# visualize epistemic uncertainty
autoplot(ngam,
  which = "terms",
  term terms = "w",
  interval = "confidence")
```

Validation on a hold-out set

A common concern when training neural networks is the risk of overfitting if the training hyperparameters are not properly tuned. **neuralGAM** includes cross-validation and hold-out splits via the `validation_split` argument, which reserves a user-specified fraction of the training data for validation purposes at each backfitting iteration. During model training, both training and validation losses are stored for every term-specific network and at every iteration, and can be visualized using the `plot_history()` function. This enables the user to monitor overfitting, adjust early stopping, and tune hyperparameters (such as `num_units`, `dropout_rate`, or `learning_rate`) in an informed way.

For more advanced evaluation using cross-validation, users can combine **neuralGAM** with standard resampling approaches in R, such as **caret** or **rsample**, by looping over training/validation folds and fitting multiple **neuralGAM** models:

```
if(!require("rsample")) install.packages("rsample")
library(rsample)

# Assuming dat is a data.frame with x1, x2 and response y

# 5-fold cross-validation
cv_folds <- vfold_cv(dat, v = 5)

cv_results <- lapply(cv_folds$splits, function(split) {
  train <- analysis(split)
  test  <- assessment(split)

  # Fit neuralGAM with a validation split
  ngam <- neuralGAM(
    y ~ s(x1) + x2,
    data = train,
    family = "gaussian",
    num_units = c(64, 64),
    validation_split = 0.2,
    uncertainty_method = "epistemic", forward_passes = 50
  )
})
```

```

# Predict on test set
preds <- predict(ngam, newdata = test, type = "response")

# Compute RMSE for this fold
sqrt(mean((test$y - preds)^2))
})

mean_rmse <- mean(unlist(cv_results))
print(mean_rmse)

```

Per-term architecture and custom loss definition

As previously introduced, beyond global hyperparameters, **neuralGAM** allows the user to specify architectural choices individually for each smooth term. For example:

```

y ~ s(x1, num_units = c(256,128), activation = "tanh") +
  s(x2, num_units = 64, kernel_regularizer = ::regularizer_l2(1e-4)) +
  x3

```

In this example, the smooth effect of x_1 is estimated with a two-layer network of sizes 256 and 128 using tanh activations, while x_2 is modeled with a smaller single-layer network with L_2 regularization. Terms not explicitly specified inherit the global defaults given in the main function call. This per-term flexibility allows customizing the training of complex covariates while keeping simpler covariates modeled with lightweight networks, which can improve both predictive performance and interpretability.

Additionally, **neuralGAM** allows the user to fit the model with any custom-built loss function, in addition to the standard losses provided by **keras**. The loss can be passed as a **keras** or **tensorflow** loss object, or defined directly by the user as an R function returning a valid tensor operation. This feature enables advanced use cases such as optimizing alternative objectives (e.g. Huber loss for robustness) or incorporating problem-specific penalties (e.g. monotonicity penalties):

```

huber_delta <- function(delta = 0.3) {
  function(y_true, y_pred) {
    tensorflow::tf$keras$losses$Huber(delta = delta)(y_true, y_pred)
  }
}

fit_ng <- neuralGAM(
  formula = y ~ s(x1) + s(x2) + s(x3),
  data = df,
  num_units = c(32, 32),
  activation = "relu",
  learning_rate = 1e-3,
  uncertainty_method = "epistemic",
  alpha = 0.95,
  # Use custom loss
  loss = huber_delta(0.3)
)

```

Diagnosis tools for model validation

When fitting any machine learning model, it is essential to verify whether model assumptions still hold. The function `diagnose()` provides a compact 2×2 grid of diagnostic plots (see Figure 1) summarizing key aspects of model fit: a Q-Q plot, residual histogram, residuals versus linear predictor, and observed versus fitted values. Residuals may be computed as

deviance, Pearson, or quantile residuals by means of the `residual_type` argument, allowing users to focus on distributional fit, or smooth normal diagnostics for discrete families. The Q-Q plot supports multiple strategies for generating theoretical quantiles by means of the `qq_method` argument, including uniform and simulation-based methods that respect the fitted response distribution, as well as a normal reference option.

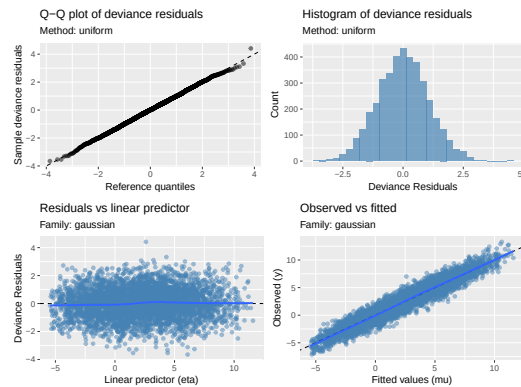


Figure 1: Example output from the `diagnose` function for a Gaussian model

These four diagnostic plots provide an easy-to-use and interpretable summary to verify model assumptions in **neuralGAM**, allowing users to assess whether residual distributions are consistent with model assumptions, detect potential problems such as heteroskedasticity, nonlinearity, or overdispersion, and to evaluate the adequacy of the link functions.

3 Illustrative examples

In this section, the use of the **neuralGAM** package is illustrated using both simulated and real data. We consider examples for different conditional distributions of the response variable, i.e., both Gaussian and binary. To contextualize the performance and interpretability of **neuralGAM**, we also include a comparative analysis with the **deepregression** package in both the simulated and real data scenarios, highlighting the differences in modeling flexibility, smoothness of the recovered partial functions, and predictive performance between the two approaches. In the application to real data, we also compare the predictive performance of **neuralGAM** and **deepregression** to a black-box neural network implemented in **keras**. The complete code for the generation of the simulated data and to reproduce the results from this section is publicly available at GitHub¹.

3.1 Application to simulated data

This subsection illustrates the capabilities of the **neuralGAM** package in a controlled simulation study. We generated a dataset of size $n = 30625$, split into 80% for training and 20% for testing. We consider the following predictor

$$\eta = \alpha + \sum_{j=1}^3 f_j(X_j),$$

with

$$f_j(X_j) = \begin{cases} X_j^2 & \text{if } j = 1, \\ 2X_j & \text{if } j = 2, \\ \sin X_j & \text{if } j = 3, \end{cases}$$

¹<https://github.com/inesortega/neuralGAM/blob/main/examples.R>

$\alpha = 2$, and covariates X_j drawn from an uniform distribution $U[-2.5, 2.5]$. The response follows a Gaussian distribution with $Y = \eta + \epsilon$, where ϵ is the error distributed in accordance to a $N(0, \sigma(x))$ in a homoscedastic situation with $\sigma(x) = 0.25$. Additional simulation scenarios including different response types and conditions on the variance of the error term ϵ are available at [Ortega-Fernandez et al. \(2024\)](#).

Table 4: Model architecture and configuration for **neuralGAM** and **deepregression** in the simulated study.

	neuralGAM	deepregression
Model type	Additive neural GAM, one independent NN per predictor (backfitting + local scoring)	Distributional regression with additive predictor, spline bases from mgcv
Hidden layers	1 per term network	1 spline basis layer
Units per layer	1024	Number of spline basis functions per smooth (thin plate regression splines, default $k = 10$)
Act. func.	ReLU (hidden layers), linear output	Linear (basis representation)
Regularization	Dropout (0.1) per hidden layer	Quadratic penalties on spline coefficients (via mgcv penalty matrices)
Optimization	Backpropagation with Adam optimizer (learning rate 0.001), trained jointly across subnetworks via backfitting	Penalized likelihood optimization in TensorFlow/Keras, initialized from spline bases
Uncertainty Quant.	Epistemic CIs via MC-Dropout (150 forward passes, Dropout rate 0.1)	No dropout-based epistemic uncertainty. CIs can be obtained via model ensembles
Training epochs	Maximum 10 iterations of backfitting algorithm (threshold 0.001)	100 epochs, early stopping with patience = 5

Both **neuralGAM** and **deepregression** were applied to this dataset, using smooth terms for each covariate. Performance was assessed in terms of mean squared error (MSE) on the training and test sets, and deviance explained on the training data. Table 5 summarizes the results, while Table 7 shows the architecture and configuration of each neural network.

Table 5: Model performance on simulated Gaussian data.

	MSE (Train)	MSE (Test)	Deviance Explained
neuralGAM	1.033	1.018	92.50%
deepregression	1.055	1.047	92.34%

In terms of predictive accuracy, both approaches generalize well from training to testing data, with nearly identical error rates, with **neuralGAM** achieving slightly better results across all metrics. The deviance explained is above 90% in both cases, highlighting that both methods are adequate for this problem.

Beyond predictive metrics, we compare the obtained partial effects, comparing them to the known true functions. Figures 2a and 2b show the estimated partial effects for each covariate for **neuralGAM** and **deepregression**, respectively. The true shape of each function is overlaid in blue for reference. Overall, both models recover the true nonlinear effects in X_1 and X_3 , as well as the linear effect in X_2 . Both methods recover the overall patterns, but **neuralGAM** more closely matches the true functional form for X_1 and X_2 where **deepregression** deviates more.

The **neuralGAM** approach provides the added benefit of direct confidence interval estimation via epistemic uncertainty quantification using MC Dropout. Unlike **neuralGAM**, **deepregression** does not natively offer epistemic uncertainty through this mechanism. Instead, users must rely on model ensembles, where multiple independently trained models are combined to capture variability in the estimated functions. In this setup, ensemble predictions are obtained from the expected response conditional on a covariate grid and

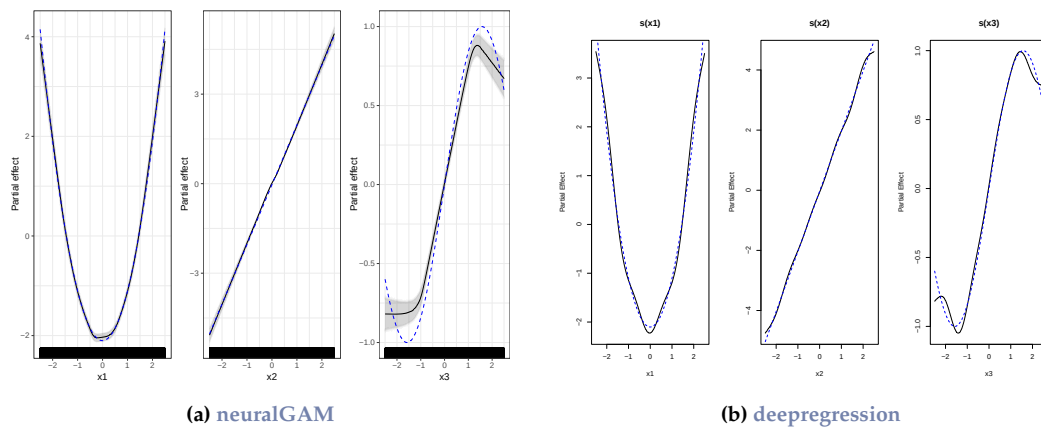


Figure 2: Estimated functions from both approaches. Black lines indicate estimated effects, blue dashed lines show the true functions. **neuralGAM** includes 95% epistemic confidence intervals obtained via MC Dropout.

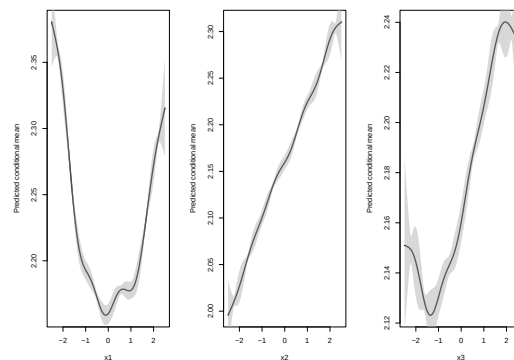


Figure 3: Estimated conditional mean functions with ensemble-based uncertainty bands from **deepregression**. Black lines indicate the ensemble mean prediction, while shaded areas show epistemic uncertainty obtained from the variability across 10 independently trained models.

reference values for the other covariates. These curves reflect the full conditional mean, which includes the intercept and all other model terms, and therefore live on the outcome scale. By contrast, the plotting method for smooth terms shown in Figure 2b displays centered partial effects, which isolates the contribution of a single smooth and makes the scale directly comparable to the true generative functions. Consequently, ensemble uncertainty bands are not on the same vertical scale as the centered smooth effects, although the shapes of the functions remain comparable.

Figure 3 highlights this difference in the uncertainty estimation approach: while **neuralGAM** produces per-term confidence intervals via MC-Dropout (Figure 2a), **deepregression** requires ensembles to approximate epistemic uncertainty, resulting in outcome-scale bands that are not directly comparable to centered smooth effects. From a computational point of view, MC Dropout concentrates cost at *inference* time, whereas ensembles multiply cost at *training* time. Let T be the time to train one model and P the time for a single forward pass over the prediction set. With 150 MC-Dropout draws, total cost is approximately $T + 150P$ and model storage is $1\times$; with a 10-member ensemble, total cost is about $10T + 10P$ and storage is $10\times$. Since in most applications $T \gg P$, MC Dropout is typically far cheaper (one trained model, P inference runs), at the expense of a larger prediction-time loop when uncertainty estimates are requested. Moreover, MC Dropout is efficient in terms of memory cost (only need to maintain one set of weights) and simplifies deployment, while ensembles can exploit hardware parallelism (training/predicting members in parallel) but require managing multiple models. In our configuration (150 draws vs. 10 members), MC Dropout trades ≈ 150 inference passes for avoiding ≈ 9 extra full trainings, which is usually

a favorable exchange whenever predictions are not orders of magnitude more expensive than training.

While both models recover the overall quadratic, linear, and sinusoidal effects, the shapes obtained with **neuralGAM** are smoother and more closely aligned with the true functions. By contrast, the **deepregression** ensemble estimates show greater variability and wider uncertainty bands, especially at the boundaries, which makes their interpretation less straightforward compared to the centered partial effects provided by **neuralGAM**.

3.2 Application to real data

We illustrate the capabilities of the **neuralGAM** package with another data set. Particularly, this section details an example of its application to real data taken from the NYC Flights 13 data set from the **nycflights13** package (Wickham, 2022). The data set includes airline on-time data for flights departing from all the airports in New York City during 2013. It also includes useful metadata on airlines, airports, weather conditions at different airports, and plane information.

We aim to predict whether a flight will be delayed (upon arrival) given departure flight information and certain weather conditions. With this in mind, firstly, we load the required data from the **nycflights13** package. We join the flights and weather data to obtain the weather conditions at a given airport (origin) and time (time_hour). We will focus on flights departing from Newark Liberty International Airport (origin == "EWR") in October, November, and December (month %in% c(10, 11, 12)), convert the temperature temp from Fahrenheit to Celsius, and construct a binary response variable delay which describes if the flight was delayed on arrival. The resulting class balance in the dataset is described in Table 6.

The following block of code shows how to load the required libraries and generate the EWR Oct–Dec subset and train/test splits from the **nycflights13** dataset:

```
if (!require("dplyr")) install.packages("dplyr", quiet = TRUE)
if (!require("nycflights13")) install.packages("nycflights13", quiet = TRUE)

suppressMessages(library(dplyr))
suppressMessages(library(nycflights13))

seed <- 1234
set.seed(seed)

dat <- flights %>%
  filter(origin == "EWR" & month %in% c(10, 11, 12)) %>%
  left_join(weather, by = c("origin", "time_hour")) %>%
  select(arr_delay, dep_delay, air_time, temp, humid) %>%
  data.frame()

dat$temp <- (dat$temp - 32) / 1.8
dat$delay <- ifelse(dat$arr_delay > 0, 1, 0)
dat <- dat[!rowSums(is.na(dat)),]

sample <- sample(nrow(dat), 0.8 * nrow(dat))
train <- dat[sample, ]; test <- dat[-sample, ]
```

In this scenario, we will compare three approaches: (i) **neuralGAM** (binomial neural additive model with per-term subnetworks and MC–Dropout for epistemic CIs), (ii) a distributional deep regression model fitted with **deepregression**, and (iii) a fully-connected, black-box neural network (Multi-Layer Perceptron, MLP) implemented in **keras**. All models use the same four covariates and are evaluated by AUC–ROC on the test set. In addition, we

Table 6: Class balance summary for the EWR Oct–Dec subset.

Class	Count	Proportion
On time (0)	16,215	56.8%
Delayed (1)	12,358	43.2%

also report the deviance explained on the training set, and perform a comparison in terms of the interpretability provided by analyzing the partial effect plots. Table 7 summarizes the architecture and hyperparameters of each method.

Table 7: Model architecture and configuration for **neuralGAM**, **deepregression**, and Keras in the simulated study.

	neuralGAM	deepregression	MLP (keras)
Model type	Additive neural GAM; binomial family	Distributional regression; additive predictor; spline bases from mgcv ; Bernoulli	Feedforward MLP
Hidden layers	Per-term subnetworks: <i>air_time</i> : 2 layers; others: 1 layer	One spline-basis “layer” per smooth	2 dense hidden layers
Units/layer	<i>air_time</i> : (256, 128); others: 128	Spline basis size (TPRS, default $k = 10$)	256, 128; output: 1
Activation	swish (<i>air_time</i> , <i>temp</i>); tanh (<i>humid</i>)	Linear (basis representation)	swish (hidden); sigmoid (output)
Regularization	Dropout 0.01 per hidden layer	Quadratic penalties on spline coefficients	No explicit weight decay; early stopping
Optimization	BF + LS; Adam (LR 0.001); bf 10^{-2} , ls 0.01; loss: MSE	Penalized likelihood; TensorFlow initialized from spline bases	Adam (LR 0.001); binary cross-entropy; batch size 128
Uncertainty	MC-Dropout CIs; 300 passes; rate 0.01, $\alpha = 0.05$	Using ensembles	None
Training	Iterative BF + LS until convergence	Up to 100 epochs; early stopping patience = 5	Up to 100 epochs; early stopping patience = 5

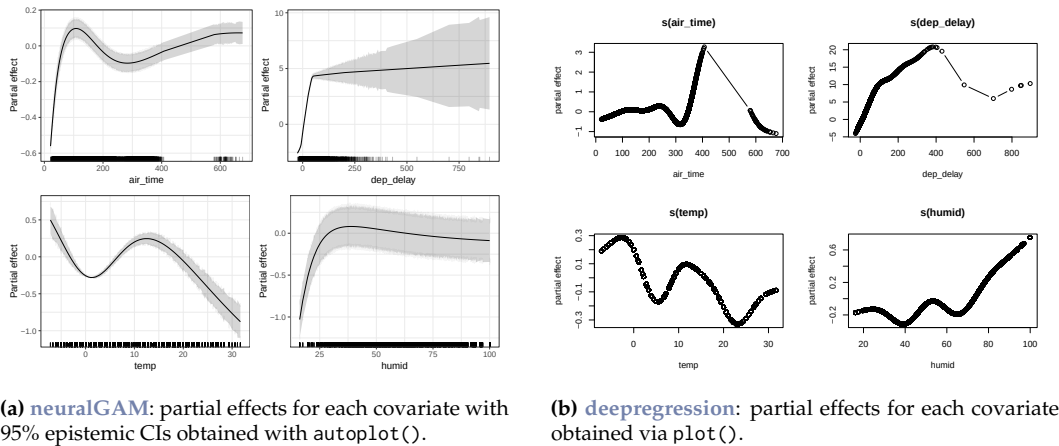


Figure 4: Estimated partial effects for the four covariates.

In this case, we showcase how **neuralGAM** can be personalized to use custom neural network architectures for each covariate for greater flexibility. For instance, the integration of **neuralGAM** with **keras** enables the direct use of different activation and loss functions. In this case, we showcase how to use *Swish* (Ramachandran et al., 2017), a self-regularization, non-monotonic activation function that has demonstrated stronger performance than ReLu across a number of challenging datasets, to model the `air_time` and `temp` covariates:

```
ngam_bin <- neuralGAM(
  delay ~ s(air_time, activation = "swish", num_units = c(256,128)) +
    s(dep_delay) +
    s(temp, activation = "swish") +
    s(humid, activation = "tanh" ),
  data = train, family = "binomial",
  num_units = 128, loss = "mse",
  bf_threshold = 1e-2, ls_threshold = 0.01, alpha = 0.05,
  uncertainty_method = "epistemic", forward_passes = 300,
  dropout_rate = 0.01,
  seed = seed, verbose = 1
)
```

In the case of **deepregression**, we specify smooth functions for all covariates to model the mean distribution and `family = "bernoulli"` to indicate a logistic regression model. We fit the model for 100 epochs (with early stopping), and compute the MSE and deviance explained by the model:

```
deepreg_mod_bin <- deepregression::deepregression(
  y = train$delay, data = train,
  list_of_formulas = list(
    loc = ~ 1 + s(air_time) + s(dep_delay) + s(temp) + s(humid)
  ),
  list_of_deep_models = NULL, family = "bernoulli"
)

deepreg_mod_bin %>% fit(
  epochs = 100, view_metrics = FALSE, early_stopping = TRUE, patience = 5
)
```

At last, to compare the performance of the neural-network based GAMs with a classic, black-box deep neural network, we fit an analogous neural network model using **keras**:

```
x_train <- as.matrix(train[, c("air_time", "dep_delay", "temp", "humid")])
y_train <- train$delay
x_test  <- as.matrix(test[, c("air_time", "dep_delay", "temp", "humid")])
y_test  <- test$delay

keras_model <- keras_model_sequential() %>%
  layer_dense(units = 256, activation = "swish", input_shape = ncol(x_train)) %>%
  layer_dense(units = 128, activation = "swish") %>%
  layer_dense(units = 1, activation = "sigmoid")

keras_model %>% compile(
  optimizer = optimizer_adam(learning_rate = 0.001),
  loss = "binary_crossentropy", metrics = c("accuracy")
)

keras_model %>% fit(x_train,
```

```

y_train,
epochs = 100,
batch_size = 128,
verbose = 1,
shuffle = FALSE,
view_metrics = FALSE,
callbacks = list(callback_early_stopping(monitor = "accuracy", patience = 5))

```

Table 8: Predictive performance on the flight delay task.

Model	AUC-ROC (Test)	Deviance Explained (Train)
neuralGAM	0.819	30.89%
deepregression	0.838	34.09%
keras	0.838	34.02%

Table 8 shows the obtained MSE in the test set and the deviance explained by each of the trained models. We can observe how the deviance explained by **deepregression** (34.09%) is similar to that obtained by **neuralGAM** (30.89%). Regarding performance on the test set, the AUC-ROC achieved by **deepregression** was 0.838, slightly higher than the value of 0.817 achieved by **neuralGAM** and comparable to the black-box neural network (0.830).

Black-box NNs can provide slightly higher predictive performance, achieving an AUC-ROC of 0.838 but offer no interpretability regarding the contribution of individual covariates. In contrast, **neuralGAM** and **deepregression** achieve comparable performance while enabling visualization of partial effects and uncertainty estimates. The black-box neural network, trained with two dense swish layers and a sigmoid output, learns a fully non-additive mapping between predictors and the outcome. This unconstrained approach allows it to capture interactions that additive models inherently miss and can sometimes improve predictive accuracy. However, it lacks the interpretability of **neuralGAM** or **deepregression**: no individual partial effects can be directly visualized, and it provides no intrinsic uncertainty quantification.

At last, focusing on the interpretability provided by the GAM-style methods, Figure 4 compares the estimated partial effects of **neuralGAM** and **deepregression** for the four covariates in the flight delay model. Again, it is worth mentioning that **deepregression** does not provide CI estimates directly, and must be obtained through computationally expensive model ensembles.

First, focusing on the partial effect of `air_time` **neuralGAM** shows a smooth non-monotonic, non-linear pattern that increases steeply for short flights (up to about 100 minutes), reaches a modest peak, and then gradually decreases for longer flights up to 400 minutes, and the delay probability increases again for flights with air time > 600 minutes. The model captures this relationship with smooth transitions and stable uncertainty, reflected by the relatively narrow confidence bands even at the extreme tail (beyond 600 minutes). In contrast, **deepregression** displays a more irregular curve for $s(\text{air\_time})$ with sharper oscillations: the shape fluctuates more abruptly, suggesting that the spline-based estimation may be more sensitive to local variations or noise. While this can offer flexibility, it comes at the cost of interpretability and stability, particularly in regions with limited data.

In the case of `dep_delay`, both models agree that higher departure delays increase the probability of arrival delay. For **neuralGAM**, the effect rises sharply up to approximately 100 minutes of departure delay and then flattens, showing a saturation effect. The narrow confidence interval in the main data region (0–300 minutes) demonstrates strong certainty and robust estimation, but the band widens beyond ~ 300 minutes, where data become sparse. In the case of **deepregression**, the effect of `dep_delay` appears almost linear across the range, with a continuous increase up to 400 min.

The temperature effect estimated by **neuralGAM** shows a smooth, nonlinear pattern in which flight delays are most likely at very cold temperatures (around -5°C), then decrease

sharply as conditions warm. The delay probability rises again between 5 °C and 10 °C, possibly reflecting transitional weather effects such as fog or humidity, and then declines steadily beyond 15°C, indicating that moderate temperatures are generally most favorable for on-time departures. Overall, the curve captures a complex relationship where both extremely low and moderately cool temperatures elevate delay risk, while warm, stable conditions correspond to smoother flight operations. The confidence interval around the estimated curve remains relatively narrow across most of the temperature range, implying that the model's uncertainty is low and the estimated effect is stable. Only at the extreme ends of the temperature distribution the interval is wider, which reflects data sparsity in those regions. Meanwhile, **deepregression** learns a more wiggly pattern with several local extrema across the range, including more oscillations between 10–30°C. These small fluctuations suggest overfitting to local noise, particularly where data density is lower.

At last, for the effect of humid, both models suggest that higher humidity implies a higher probability of delay. In **neuralGAM**, this pattern is monotonic and smooth, with an initially steep rise that levels off beyond 50% humidity. The narrow confidence bands indicate a stable and well-regularized estimate. By contrast, **deepregression** shows a wigglier response, with alternating increases and decreases before ultimately trending upward.

Overall, the **neuralGAM** results exhibit smooth, stable, and interpretable effects due to its neural network based regularization. Moreover, the Monte Carlo Dropout based intervals are coherent and narrow in data-dense regions, indicating reliable uncertainty quantification. In contrast, **deepregression**, while grounded in the spline framework from **mgcv**, tends to produce more irregular and wiggly smooths, especially in sparse data regions. This flexibility allows it to capture more complex shapes but also makes it more susceptible to overfitting and harder to interpret.

4 Summary and discussion

In this paper, we introduced **neuralGAM**, an R package for fitting Generalized Additive Neural Networks (GANNs) which combines the interpretability of additive models with the flexibility of neural networks. The proposed methodology extends the classical Generalized Additive Model (GAM) framework by using independent neural networks to estimate the contribution of each covariate to the response, ensuring additivity through the local scoring and backfitting algorithms. This white-box modeling approach enables the visualization and interpretation of each feature's partial effect while maintaining the ability to learn complex, non-linear patterns from data.

To contextualize the contribution of **neuralGAM**, we compare it against **deepregression**, a package that embeds **mgcv**-style spline bases and associated penalty matrices into deep learning architectures. Both approaches combine additive modeling with neural networks, but they differ substantially in scope and estimation strategy. **deepregression** is designed for distributional regression, allowing simultaneous modeling of mean, scale, and shape parameters, with smooth terms regularized through penalized likelihood optimization in TensorFlow / Keras. In contrast, **neuralGAM** focuses on interpretable additive predictors for the mean response, fitting one independent neural network per predictor via local scoring and backfitting.

Through comprehensive examples on both simulated and real data, we demonstrated the ability of **neuralGAM** to recover smooth effect estimates comparable to those obtained using other methods such as **deepregression** and generalize well in predictive tasks, while offering high flexibility, particularly in modeling sharp nonlinearities and capturing complex patterns in sparse regions. Importantly, **neuralGAM** is, to the best of our knowledge, the only R-based implementation of a GANN using independently-fitted (deep) neural networks, positioning it as a valuable tool for researchers seeking interpretable neural network implementations.

Overall, **neuralGAM** offers a strong compromise between interpretability and the flexibility provided by neural network-based estimation, producing stable, smooth, and easily interpretable effect estimates with uncertainty quantification. **deepregression** remains a

valuable benchmark for highly flexible spline-based additive modeling, but our initial experiments show that it can be more prone to overfitting and less stable in sparse regions. The Keras black-box neural network represents the opposite end of the spectrum: a powerful, unconstrained model focused on predictive performance but with limited transparency. These three approaches illustrate the trade-offs between interpretability, flexibility, and predictive power that need to be balanced in modern statistical learning.

The current version of the package supports Gaussian, binomial, and Poisson response distributions, making it suitable for multiple applications. As future work, several research directions have been identified to further extend the proposed algorithm. A first venue concerns the generalization of the model to allow multinomial logistic regression, thereby enabling its application to multi-class classification problems.

A second line of extension concerns the incorporation of interaction terms, allowing the model to capture joint effects between covariates. The local scoring algorithm can be adapted to estimate such terms, enabling the model to learn not only the marginal contributions of individual predictors but also their potential interactions.

At last, we also foresee the incorporation of support for automatic feature selection procedures, support for cyclic and spatial features, and the inclusion of additional uncertainty estimation methods, including conformal prediction (Karimi and Samavi, 2023), bootstrap-based methods, or the use of empirical quantiles to obtain confidence intervals from MC Dropout.

In terms of future software-development improvements, we foresee the support for additional deep learning backends and APIs beyond Keras and Tensorflow.

These additions will further enhance the interpretability, flexibility, and applicability of the model across a broader range of statistical and machine learning problems.

Computational details

The results in this paper were obtained using R 4.4.1 (2024-06-14 ucrt) with the **neuralGAM** 2.0.1 package. R itself and all packages used are available from the [Comprehensive R Archive Network \(CRAN\)](https://cran.r-project.org/).

Acknowledgments

This work was supported by the project “FAIR”, financed by the “European Union NextGeneration-EU” (https://next-generation-eu.europa.eu/index_es), the Recovery Plan, Transformation and Resilience (<https://planderecuperacion.gob.es/>), through Spanish National Cybersecurity Institute (INCIBE) (<https://www.incibe.es/>)” program under the fourth CPI call (CPI-2023) and the Xunta de Galicia (Centro singular de investigación de Galicia accreditation 2019-2022) and the European Union (European Regional Development Fund - ERDF); and the Grant PID2023-148811NB-I00 (MINECO/AEI/FEDER, UE).

References

- M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <https://www.tensorflow.org/>. Software available from tensorflow.org. [p260]
- R. Agarwal, L. Melnick, N. Frosst, X. Zhang, B. Lengerich, R. Caruana, and G. E. Hinton. Neural additive models: Interpretable machine learning with neural nets. *Advances in Neural Information Processing Systems*, 34, 2021. [p254]
- M. W. Beck. NeuralNetTools: Visualization and analysis tools for neural networks. *Journal of Statistical Software*, 85(11):1, 2018. [p255]
- P. Biecek. DALEX: Explainers for complex predictive models in R. *The Journal of Machine Learning Research*, 19(1):3245–3249, 2018. [p255]
- C. Brás-Geraldes, A. Papoila, and P. Xufre. Generalized additive neural network with flexible parametric link function: model estimation using simulated and real clinical data. *Neural Computing and Applications*, 31(3):719–736, 2019. ISSN 1433-3058. doi: 10.1007/s00521-017-3105-6. [p254]
- F. Chollet et al. Keras. <https://keras.io>, 2015. [p260]
- K. A. Copeland. Local polynomial modelling and its applications, 1997. [p257]
- C. De Boor, C. De Boor, E.-U. Mathématicien, C. De Boor, and C. De Boor. *A practical guide to splines*, volume 27. Springer-Verlag New York, 1978. [p257]
- F. K. Došilović, M. Brčić, and N. Hlupić. Explainable artificial intelligence: A survey. In *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*, pages 0210–0215. IEEE, 2018. [p254]
- Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059. PMLR, 2016. [p258]
- X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterton, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL <https://proceedings.mlr.press/v9/glorot10a.html>. [p261]
- T. Hastie and R. Tibshirani. Generalized Additive Models. *London: Chapman and Hall*, 1931 (11):683–741, 1990. [p256]
- K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080. doi: [https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8). URL <https://www.sciencedirect.com/science/article/pii/0893608089900208>. [p257]
- H. Karimi and R. Samavi. Quantifying deep learning model uncertainty in conformal prediction. In *Proceedings of the AAAI Symposium Series*, volume 1, pages 142–148, 2023. [p274]
- D. P. Kingma and J. Ba. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. [p261, 262]

- M. Kraus, D. Tschernutter, S. Weinzierl, and P. Zschech. Interpretable Generalized Additive Neural Networks. *European Journal of Operational Research*, 2023. [p254]
- S. Lang and A. Brezger. Bayesian P-splines. *Journal of Computational and Graphical Statistics*, 13(1):183–212, 2004. [p257]
- C. Molnar, G. Casalicchio, and B. Bischl. iml: An R package for interpretable machine learning. *Journal of Open Source Software*, 3(26):786, 2018. [p255]
- J. A. Nelder and R. W. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society Series A: Statistics in Society*, 135(3):370–384, 1972. [p255]
- I. Ortega-Fernandez, M. Sestelo, and N. M. Villanueva. Explainable generalized additive neural networks with independent neural network training. *Statistics and Computing*, 34(1):6, 2024. ISSN 1573-1375. doi: 10.1007/s11222-023-10320-5. URL <https://doi.org/10.1007/s11222-023-10320-5>. [p254, 267]
- W. J. E. Potts. Generalized Additive Neural Networks. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, pages 194–200, New York, NY, USA, 1999. Association for Computing Machinery. ISBN 1581131437. doi: 10.1145/312129.312228. [p254]
- P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017. [p271]
- D. Rügamer, C. Kolb, C. Fritz, F. Pfisterer, P. Kopper, B. Bischl, R. Shen, C. Bukas, L. Barros de Andrade e Sousa, D. Thalmeier, P. F. M. Baumann, L. Kook, N. Klein, and C. L. Müller. deepregression: A flexible neural network framework for semi-structured deep distributional regression. *Journal of Statistical Software*, 105(2):1–31, 2023. doi: 10.18637/jss.v105.i02. [p255]
- C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013. URL <https://arxiv.org/abs/1312.6199>. [p254]
- N. Umlauf, N. Klein, and A. Zeileis. BAMLSS: Bayesian additive models for location, scale and shape (and beyond). *Journal of Computational and Graphical Statistics*, 27(3):612–627, 2018. doi: 10.1080/10618600.2017.1407325. [p255]
- M. P. Wand and M. C. Jones. *Kernel smoothing*. CRC press, 1994. [p257]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p260, 263]
- H. Wickham. *nycflights13: Flights that Departed NYC in 2013*, 2022. URL <https://github.com/hadley/nycflights13>. R package version 1.0.2. [p269]
- S. N. Wood. Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. *Journal of the Royal Statistical Society (B)*, 73(1):3–36, 2011. doi: 10.1111/j.1467-9868.2010.00749.x. [p255]
- Z. Yang, A. Zhang, and A. Sudjianto. GAMI-Net: An explainable neural network based on generalized additive models with structured interactions. *Pattern Recognition*, 120:108192, 2021. [p254]

Ines Ortega-Fernandez

Galician Research and Development Center in Advanced Telecommunications (Gradiant), Vigo (Spain)

ORCID: 0000-0002-8041-6860

iortega@gradient.org

Marta Sestelo

Galician Centre for Mathematical Research and Technology (CITMAga), Santiago de Compostela (Spain)

Universidade de Vigo, Department of Statistics and O.R. & SiDOR Group, 36310 Vigo, Spain

ORCID: [0000-0003-4284-6509](https://orcid.org/0000-0003-4284-6509)

sestelo@uvigo.es

generalRSS: Sampling and Inference for Balanced and Unbalanced Ranked Set Sampling

by Chul Moon and Soohyun Ahn

Abstract Ranked set sampling (RSS) is a stratified sampling method that improves efficiency over simple random sampling (SRS) by utilizing auxiliary information for ranking and stratification. While balanced RSS (BRSS) assumes equal allocation across strata, unbalanced RSS (URSS) allows unequal allocation, making it particularly effective for skewed distributions. The **generalRSS** package provides extensive tools for both BRSS and URSS, addressing limitations in existing RSS software that primarily focus on balanced designs. It supports RSS data generation, efficient sample allocation strategies for URSS, and statistical inference for both balanced and unbalanced designs. This paper presents the RSS methodology and demonstrates the utility of **generalRSS** through two medical data applications: a one-sample mean inference and a two-sample area under the curve (AUC) comparison using NHANES datasets. These applications illustrate the practical implementation of URSS and show how **generalRSS** facilitates ranked set sampling and inference in real-world data analysis.

1 Introduction

Randomized experiments are fundamental to modern scientific discovery and typically depend on simple random sampling (SRS) to select units. While increasing the sample size can enhance the efficiency of such experiments, this approach may be impractical under resource constraints. Ranked set sampling (RSS), first introduced by McIntyre (1952), offers a cost-effective alternative. RSS is a stratified sampling method that utilizes auxiliary ranking information to create strata (Stokes and Sager, 1988). When each rank stratum includes an equal number of samples, it is called balanced RSS (BRSS). In contrast, unbalanced RSS (URSS) occurs when the number of samples differs across strata.

RSS has a rich history of research and development (Chen et al., 2004; Wolfe, 2012, and references therein). Research has shown that BRSS consistently outperforms SRS with an equivalent sample size in estimation precision, as demonstrated by Takahashi and Wakimoto (1968). Numerous studies have explored the efficiency of RSS estimators (Dell and Clutter, 1972; David and Levine, 1972; Stokes, 1980). For skewed distributions, URSS estimators can achieve even greater efficiency than BRSS and SRS counterparts (Ahn et al., 2017, 2024; Bocci et al., 2010; Chen and Bai, 2000; Ozturk and Wolfe, 2000). However, the performance of URSS depends heavily on the allocation of replicates among strata. Improper allocation can lead to inefficiencies, sometimes even worse than SRS. To address this, several studies (Bhoj and Chandra, 2020; Chen and Bai, 2000; McIntyre, 1952; Wang et al., 2004) have proposed appropriate allocation rules, with Neyman allocation being the most widely used due to its optimal variance-minimizing properties when estimating the population mean. As a result, most of the existing literature on URSS has focused on this allocation strategy (Chen et al., 2004; Takahashi and Wakimoto, 1968; Wang et al., 2017). Recently, Ahn et al. (2022a) defined a sufficient set of allocation schemes to ensure that URSS achieves greater efficiency in population mean estimation compared to BRSS. They also introduced two practical allocation adjustments to enhance the efficiency of URSS designs beyond that of BRSS.

Several R packages are available for RSS, including **RSSampling** (Sevinc et al., 2018), **NSM3** (Schneider et al., 2024), **RSStest** (Gökpınar et al., 2023), and **RankedSetSampling** (Ozturk et al., 2021). **RSSampling** supports a wide range of RSS extensions, offering both sampling tools and statistical inference. **NSM3** includes only classical RSS as a sampling procedure and provides critical value calculations for nonparametric inference. **RSStest**

focuses on sampling and mean testing for RSS and MRSS and includes simulation tools under normal distributions. Finally, [RankedSetSampling](#) extends its scope to joint probability sampling (JPS) and offers mean and variance estimation for RSS. However, these packages are generally limited to balanced RSS designs, restricting their applicability for both sampling and inference in scenarios that require more flexibility.

BRSS designs are commonly implemented due to their simplicity and ease of implementation. However, in real-world survey studies, missing data often result in URSS scenarios, where BRSS methodologies become impractical. URSS offers a flexible alternative by allowing unequal allocation across strata, making it particularly effective for skewed distributions where unequal allocation can significantly improve estimation efficiency. Despite its advantages, implementing URSS has been challenging due to the lack of accessible tools in existing software packages.

R package [generalRSS](#) (Ahn and Moon, 2025) was developed to address these limitations by fully supporting URSS while maintaining compatibility with BRSS. It provides parametric and nonparametric inference tools for population means, medians, AUCs, and proportions. Additionally, it offers functions for generating ranked set samples with specified allocations and optimizing allocation strategies to improve the efficiency of URSS designs. The flexibility of URSS allows users to tailor sampling designs to real-world constraints, making it particularly valuable when balanced designs are impractical. By treating BRSS as a special case of URSS with equal allocation across strata, [generalRSS](#) extends the applicability of RSS beyond BRSS methods. This versatility makes it a practical tool for a wide range of applications, including environmental or medical studies, where flexible sampling strategies are essential.

The rest of this paper is organized as follows. Section 2 provides an overview of ranked set sampling, with Section 2.1 covering BRSS and Section 2.2 discussing URSS, including their respective procedures. Section 3 describes the analysis approaches and functions available in [generalRSS](#) and compares them with existing R packages. Section 4 evaluates the performance of RSS relative to SRS in two inference problems using data from the US National Health and Nutrition Examination Survey (NHANES) (Pruim, 2015; Centers for Disease Control and Prevention, 2023). Finally, Section 5 summarizes the capabilities of [generalRSS](#), its applications in inference problems, and future directions for inference methods and sampling designs.

2 Ranked set sampling

In this section, we illustrate the sampling procedure of RSS by detailing its design and associated notations. An RSS dataset of size n can be represented as

$$\{(y_i, h_i, r_i), i = 1, 2, \dots, n\}, \quad (1)$$

where y_i denotes the i -th observation, h_i represents its rank among r_i independent observations, and r_i denotes the set size. Typically, RSS assumes a fixed set size, i.e., $r_i = H$ for all $i = 1, 2, \dots, n$, and h_i takes values in $\{1, \dots, H\}$. The number of observations assigned to rank h is given by $n_h = \sum_{i=1}^n I(h_i = h)$, where $I(\cdot)$ denotes the indicator function. The total sample size of the RSS data is then given by $n = \sum_{h=1}^H n_h$. These notations define the structure of RSS data and its implementation in [generalRSS](#).

RSS data can also be represented using the notation $Y_{[h],r}$ denoting the r -th observation assigned to rank h , with $h = 1, \dots, H$ and $r = 1, \dots, n_h$. The RSS sampling process consists of the following steps:

- Step 1: Draw an i.i.d. sample of size H from the target population.
- Step 2: Rank the H sampled units using an auxiliary variable without measuring the variable of primary interest.
- Step 3: Measure the unit ranked as the h -th smallest and discard the remaining units.

Step 4: Repeat Steps 1-3 for each rank h up to H .

These steps define one cycle of RSS, as illustrated in Figure 1. In each cycle, an i.i.d. random sample of size H is selected, ranked, and one unit is measured per stratum, yielding a sample consisting of H strata with one observed unit. In **generalRSS**, we adopt the infinite population framework, where independently repeating the cycles (i.e., with replacement across cycles) yields multiple observations within each stratum and ultimately forms the final ranked set sample.

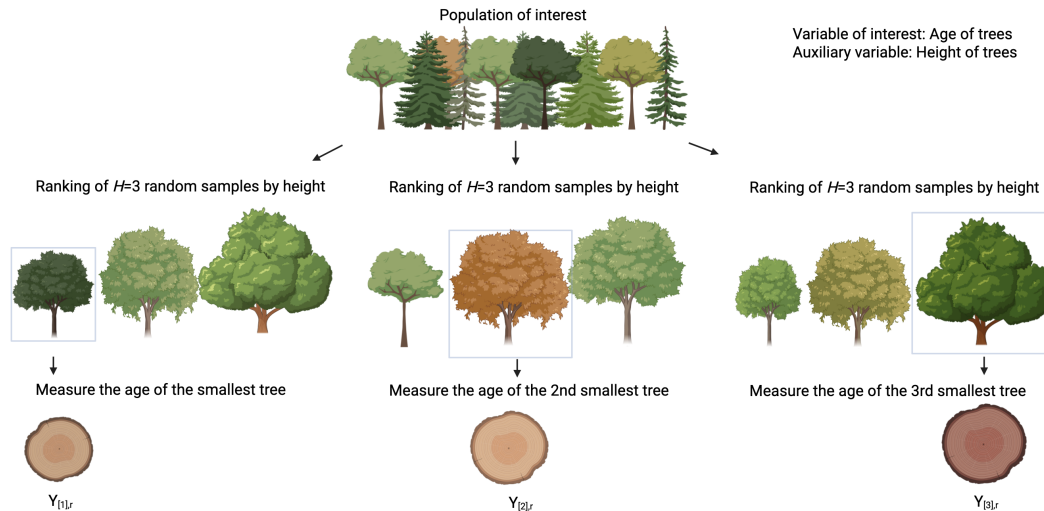


Figure 1: Example of a cycle r of RSS of a set size of $H = 3$ and ranked set samples. The variable of interest (Y) is tree age and the auxiliary variable (X) is tree height.

2.1 Balanced RSS

BRSS assumes equal allocation across strata, meaning that each rank h is assigned the same number of observations, i.e., $n_h = m$ for every $h = 1, 2, \dots, H$. To obtain this, the RSS sampling cycle is repeated m times, resulting in a BRSS design with the sample allocation $(m, m, \dots, m)$. For $H = 3$, followed by [Chen et al. \(2004\)](#), a BRSS procedure with m cycles can be represented as:

$$\begin{aligned}
 &\text{Cycle } r = 1 \\
 &X_{[1]11} \leq X_{[2]11} \leq X_{[3]11} \implies Y_{[1],1} \\
 &X_{[1]21} \leq X_{[2]21} \leq X_{[3]21} \implies Y_{[2],1} \\
 &X_{[1]31} \leq X_{[2]31} \leq X_{[3]31} \implies Y_{[3],1} \\
 &\text{Cycle } r = 2 \\
 &X_{[1]12} \leq X_{[2]12} \leq X_{[3]12} \implies Y_{[1],2} \\
 &X_{[1]22} \leq X_{[2]22} \leq X_{[3]22} \implies Y_{[2],2} \\
 &X_{[1]32} \leq X_{[2]32} \leq X_{[3]32} \implies Y_{[3],2} \\
 &\dots\dots\dots \\
 &\text{Cycle } r = m \\
 &X_{[1]1m} \leq X_{[2]1m} \leq X_{[3]1m} \implies Y_{[1],m} \\
 &X_{[1]2m} \leq X_{[2]2m} \leq X_{[3]2m} \implies Y_{[2],m} \\
 &X_{[1]3m} \leq X_{[2]3m} \leq X_{[3]3m} \implies Y_{[3],m}
 \end{aligned}$$

where $X_{[h]ir}$ denotes the h -th smallest auxiliary variable of the i -th SRS of size H , selected in the r -th cycle. The resulting BRSS dataset is given by:

$$\{Y_{[1],1}, \dots, Y_{[1],m}, Y_{[2],1}, \dots, Y_{[2],m}, \dots, Y_{[H],1}, \dots, Y_{[H],m}\}.$$

2.2 Unbalanced RSS

While BRSS assumes equal allocation across strata, real-world applications often involve scenarios where such a balance is impractical. For instance, skewed data, missing observations, or resource constraints can result in an unbalanced design. URSS allows for greater flexibility by permitting unequal allocations across strata, enabling more efficient sampling when strata exhibit higher variability or measurement difficulty. For example, if the population of interest has strata with differing levels of variability in measurements, allocating more samples to strata with higher variability can improve estimation efficiency, which BRSS cannot achieve.

Unlike BRSS, which completes a full cycle by selecting one sample for each stratum in every iteration, URSS follows an incomplete cycle structure, where strata exit the sampling process once they reach their allocated sample sizes n_h . This means that different strata complete different numbers of iterations, leading to an unbalanced design. In Step 4 of the URSS procedure, Steps 1-3 are repeated only for strata that have not yet reached n_h , ensuring that each stratum meets its predefined sample allocation.

For $H = 3$, without loss of generality, we may assume an allocation such that $n_1 \leq n_2 \leq n_3$. Then, a URSS with sample allocation (n_1, n_2, n_3) is obtained as:

$$\begin{aligned} &\textbf{Cycle } r = 1, \dots, n_1 \\ &X_{[1]1r} \leq X_{[2]1r} \leq X_{[3]1r} \implies Y_{[1],r} \\ &X_{[1]2r} \leq X_{[2]2r} \leq X_{[3]2r} \implies Y_{[2],r} \\ &X_{[1]3r} \leq X_{[2]3r} \leq X_{[3]3r} \implies Y_{[3],r} \\ \\ &\textbf{Cycle } r = n_1 + 1, \dots, n_2 \\ &X_{[1]2r} \leq X_{[2]2r} \leq X_{[3]2r} \implies Y_{[2],r} \\ &X_{[1]3r} \leq X_{[2]3r} \leq X_{[3]3r} \implies Y_{[3],r} \\ \\ &\textbf{Cycle } r = n_2 + 1, \dots, n_3 \\ &X_{[1]3r} \leq X_{[2]3r} \leq X_{[3]3r} \implies Y_{[3],r} \end{aligned}$$

For a general H , the resulting URSS data can be represented as:

$$\{Y_{[1],1}, \dots, Y_{[1],n_1}, Y_{[2],1}, \dots, Y_{[2],n_2}, \dots, Y_{[H],1}, \dots, Y_{[H],n_H}\}.$$

In practice, missing values during the sampling process in BRSS often lead to a URSS design. This highlights the importance of methodologies that can accommodate both balanced and unbalanced designs seamlessly.

3 **generalRSS** package

In Section 2.2, we introduced the RSS procedure, which involves selecting samples from a population and ranking them based on an auxiliary variable. In this section, we demonstrate how the **generalRSS** package facilitates the implementation of RSS methodologies, enabling efficient sampling and accurate inference of population parameters.

The **generalRSS** package, available on CRAN at <https://cran.r-project.org/package=generalRSS>, can be installed and loaded using the following commands:

```
install.packages("generalRSS")
library("generalRSS")
```

The package incorporates some functions from R packages **emplik** (Zhou et al., 2024) and **rootSolve** (Soetaert and Soetaert, 2023) for solving empirical likelihood problems. It is designed to address both sampling and inference problems in RSS with a focus on unbalanced designs. The package comprises two main components: (i) functions for sampling and sample allocation and (ii) functions for statistical inference.

3.1 Sampling and allocation design

The **generalRSS** package introduces two sampling functions to facilitate RSS procedures as well as a design function to calculate efficient sample allocations for URSS as summarized in Table 1. The function `rss.sampling` generates ranked set samples directly from a population

Function	Description
<code>rss.sampling</code>	Generate ranked set samples
<code>rss.simulation</code>	Generate example ranked set samples
<code>rss.prop.sampling</code>	Generate ranked set samples for proportions
<code>rss.prop.simulation</code>	Generate example ranked set samples for proportions
<code>rss.design</code>	Calculate efficient sample allocations for RSS

Table 1: Functions for the sampling and allocation design in the **generalRSS** package.

dataset containing an auxiliary variable (X) and, optionally, a variable of interest (Y). Internally, the function repeatedly draws sets of size H , ranks them by X , and selects the unit of rank h in each set, following the RSS procedure described in Section 2.2. This process is iterated across strata until the user-specified allocation $(n_1, \dots, n_H)$ is satisfied. When the auxiliary variable coincides with the variable of interest (e.g., $X = Y$), perfect ranking is assumed. If the variable of interest (Y) is not provided, the function selects sample IDs based on X , returning a data frame containing IDs and ranks (r_i). If Y is provided, the function directly returns RSS data with observation (y_i). For illustration, we use the `iris` dataset as the population, treating `Sepal.Length` as the outcome variable (Y) and `Petal.Length` as the auxiliary variable (X), with a set size $H = 3$ and sample allocations $(n_1 = 2, n_2 = 2, n_3 = 2)$:

```
data("iris")
id = 1:nrow(iris)
rss.data = rss.sampling(ID = id, Y = NULL, X = iris$Petal.Length, H = 3,
+ nsamp = c(2,2,2))
head(rss.data)
```

rank	ID
1	1 80
2	1 34
3	2 91
4	2 121
5	3 106
6	3 122

Since Y was not provided, the function returns only the selected IDs and ranks. The actual measurement of Y must be collected for these selected samples to complete the dataset. When Y is included in the population dataset, the function automatically extracts the corresponding values and returns a data frame containing ranks (r_i), IDs, and observations (y_i).


```

rss.data = rss.sampling(ID = id, Y = iris$Sepal.Length, X = iris$Petal.Length, H = 3,
                        nsamp = c(2,2,2))
head(rss.data)

  rank ID   y
1    1  80 5.7
2    1  34 5.5
3    2  91 5.5
4    2 121 6.9
5    3 106 7.6
6    3 122 5.6

```

The `nsamp` parameter specifies the number of cycles (n_h) for each stratum. When $n_h = m$ for all h , the output is BRSS data. If n_h varies across strata, the design corresponds to a URSS scheme.

The function `rss.simulation` generates ranked set samples by simulating data from selected probability distributions, including normal, t , and lognormal distributions, for illustrative purposes. Users can specify the set size (H), the number of cycles (n_h), and ranking accuracy (ρ) through the `rho` parameter, and a mean shift through `delta`, enabling the simulation of populations with group-specific differences. The key feature is that ρ directly controls the quality of ranking, from perfect ranking $\rho = 1$ (i.e., no ranking errors) to increasingly noisy ranking as ρ decreases.

The ranking accuracy in the simulated data is controlled through a linear ranking model defined as $X_i = Y_i + \epsilon_i$ for $i = 1, 2, \dots, n$, where ϵ_i represents independent normal random variables with mean zero and variance chosen to achieve a specific correlation $\rho = \text{Corr}(X, Y)$. The `rho` parameter determines the correlation between the outcome variable (Y) and the auxiliary variable (X). The outcome variable Y is drawn from one of three fixed shapes: $N(0, 1)$, $t_5/\sqrt{5/3}$, or $\text{Lognormal}(0, \sqrt{0.481}) - 1.27$. Thus, in all cases Y has approximately mean 0 and variance 1. This standardization ensures that simulation results reflect distributional shape and ρ , rather than arbitrary parameters. The function is intended as an illustrative tool and therefore does not allow user-specified distributional parameters such as degrees of freedom or scale. The `dist` parameter determines the underlying distribution shape. The function returns a data frame containing ranks (r_i) and outcomes (y_i). For example, the following code generates RSS data from a normal distribution with a mean shift of $\delta = 0$ and a ranking quality of $\rho = 0.8$:

```

rss.simulation(H=3, nsamp=c(2,2,2), dist="normal", rho=0.8, delta=0)

```

The functions `rss.prop.sampling` and `rss.prop.simulation` are the sampling and simulation functions for proportions. The variable of interest (Y) is a binary variable of 0 and 1, corresponding to failure and success, respectively. We assume the perfect ranking ($X = Y$) for proportions. For illustration, we use the `iris` dataset as the population, defining both the outcome and the auxiliary variable ($X = Y$) as a binary indicator of whether `Sepal.Length` is less than 5.8:

```

data(iris)
id = 1:nrow(iris)
X = ifelse(iris$Sepal.Length < 5.8, 0, 1)
head(X)

0 0 0 0 0 0

```

Once the binary variable is defined, we apply `rss.prop.sampling` to generate RSS data with a set size of $H = 3$ and sample allocations ($n_1 = 2, n_2 = 2, n_3 = 2$):

```

rss.prop.data = rss.prop.sampling(ID = id, X = X, H = 3, nsamp = c(2,2,2))
rss.prop.data

```

	rank	ID
1	1	14
2	1	43
3	2	91
4	2	92
5	3	72
6	3	78

The `rss.prop.simulation` function generates ranked set samples for proportions by simulating data based on a true population proportion p . The `p` parameter represents the true proportion of successes and serves as the success probability in a binomial distribution of Y . For example, the following code generates RSS data with a proportion of $p = 0.6$:

```
rss.prop.data = rss.prop.simulation(H = 3, nsamp = c(2,2,2), p = 0.6)
rss.prop.data
```

	rank	y
1	1	0
2	1	0
3	2	1
4	2	1
5	3	1
6	3	1

The `rss.design` function evaluates the efficiency of a current URSS design for mean or proportion estimation based on either an initial RSS dataset or predefined sample allocation. If the design is inefficient compared to SRS or BRSS, the function calculates improved sample allocations to enhance efficiency. The function offers three allocations for estimating a population mean: integer Neyman allocation (Wright, 2012), local ratio consistent (LRC) allocation (Ahn et al., 2022a), and adjusted Neyman (AN) allocation (Ahn et al., 2022a).

Let $\mathbf{n} = (n_1, \dots, n_H)$ with total $n = \sum_{h=1}^H n_h$, and let σ_h^2 be the variance of the h -th rank stratum. The Neyman allocation $\tilde{\mathbf{n}} = (\tilde{n}_1, \tilde{n}_2, \dots, \tilde{n}_H)$ minimizes the variance of the RSS mean estimator with

$$\tilde{n}_h = \frac{\sigma_h}{\sum_{l=1}^H \sigma_l} \cdot n, \quad h = 1, \dots, H.$$

Since $\tilde{n}_h$ is generally noninteger, the function `rss.design` implements the integer allocation proposed by (Wright, 2012), which applies the method of equal proportions to the noninteger Neyman allocation. This procedure allocates one unit to each stratum and then distributes the remaining $(n - H)$ units according to priority values $\sigma_h / \sqrt{k(k+1)}$ ($k = 1, 2, \dots$), thereby yielding integer Neyman allocations $(n_1, \dots, n_H)$ that sum exactly to n .

Ahn et al. (2022a) proposed the allocation set

$$\mathcal{N} = \left\{ \mathbf{n} : 1 \leq \frac{n_{h+1}}{n_h} \leq \frac{\sigma_{h+1}^2}{\sigma_h^2}, \quad h = 1, \dots, H-1 \right\}$$

and showed that this set provides a sufficient condition for efficiency. Whenever $\mathbf{n} \in \mathcal{N}$, the URSS estimator is guaranteed to be at least as efficient as the BRSS estimator, that is, $\text{Var}(\hat{\mu}_{URSS}) \leq \text{Var}(\hat{\mu}_{BRSS})$. The LRC allocation is obtained by an adjustment procedure that moves a given allocation into $\mathcal{N}$ with minimal changes. Specifically, if the upper ratio condition $n_{h+1}/n_h \leq \sigma_{h+1}^2/\sigma_h^2$ is violated, one unit is added to the smaller stratum n_h ; if the lower condition $n_{h+1}/n_h \geq 1$ is violated, one unit is added to the larger stratum n_{h+1} . These updates are repeated until all local ratio conditions hold, producing an allocation that lies in $\mathcal{N}$.

The AN allocation raises each stratum to at least its Neyman level,

$$n_h^{AN} = \max(n_h^N, n_h), \quad h = 1, \dots, H,$$

so that $\mathbf{n}^{AN}$ is closer to Neyman and typically more efficient than the initial $\mathbf{n}$ because it uses more samples and moves toward the Neyman allocation, even though it is not guaranteed to lie in $\mathcal{N}$.

These methods ensure greater efficiency with minimal additional samples compared to the original URSS design. For example, we consider a simulated URSS dataset with an initial allocation of (3, 10, 5). The function calculates optimized allocations as follows:

```
rss.data = rss.simulation(H = 3, nsamp = c(3,10,5), dist = "t", rho = 1, delta = 0)
rss.design(rss.data)

$original.n
n1 n2 n3
 3 10  5

$Integer.Neyman
n1 n2 n3
 4  5  9

$Adj.Neyman
n1 n2 n3
 4 10  9

$LRC.allocation
n1 n2 n3
 6 10 10
```

In this example, the function first computes the integer Neyman allocation ($n_1 = 4, n_2 = 5, n_3 = 9$) and then recommends adding 1 sample to the first stratum and 4 samples to the third stratum, resulting in an adjusted Neyman allocation ($n_1 = 4, n_2 = 10, n_3 = 9$). Similarly, it computes the LRC allocation ($n_1 = 6, n_2 = 10, n_3 = 10$) by adding 3 samples to the first stratum and 5 samples to the third stratum. These adjusted allocations improve the efficiency of the original URSS design.

The `rss.design` function also computes the optimal Neyman allocation for estimating a population proportion under perfect rankings (Chen et al., 2006). Let p_h denote the success probability in the h -th stratum. The variance of the URSS proportion estimator is minimized with the Neyman allocation for proportions given by

$$\tilde{n}_h = n \frac{\sqrt{p_h(1-p_h)}}{\sum_{l=1}^H \sqrt{p_l(1-p_l)}}, \quad h = 1, \dots, H.$$

The following example compares an initial URSS allocation ($n_1 = 10, n_2 = 15, n_3 = 20$) to the computed Neyman allocation:

```
rss.prop.data = rss.prop.simulation(H = 3, nsamp = c(10,15,20), p = 0.5)
rss.design(rss.prop.data, prop = TRUE)

$original.n
n1 n2 n3
10 15 20

$Neyman.proportion
      n1      n2      n3
12.45017 19.36741 13.18242
```

Here, the computed Neyman allocation aligns more closely to the true Neyman allocation ($n_1 = 12.814, n_2 = 19.373, n_3 = 12.814$), as suggested by Chen et al. (2006), compared to the initial URSS design.

3.2 Statistical inference

The **generalRSS** provides six statistical inference functions for estimating and testing the population means, medians, proportions, and AUCs using RSS data. These functions include both parametric and nonparametric methods, as summarized in Table 2.

Function	Description
<code>rss.z.test</code>	RSS z-test for one-sample and two-sample problems
<code>rss.t.test</code>	RSS t-test for one-sample and two-sample problems
<code>rss.ELR.test</code>	RSS empirical likelihood ratio test for one-sample problems
<code>rss.sign.test</code>	RSS Sign test for one-sample problems
<code>rss.prop.test</code>	RSS proportion test for one-sample problems
<code>rss.AUC.test</code>	RSS empirical likelihood ratio test for two-sample problems

Table 2: Statistical inference functions in the **generalRSS** package.

The `rss.z.test` function provides point estimation, confidence intervals (CIs), and hypothesis testing for the population mean using a normal approximation for RSS data (Chen et al., 2004; Ahn et al., 2024). It uses the asymptotic pivotal method to test $H_0 : \mu = \mu_0$, where the test statistic is:

$$\frac{\hat{\mu}_{\text{RSS}} - \mu_0}{\hat{\sigma}_{\mu_{\text{RSS}}}} \xrightarrow{d} N(0, 1) \quad (2)$$

where $\hat{\mu}_{\text{RSS}} = \frac{1}{H} \sum_{h=1}^H \frac{1}{n_h} \sum_{r=1}^{n_h} Y_{[h],r}$, $\hat{\sigma}_{\mu_{\text{RSS}}}^2 = \frac{1}{H^2} \sum_{h=1}^H \frac{1}{n_h(n_h-1)} \sum_{r=1}^{n_h} (Y_{[h],r} - \bar{Y}_{[h]})^2$, and $\bar{Y}_{[h]} = \sum_{r=1}^{n_h} Y_{[h],r} / n_h$.

Similarly, the `rss.t.test` function performs mean inference using a t-distribution approximation of the pivot statistics in equation (2) (Ahn et al., 2014). It supports two methods for estimating degrees of freedom: (i) **sample** applies a Welch-type approximation replacing the in-stratum variances with their sample estimates and (ii) **naive** uses $n - H$. The choice is controlled by the `method` parameter. In practice, we recommend **naive** for BRSS due to its simplicity, except for small-sample skewed data where **sample** performs better. For URSS, the sample-based estimator **sample** is generally preferred. Both `rss.z.test` and `rss.t.test` functions handle one-sample and two-sample problems using the same pivotal statistics but differ in their approximation methods. The following example demonstrates the `rss.t.test` function for a two-sample problem, testing the hypothesis that the population mean difference is 0:

```

rss.data1 = rss.simulation(H = 3, nsamp = c(6,6,6), dist = "normal",
                           rho = 0.8, delta = 0)
rss.data2 = rss.simulation(H = 3, nsamp = c(6,8,10), dist = "normal",
                           rho = 0.8, delta = 0.5)
rss.t.test(data1 = rss.data1, data2 = rss.data2, alpha = 0.05,
           alternative = "two.sided", mu0 = 0, method = "naive")

$RSS_mean
[1] -0.05269583  0.59074771

$CI
[1] -0.9032096 -0.3836775

$t
[1] -5.023613

$df
[1] 36

```

```
$p.value
[1] 1.397776e-05
```

The `rss.ELR.test` function provides a semi-parametric one-sample mean test using an empirical likelihood ratio (ELR) test (Ahn et al., 2024). This method offers a flexible alternative to traditional parametric approaches, particularly when normality assumptions are not appropriate. `rss.ELR.test` uses the empirical likelihood function which is defined as:

$$L(\mu_0) = \sup \left\{ \prod_{h=1}^H \prod_{r=1}^{n_h} p_{h,r} : \sum_{r=1}^{n_h} p_{h,r} = \frac{1}{H} \text{ and } \sum_{r=1}^{n_h} p_{h,r} Y_{[h],r} = \mu_0 \right\},$$

where $p_{h,r}$ is a mass probability on $Y_{[h],r}$. The test statistic is the empirical likelihood ratio $-2 \log LR(\mu_0)$, which asymptotically follows a chi-square distribution with one degree of freedom. The following example demonstrates how to test the hypothesis $H_0 : \mu = 0$ using `rss.ELR.test`:

```
rss.ELR.test(data = rss.data1, alpha = 0.05, mu0 = 0)
```

```
$RSS_mean
[1] -0.05269583
```

```
$CI
[1] -0.3153479 0.2204137
```

```
$-2*log.LR`
[1] 0.1488371
```

```
$p.value
[1] 0.6996491
```

The `rss.sign.test` function performs a nonparametric one-sample sign test, providing median estimation, CIs, and hypothesis testing. For BRSS, the function follows the method of Hettmansperger (1995) whereas it implements the approach of Barabesi (2001) for URSS. Under the null hypothesis $H_0 : M = M_0$, the test statistic for BRSS is:

$$n^{-1/2} \left(S_{\text{RSS}}^+ - \frac{n}{2} \right) \xrightarrow{d} N \left(0, \frac{\eta^2}{4} \right),$$

where $S_{\text{RSS}}^+ = \sum_{i=1}^n \mathbf{I}(y_i - M_0 > 0)$, $\eta^2 = 1 - \frac{4}{H} \sum_{h=1}^H \left\{ B \left(h, H - h + 1, \frac{1}{2} \right) - \frac{1}{2} \right\}^2$, and $B(h, s, q)$ is the cumulative distribution function of the beta distribution with parameters h and s for $0 \leq q \leq 1$. For URSS, the test statistic follows:

$$\left(S_{\text{RSS}}^+ - \sum_{h=1}^H n_h (1 - \beta_h) \right) \xrightarrow{d} N \left(0, \sum_{h=1}^H n_h \beta_h (1 - \beta_h) \right),$$

where $\beta_h = B \left(h, H - h + 1, \frac{1}{2} \right)$. Thus, the function automatically applies the appropriate statistic depending on whether the input design is balanced or unbalanced. The following example demonstrates how to test whether the population median is 0 using `rss.sign.test`:

```
rss.sign.test(data = rss.data1, alpha = 0.05, alternative = "two.sided", median0 = 0)
```

```
$RSS_median
[1] -0.09226903
```

```
$sign
[1] 12
```

```
$CI
[1] -0.3059627 0.3796395

$z
[1] -0.7745967

$p.value
[1] 0.438578
```

The `rss.prop.test` function offers proportion estimation, CIs, and hypothesis testing for binary outcome variables (Y). The unbiased RSS estimator for p is $\hat{p} = \frac{1}{H} \sum_{h=1}^H \frac{1}{n_h} \sum_{r=1}^{n_h} Y_{[h]r}$ and its variance is $\text{Var}(\hat{p}) = \frac{1}{H^2} \sum_{h=1}^H \frac{1}{n_h} \hat{p}(1 - \hat{p})$. Under perfect rankings, we can use a normal approximation for proportions of RSS data (Chen et al., 2006; Ahn et al., 2022b):

$$\hat{p} - p \xrightarrow{d} N \left(0, \frac{1}{H} \sum_{l=H-h+1}^H \binom{H}{l} p^l (1-p)^{H-l} \right).$$

We extend the CI method proposed by Zamanzade and Mahdizadeh (2020) for BRSS to URSS by $\hat{p} \pm z_{1-\frac{\alpha}{2}} \sqrt{\frac{1}{H^2} \sum_{h=1}^H \frac{1}{n_h} \hat{p}_h (1 - \hat{p}_h)}$ where $\hat{p}_h = \sum_{l=H-h+1}^H \binom{H}{l} \hat{p}^l (1 - \hat{p})^{H-l}$. The following example illustrates how to use `rss.prop.test` to test the null hypothesis $H_0 : p = 0.2$ when the true population proportion is $p = 0.6$.

```
rss.prop.data = rss.prop.simulation(H = 3, nsamp = c(12,9,6), p = 0.6)
rss.prop.test(data = rss.prop.data, alpha = 0.05, alternative = "two.sided", p0 = 0.2)

$RSS_prop
[1] 0.4907407

$CI
[1] 0.3367646 0.6447169

$pstat
[1] 3.700841

$p.value
[1] 0.0002148859
```

Additionally, the `rss.AUC.test` function conducts a semi-parametric ELR test for comparing the AUC between two groups based on RSS data (Moon et al., 2022). The AUC can be interpreted as the probability that a randomly chosen observation from one group exceeds a randomly chosen observation from the other group. This test is equivalent to the Mann-Whitney U test, which assesses whether there is a significant difference in the distributions between the two groups. It provides AUC estimation, CIs, and hypothesis testing. For example, the `rss.AUC.test` can be used to test the null hypothesis of no difference in AUC between two groups ($H_0 : AUC = \delta_0 = 0.5$):

```
rss.AUC.test(data1 = rss.data1, data2 = rss.data2, alpha = 0.05, delta0 = 0.5)

$RSS_AUC
[1] 0.7123457

$CI
[1] 0.5586732 0.8156125

$`-2*log.LR`
```

```
[1] 6.63908
```

```
$p.value
```

```
[1] 0.009976544
```

3.3 Comparison with existing packages

To our knowledge, the existing R packages that support RSS include [RSSampling](#), [NSM3](#), [RSStest](#), and [RankedSetSampling](#). While all these packages support RSS from population data using auxiliary variables for imperfect ranking - a standard feature in RSS methodologies - they vary in their scope and additional functionalities, with most focusing primarily on BRSS.

[RSSampling](#) provides sampling functions for both classical RSS and several modified RSS variants, such as Median RSS (MRSS), Percentile RSS (PRSS), Extreme RSS (ERSS), and Double RSS (DRSS). Additionally, it includes statistical inference methods for classical RSS, assuming balanced designs for both sampling and inference. [NSM3](#) includes only the classical RSS procedure as a sampling tool, along with critical value calculations for a nonparametric test. [RSStest](#) primarily focuses on mean testing for RSS and MRSS and generating RSS data under a normal distribution. Lastly, [RankedSetSampling](#) incorporates Judgment Post-Stratified Sampling (JPS) and RSS, offering sampling, mean estimation, and variance calculation, but it is limited to balanced designs.

The [generalRSS](#) package distinguishes itself by fully supporting both BRSS and URSS, offering advanced tools for flexible sampling and efficient sample allocation in URSS. Additionally, it includes parametric and nonparametric inference procedures for population means, medians, proportions, and AUC, along with simulation tools under predefined distributions (Normal, t, and Log-normal), leveraging a linear ranking model to incorporate imperfect ranking. Unlike other packages, it provides methods to calculate efficient sample allocations for URSS, improving estimation efficiency for both continuous and binary data.

For BRSS, the methods implemented in [generalRSS](#) align with the standard approaches available in other packages and are therefore not explicitly compared here. Instead, the focus is on the unique features of [generalRSS](#) for URSS and its broader applicability. Table 3 summarizes the capabilities of [generalRSS](#) compared to other existing packages.

4 Applications

In this section, we consider two inference problems comparing two sampling methods: RSS and SRS. The first problem involves a one-sample mean inference, and the second involves a two-sample AUC inference. The goal is to estimate the population mean and AUC and compare the efficiency of RSS and SRS in terms of the CI length of each estimator.

4.1 One-sample problem

We demonstrate the RSS sampling and inference procedures by estimating the mean body mass index (BMI) and testing the hypothesis $H_0 : \mu = \mu_0$. BMI is a widely used indicator of body fat based on weight and height, commonly employed to assess an individual's health status.

For this analysis, we use the US National Health and Nutrition Examination Survey (NHANES) dataset, available in the R package [NHANES](#) (Pruim, 2015). NHANES provides comprehensive health-related measurements collected from individuals across the United States. In this study, we focus on two key variables in the NHANES dataset: BMI as the outcome of interest (Y) and `weight` as the auxiliary variable (X) in RSS.

The NHANES dataset originally contains BMI records for approximately 10,000 subjects. However, due to duplicated entries, only the first observation per subject is retained, resulting

Feature	R package			
	generalRSS	RSSampling	NSM3	RSStest
Sampling	Continuous • BRSS • URSS Binary • BRSS • URSS	Continuous • BRSS • MRSS • PRSS • ERSS • DRSS etc.	Continuous • BRSS	Continuous • BRSS • MRSS Continuous • BRSS
Simulation Support	Continuous • Normal • t • Log-normal Binary • Binomial	–	–	Continuous • Normal –
Sample Allocation	Continuous • Integer Neyman • Adjusted Neyman • LRC Allocations Binary • Neyman	–	–	–
Parametric Inference	• Mean Tests for BRSS, URSS • Proportion Tests for BRSS, URSS	• Mean Tests for BRSS	–	• Mean Tests for BRSS, MRSS • Mean Tests for BRSS
Nonparametric Inference	• Mean Tests for BRSS, URSS • Median Tests for BRSS, URSS • AUC Tests for BRSS, URSS	• Median Tests for BRSS	• Median Values	–

Table 3: Comparison of existing R packages for RSS.

in a final dataset of 6,779 unique individuals. We treat this pre-processed dataset as the underlying population for our analysis.

We note that NHANES data are used strictly for illustrative purposes to demonstrate the functionality and flexibility of the generalRSS package, rather than to produce population-level estimates. The NHANES dataset is derived from a stratified, multi-stage, unequal probability cluster sampling design. Consequently, the results presented should not be

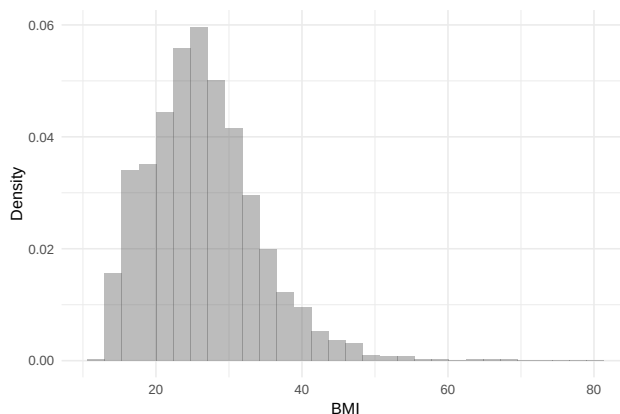


Figure 2: BMI distribution in the NHANES dataset (Pruim, 2015).

interpreted as valid inferences about the U.S. population without applying appropriate survey-weighted methods and design-based variance estimation techniques.

We load the NHANES dataset from the R package `NHANES` as follows:

```
library("NHANES")
```

The duplicated data are pre-processed by running:

```
library(dplyr)
dat = NHANES |> distinct(ID, .keep_all = TRUE) |> select(ID, BMI, Weight)
mu0 = mean(dat$BMI, na.rm=T)
mu0

[1] 26.48768

cor(na.omit(cbind(dat$BMI, dat$Weight)))

      BMI      Weight
BMI    1.0000000 0.9027414
Weight 0.9027414 1.0000000
```

The true mean of BMI is 26.488 and the correlation between the auxiliary variable `Weight` and the outcome variable BMI is 0.903, indicating high-ranking quality. The histogram of the BMI in Figure 2 shows a skewed distribution, suggesting that URSS may be more efficient than SRS.

Using the NHANES data as the underlying population, we start with a BRSS design, using a set size of $H = 3$ and a total sample size of $n = 30$, resulting in a balanced allocation of $n_h = 10$ for each stratum ($h = 1, 2, 3$). However, missing data in the outcome variable can lead to a URSS design. In such cases, we apply `rss.design` to evaluate the efficiency of the initial sample allocation. If the allocation is deemed inefficient, adjusted allocations are calculated, and additional sampling is performed to obtain the final RSS data. Inference methods are then applied to the final RSS data.

In practice, we first generate BRSS data using the auxiliary variable (`Weight`) to rank and select samples without measuring BMI. The outcome variable is then measured for the selected samples as described in Section 2.2.

```
org.nsamp = c(10,10,10)
rss.dat = rss.sampling(ID=dat$ID, X=dat$Weight, H=3, nsamp=org.nsamp)
ind<- match(rss.dat$ID,dat$ID)
rss.dat$y = dat$BMI[ind]
head(rss.dat)
```

	rank	ID	y
1	1	64759	24.80
2	1	58179	16.93
3	1	54775	18.84
4	1	63972	19.90
5	1	52695	28.04
6	1	64076	NA

Here, we observe missing values in the outcome y , resulting in a URSS sample allocation ($n_1 = 9, n_2 = 10, n_3 = 10$) and total sample size $n = 29$ as shown below:

```
rss.dat = na.omit(rss.dat)
org.nsamp = table(rss.dat$rank)
print(org.nsamp)
```

```
1  2  3
9 10 10
```

Using this original URSS data, we test the null hypothesis $H_0 : \mu = \mu_0$ using a t-test with `rss.t.test` and the `sample` method.

```
org.t = rss.t.test(data1 = rss.dat, mu0 = mu0, method = "sample")
print(org.t)
```

```
$RSS_mean
[1] 27.02993
```

```
$CI
[1] 24.90905 29.15080
```

```
$t
[1] 0.5323651
```

```
$df
[1] 20.57505
```

```
$p.value
[1] 0.6001757
```

```
diff(org.t$CI)
[1] 4.241756
```

In Figure 2, the outcome variable shows a skewed distribution and URSS may offer improved efficiency, but the current sample allocation `org.nsamp` ($n_1 = 9, n_2 = 10, n_3 = 10$) is not guaranteed to be optimal. To address this, we use `rss.design` to evaluate sample efficiency and adjust allocations accordingly:

```
alloc = RSS.design(data = rss.dat[,c("rank", "y")])
add.samp = alloc$Adj.Neyman - org.nsamp
add.samp
```

```
1 2 3
0 0 3
```

We adopt the adjusted Neyman allocation `Adj.Neyman` and identify the need for three additional samples in the third stratum. From the remaining population, except for the pre-selected samples, we sample again using the auxiliary variable without measuring the outcome. Then, we measure the outcome for the three newly selected samples:

```
add.dat = rss.sampling(ID=dat$ID, X=dat$Weight, H=3, nsamp=add.samp)
add.ind = match(add.dat$ID, dat$ID)
add.dat$y = dat$BMI[add.ind]
```

By merging the additional data with the original data, we obtain efficient URSS data.

```
update.dat = rbind(rss.dat, add.dat)
table(update.dat$rank)
```

```
1  2  3
9 10 13
```

Using this updated RSS data, we reapply the t-test with `rss.t.test`:

```
update.t = rss.t.test(data1 = update.dat, mu0 = mu0, method = "sample")
print(update.t)
```

```
$RSS_mean
[1] 26.59152
```

```
$CI
[1] 24.59261 28.59042
```

```
$t
[1] 0.1067613
```

```
$df
[1] 26.07755
```

```
$p.value
[1] 0.9157954
```

```
diff(update.t$CI)

[1] 3.997811
```

When we compare the results before and after adding extra samples, we observe that the CI length decreases from 4.242 to 3.998 for RSS data.

We repeated the following procedure for 500 replicates to compare the performance of the original URSS, updated RSS, and SRS. For a fair comparison, SRS was assigned the same total sample size as the updated RSS. To generate the original URSS in each replicate, we introduced 10% missing values into a BRSS dataset with $n_h = m = 10$, resulting in a URSS design. For the updated allocations, we selected the sample allocation provided by the `rss.design` function that required the smallest number of additional samples. Table 4 presents the average sample size, coverage probability, and 95% CI over 500 replicates. Compared to the original URSS, the updated RSS achieves a significant reduction in CI length while maintaining a coverage probability close to the nominal 95% level. Additionally, compared to SRS, the updated RSS demonstrates superior performance, achieving both a higher coverage probability and a shorter CI length. These findings underscore the efficiency of updated RSS in handling missing data and enhancing estimation precision.

4.2 Two-sample problem

We demonstrate the RSS sampling and inference procedures by comparing fasting plasma glucose (FPG) levels between two groups with and without diabetes through the estimation of the AUC (δ). The FPG test determines whether blood sugar levels stay elevated after

Sampling	Sample size	Coverage Probability	CI Length
Original URSS	25.72	0.948	4.981
Updated RSS	27.15	0.950	4.686
SRS	27.15	0.942	5.924

Table 4: Comparison of mean inference results from RSS and SRS over 500 replicates.

an extended fasting period, signaling potential issues with sugar metabolism ([American Diabetes Association, 2022](#)). However, the FPG test may sometimes be inconvenient because it requires a fasting period of at least 8 hours. The HbA1c test, also called glycohemoglobin or hemoglobin A1c, is another valuable tool for diagnosing diabetes. It provides an average of blood sugar levels over the past 2-3 months and does not require any prior preparation, unlike the FPG test ([American Diabetes Association, 2022](#)). In this application, we set the FPG as the outcome variable (Y) and glycohemoglobin as the auxiliary variable (X).

AUC represents the probability that a randomly selected individual from the diabetes group has a higher FPG than a randomly selected individual from the non-diabetes group. For example, under $H_0 : \delta = 0.5$, the two distributions are identical, indicating no discriminatory ability between the groups based on FPG.

We analyze the NHANES dataset collected between 2021 and 2023, focusing on 3,540 subjects with data on FPG (LBXGLU), glycohemoglobin (LBXGH), and diabetes-related information (DIQ010) ([Centers for Disease Control and Prevention, 2023](#)). These variables are provided in the supplementary file ‘diabetesAUC.RData’. We first separate the data into two groups by diabetes status.

```
load("diabetesAUC.RData") # load data
grp1 = data |> filter(DIQ010 == 0) # non-diabetic
grp2 = data |> filter(DIQ010 == 1) # diabetic
mean(grp1$LBXGLU)

[1] 100.4777

mean(grp2$LBXGLU)

[1] 156.5644

cor(data$LBXGH, data$LBXGLU)

[1] 0.813826
```

The true mean FPG levels are 100.478 for non-diabetes and 156.564 for diabetes populations, respectively. The histograms of FPG by diabetes status in Figure 3 highlight that the FPG distribution in the diabetes group is shifted towards higher values compared to the non-diabetes group. Also, the correlation between the outcome and auxiliary variables (i.e., FPG and glycohemoglobin) is 0.814, confirming strong ranking quality.

Using R package **pROC**, we calculate the true AUC:

```
library(pROC)
delta0 = roc(data$DIQ010, data$LBXGLU, direction = c("<"), levels = c(0,1))$auc
delta0

Area under the curve: 0.8861
```

In this section, we assume that the FPG level (Y) of the population is given, compared to Section 2.4.1. We use a set size of $H = 3$ and a total sample size of $n = 30$, a balanced allocation of $n_h = 10$ for each stratum ($h = 1, 2, 3$). Samples are ranked by the glycohemoglobin (X) and measure the outcome (Y) simultaneously for each group with a BRSS design:

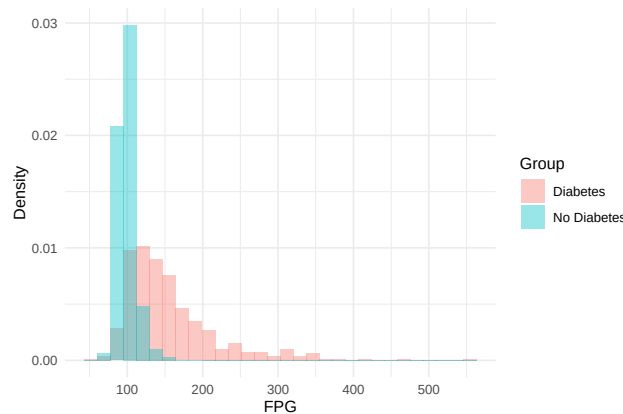


Figure 3: FPG distributions by diabetes status in the NHANES dataset (Centers for Disease Control and Prevention, 2023).

```
org.nsamp1 = c(10,10,10)
org.nsamp2 = c(10,10,10)
H = length(org.nsamp1)
brss.grp1 = rss.sampling(ID = grp1$SEQN, X = grp1$LBXGH, Y = grp1$LBXGLU, H = H,
                        nsamp=org.nsamp1)
brss.grp2 = rss.sampling(ID = grp2$SEQN, X = grp2$LBXGH, Y = grp2$LBXGLU, H = H,
                        nsamp = org.nsamp2)
```

We then apply `rss.AUC.test` to these two RSS datasets to test the null hypothesis $H_0 : AUC = 0.8861$.

```
brss.auc = rss.AUC.test(data1 = brss.grp1, data2 = brss.grp2, alpha = 0.05,
                        delta0 = delta0)
```

```
brss.auc
```

```
$RSS_AUC
```

```
[1] 0.8477778
```

```
$CI
```

```
[1] 0.7046560 0.9178329
```

```
$`-2*log.LR`
```

```
[1] 0.818453
```

```
$p.value
```

```
[1] 0.3656329
```

`rss.AUC.test` returns the RSS AUC point estimate, CI, the ELR test statistics, and the p value for the test.

We can improve the efficiency of RSS sampling using URSS design compared to BRSS. If AUC is expected to be high, the smallest rank group in the non-diabetic group and the largest rank group in the diabetic group provide limited information in AUC inference because the FPG levels are expected to be too small and too large, respectively. Therefore, a more precise inference is possible if we could have a larger number of samples from the largest rank group in the non-diabetic group and a larger number of samples from the smallest rank group in the diabetes group. To illustrate this, we set $(n_1 = 5, n_2 = 10, n_3 = 15)$ for the non-diabetic group and $(n_1 = 15, n_2 = 10, n_3 = 5)$ for the diabetic group. The URSS sampling and AUC inference can be done:

```
org.nsamp1 = c(5,10,15)
```

```

org.nsamp2 = c(15,10,5)
H = length(org.nsamp1)
urss.grp1 = rss.sampling(ID = grp1$SEQN, X = grp1$LBXGH, Y = grp1$LBXGLU, H = H,
                        nsamp=org.nsamp1)
urss.grp2 = rss.sampling(ID = grp2$SEQN, X = grp2$LBXGH, Y = grp2$LBXGLU, H = H,
                        nsamp=org.nsamp2)
urss.auc = rss.AUC.test(data1 = urss.grp1, data2 = urss.grp2, alpha = 0.05,
                        delta0 = delta0)

urss.auc

$RSS_AUC
[1] 0.8977778

$CI
[1] 0.7933407 0.9450542

$`-2*log.LR`
[1] 0.09911326

$p.value
[1] 0.7528964

```

Using URSS, we estimate AUC as 0.898 with a CI length of 0.152, which is shorter compared to the CI length of 0.213 from BRSS data.

```

diff(brss.auc$CI)

[1] 0.2131769

diff(urss.auc$CI)

[1] 0.1517134

```

We repeat the following procedure for 500 different samples and compare URSS and BRSS. We also compare the efficiency with SRS data. We sample two SRS data with the same total sample size of $n = 30$ from the non-diabetes and the diabetes group, respectively, and estimate the AUC and its CI, followed by [Moon et al. \(2022\)](#). Table 5 shows the average coverage probability and the length of 95% CI. URSS yields the highest coverage probability with CI length comparable to BRSS and shorter than SRS.

Sampling	Coverage Probability	CI Length
URSS	0.944	0.175
BRSS	0.926	0.175
SRS	0.916	0.181

Table 5: Comparison of AUC inference results from RSS and SRS over 500 replicates.

5 Conclusion

The [generalRSS](#) package provides a comprehensive framework for ranked set sampling (RSS), addressing challenges such as missing data, skewed population distributions, and binary data. Unlike traditional tools, it supports both balanced (BRSS) and unbalanced (URSS) designs, offering functions for flexible sampling, efficient sample allocation, and statistical inference for means, medians, proportions, and AUCs.

This paper compares **generalRSS** with existing RSS packages (Table 3) and demonstrates its application through real-world medical case studies. In the one-sample problem, an initial BRSS design became unbalanced due to missing data. By applying the `rss.design` function, we optimized the URSS allocation, reducing CI lengths while maintaining coverage probability. In the two-sample problem, URSS was applied to AUC inference by designing the sample allocations based on the distributions of the two groups. Simulations confirmed that URSS improved estimation efficiency over both SRS and BRSS.

These findings highlight the advantages of unbalanced designs in RSS-based studies, particularly when missing data or skewed distributions limit traditional methods. The case studies illustrate the effectiveness of **generalRSS** in applied research and its broader applicability to inference problems beyond medical studies. Future work will focus on expanding inferential methods and refining sampling strategies to further enhance **generalRSS** as a tool for RSS-based analysis.

Computational details

The results in this paper were obtained using R 4.3.1 with the following packages: **generalRSS** 0.2.0, **MASS** 7.3-60, **dplyr** 1.1.4, **ggplot2** 3.5.1, **pROC** 1.18.5, **tidyverse** 2.0.0, **haven** 2.5.4, **rootSolve** 1.8.2.4, **emplik** 1.3-1, and **NHANES** 2.1.0.

All computations were carried out on a 64-bit Windows 11 Pro system (version 24H2) with a 13th Gen Intel(R) Core(TM) i7-13700 2.10GHz processor and 32 GB of RAM.

Acknowledgments

This research was supported by the Learning & Academic research institution for Master's · PhD students, and Postdocs (LAMP) Program of the National Research Foundation of Korea (NRF) grant funded by the Ministry of Education (No. RS-2023-00285390) and Basic Science Research Program through the National Research Foundation of Korea (NRF) grant funded by the Ministry of Education (NRF-2021R1A6A1A10044950).

References

- S. Ahn and C. Moon. *generalRSS: Statistical Tools for Balanced and Unbalanced Ranked Set Sampling*, 2025. URL <https://CRAN.R-project.org/package=generalRSS>. R package version 0.1.3. [p278]
- S. Ahn, J. Lim, and X. Wang. The student's t approximation to distributions of pivotal statistics from ranked set samples. *Journal of the Korean Statistical Society*, 43:643–652, 2014. [p285]
- S. Ahn, X. Wang, and J. Lim. On unbalanced group sizes in cluster randomized designs using balanced ranked set sampling. *Statistics & Probability Letters*, 123:210–217, 2017. [p277]
- S. Ahn, X. Wang, and J. Lim. Efficient sample allocation by local adjustment for unbalanced ranked set sampling. In *Recent Advances on Sampling Methods and Educational Statistics: In Honor of S. Lynne Stokes*, pages 45–60. Springer, 2022a. [p277, 283]
- S. Ahn, X. Wang, M. Wang, and J. Lim. On continuity correction for RSS-structured cluster randomized designs with binary outcomes. *Metron*, 80(3):383–397, 2022b. [p287]
- S. Ahn, X. Wang, C. Moon, and J. Lim. New scheme of empirical likelihood method for ranked set sampling: Applications to two one sample problems. *International Statistical Review*, 2024. doi: 10.1111/insr.12589. [p277, 285, 286]

- American Diabetes Association. Standards of medical care in diabetes—2022 abridged for primary care providers. *Clinical Diabetes*, 40(1):10–38, 2022. [p293]
- L. Barabesi. The unbalanced ranked-set sample sign test. *Journal of Nonparametric Statistics*, 13(2):279–289, 2001. [p286]
- D. Bhoj and G. Chandra. Simple unequal allocation procedure for ranked set sampling with skew distributions. *Journal of Modern Applied Statistical Methods*, 18(2):22, 2020. [p277]
- C. Bocci, A. Petrucci, and E. Rocco. Ranked set sampling allocation models for multiple skewed variables: An application to agricultural data. *Environmental and Ecological Statistics*, 17(3):333–345, 2010. [p277]
- Centers for Disease Control and Prevention. National health and nutrition examination survey data. *Hyattsville, MD: US Department of Health and Human Services, Centers for Disease Control and Prevention*, 2023. URL <https://wwwn.cdc.gov/nchs/nhanes/continuousnhanes/default.aspx?Cycle=2021-2023>. [p278, 293, 294]
- H. Chen, E. A. Stasny, and D. A. Wolfe. Unbalanced ranked set sampling for estimating a population proportion. *Biometrics*, 62(1):150–158, 2006. [p284, 287]
- Z. Chen and Z. Bai. The optimal ranked-set sampling scheme for parametric families. *Sankhya: The Indian Journal of Statistics, Series A*, 62(2):178–192, 2000. [p277]
- Z. Chen, Z. Bai, and B. K. Sinha. *Ranked Set Sampling: Theory and Applications*. Springer, 2004. [p277, 279, 285]
- H. David and D. Levine. Ranked set sampling in the presence of judgment error. *Biometrics*, 28:553–555, 1972. [p277]
- T. Dell and J. Clutter. Ranked set sampling theory with order statistics background. *Biometrics*, 28(2):545–555, 1972. [p277]
- F. Gökpınar, M. Ebegil, Y. A. Özdemir, and E. Gökpınar. **RSStest: Testing the Equality of Two Means Using RSS and MRSS**, 2023. URL <https://CRAN.R-project.org/package=RSStest>. R package version 1.0. [p277]
- T. P. Hettmansperger. The ranked-set sample sign test. *Journal of Nonparametric Statistics*, 4(3):263–270, 1995. [p286]
- G. McIntyre. A method for unbiased selective sampling, using ranked sets. *Australian journal of agricultural research*, 3(4):385–390, 1952. [p277]
- C. Moon, X. Wang, and J. Lim. Empirical likelihood inference for area under the receiver operating characteristic curve using ranked set samples. *Pharmaceutical Statistics*, 21(6):1219–1245, 2022. [p287, 295]
- O. Ozturk and D. Wolfe. Optimal allocation procedure in ranked set sampling for unimodal and multi-modal distributions. *Environmental and Ecological Statistics*, 7:343–356, 2000. [p277]
- O. Ozturk, S. Rogers, O. Kravchuk, and P. Kasprzak. **RankedSetSampling: Easing the Application of Ranked Set Sampling in Practice**, 2021. URL <https://biometryhub.github.io/RankedSetSampling/>. R package version 0.1.0. [p277]
- R. Pruim. **NHANES: Data from the US National Health and Nutrition Examination Study**, 2015. URL <https://CRAN.R-project.org/package=NHANES>. R package version 2.1.0. [p278, 288, 290]
- G. Schneider, E. Chicken, and R. Becvarik. **NSM3: Functions and Datasets to Accompany Hollander, Wolfe, and Chicken - Nonparametric Statistical Methods, Third Edition**, 2024. URL <https://CRAN.R-project.org/package=NSM3>. R package version 1.19. [p277]

- B. Sevinc, B. Cetintav, M. Esemeyen, and S. Gurler. **RSSampling**: *Ranked Set Sampling*, 2018. URL <https://CRAN.R-project.org/package=RSSampling>. R package version 1.0. [p277]
- K. Soetaert and M. K. Soetaert. **rootSolve**: *Nonlinear Root Finding, Equilibrium and Steady-State Analysis of Ordinary Differential Equations*, 2023. URL <https://CRAN.R-project.org/package=rootSolve>. R package version 1.8.2.4. [p281]
- S. L. Stokes. Inferences on the correlation coefficient in bivariate normal populations from ranked set samples. *Journal of the American Statistical Association*, 75(372):989–995, 1980. [p277]
- S. L. Stokes and T. W. Sager. Characterization of a ranked-set sample with application to estimating distribution functions. *Journal of the American Statistical Association*, 83(402):374–381, 1988. [p277]
- K. Takahashi and K. Wakimoto. On the unbiased estimates of the population mean based on the sample stratified by means of ordering. *Annals of Institute of Statistical Mathematics*, 20(1):1–31, 1968. [p277]
- X. Wang, S. Ahn, and J. Lim. Unbalanced ranked set sampling in cluster randomized studies. *Journal of Statistical Planning and Inference*, 187:1–16, 2017. [p277]
- Y. Wang, Z. Chen, and J. Liu. General ranked set sampling with cost considerations. *Biometrics*, 60(2):556–561, 2004. [p277]
- D. A. Wolfe. Ranked set sampling: its relevance and impact on statistical inference. *International scholarly research notices*, 2012(1):568385, 2012. [p277]
- T. Wright. The equivalence of neyman optimum allocation for sampling and equal proportions for apportioning the US house of representatives. *The American Statistician*, 66(4):217–224, 2012. [p283]
- E. Zamanzade and M. Mahdizadeh. Using ranked set sampling with extreme ranks in estimating the population proportion. *Statistical methods in medical research*, 29(1):165–177, 2020. [p287]
- M. Zhou, Y. Yang, and A. Owen. **emplik**: *Empirical Likelihood Ratio for Censored/Truncated Data*, 2024. URL <https://CRAN.R-project.org/package=emplik>. R package version 1.3-2. [p281]

Chul Moon

Department of Statistics and Data Science
Southern Methodist University
Dallas, Texas, USA
(0000-0002-2892-5449)
chulm@smu.edu

Soohyun Ahn

Department of Mathematics
Ajou University
Suwon, Republic of Korea
(0000-0001-5016-5469)
Corresponding author: shahn@ajou.ac.kr

GeoThinneR: An R Package for Efficient Spatial Thinning of Species Occurrences and Point Data

by Jorge Mestre-Tomás

Abstract In this paper, we present GeoThinneR, an R package for efficient and flexible spatial thinning of species occurrence data. Spatial thinning is a widely used preprocessing step in species distribution modelling (SDM) that can help reduce sampling bias, but existing R implementations rely on brute-force algorithms that scale poorly with large datasets. GeoThinneR implements multiple thinning approaches, including ensuring a minimum distance between points, subsampling points on a grid, and filtering based on decimal precision. To handle large datasets, it introduces two optimized algorithms based on local *kd*-trees and adaptive neighbour estimation, which greatly reduce memory usage and execution time. Additional functionalities such as group-wise thinning and point prioritization are included to facilitate its use in SDM workflows. Here, we provide performance benchmarks using both simulated and real-world data to demonstrate substantial performance improvements over existing tools.

1 Introduction

Presence-only data are often the main source of information for modelling species distributions, as systematic survey data are rarely available (Pearce and Boyce, 2006). In recent years the availability of spatial biodiversity data has increased, with repositories such as the Global Biodiversity Information Facility (GBIF, <https://www.gbif.org>) and the Ocean Biodiversity Information System (OBIS, <https://obis.org/>) providing datasets containing hundreds of thousands of species occurrence records. These larger datasets provide new opportunities in ecological research and species distribution modelling (SDM), which are used to model the species geographic distributions based on occurrence records and environmental variables (Pulliam, 2000; Elith and Leathwick, 2009; Miller, 2010; Guisan et al., 2013). However, as dataset sizes grow, the computational requirements for processing and filtering them increase too. Species occurrence data often show spatial sampling bias due to opportunistic observations, uneven sampling effort or a combination of information from diverse sources that vary in how and where data were collected, resulting in heterogeneous datasets (Boakes et al., 2010; Lobo and Tognelli, 2011; Beck et al., 2014; Meyer et al., 2016; Hughes et al., 2021). These biases result in clusters of records that can artificially increase spatial autocorrelation among occurrences and cause the model to overfit the environmental conditions in these oversampled regions, leading to models that reflect patterns in sampling effort rather than true species distributions (Wisz et al., 2008; Phillips et al., 2009; Higa et al., 2015; Cosentino and Maiorano, 2021; Steen et al., 2021).

Several approaches exist to address sampling bias (Inman et al., 2021; Baker et al., 2024; Moudry et al., 2024), including adjusting background points (Barber et al., 2022; Vollering et al., 2019), applying environmental filtering (Castellanos et al., 2019), or incorporating bias-related covariates (Varela et al., 2014; Chauvier et al., 2021). Some modelling frameworks attempt to account for sampling bias directly, for example by modelling preferential sampling through point processes (Diggle et al., 2010; Pennino et al., 2019; Amaral et al., 2024) or by combining opportunistically collected presence-only data with systematically sampled data within integrated SDMs (Koshkina et al., 2017; Simmonds et al., 2020; Mäkinen et al., 2024). Here we focus on one of the most widely used preprocessing methods to mitigate these biases across the geographic space, known as spatial thinning, which selectively filters points based on specified criteria to reduce the overrepresentation of the densest locations (Veloz, 2009; Boria et al., 2014). The two most common thinning methods in SDM are distance-based thinning, where points are removed if they fall within a specified minimum distance of another point (Aiello-Lammens et al., 2015), and grid-based thinning, which retains only a specified number of points per grid cell (Hijmans et al., 2023).

While existing distance-based thinning tools in R, such as **spThin** (Aiello-Lammens et al., 2015) and **enmSdmX** (Smith et al., 2023), are widely used, they have several limitations that become more challenging as dataset sizes continue to increase. These tools rely on brute-force algorithms for nearest-neighbour searches given a distance threshold, which are unfeasible to run on personal computers as they scale poorly for datasets containing hundreds of thousands of points. On the other hand, grid-based thinning, such as the one implemented in the **dismo** package (Hijmans et al., 2023), offers better computational efficiency but lacks precision when requiring a specified distance threshold. Additionally, these tools have limited functionality and usability as they often focus on a

single thinning approach and offer few customization options for specific SDM applications, such as thinning occurrences across multiple species (Noori et al., 2024). This can be useful when modelling several species or community-level patterns, for example in stacked SDMs (SSDMs) where species are modelled independently and then combined (Calabrese et al., 2014), or in joint species distribution models (JSDMs) where multiple species are analysed simultaneously accounting for the multivariate nature of the data (Pollock et al., 2014; Ovaskainen et al., 2017; Ovaskainen and Abrego, 2020). Other useful features include retaining an exact number of occurrences (Steen et al., 2021) and prioritizing points based on data quality or other features (Yu et al., 2023).

To overcome these challenges, we introduce **GeoThinner** (Mestre-Tomás, 2025), an R package designed to provide efficient and flexible spatial thinning for large-scale datasets. **GeoThinner** integrates multiple thinning approaches into a single package simplifying user experience (distance, grid, and decimal precision-based thinning), offers an optimized distance-based thinning using enhanced algorithms based on *kd*-tree structures, which improve nearest-neighbour search efficiency (Friedman et al., 1975), together with additional specific functionalities for SDM workflows including thinning by species or groups, the ability to retain an exact number of occurrences, and prioritization of points based on user-defined variables.

In Section 2, we present the methods and features implemented in **GeoThinner**. Section 3 describes the optimized algorithms for large datasets together with a performance comparison between the distance-based methods available. Section 4 benchmarks the performance of the package with existing software, both with simulated and real-world datasets. Finally, Section 5 presents a short conclusion of the work and discusses future directions.

2 GeoThinner package description

Given a set of points $S = \{x_1, x_2, \dots, x_n\}$, spatial thinning aims to filter S to obtain a subset $S' \subseteq S$ that meets a specified selection criteria. Different thinning methods can be employed to apply this criteria, such as enforcing a minimum separation distance between retained points, limiting the number of points per grid cell, or rounding coordinates to a specified precision. The thinning process iteratively evaluates each point, referred to as the query point q , identifies neighbour points within S , and applies the selection criteria to return a thinned subset S' .

The **GeoThinner** package is implemented in R and can be downloaded from the Comprehensive R Archive Network (CRAN) at <https://cran.r-project.org/package=GeoThinner>. Comprehensive documentation and usage examples are available on GitHub and CRAN. One of the advantages of **GeoThinner** is its flexibility and ease of use, allowing users to efficiently apply different spatial thinning configurations to their datasets. In this section, we demonstrate how to use the package and explore its various thinning methods and additional functionalities.

The main function to apply spatial thinning is `thin_points()`, which takes a data frame containing a set of points with longitude and latitude columns. The key arguments are the thinning method ("distance", "grid", or "precision"), the thinning criterion (`thin_dist`, resolution, or precision), and the number of independent thinning attempts (`trials`). By default, only the subset retaining the largest number of points is returned; to keep all thinning trials, the argument `all_trials = TRUE` must be specified. To illustrate the basic usage, we simulate a toy dataset of 100 uniformly distributed points for two species across an area of 1 × 1 degree and apply spatial thinning using the `thin_points()` function:

```
library(GeoThinner)
# Simulate data
set.seed(2547)
sim_data <- data.frame(
  lon = runif(100, 0, 1),
  lat = runif(100, 0, 1),
  group = sample(c("species_1", "species_2"), 100, replace = TRUE)
)

thin_results <- thin_points(sim_data,
  method = "distance", thin_dist = 10,
  trials = 10, all_trials = TRUE, seed = 567
)
```

GeoThinner returns the results structured as a `GeoThinned` object, which contains: a list of logical vectors indicating which points were retained in each trial (`x$retained`), the method used for thinning (`x$method`), the parameters used for thinning (`x$params`), and the original dataset (`x$original_data`).

The `summary()` function provides information about the thinning process, including the number of points retained after thinning, the nearest neighbour distance (NND), and the spatial coverage of the retained points. By default, the summary shows the metrics for the trial that retains the largest number of points, but users can also specify a specific trial using the `trial` argument.

```
summary(thin_results)

#> Summary of GeoThinner Results
#> -----
#> Trial summarized   : 2
#> Method used       : distance
#> Distance metric    : haversine
#>
#> Number of points:
#>   Original          100
#>   Thinned           46
#>   Retention ratio    0.460
#>
#> Nearest Neighbor Distances [km]
#>      Min 1st Qu. Median   Mean 3rd Qu.  Max
#> Original  1.054   3.699  5.349  5.917   7.474 16.4
#> Thinned  10.254  10.899 11.700 11.968  12.646 16.4
#>
#> Spatial Coverage (Convex Hull Area) [km2]
#>   Original      10453.599
#>   Thinned       10111.415
```

The `summary()` output shows that 46 of the 100 points were retained (a retention ratio of 0.46), increasing the minimum nearest-neighbour distance from 1.05 km in the original data to 10.25 km after thinning. Spatial coverage changes only slightly (from 10,453.59 km² to 10,111.42 km²), indicating that the overall extent of the points is mostly maintained. Using the `plot()` function, we can visualize the original and thinned datasets.

```
# Visualize largest thinned trial
plot(thin_results)
# Show retained points only
plot(thin_results, show_original = FALSE)
# Change the colors
plot(thin_results, col_original = "red", col_thinned = "blue")
```

For grid-based and precision-based thinning, retaining the maximum number of points that meet the thinning constraint is simple and straightforward. In contrast, distance-based thinning is a non-deterministic problem related to the set packing problem in combinatorial mathematics. Therefore, as in **spThin**, **GeoThinner** allows users to perform multiple independent thinning trials, which may produce different subsets of retained points. Using the `largest()` and `get_trial()` functions, users can extract the largest thinned dataset or any of the trials, respectively.

```
# Extract the largest thinned dataset
largest(thin_results)
# Extract the index of the largest dataset
largest_index(thin_results)
# Extract the second trial
get_trial(thin_results, 2)
```

2.1 Overview of thinning methods

GeoThinner integrates three distinct methods to provide flexibility: distance-based, grid-based, and precision-based thinning (Figure 1). For better usability, all methods are implemented within a modular wrapper function, `thin_points()`, which includes the thinning methods with optimized algorithms for large datasets.

Distance-based thinning (`method="distance"`) ensures that all retained points are separated by at least a user-defined minimum distance d (Aiello-Lammens et al., 2015). However, efficiently identifying neighbour points becomes computationally challenging as dataset sizes increase. For a

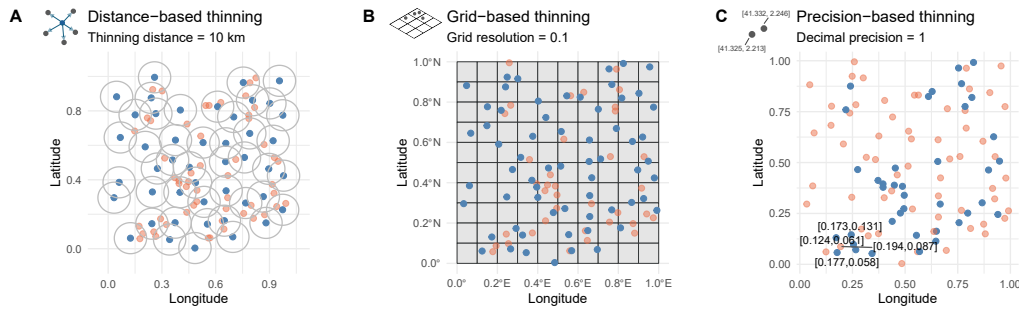


Figure 1: Spatial thinning examples using the three methods implemented in *GeoThinner*: (A) distance-based thinning with a thinning distance of 10 km, (B) grid-based thinning with a grid resolution of 0.1 degrees and retaining one point per grid cell, and (C) precision-based thinning with coordinates rounded to one decimal place. Blue points indicate retained points, and red points indicate removed points.

given set of points $\mathbf{S}$, the algorithm iteratively removes points with neighbours within d based on either Haversine (geographic coordinates) or Euclidean (Cartesian coordinates) distance. By default, *GeoThinner* uses the Haversine formula (Chopde and Nichat, 2013) to compute great-circle distances:

$$D(\mathbf{x}_i, \mathbf{x}_j) = 2R \cdot \arcsin \left(\sqrt{\sin^2 \left(\frac{\text{lat}_j - \text{lat}_i}{2} \right) + \cos(\text{lat}_j) \cdot \cos(\text{lat}_i) \cdot \sin^2 \left(\frac{\text{lon}_j - \text{lon}_i}{2} \right)} \right), \quad (1)$$

where R is the radius of the Earth (by default, 6371 km), and (lon, lat) are the longitude and latitude coordinates of $\mathbf{x}_i$ and $\mathbf{x}_j$. The Haversine formula assumes a smooth spherical Earth, providing sufficient accuracy for most spatial analyses (Maria et al., 2020). For example, to obtain points separated at least by 10 km we could use a similar code as before:

```
# Distance-based thinning
dist_thin <- thin_points(sim_data, method = "distance", thin_dist = 10, seed = 8237)
```

Grid-based thinning (method="grid") is an alternative that stratifies points $\mathbf{S}$ into equal-sized spatial grid cells $\mathcal{G} = \{G_1, G_2, \dots, G_m\}$, selecting a subset of points $\mathbf{S}'_{G_j} \subseteq \{\mathbf{x}_i \in \mathbf{S} : \mathbf{x}_i \in G_j\}$ from each cell $G_j \in \mathcal{G}$ (Hijmans et al., 2023). This method is computationally more efficient than distance-based thinning as it does not need to compute neighbour distances, making it suitable for large datasets where strict separation distances are unnecessary.

We can provide the grid size in kilometers (thin_dist), the resolution of the grid (resolution), or a raster object to use as a grid template (raster_obj).

```
# Grid-based thinning using resolution
grid_thin <- thin_points(sim_data, method = "grid", resolution = 0.1, seed = 9137)
```

```
# Using raster layer
rast_obj <- terra::rast(xmin = 0, xmax = 1, ymin = 0, ymax = 1, res = 0.1)
grid_thin <- thin_points(sim_data, method = "grid", raster_obj = rast_obj, seed = 54898)
```

Precision-based thinning (method="precision") removes duplicate points by rounding coordinates to a specified decimal precision, which is a very common procedure when dealing with georeferenced data in SDM (Kindt et al., 2023). This method is particularly useful when dealing with heterogeneous datasets with varying levels of spatial precision or when reducing density without enforcing a strict minimum distance is sufficient. Since it avoids distance calculations entirely, precision-based thinning is particularly fast and scales efficiently with dataset size.

```
# Precision-based thinning
prec_thin <- thin_points(sim_data, method = "precision", precision = 1, seed = 5674)
```

The optimized algorithms described in the next section focus on distance-based thinning (method = "distance"), since the other methods do not rely on neighbour searches and have lower computational cost.

2.2 Optimization for large datasets

As the size of spatial datasets increases, finding exact nearest neighbours by scanning the complete set S and computing all pairwise distances for each query point q becomes computationally unfeasible. This makes nearest neighbour searches one of the primary bottlenecks in spatial thinning methods with exact distance thresholds. To address this, **GeoThinnerR** implements four different search strategies:

- `search_type="brute"` is a greedy algorithm similar to the one implemented in **spThin** and **enmSdmX**, which calculates all pairwise distances between points using the **fields** package (Douglas Nychka et al., 2021). While being the most straightforward approach, it scales poorly with large datasets due to its $O(n^2)$ complexity.
- `search_type="kd_tree"` uses *kd*-tree structures from the **nabor** package (Elseberg et al., 2012), which were proposed to theoretically reduce the time complexity of nearest-neighbour searches to $O(\log n)$ (Friedman et al., 1975). Although *kd*-trees can significantly improve performance for large datasets, they may suffer from the “curse of dimensionality” or in cases where exhaustive searches are required to identify exact nearest neighbours, their performance can be equivalent to or worse than brute-force algorithms (Ram and Sinha, 2019).
- `search_type="local_kd_tree"` subdivides the spatial domain into multiple subregions, each with an independent and smaller *kd*-tree. This avoids creating a single large *kd*-tree where lots of queries have to be evaluated and reduces both run time and memory usage for large datasets by building smaller and more manageable search trees.
- `search_type="k_estimation"` is an alternative approach to improve *kd*-tree performance and scalability. When searching for nearest-neighbours within a given distance, the exact number of neighbours k per query point q is unknown, so the algorithm evaluates whether all points in the tree lie within the search radius bound from q . To reduce the number of searches performed in the tree for each query point q , we propose a method to estimate the maximum possible number of neighbours $k_{\max}$. This estimation reduces the number of searches per point from n to $k_{\max}$, reducing unnecessary computations of distant points while still finding the exact number of neighbours.

These optimizations significantly improve computational efficiency, making **GeoThinnerR** suitable for large-scale SDM applications. The underlying algorithms and their performance trade-offs are further detailed in Section 3.

2.3 Additional functionalities

In addition to its core thinning methods, **GeoThinnerR** includes several functionalities designed for SDM workflows that users may require when running their analysis. Firstly, **GeoThinnerR** allows performing group-specific thinning within a single dataset. This is useful when working with datasets containing multiple groups, such as different species or presence/absence data, where thinning needs to be applied independently within each group (Figure 2a).

```
# Thinning by group
precision_thin_group <- thin_points(sim_data,
  method = "precision", precision = 1,
  group_col = "group", seed = 2948
)
```

Secondly, in some cases, users may need to retain a fixed number of points while maintaining spatial representativeness, such as for class balancing in presence/absence datasets. **GeoThinnerR** allows users to specify a target number of retained points and attempts to retain the specified number of points while guaranteeing a minimum distance threshold between them (Figure 2b). The process for retaining points selects the most spatially distant points to maximize geographical representativeness. This approach is supported only via the brute-force method as it requires all pairwise distance computations to select the most distant points.

```
# Thinning target points
target_thin <- thin_points(sim_data, target_points = 20, thin_dist = 10, seed = 675)
```

Finally, when applying the thinning algorithm and deciding which points to remove and retain, users can specify variables that influence which points are preferentially retained, making it possible to prioritize records based on ecological relevance, data quality, or uncertainty (Figure 2c). The thinning algorithm follows the standard procedure, but when multiple candidate points violate the thinning constraint, such as falling within the same grid cell or minimum distance threshold, the algorithm

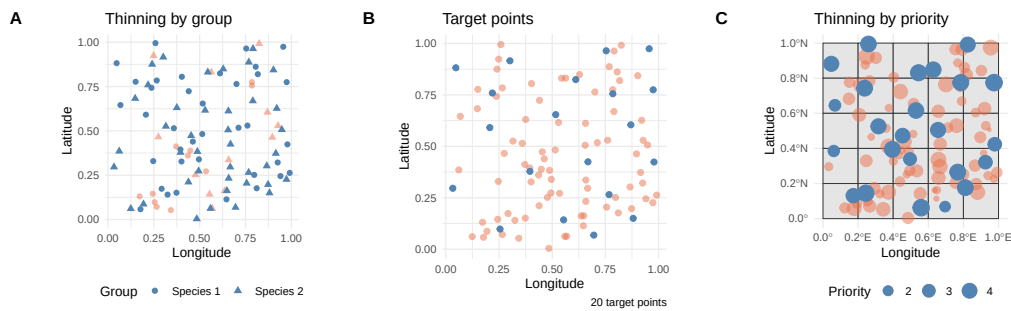


Figure 2: Additional spatial thinning functionalities available in GeoThinnerR: (A) group-wise thinning applied independently for each species, (B) retaining a fixed number of points (20 points in this example) while ensuring spatial separation, and (C) prioritizing points based on larger variable values within each grid cell. Retained points are shown in blue, and removed points are in red.

retains the point with the highest priority value. This gives users more control over the characteristics of the retained dataset. For example, heterogeneous datasets from sources such as GBIF often include records of varying spatial precision. In such cases, prioritization allows users to preferentially retain higher-quality records instead of selecting points at random.

```
# Thinning priority
sim_data$priority <- runif(100, 1, 5)
rast_obj <- terra::rast(xmin = 0, xmax = 1, ymin = 0, ymax = 1, res = 0.2)
priority_thin <- thin_points(sim_data,
  method = "grid", raster_obj = rast_obj,
  priority = sim_data$priority, seed = 3455
)
```

3 Modified distance-based thinning algorithm based on *kd*-tree

The distance-based thinning process can be conceptualized as a two-step procedure: (1) identifying neighbouring points within a given distance threshold, and (2) selectively removing points to meet the selection criteria. Current methods in SDM workflows use greedy approaches in the first step, computing all pairwise distances between points and resulting in a time complexity of $O(n^2)$ and high memory usage. **GeoThinnerR**, like **spThin** (Aiello-Lammens et al., 2015) and **enmSdmX** (Smith et al., 2023), implements a brute-force method using the `rdist.earth()` function from the **fields** package (Douglas Nychka et al., 2021). To improve performance, we propose two minor modifications to *kd*-tree nearest-neighbour search algorithms that improve scalability for large datasets (Figure 3). These methods reduce computational complexity while ensuring exact thinning distances, so they do not change which points are retained.

3.1 Traditional *kd*-tree nearest neighbour search

A *kd*-tree (Friedman et al., 1975) is a data structure that organizes points in a *k*-dimensional space by recursively partitioning the data along hyperplanes creating a binary search tree. Each node in the *kd*-tree represents a data point in the multidimensional space, and internal (non-leaf) nodes define the splitting hyperplanes that divide the space into two subsets of points based on their position relative to the split (Ram and Sinha, 2019). Many *kd*-tree variants exist, but the essence is that they recursively partition the space containing the set of points S into hyper-rectangles that contain a small subset $S'_i \subset S$ of the input points (Panigrahy, 2008).

GeoThinnerR implements *kd*-tree construction and nearest-neighbour searches using the **nabor** package, which wraps the **libnabo** library (Elseberg et al., 2012). Since these *kd*-trees are built on Euclidean distances, geographic coordinates (latitude and longitude) are transformed into Cartesian coordinates (x, y, z) before constructing the binary tree:

$$\begin{aligned} x &= R \cdot \cos(\text{lat}) \cdot \cos(\text{lon}) \\ y &= R \cdot \cos(\text{lat}) \cdot \sin(\text{lon}) \\ z &= R \cdot \sin(\text{lat}), \end{aligned} \quad (2)$$

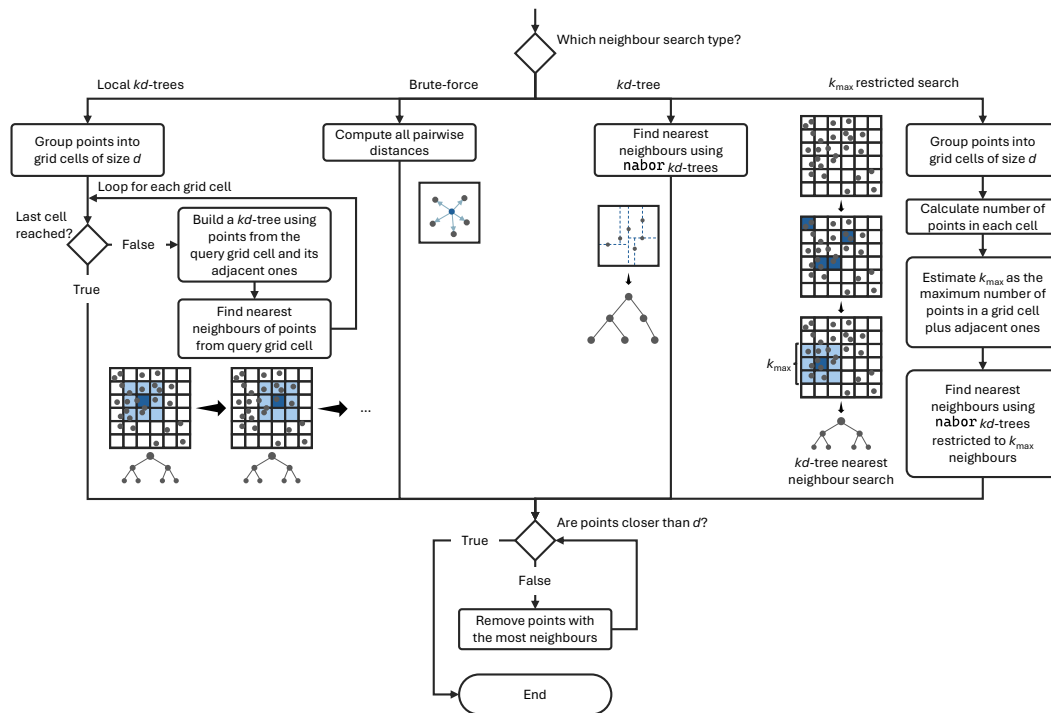


Figure 3: Diagram illustrating the four nearest neighbour search strategies implemented in GeoThinner for distance-based thinning. The methods include brute-force, standard $*kd$ -trees, local $*kd$ -trees, and restricted $*kd$ -trees using estimated k_{max} . Each approach computes neighbouring points within a thinning distance before proceeding with point removal.

where lat and lon are the latitude and longitude in radians, respectively ($\frac{\pi}{180} (lon, lat)$), and R is the Earth's radius. This results in a three-dimensional kd -tree.

However, performance may be worsened in higher dimensions or when many points must be evaluated as potential neighbours of the query point q . The time required to find the neighbours of q is affected by the search for candidate neighbours and the time it takes to evaluate among all candidates which ones are actually neighbours (Ram and Sinha, 2019). While approximate searches can reduce computational time by tuning the precision-recall trade-off, exact searches in large datasets require exhaustive searches, leading to higher computational complexity. To improve scalability, we introduce two additional optimizations: local kd -trees partitioning and restricted neighbour searches.

3.2 Local kd -trees for scalable searches

To reduce the complexity of searching within a single kd -tree constructed for the entire set of points S , which requires evaluating all points to identify neighbours within the distance threshold, we propose constructing local kd -trees by subdividing the spatial domain into multiple smaller regions, each with its own tree. Although creating multiple kd -trees is time-consuming, memory usage and overall runtime are improved for large datasets, as neighbour searches are less exhaustive when evaluating smaller subsets $S'_i \subset S$.

To find exact neighbours using this approach, we first divide the spatial domain into a grid with cell sizes equal to the thinning distance and assign each point to a grid cell. Then, for each grid cell, we identify points within that cell and its adjacent cells, as these are potential neighbours. Recursively, an independent kd -tree is constructed for each grid cell using the `nabor` package, and neighbours are identified for each query point within the cell.

This strategy maintains computational efficiency while avoiding memory overloads of constructing and querying large trees. Furthermore, `GeoThinner` uses parallel processing when `n_cores` is set to a value greater than one, providing a straightforward way to scale thinning operations across large datasets.

3.3 Restricted neighbour searches

When performing the nearest-neighbour searches using *kd*-trees from [nabor](#), we have to specify the number of neighbours to find k which increases or decreases the search complexity. As the number of potential neighbours for a point q is unknown, we set k equal to the total number of points n in the dataset, ensuring that all points are checked to determine whether they are neighbours of the query point. However, for large datasets, this approach introduces unnecessary computational costs by evaluating distant points. Instead, we propose an adaptive neighbour search that estimates $k_{\max}$ based on local point densities to reduce the number of nearest neighbours to find for each query point.

The process involves dividing the spatial domain into grid cells of size equal to the thinning distance d , then counting the number of points in each grid cell and identifying the densest cells. The maximum number of neighbours a point may have $k_{\max}$ is estimated as the maximum number of points found within any dense cell and its adjacent cells. This value provides a practical upper bound on the neighbour search and is used to limit the number of points evaluated during the *kd*-tree query, reducing unnecessary searches while still ensuring exact neighbour identification.

This method significantly reduces computational time, particularly in datasets where the maximum number of neighbours per point is much smaller than the total number of points. However, one limitation is that densely populated areas or large thinning distances may inflate $k_{\max}$, making the method less effective. In such cases, the local *kd*-tree method with parallelization provides a suitable alternative.

3.4 Performance comparison of neighbour search methods

In this section, we evaluate the computational efficiency of the different neighbour search methods implemented in [GeoThinneR](#) using simulated datasets of varying sizes and spatial structures. All tests were conducted using R 4.3.3 on a Windows 11 computer with 128 GB of RAM and an Intel(R) Xeon(R) Gold 6240R processor featuring 24 cores, 48 threads, and a base clock speed of 2.40 GHz. The benchmark code is available in the supplementary R files.

We simulated datasets within a unit square domain $[0, 1] \times [0, 1]$, ranging from 1,000 to 50,000 points, under three spatial processes to mimic different real-world scenarios: (i) a homogeneous Poisson process simulating complete spatial randomly distributed points, (ii) a Matern clustered point process generated using the `rMatClust` function from the [spatstat](#) package ([Baddeley et al., 2015](#)) with 10 cluster centers (κ), a cluster radius of 0.15 (scale), and a varying number of points per cluster (μ), and (iii) a mixed spatial pattern dataset combining the randomly distributed points and the clustered processes. Finally, we benchmarked the methods using two thinning distances (1 km and 10 km) to highlight the impact of increasing the number of points that need to be removed and how each method scales with different search complexities.

We compared the neighbour search methods in [GeoThinneR](#) using execution time (seconds) and peak RAM usage (MB) to evaluate the computational speed and memory requirements, which are critical for large-scale spatial datasets. Each method was tested three times, and the mean values were calculated.

The two modified algorithms ($k_{\max}$ estimation and local *kd*-trees) consistently outperform the brute-force and traditional *kd*-tree methods in both peak memory usage (Figure 4a) and execution time (Figure 4b) across all spatial data types. The traditional *kd*-tree method provides better performance compared to the brute-force method for small datasets; however, for larger datasets, memory usage and execution time increase due to exhaustive searches in deeper trees to find all neighbours of each point. However, the two optimized methods are orders of magnitude faster and less memory-intensive. For small thinning distances, the $k_{\max}$ estimation algorithm shows the best performance in terms of both time and memory usage, because the estimated number of neighbours $k_{\max}$ remains low, reducing unnecessary searches. However, at larger thinning distances, and particularly for the clustered and mixed datasets, as the dataset size increases, this algorithm becomes slower and uses more memory than the local *kd*-trees approach. This is because the algorithm estimates a high value for $k_{\max}$ and, therefore, increases search complexity.

Alternatively, the local *kd*-tree approach, which generates an independent *kd*-tree for each subregion, offers improved memory efficiency and outperforms the other methods as thinning distances increase, as fewer trees need to be built. This method can also be executed in parallel, which can further improve run time for large datasets, especially when many partitions are created due to a small thinning distance or a large spatial extent (Figure 4c). However, if the number of partitions is not very large, as in the case of the 1 km thinning distance in this example, parallel execution may not bring any improvement.

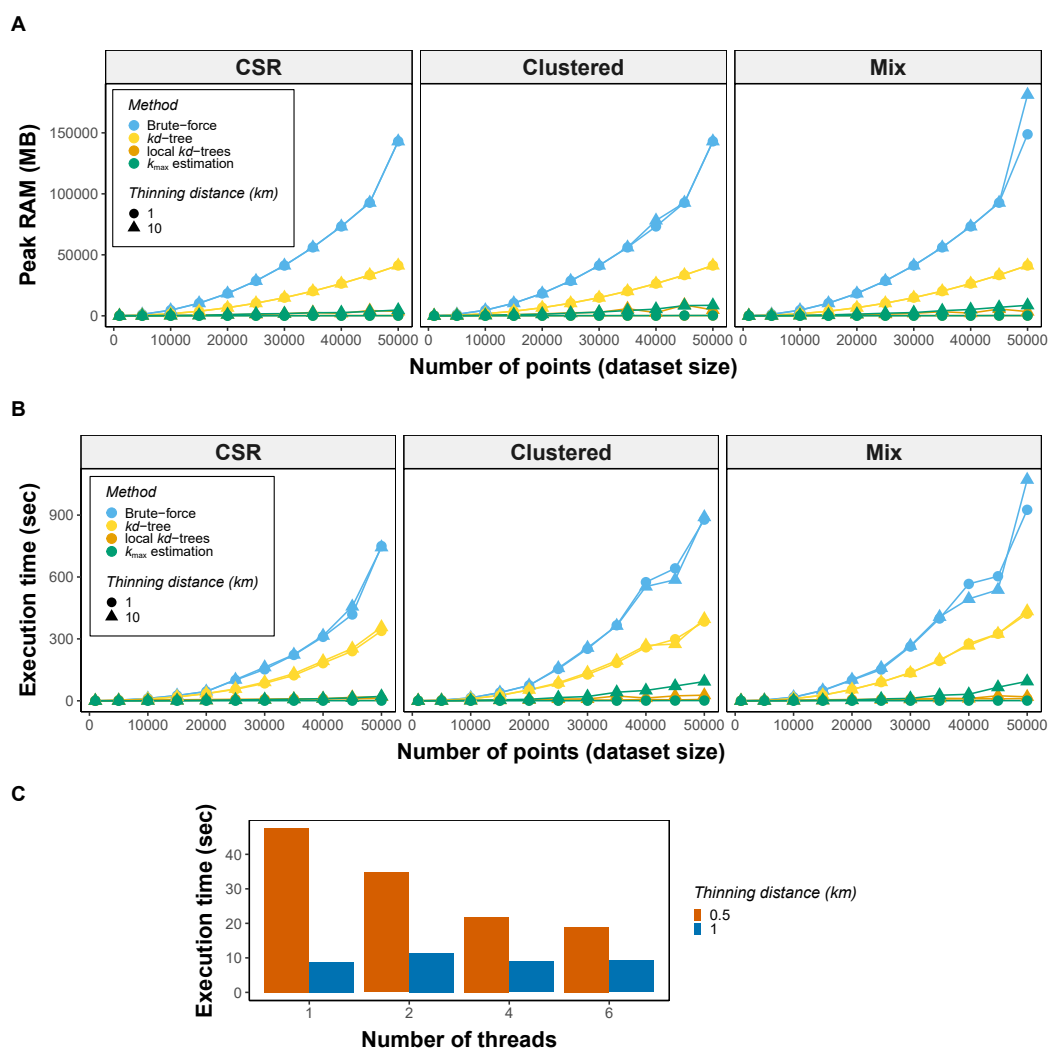


Figure 4: Performance benchmark for neighbour searches implemented in GeoThinneR distance-based thinning. (A) Peak RAM usage (MB) and (B) execution time (seconds) across three spatial data types (CSR, clustered, and mixed spatial pattern) for dataset sizes ranging from 1,000 to 50,000 points. (C) Performance comparison of local *kd*-trees with parallelization for 50,000 randomly distributed points using two thinning distances (0.5 km and 1 km). The optimized algorithms (local *kd*-trees and *k*-estimation) scale better and provide superior computational efficiency than brute-force and standard *kd*-tree, especially for large datasets.

4 Performance benchmark

In this section, we compare the performance of **GeoThinnerR** with two widely used R packages for spatial thinning: **spThin** and **enmSdmX**. We compare the execution time and memory usage of the distance-based thinning algorithms available in **GeoThinnerR** version 2.0.0 against the `thin()` function from **spThin** version 0.2.0 and the `geoThin()` function from **enmSdmX** version 1.2.10. In Section 4.1, we compare the performance using simulated data, and in Section 4.2, we benchmark the methods using a real-world large dataset.

```
brute <- thin_points(data,
  method = "distance", search_type = "brute",
  thin_dist = thin_dist, trials = trials
)
kd_tree <- thin_points(data,
  method = "distance", search_type = "kd_tree",
  thin_dist = thin_dist, trials = trials
)
local_kd_tree <- thin_points(data,
  method = "distance", search_type = "local_kd_tree",
  thin_dist = thin_dist, trials = trials
)
k_estimation <- thin_points(data,
  method = "distance", search_type = "k_estimation",
  thin_dist = thin_dist, trials = trials
)
spThin <- thin(data,
  lat.col = "lat", long.col = "lon", spec.col = "group",
  thin.par = thin_dist, reps = trials, locs.thinned.list.return = TRUE,
  write.files = FALSE, write.log.file = FALSE, verbose = FALSE
)
enmSdmX <- geoThin(data,
  minDist = thin_dist * 1000, longLat = c("lon", "lat"), method = "complete"
)
```

4.1 Benchmark with simulated data

The benchmark tests were performed on the complete thinning workflow using the simulated datasets described in Section 3.4. Each thinning method was executed with a single trial to obtain the maximum number of retained points per run, and each test was repeated three times to ensure reliable timing estimates. Here, we compared all methods across three spatial data types (random, clustered, and mixed) using again two thinning distances of 1 and 10 kilometers.

The optimized algorithms implemented in **GeoThinnerR** demonstrate significant improvements in both memory efficiency (Figure 5a) and execution time (Figure 5b) compared to existing tools. The **enmSdmX** package exhibited the longest execution times, making it unsuitable for large datasets, so we limited its tests to 20,000 points. **spThin**, although it's more efficient, shows similar performance to the brute-force method from **GeoThinnerR**. In contrast, our new modified methods demonstrate better scalability in both runtime and peak memory usage, making them more practical for large-scale spatial thinning.

The local *kd*-tree and $k_{\max}$ estimation algorithms in **GeoThinnerR** present the best scalability for large datasets. The patterns are similar to those observed when comparing the neighbour search methods. The $k_{\max}$ estimation method is best suited for small thinning distances and for datasets without relatively dense regions, while the local *kd*-tree method offers a good alternative for highly clustered datasets or large thinning distances. This flexibility ensures that **GeoThinnerR** can adapt to various real-world SDM workflows more efficiently than existing tools.

4.2 Application to real-world data

To illustrate the performance of **GeoThinnerR** on real-world datasets, we applied its optimized thinning methods and compared them with **spThin**. We used a small dataset of 8,340 occurrence records for the loggerhead sea turtle (*Caretta caretta*) in the Mediterranean Sea (GBIF.org, 2025b), and a large and highly clustered dataset of 80,163 points for the yellowfin tuna (*Thunnus albacares*) collected globally from 1950 to 2025 (GBIF.org, 2025c). Both datasets were downloaded from GBIF (GBIF.org, 2025a) and

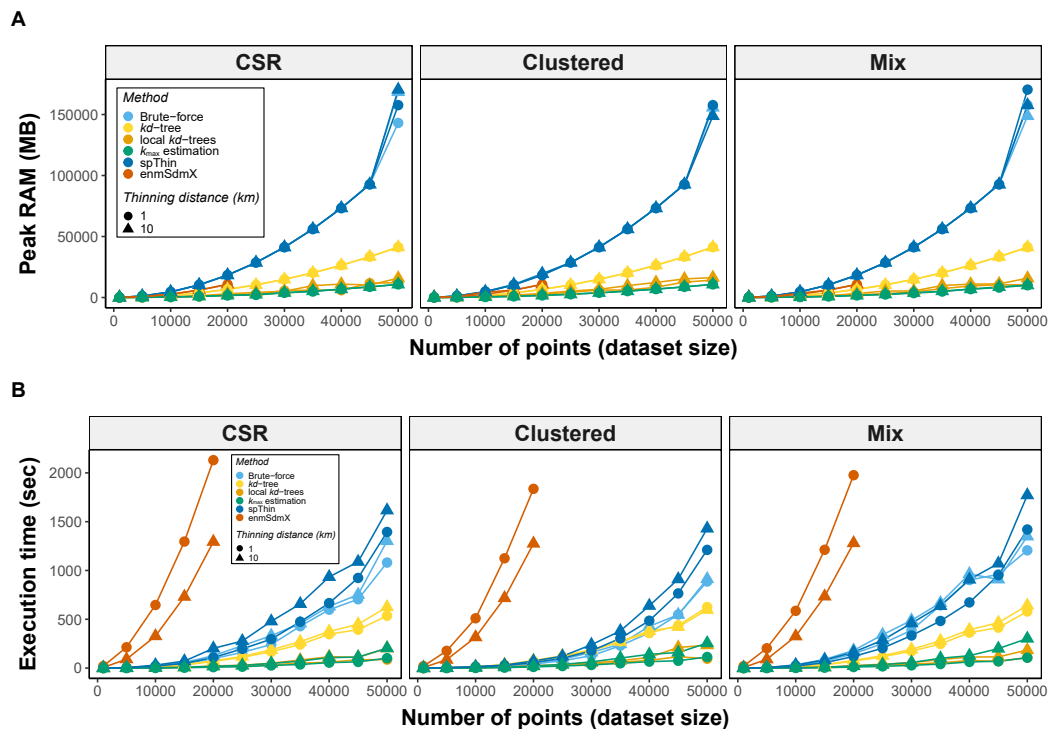


Figure 5: Performance benchmark for distance-based thinning across GeoThinnerR, spThin, and enmSdmX. (A) Peak RAM usage (MB) and (B) execution time (seconds) across three spatial data types for varying dataset sizes. The enmSdmX method was only evaluated up to 20,000 points due to its high computational cost on larger datasets. The optimized methods of GeoThinnerR (local **kd**-trees and **k**-estimation) offer better memory and runtime performance across all settings.

are heterogeneous, including data from different sources, locations, and sampling methods. The data are available in the [GeoThinnerR](#) package and can be loaded using the `data()` function.

```
# Loggerhead sea turtle
data(caretta)
# Yellowfin tuna
data(thunnus)
```

Here we show the performance of the local *kd*-tree and k_{max} estimation algorithms implemented in [GeoThinnerR](#), comparing their computational performance against [spThin](#) using varying thinning distances. The objective is to show the trade-offs, best- and worst-case scenarios for each method, that's why we use varying thinning distances and dataset sizes. So we are not trying to identify the optimal thinning distance for SDM applications, this comparison highlights the scalability and efficiency of the algorithms. It is important to note that choosing an appropriate thinning distance for SDM is challenging and often requires being tuned empirically (Veloz, 2009; Soley-Guardia et al., 2019).

The results in Figure 6 show significant performance differences even for the smaller dataset. [GeoThinnerR](#) not only executed faster but also demonstrated substantially lower memory usage, addressing one of the key limitations of current spatial thinning methods in SDM (Figure 6a). The k_{max} estimation algorithm was particularly efficient, with run times between one and two seconds due to low neighbour density that minimizes search complexity.

In contrast, the large yellowfin tuna dataset presented a different scenario (Figure 6b). [spThin](#) failed to execute due to excessive memory demands, surpassing our 128 GB RAM limit after several minutes, which highlights the scalability challenges of brute-force methods. Both algorithms in [GeoThinnerR](#) maintained low memory usage, even at this dataset size. However, the k_{max} estimation method experienced slower performance as thinning distances increased because the dataset presents highly clustered areas which inflate k_{max} estimates and increase search complexity. To address this, the local *kd*-tree approach, executed here with multithreading (6 threads), proved both faster and less memory-intensive, being the method with best performance in this context. These results illustrate that while k_{max} estimation is highly efficient for moderately sized or uniformly distributed datasets,

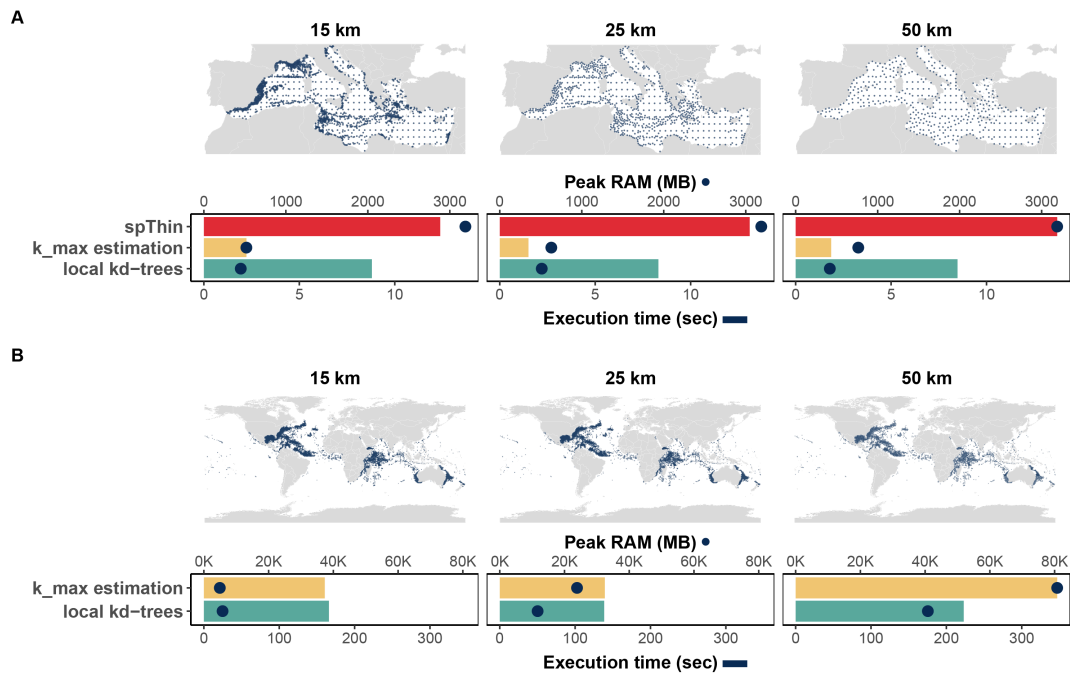


Figure 6: Execution time (bars) and memory usage (points) of GeoThinnerR and spThin across varying thinning distances for occurrence data of (A) loggerhead turtles and (B) yellowfin tuna from GBIF. For the yellowfin tuna dataset, the results for spThin are not shown because the run could not be completed due to excessive memory usage.

the local *kd*-tree method with parallelization offers a memory-efficient solution for large and dense datasets.

Here we present the code used to generate the thinned datasets shown in Figure 6:

```
caretta_thin <- thin_points(
  caretta,
  lon_col = "decimalLongitude",
  lat_col = "decimalLatitude",
  method = "distance",
  thin_dist = thin_dist, # One of 10, 25, 50
  search_type = "k_estimation",
  trials = 10
)

thunnus_thin <- thin_points(
  thunnus,
  lon_col = "decimalLongitude",
  lat_col = "decimalLatitude",
  method = "distance",
  thin_dist = thin_dist, # One of 10, 25, 50
  search_type = "local_kd_tree",
  trials = 1,
  n_cores = 6
)
```

5 Conclusions

The **GeoThinnerR** R package offers a flexible and scalable implementation of spatial thinning methods, addressing key limitations of existing tools. It integrates multiple thinning approaches (distance-, grid-, and precision-based) into a single wrapper function, enhancing usability. By incorporating optimized *kd*-tree algorithms for distance-based thinning, such as local *kd*-tree partitioning and adaptive $k_{\max}$ estimation, the package significantly reduces execution time and memory usage for

large datasets. Additionally, **GeoThinneR** includes customization features designed for species distribution modelling workflows, such as group-wise thinning and point prioritization. While future developments should focus on improving the computational performance of distance-based methods, incorporating additional thinning approaches, and estimating an optimal thinning distance, **GeoThinneR** represents a step forward in improving the efficiency of spatial thinning methods in SDM.

6 Acknowledgments

This package has been developed as part of the ProOceans (PID2020-118097RB-I00) and SOSPen (PID2021-124831OA-I00) projects funded by the Spanish Ministry of Science and Innovation. The author also acknowledges the support from the MOIRAI project (grant agreement no. 101180994) funded by the European Union under the Horizon Europe program. The author acknowledges the institutional support of the “Severo Ochoa Center of Excellence” accreditation (CEX2024-001494-S funded by AEI 10.13039/501100011033) to the Institute of Marine Science (ICM-CSIC). Additionally, this work is part of the Integrated Marine Ecosystem Assessments (iMARES) research group, funded by the Agència de Gestió d’Ajuts Universitaris i de Recerca of the Generalitat de Catalunya (2021 SGR 00435).

7 Supplementary materials

Supplementary materials are available in addition to this article:

- *GeoThinneR_replication_small.R*: reproduces examples and figures from Sections 2 and 3. Includes a short benchmark and demonstrations of package features using small datasets.
- *GeoThinneR_replication_large.R*: reproduces figures and results from Section 4. Includes a larger benchmark to evaluate performance of GeoThinneR and other packages using simulated and real-world datasets.
- *caretta_download.R*: downloads occurrence data for loggerhead turtles (*Caretta caretta*) from GBIF.
- *thunnus_download.R*: downloads occurrence data for yellowfin tuna (*Thunnus albacares*) from GBIF.
- *data/*: contains benchmark results and the cleaned loggerhead turtle and yellowfin tuna datasets used.

References

- M. E. Aiello-Lammens, R. A. Boria, A. Radosavljevic, B. Vilela, and R. P. Anderson. *spThin*: An R package for spatial thinning of species occurrence records for use in ecological niche models. *Ecography*, 38(5):541–545, 2015. doi: 10.1111/ecog.01132. [p299, 301, 304]
- A. V. R. Amaral, E. T. Krainski, R. Zhong, and P. Moraga. Model-based geostatistics under spatially varying preferential sampling. *Journal of Agricultural, Biological and Environmental Statistics*, 29(4): 766–792, 2024. doi: 10.1007/s13253-023-00571-0. [p299]
- A. Baddeley, E. Rubak, and R. Turner. *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press, London, 2015. ISBN 9781482210200. URL <https://www.routledge.com/Spatial-Point-Patterns-Methodology-and-Applications-with-R/Baddeley-Rubak-Turner/p/book/9781482210200/>. [p306]
- D. J. Baker, I. M. Maclean, and K. J. Gaston. Effective strategies for correcting spatial sampling bias in species distribution models without independent test data. *Diversity and Distributions*, 30(3):e13802, 2024. doi: 10.1111/ddi.13802. [p299]
- R. A. Barber, S. G. Ball, R. K. Morris, and F. Gilbert. Target-group backgrounds prove effective at correcting sampling bias in Maxent models. *Diversity and Distributions*, 28(1):128–141, 2022. doi: 10.1111/ddi.13442. [p299]
- J. Beck, M. Böller, A. Erhardt, and W. Schwanghart. Spatial bias in the GBIF database and its effect on modeling species’ geographic distributions. *Ecological Informatics*, 19:10–15, 2014. doi: 10.1016/j.ecoinf.2013.11.002. [p299]

- E. H. Boakes, P. J. McGowan, R. A. Fuller, D. Chang-qing, N. E. Clark, K. O'Connor, and G. M. Mace. Distorted views of biodiversity: Spatial and temporal bias in species occurrence data. *PLoS Biology*, 8(6):e1000385, 2010. doi: 10.1371/journal.pbio.1000385. [p299]
- R. A. Boria, L. E. Olson, S. M. Goodman, and R. P. Anderson. Spatial filtering to reduce sampling bias can improve the performance of ecological niche models. *Ecological Modelling*, 275:73–77, 2014. doi: 10.1016/j.ecolmodel.2013.12.012. [p299]
- J. M. Calabrese, G. Certain, C. Kraan, and C. F. Dormann. Stacking species distribution models and adjusting bias by linking them to macroecological models. *Global Ecology and Biogeography*, 23(1): 99–112, 2014. doi: 10.1111/geb.12102. [p300]
- A. A. Castellanos, J. W. Huntley, G. Voelker, and A. M. Lawing. Environmental filtering improves ecological niche models across multiple scales. *Methods in Ecology and Evolution*, 10(4):481–492, 2019. doi: 10.1111/2041-210X.13142. [p299]
- Y. Chauvier, N. E. Zimmermann, G. Poggiato, D. Bystrova, P. Brun, and W. Thuiller. Novel methods to correct for observer and sampling bias in presence-only species distribution models. *Global Ecology and Biogeography*, 30(11):2312–2325, 2021. doi: 10.1111/geb.13383. [p299]
- N. R. Chopde and M. Nichat. Landmark based shortest path detection by using A* and Haversine formula. *International Journal of Innovative Research in Computer and Communication Engineering*, 1(2):298–302, 2013. URL <https://scispace.com/pdf/landmark-based-shortest-path-detection-byusing-a-and-2nsrbowy4m.pdf>. [p302]
- F. Cosentino and L. Maiorano. Is geographic sampling bias representative of environmental space? *Ecological Informatics*, 64:101369, 2021. doi: 10.1016/j.ecoinf.2021.101369. [p299]
- P. J. Diggle, R. Menezes, and T.-I. Su. Geostatistical inference under preferential sampling. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 59(2):191–232, 2010. doi: 10.1111/j.1467-9876.2009.00701.x. [p299]
- Douglas Nychka, Reinhard Furrer, John Paige, and Stephan Sain. fields: Tools for spatial data, 2021. URL <https://github.com/dnychka/fieldsRPackage>. R package version 16.3. [p303, 304]
- J. Elith and J. R. Leathwick. Species distribution models: Ecological explanation and prediction across space and time. *Annual Review of Ecology, Evolution, and Systematics*, 40(1):677–697, 2009. doi: 10.1146/annurev.ecolsys.110308.120159. [p299]
- J. Elseberg, S. Magnenat, R. Siegwart, and A. Nüchter. Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration. *Journal of Software Engineering for Robotics (JOSER)*, 3(1):2–12, 2012. ISSN 2035-3928. URL <https://robotik.informatik.uni-wuerzburg.de/telematics/download/joser2012.pdf>. [p303, 304]
- J. H. Friedman, J. L. Bentley, and R. A. Finkel. *An Algorithm for Finding Best Matches in Logarithmic Time*. Department of Computer Science, Stanford University, 1975. doi: 10.1145/355744.355745. [p300, 303, 304]
- GBIF.org. GBIF Home Page, 2025a. URL <https://www.gbif.org/>. [26 November 2025]. [p308]
- GBIF.org. GBIF Occurrence Download. 2025b. doi: 10.15468/dl.9jcjrm. [p308]
- GBIF.org. GBIF Occurrence Download. 2025c. doi: 10.15468/dl.xsyrrh. [p308]
- A. Guisan, R. Tingley, J. B. Baumgartner, I. Naujokaitis-Lewis, P. R. Sutcliffe, A. I. Tulloch, T. J. Regan, L. Brotons, E. McDonald-Madden, C. Mantyka-Pringle, et al. Predicting species distributions for conservation decisions. *Ecology Letters*, 16(12):1424–1435, 2013. doi: 10.1111/ele.12189. [p299]
- M. Higa, Y. Yamaura, I. Koizumi, Y. Yabuhara, M. Senzaki, and S. Ono. Mapping large-scale bird distributions using occupancy models and citizen data with spatially biased sampling effort. *Diversity and Distributions*, 21(1):46–54, 2015. doi: 10.1111/ddi.12255. [p299]
- R. J. Hijmans, S. Phillips, J. Leathwick, and J. Elith. *dismo: Species Distribution Modeling*, 2023. URL <https://github.com/rspsatial/dismo>. R package version 1.3-14. [p299, 302]
- A. C. Hughes, M. C. Orr, K. Ma, M. J. Costello, J. Waller, P. Provoost, Q. Yang, C. Zhu, and H. Qiao. Sampling biases shape our view of the natural world. *Ecography*, 44(9):1259–1269, 2021. doi: 10.1111/ecog.05926. [p299]

- R. Inman, J. Franklin, T. Esque, and K. Nussear. Comparing sample bias correction methods for species distribution modeling using virtual species. *Ecosphere*, 12(3):e03422, 2021. doi: 10.1002/ecs2.3422. [p299]
- R. Kindt, A. Abiyu, P. Borchardt, I. Dawson, S. Demissew, L. Gaudal, R. Jamnadass, J.-P. Lillesø, S. Moestrup, F. Pedercini, et al. *The Climate Change Atlas for Africa of Tree Species Prioritized for Forest Landscape Restoration in Ethiopia: A Description of Methods Used to Develop the Atlas*. CIFOR, 2023. doi: 10.17528/cifor-icraf/008977. [p302]
- V. Koshkina, Y. Wang, A. Gordon, R. M. Dorazio, M. White, and L. Stone. Integrated species distribution models: Combining presence-background data and site-occupancy data with imperfect detection. *Methods in Ecology and Evolution*, 8(4):420–430, 2017. doi: 10.1111/2041-210X.12738. [p299]
- J. M. Lobo and M. F. Tognelli. Exploring the effects of quantity and location of pseudo-absences and sampling biases on the performance of distribution models with limited point occurrence data. *Journal for Nature Conservation*, 19(1):1–7, 2011. doi: 10.1016/j.jnc.2010.03.002. [p299]
- J. Mäkinen, C. Merow, and W. Jetz. Integrated species distribution models to account for sampling biases and improve range-wide occurrence predictions. *Global Ecology and Biogeography*, 33(3): 356–370, 2024. doi: 10.1111/geb.13792. [p299]
- E. Maria, E. Budiman, M. Taruk, et al. Measure distance locating nearest public facilities using Haversine and euclidean methods. In *Journal of Physics: Conference Series*, volume 1450, page 012080. IOP Publishing, 2020. doi: 10.1088/1742-6596/1450/1/012080. [p302]
- J. Mestre-Tomás. *GeoThinnerR: Efficient Spatial Thinning of Species Occurrences*, 2025. URL <https://github.com/jmestret/GeoThinnerR>. R package version 2.1.0. [p300]
- C. Meyer, W. Jetz, R. P. Guralnick, S. A. Fritz, and H. Kreft. Range geometry and socio-economics dominate species-level biases in occurrence information. *Global Ecology and Biogeography*, 25(10): 1181–1193, 2016. doi: 10.1111/geb.12483. [p299]
- J. Miller. Species distribution modeling. *Geography Compass*, 4(6):490–509, 2010. doi: 10.1111/j.1749-8198.2010.00351.x. [p299]
- V. Moudry, M. Bazzichetto, R. Remelgado, R. Devillers, J. Lenoir, R. G. Mateo, J. J. Lembrechts, N. Sillero, V. Lecours, A. F. Cord, et al. Optimising occurrence data in species distribution models: Sample size, positional uncertainty, and sampling bias matter. *Ecography*, 2024(12):e07294, 2024. doi: 10.1111/ecog.07294. [p299]
- S. Noori, A. Hofmann, D. Rödder, M. Husemann, and H. Rajaei. A window to the future: Effects of climate change on the distribution patterns of iranian zygaenidae and their host plants. *Biodiversity and Conservation*, 33(2):579–602, 2024. doi: 10.1007/s10531-023-02760-2. [p300]
- O. Ovaskainen and N. Abrego. *Joint Species Distribution Modelling: With Applications in R*. Cambridge University Press, 2020. ISBN 9781108591720. doi: 10.1017/9781108591720. [p300]
- O. Ovaskainen, G. Tikhonov, A. Norberg, F. Guillaume Blanchet, L. Duan, D. Dunson, T. Roslin, and N. Abrego. How to make more out of community data? a conceptual framework and its implementation as models and software. *Ecology Letters*, 20(5):561–576, 2017. doi: 10.1111/ele.12757. [p300]
- R. Panigrahy. An improved algorithm finding nearest neighbor using kd-trees. In *Latin American Symposium on Theoretical Informatics*, pages 387–398. Springer, 2008. doi: 10.1007/978-3-540-78773-0_34. [p304]
- J. L. Pearce and M. S. Boyce. Modelling distribution and abundance with presence-only data. *Journal of Applied Ecology*, 43(3):405–412, 2006. doi: 10.1111/j.1365-2664.2005.01112.x. [p299]
- M. G. Pennino, I. Paradinas, J. B. Illian, F. Muñoz, J. M. Bellido, A. López-Quílez, and D. Conesa. Accounting for preferential sampling in species distribution models. *Ecology and Evolution*, 9(1): 653–663, 2019. doi: 10.1002/ece3.4789. [p299]
- S. J. Phillips, M. Dudík, J. Elith, C. H. Graham, A. Lehmann, J. Leathwick, and S. Ferrier. Sample selection bias and presence-only distribution models: Implications for background and pseudo-absence data. *Ecological Applications*, 19(1):181–197, 2009. doi: 10.1890/07-2153.1. [p299]

- L. J. Pollock, R. Tingley, W. K. Morris, N. Golding, R. B. O'Hara, K. M. Parris, P. A. Vesk, and M. A. McCarthy. Understanding co-occurrence by modelling species simultaneously with a joint species distribution model (JSJM). *Methods in Ecology and Evolution*, 5(5):397–406, 2014. doi: 10.1111/2041-210X.12180. [p300]
- H. R. Pulliam. On the relationship between niche and distribution. *Ecology Letters*, 3(4):349–361, 2000. doi: 10.1046/j.1461-0248.2000.00143.x. [p299]
- P. Ram and K. Sinha. Revisiting kd-tree for nearest neighbor search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1378–1388, 2019. doi: 10.1145/3292500.3330875. [p303, 304, 305]
- E. G. Simmonds, S. G. Jarvis, P. A. Henrys, N. J. Isaac, and R. B. O'Hara. Is more data always better? a simulation study of benefits and limitations of integrated distribution models. *Ecography*, 43(10):1413–1422, 2020. doi: 10.1111/ecog.05146. [p299]
- A. B. Smith, S. J. Murphy, D. Henderson, and K. D. Erickson. Including imprecisely georeferenced specimens improves accuracy of species distribution models and estimates of niche breadth. *Global Ecology and Biogeography*, 32(3):342–355, 2023. doi: 10.1111/geb.13628. [p299, 304]
- M. Soley-Guardia, A. C. Carnaval, and R. P. Anderson. Sufficient versus optimal climatic stability during the late quaternary: Using environmental quality to guide phylogeographic inferences in a neotropical montane system. *Journal of Mammalogy*, 100(6):1783–1807, 2019. doi: 10.1093/jmammal/gyz162. [p309]
- V. A. Steen, M. W. Tingley, P. W. Paton, and C. S. Elphick. Spatial thinning and class balancing: Key choices lead to variation in the performance of species distribution models with citizen science data. *Methods in Ecology and Evolution*, 12(2):216–226, 2021. doi: 10.1111/2041-210X.13525. [p299, 300]
- S. Varela, R. P. Anderson, R. García-Valdés, and F. Fernández-González. Environmental filters reduce the effects of sampling bias and improve predictions of ecological niche models. *Ecography*, 37(11):1084–1091, 2014. doi: 10.1111/j.1600-0587.2013.00441.x. [p299]
- S. D. Veloz. Spatially autocorrelated sampling falsely inflates measures of accuracy for presence-only niche models. *Journal of Biogeography*, 36(12):2290–2299, 2009. doi: 10.1111/j.1365-2699.2009.02174.x. [p299, 309]
- J. Vollering, R. Halvorsen, I. Auestad, and K. Rydgren. Bunching up the background betters bias in species distribution models. *Ecography*, 42(10):1717–1727, 2019. doi: 10.1111/ecog.04503. [p299]
- M. S. Wisz, R. Hijmans, J. Li, A. T. Peterson, C. Graham, A. Guisan, and NCEAS Predicting Species Distributions Working Group. Effects of sample size on the performance of species distribution models. *Diversity and Distributions*, 14(5):763–773, 2008. doi: 10.1111/j.1472-4642.2008.00482.x. [p299]
- H. Yu, T. Wang, A. Skidmore, M. Heurich, and C. Bässler. How future climate and tree distribution changes shape the biodiversity of macrofungi across europe. *Diversity and Distributions*, 29(5):666–682, 2023. doi: 10.1111/ddi.13688. [p300]

Jorge Mestre-Tomás

Renewable Marine Resources Department, Institut de Ciències del Mar (ICM-CSIC), Barcelona, Spain
Department of Applied Statistics and Operational Research, and Quality, Universitat Politècnica de València (UPV), Valencia, Spain

<https://github.com/jmestret>

ORCID: 0000-0002-8983-3417

jorgemestre@icm.csic.es

Kernopt: A Package for Estimating Count Data Distributions with a Discrete Symmetric Optimal Kernel.

by Gilles Durrieu, Thomas Fillon, and Tristan Senga Kiessé

Abstract kernopt is an R package that implements a discrete symmetric optimal kernel for estimating count data distributions. The nonparametric estimator using the discrete symmetric optimal kernel was illustrated on simulated and real-world data included in the package, in comparison with two other discrete symmetric kernels.

1 Introduction

Since the original works of Rosenblatt (Rosenblatt, 1956) and Parzen (Parzen, 1962), nonparametric kernel smoothing was often studied for estimating continuous data distributions by using symmetric and asymmetric continuous kernels, e.g., Gaussian, Epanechnikov (Epanechnikov, 1969), on the one hand, beta (Chen, 1999), inverse Gaussian (Scaillet, 2004), on the other hand, respectively. Then, discrete kernel smoothing was developed to specifically address the nonparametric estimation of count data distributions by using discrete kernels associated to discrete probability distributions, such as the kernel of Racine et al. (2020) for ordered data, Conway-Maxwell-Poisson kernel (Huang et al., 2022) and, more recently, underdispersed count kernels (Kokonendji et al., 2023). That was a valuable alternative to the traditional parametric Poisson modeling, which is often of limited use because empirical count data sets typically exhibit over/under-dispersion and/or an excess number of zeros (Kokonendji and Senga Kiessé, 2011). One R package implements the kernel method for modeling distributions of continuous (bounded and positive) and discrete (categorical and count) data (Ake package by Wansouwé et al. (2016)). Nevertheless, kernel density estimation using traditional continuous symmetric kernels (e.g., Gaussian, Epanechnikov, rectangular, triangular) is available by using the function `density` available in R. Various functions for kernel smoothing from the works of Wand and Jones (1995) were also implemented in **KernSmooth** package (Wand, 2025). In addition, multivariate kernel density estimation was developed in **ks** package (Duong, 2007).

The wide variety of discrete associated kernels available in the literature raised the issue of finding one discrete “optimal” associated kernel, which was developed by Senga Kiessé and Durrieu (2024). That new discrete symmetric “optimal” kernel is implemented in the R package **kernopt** which is also available at <https://github.com/thomasfillon/kernopt>. Other discrete associated kernels are also available, e.g., Epanechnikov (Chu et al., 2017), triangular and binomial (Wansouwé et al., 2016), for a comparison with the new discrete kernel developed. In addition to the choice of discrete kernel, the choice of the bandwidth parameter is challenging. We propose a cross-validation method for selecting the smoothing bandwidth to be used with the discrete kernel. Illustrations are provided on discrete simulated and real-world application data.

One can refer to the article of Senga Kiessé and Durrieu (2024) for technical details on mathematical results (propositions, theorem and proofs) provided in that paper. The **kernopt** package not only implements the discrete symmetric optimal kernel but also introduces a data-driven bandwidth selection method based on cross-validation, enhancing the estimator’s adaptability and performance to various data.

By providing a comprehensive and user-friendly implementation, the **kernopt** package plays a pivotal role in enabling researchers and practitioners to effectively apply nonparametric optimal kernel estimation to discrete count data. Available on both CRAN and GitHub, the package ensures broad accessibility and facilitates reproducibility in statistical analyses. Example usages of the package on both simulated data and real aquaculture data

from the SIMTAP project demonstrate the better estimation performance of our proposed method.

The paper is organized as follows. [Section 2](#) presents the fundamental principles of the nonparametric estimation method using discrete associated kernels for probability mass functions. In [Section 3](#), we describe the new discrete symmetric “optimal” kernel developed by [Senga Kiessé and Durrieu \(2024\)](#), with the procedure for bandwidth selection. The performance of the estimator with the discrete symmetric “optimal” kernel is illustrated in [Section 4](#) on simulated data, then applications on real data are provided in [Section 5](#).

2 Nonparametric discrete kernel density estimator

Let $X_1, \dots, X_n$ be a sequence of independent and identically distributed discrete random variable having an unknown probability mass function (pmf) f to estimate on discrete support $\mathcal{S} \subseteq \mathbb{Z}$, with $\mathbb{Z}$ being the set of all integer numbers. For a target point $x \in \mathcal{S}$ and a bandwidth parameter $h > 0$, the nonparametric estimator $\hat{f}_{K,h}$ of f is given by

$$\hat{f}_{K,h}(x) = \frac{1}{n} \sum_{i=1}^n K_{x,h}(X_i), \quad (1)$$

where $K_{x,h}$ is a symmetric pmf on discrete support $\mathcal{S}_x$, i.e. $\sum_{y \in \mathcal{S}_x} K_{x,h}(y) = 1$, $0 \leq K_{x,h}(y) \leq 1$ and $K_{x,h}(y) = K_{x,h}(-y)$. More precisely, a discrete associated kernel $K_{x,h}(\cdot)$ with a support $\mathcal{S}_x$ (which contains x) is a pmf of random variable $\mathcal{K}_{x,h}$, whose modal probability, mean and variance behave asymptotically as follows when $h \rightarrow 0$:

$$K_{x,h}(x) = \Pr(\mathcal{K}_{x,h} = x) \rightarrow 1, \mathbb{E}(\mathcal{K}_{x,h}) \rightarrow x \text{ and } \text{Var}(\mathcal{K}_{x,h}) \rightarrow 0. \quad (2)$$

The discrete kernel estimator (1) is defined up to the normalizing constant $C_n = \sum_{x \in \mathcal{S}} \hat{f}_{K,h}(x)$ in [Senga Kiessé \(2017\)](#).

We particularly considered a symmetric pmf $K_{x,h}$ satisfying the conditions in (2) such that

$$\mathbb{E}(\mathcal{K}_{x,h}) = x, \mathbb{E}(\mathcal{K}_{x,h}^2) = \tau^2(x, h) < \infty \text{ and } \text{Var}(\mathcal{K}_{x,h}) = \tau^2(x, h) - x^2 < \infty, \quad (3)$$

where $\text{Var}(\mathcal{K}_{x,h})$ does not depend on x and tends to 0 as h goes to 0.

2.1 Choice of discrete kernels

We consider two examples of discrete symmetric kernels with mean value equal to x , i.e., $\mathbb{E}(\mathcal{K}_{x,h}) = x$.

For a positive integer $h \in \mathbb{N} \setminus \{0\}$, the first example is a discrete symmetric Epanechnikov kernel with $\mathcal{S}_x = \{x - h, \dots, x, \dots, x + h\}$ given by

$$K_{x,h}^{Epan}(y) = \left(a \left(\frac{x-y}{h} \right)^2 + b \right) \mathbf{1}_{\{x-h, \dots, x, \dots, x+h\}}(y),$$

with $a = -b = 3h/(1 - 4h^2)$; see [Chu et al. \(2017\)](#). That kernel only involves the parameter h , which is a positive integer that scales the distance between x and y . The discrete Epanechnikov kernel is less often used in the literature, compared to its continuous counterpart.

Proposition 1

The random variable $\mathcal{K}_{x,h}^{Epan}$ associated to the discrete Epanechnikov kernel with support $\mathcal{S}_x = \{x - h, \dots, x, \dots, x + h\}$, with $h \in \mathbb{N} \setminus \{0\}$, has a Dirac distribution at x as $h \rightarrow 1$ and a discrete uniform distribution as $h \rightarrow \infty$.

The **kernopt** R package computes that kernel at a target x for various values of observations z and a fixed bandwidth h by using the function `discrete_kernel` such that (Figure 1):

```
x <- 5 # Target
z <- 0:10 # Observations
h <- c(1, 2, 3, 4) # Set of bandwidths
K_epan <- matrix(
  data = 0,
  nrow = length(z),
  ncol = length(h)
)
for (i in 1:length(h))
{
  K_epan[, i] <- discrete_kernel(kernel = "epanech", x, z, h[i])
}
plot(
  z,
  K_epan[, 1],
  xlab = "x",
  ylab = "Probability",
  ylim = c(0, 1),
  pch = 1
)
lines(z, K_epan[, 1], lty = 1)
for (i in 2:length(h))
{
  points(z, K_epan[, i], xlab = "z", pch = i)
  lines(z, K_epan[, i], lty = i)
}
legend(
  "topleft",
  c("h=1", "h=2", "h=3", "h=4"),
  lty = 1:4,
  pch = 1:4,
  cex = 1.6
)
```

For $h > 0$ and non-negative integer $a \in \mathbb{N}$, the second example is the discrete symmetric triangular kernel with $\mathcal{S}_x = \{x - a, \dots, x, \dots, x + a\}$, which has a pmf defined by

$$K_{a,x,h}^{Triang}(y) = \frac{(a+1)^h}{C(a,h)} \left(1 - \frac{|y-x|^h}{(a+1)^h}\right) \mathbf{1}_{\{x-a, \dots, x, \dots, x+a\}}(y),$$

with $C(a,h) = (2a+1)(a+1)^h - 2 \sum_{j=0}^a j^h$, see Kokonendji et al. (2007). That kernel depends on two parameters h and a . That kernel was implemented in the **kernopt** package through the function `discrete_kernel` with its specific arguments as follows (Figure 2):

```
x <- 5
z <- 0:10
h <- c(0.1, 0.4, 1, 2)
a <- 1

K_trg <- matrix(
  data = 0,
  nrow = length(z),
```

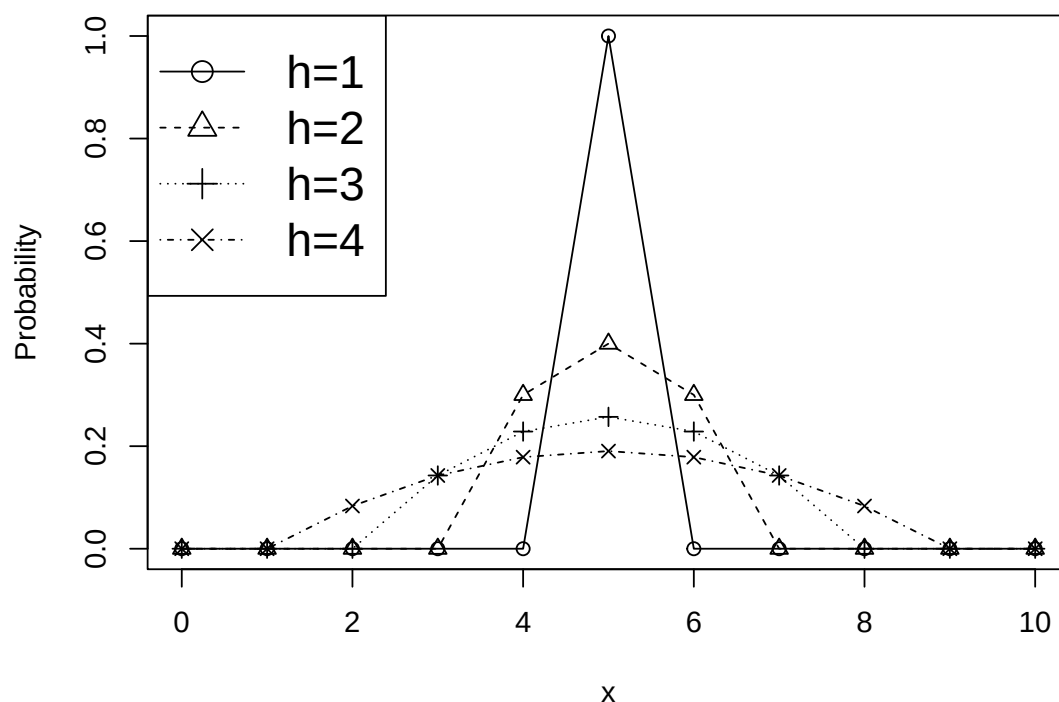


Figure 1: Distribution of discrete symmetric Epanechnikov kernel at the target point $x = 5$ for various bandwidth parameters h .

```

ncol = length(h)
)

for (i in 1:length(h))
{
  K_trg[, i] <- discrete_kernel(kernel = "triang", x, z, h[i], a)
}
plot(
  z,
  K_trg[, 1],
  xlab = "x",
  ylab = "Probability",
  ylim = c(0, 1),
  pch = 1
)
lines(z, K_trg[, 1], lty = 1)
for (i in 2:length(h))
{
  points(z, K_trg[, i], xlab = "z", pch = i)
  lines(z, K_trg[, i], lty = i)
}
legend(
  "topleft",
  c("h=0.1", "h=0.4", "h=1", "h=2"),
  lty = 1:4,
  pch = 1:4,
  cex = 1.6
)

```

The discrete symmetric triangular kernel is also available in the [Ake](#) R package (Wan-souwé et al., 2016).

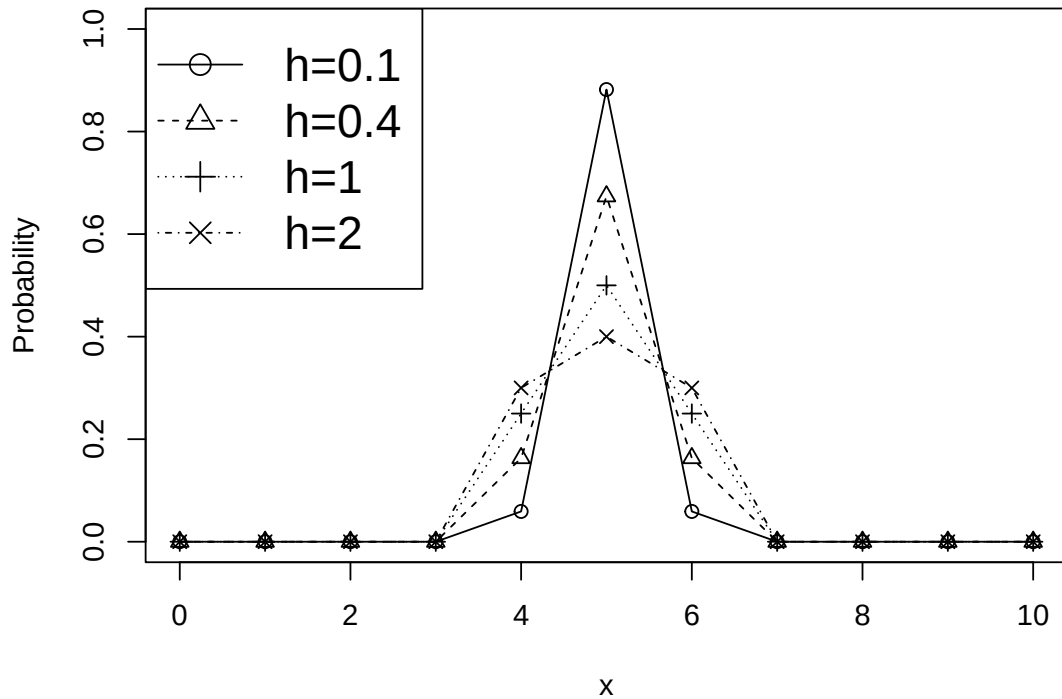


Figure 2: Distribution of discrete symmetric triangular kernel ($a = 1$) at the target point $x = 5$ for various bandwidth parameters h .

Other discrete kernels are available in the literature such as the Habbema kernel (Tutz and Pritscher, 1996), which is implemented in the **NPHazardRate** R package (Bagkavos, 2018) for estimating discrete data of time failure.

Mean Summed Squared Error

For $x \in \mathcal{S}$ and $h > 0$, the global performance of the estimator $\hat{f}_{K,h}$ of pmf f can be assessed by calculating its Mean Summed Squared Error (MSSE) given by

$$\text{MSSE}(\hat{f}_{K,h}(x)) = \frac{1}{n} \sum_{x \in \mathcal{S}} f(x) \sum_{y \in \mathcal{S}_x} K_{x,h}^2(y) + \sum_{x \in \mathcal{S}} \left(\frac{f^{(2)}(x)}{2} \text{Var}(\mathcal{K}_{x,h}) \right)^2 + o\left(\frac{1}{n} + h^2\right) \quad (4)$$

where the finite difference $f^{(2)}$ of order 2 of f is such that $f^{(k)}(x) = \{f^{(k-1)}(x)\}^{(1)}$ with

$$f^{(1)}(x) = \begin{cases} (f(x+1) - f(x-1)) / 2, & \text{if } x \in \mathbb{N} \setminus \{0\}, \\ f(1) - f(0), & \text{if } x = 0 \end{cases}$$

and

$$f^{(2)}(x) = \begin{cases} (f(x+2) - 2f(x) + f(x-2)) / 4, & \text{if } x \in \mathbb{N} \setminus \{0,1\}, \\ (f(3) - 3f(1) + f(0)) / 4, & \text{if } x = 1, \\ (f(2) - 2f(1) + f(0)) / 2 & \text{if } x = 0. \end{cases}$$

At the end of Section 3 we provide detailed examples of the expression of MSSE of $\hat{f}_{K,h}$ of the pmf f as a function of the discrete symmetric kernel considered.

2.2 Bandwidth selection

For $x \in \mathcal{S}$ and $h > 0$, the selection of the bandwidth parameter to be used with a given kernel $K_{x,h}$ was performed by the cross-validation method in **kernopt** package. Nevertheless, other procedures are available in the literature such as Bayesian bandwidth selection. We

determined the bandwidth h_{CV} by minimizing in h the cross-validation function

$$CV(h) = \sum_{x \in \mathcal{S}} \left(\hat{f}_{K,h}(x) \right)^2 - \frac{2}{n} \sum_{i=1}^n \hat{f}_{K,h,-i}(X_i),$$

where $\hat{f}_{K,h,-i}$ is calculated as $\hat{f}_{K,h}$ but by excluding the observation X_i . The cross-validation procedure was implemented through the function `cv_bandwidth` for the discrete kernels considered.

3 Discrete optimal symmetric kernel

For $x \in \mathcal{S}$ and $h > 0$, to determine a discrete symmetric “optimal” associated kernel $K_{x,h}^{opt}(\cdot) \geq 0$, we minimize the sum $\sum_{y \in \mathcal{S}_x} K_{x,h}^2(y)$ from the expression of MSSE in (4). That consists in a minimization problem under the following constraints (see conditions in (3)):

$$C_1 : \sum_{y \in \mathcal{S}_x} K_{x,h}(y) = 1, C_2 : \sum_{y \in \mathcal{S}_x} y K_{x,h}(y) = x \text{ and } C_3 : \sum_{y \in \mathcal{S}_x} y^2 K_{x,h}(y) = \tau^2,$$

which can be solved by using the method of Lagrange multiplier. A discrete expression of the Lagrange function is given by

$$\mathcal{L}(K) = \sum_{y \in \mathcal{S}_x} \left(K_{x,h}^2(y) + \lambda_1 K_{x,h}(y) + \lambda_2 y K_{x,h}(y) + \lambda_3 y^2 K_{x,h}(y) \right) + \beta,$$

where the constant β does not depend on $K_{x,h}$ and λ_i for $i \in \{1, 2, 3\}$, are the Lagrange multipliers corresponding to constraints C_1 - C_3 .

The discrete symmetric “optimal” kernel has the general form

$$K_{x,h}^{opt}(y, \lambda_1, \lambda_2, \lambda_3) = -\frac{1}{2}(\lambda_1 + \lambda_2 y + \lambda_3 y^2).$$

Without loss of generality, we investigate discrete symmetric “optimal” kernels $K_{k;x,h}^{opt}$ on the support $\mathcal{S}_x = \{x - k, \dots, x, \dots, x + k\}$, with $k \geq 1$ a fixed integer. We obtain the following result when solving the minimisation problem described above.

Theorem

For $x \in \mathcal{S}$ and a fixed integer $k \geq 1$, the discrete symmetric “optimal” kernel on $\mathcal{S}_x = \{x - k, \dots, x, \dots, x + k\}$ that minimises the mean integrate squared error of the estimator $\hat{f}_{K,h}$ in (1) is given by

$$K_{k;x,h}^{opt}(y) = \lambda \left(\frac{3k^2 + 3k - 1}{5} - (x - y)^2 \right) + \frac{h}{2k + 1}, y \in \mathcal{S}_x,$$

where $\lambda = 15(1 - h) / ((2k + 1)(4k^2 + 4k - 3)) > 0$ and $3/5(1 - 1/k) < h < 1$.

The bandwidth parameter h of the “optimal” kernels $K_{k;x,h}^{opt}$ is bounded with a lower bound $h_{lower} = 3/5(1 - 1/k) \geq 0$, which particularly depends on $k \geq 1$. The distribution of that kernel can be plotted as follows in the package (Figure 3):

```
x <- 5
z <- 0:10
h <- c(0.1, 0.4, 0.7, 0.9)
k <- 1
```

```
K_opt <- matrix(
  data = 0,
```

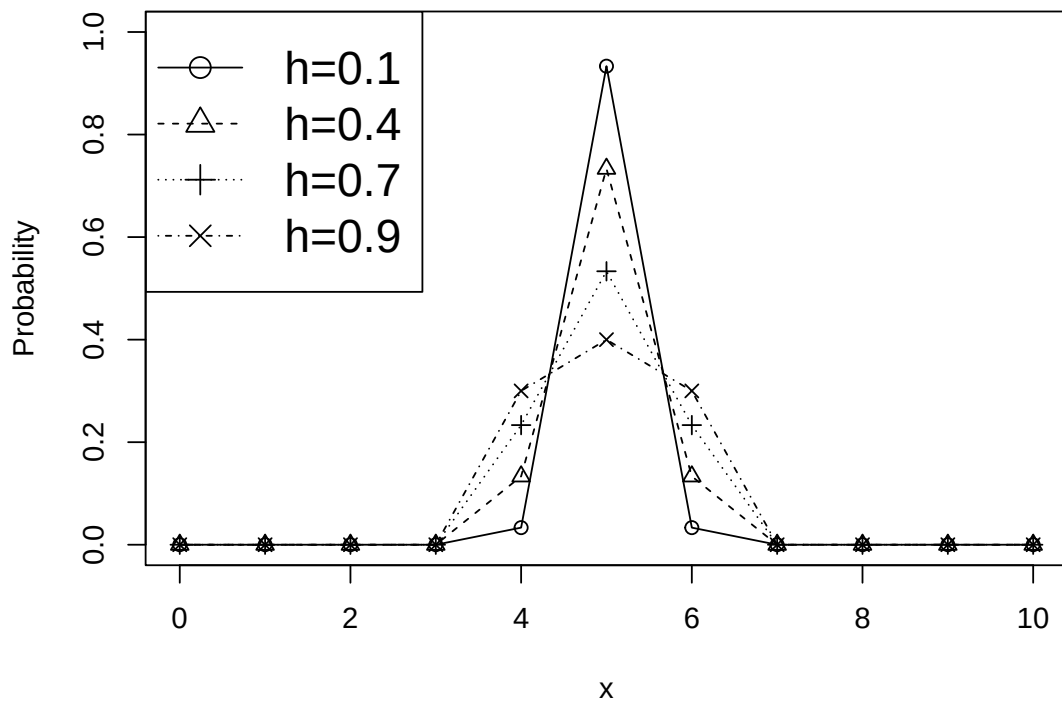


Figure 3: Distribution of discrete symmetric “optimal” kernel ($k = 1$) at the target point $x = 5$ for various bandwidth parameters h .

```

nrow = length(z),
ncol = length(h)
)

for (i in 1:length(h))
{
  K_opt[, i] <- discrete_kernel(kernel = "optimal", x, z, h[i], k)
}
plot(
  z,
  K_opt[, 1],
  xlab = "x",
  ylab = "Probability",
  ylim = c(0, 1),
  pch = 1
)
lines(z, K_opt[, 1], lty = 1)
for (i in 2:length(h))
{
  points(z, K_opt[, i], xlab = "z", pch = i)
  lines(z, K_opt[, i], lty = i)
}
legend(
  "topleft",
  c("h=0.1", "h=0.4", "h=0.7", "h=0.9"),
  lty = 1:4,
  pch = 1:4,
  cex = 1.6
)

```

We can formulate the following proposition on the limit distributions of $K_{k;x,h}^{opt}$ as a function of h .

Proposition 2

For $x \in \mathcal{S}$, $h \in (0, 1)$ and integer $k \geq 1$, we consider the discrete symmetric “optimal” kernel $K_{k,x,h}^{opt}(\cdot)$ with the support $\mathcal{S}_x = \{x - k, \dots, x, \dots, x + k\}$.

- i. When $h \rightarrow 1$, the limit “optimal” kernel $K_{k,x,1}^{opt}$ follows the discrete uniform distribution on the support $\mathcal{S}_x$.
- ii. When $h \rightarrow 0$, the limit “optimal” kernel $K_{k,x,0}^{opt}$ follows the Dirac distribution on the support $\mathcal{S}_x$.

MISE of estimators as a function of discrete symmetric kernels. For the three discrete symmetric associated kernels considered, the MSSE of $\hat{f}_{K,h}$ in (4) can be also given by

$$\begin{aligned} \text{MSSE}(\hat{f}_{K,h}(x)) &= \frac{1}{n}(\Pr(\mathcal{K}_{x,h} = x))^2 C_1 + \frac{1}{4}(\text{Var}(\mathcal{K}_{x,h}))^2 C_2 + o\left(\frac{1}{n} + h^2\right) \\ &= \text{AMSSE}(\hat{f}_{K,h}(x)) + o\left(\frac{1}{n} + h^2\right), \end{aligned}$$

with $C_1 = \sum_{x \in \mathcal{S}} f(x)(1 - f(x)) > 0$ and $C_2 = \sum_{x \in \mathcal{S}} (f^{(2)}(x))^2 > 0$. For instance, the AMSSE of the estimator using the discrete symmetric “optimal” kernel is given by

$$\text{AMSSE}(\hat{f}_{K^{opt},h}(x)) = \frac{1}{n} \frac{(3(3k^2 + 3k - 1) - 5hk(k + 1))^2}{(2k + 1)^2(4k^2 + 4k - 3)^2} C_1 + \frac{h^2 k^2 (k + 1)^2}{36} C_2,$$

with $\lambda = 15(1 - h)/((2k + 1)(4k^2 + 4k - 3))$. Likewise, the AMSSE is given as follows, when considering the estimator using the discrete symmetric triangular kernel:

$$\text{AMSSE}(\hat{f}_{K^{Triang},h}(x)) = \frac{1}{n} (1 - 2hA(p))^2 C_1 + \frac{1}{4} h^2 V^2(p) C_2$$

and the estimator using the discrete Epanechnikov kernel:

$$\text{AMSSE}(\hat{f}_{K^{Epan},h}(x)) = \frac{1}{n} \frac{9h^2}{(4h^2 - 1)^2} C_1 + \frac{1}{100} (h^2 - 1)^2 C_2.$$

When considering discrete symmetric “optimal” and triangular kernels, an analytical expression of $h > 0$ that minimizes AMSSE of $\hat{f}_{K,h}$ could be found, unlike Epanechnikov kernel for which only a numeric approximation was available.

4 Package presentation on simulated data

We illustrate the **kernopt** package by applying it to simulated data. [Senga Kiessé and Durrieu \(2024\)](#) performed a simulation study using different sample sizes n from various pmfs f , i.e., a Poisson distribution, a geometric distribution and a negative binomial distribution. The code below presents simulations for estimating count data sets of size $n = 100$ that follow the Poisson distribution $\mathcal{P}(\lambda = 2)$ (Figure 4). The kernel estimator was applied with each of three discrete symmetric kernels presented above in comparison with the count binomial asymmetric kernel that follows the binomial distribution $\mathcal{B}(x + 1, (x + h)/(x + 1))$ with support $\mathcal{S}_x = \{0, 1, \dots, x + 1\}$, for $x \in \mathcal{S} \subseteq \mathbb{N}$ and $h \in (0, 1]$ as seen in [Kokonendji and Senga Kiessé \(2011\)](#). The cross-validation method was used to select the bandwidth parameter h_{cv} .

For a fixed kernel K and a bandwidth parameter $h = h_{cv} > 0$, the performance of the nonparametric kernel estimator $\hat{f}_{K,h_{cv}}$ was compared to the true pmf f of simulated data by

using the Summed Squared Error (SSE) given by

$$\text{SSE}(h_{cv}) = \sum_{x \in \mathbb{N}} (\hat{f}_{K,h_{cv}}(x) - f(x))^2.$$

In addition, we can consider a goodness-of-fit test derived from the chi-squared statistic to test the null hypothesis H_0 : “the true pmf f is distributed according to the pmf P_0 ”, for fixed $\alpha = 0.05$ for example. As the P_0 is unknown, we replace P_0 by a kernel estimator $\hat{f}_{K,h}$ for a fixed kernel K and a bandwidth parameter $h = h_{cv} > 0$. The statistic can be given by

$$\chi_0^2 = \sum_{x \in \{0,1,\dots,N_0\}} \frac{(n\hat{f}_{K,h_{cv}}(x) - n_i)^2}{n\hat{f}_{K,h_{cv}}(x)},$$

where n_i is the number of observations in each class $x \in \{0,1,\dots,N_0\}$. Since the distribution of the statistics test under H_0 is unknown, we can use the bootstrap procedure to determine the distribution and the p -value. The Kolmogorov-Smirnov test can also be used for discrete data using the discrete kernel estimator $\hat{f}_{K,h_{cv}}$ of the pmf f .

```
# Simulated data
#-----
set.seed(18)
lambda <- 2
x <- 0:10
f <- dpois(x, lambda)
n <- 100
y <- sort(rpois(n, lambda))
data_sim <- as.data.frame(table(y))
f_sim <- c(f[1], f[2], f[3], f[4], f[5], sum(f[6:8])) # classes for chi-2 test

# Estimation with optimal kernel
#-----
k <- 1
H <- seq((max(y) - min(y)) / 200, (max(y) - min(y)) / 2, length.out = 50)
hcv_opt_k1 <- cv_bandwidth(kernel = "optimal", y, H, k)
Fn_opt_k1 <- estim_kernel(kernel = "optimal", x, hcv_opt_k1, y, k)
SSE_opt_k1 <- sum((Fn_opt_k1 - f)^2)

# Estimation with triangular kernel
#-----
a <- 1
hcv_trg_a1 <- cv_bandwidth(kernel = "triang", y, H, a)
Fn_triang_a1 <- estim_kernel(kernel = "triang", x, hcv_trg_a1, y, a)
SSE_triang_a1 <- sum((Fn_triang_a1 - f)^2)

# Estimation with Epanechnikov kernel
#-----
H <- seq(2, 10, 1)
hcv_epanech <- cv_bandwidth(kernel = "epanech", y, H, k = NULL)
Fn_epanech <- estim_kernel(kernel = "epanech", x, hcv_epanech, y, k = NULL)
SSE_epanech <- sum((Fn_epanech - f)^2)

# Estimation with Binomial kernel
#-----
H <- seq((max(y) - min(y)) / 500, 1, length.out = 50)
hcv_bin <- cv_bandwidth(kernel = "binomial", y, H)
Fn_bino <- estim_kernel(kernel = "binomial", x, hcv_bin, y, k = NULL)
```

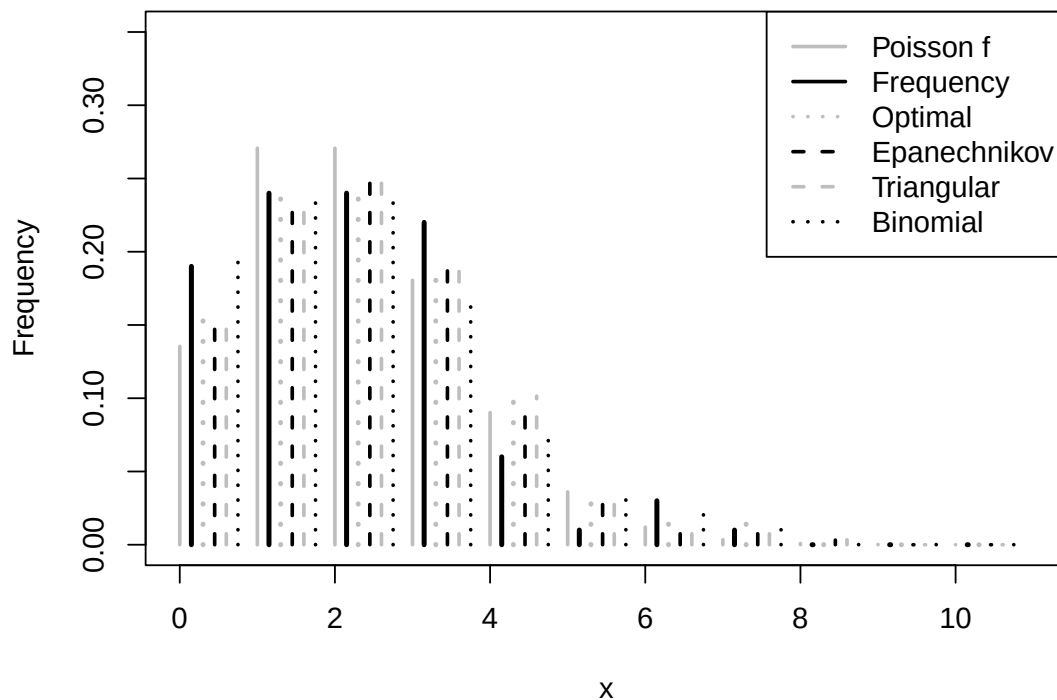



Figure 4: Barplot of the frequency estimation of a count data set of sample size $n = 100$ (black lines) simulated from a Poisson distribution $f := \mathcal{P}(\lambda = 2)$ (grey lines), compared with estimations using discrete kernels (dotted lines). Summed Squared Error (SSE) was also calculated resulting in the following results as a function of the kernel used: Optimal $k=1$, $\text{SSE}=0.0022$ ($h_{cv} = 0.95$); Triangular $a=1$, $\text{SSE}=0.0022$ ($h_{cv} = 3.22$); Epanechnikov, $\text{SSE}=0.0023$ ($h_{cv} = 2$); Binomial, $\text{SSE}=0.0069$ ($h_{cv} = 0.11$); Frequency, $\text{SSE}=0.0084$.

```
SSE_bino <- sum((Fn_bino - f)^2)

# Frequency estimation
#-----
freq_sim <- as.numeric(as.character(data_sim[, 2])) # count data
f0_sim <- c(freq_sim / sum(freq_sim), rep(0, length(x) - length(freq_sim))) # frequency
SSE_0_sim <- sum((f0_sim - f)^2)

# Graph
#-----
plot(x, f,
     xlab = "x", ylab = "Frequency", xlim = c(0, 11), ylim = c(0, 0.35), type = "h",
     lwd = 2, col = "grey", main = "")
)
points(x + 0.15, f0_sim, type = "h", lty = 1, lwd = 2.75)
points(x + 0.3, Fn_opt_k1, type = "h", lty = 3, col = "grey", lwd = 2.75)
points(x + 0.45, Fn_epanech, type = "h", lty = 2, lwd = 2)
points(x + 0.6, Fn_triangular_a1, type = "h", lty = 2, col = "grey", lwd = 2)
points(x + 0.75, Fn_bino, type = "h", lty = 3, lwd = 2)
legend("topright", c("Poisson f", "Frequency", "Optimal", "Epanechnikov", "Triangular", "Binomial"),
     lty = c(1, 1, 3, 2, 2, 3), lwd = 2, col = c("grey", "black", "grey", "black", "grey", "black"), inset =
)
```

The choice of the bandwidth parameter h is crucial, as usual in discrete and continuous nonparametric kernel estimations. Nevertheless, the quality of the fit is more sensitive to the kernel chosen for discrete kernel estimates K than for continuous kernel estimates.

Broader comparisons of the discrete symmetric “optimal” kernel with discrete symmetric and asymmetric kernels, and the frequency estimator, were carried out on $N = 100$ replicates

of different samples sizes $n \in \{25, 100, 250, 500\}$ (i.e., small, medium and large) in [Senga Kiessé and Durrieu \(2024\)](#), to strengthen evidence for the advantages of the “optimal” kernel, in particular for the small or moderate sample sizes $n \in \{25, 100\}$.

5 Application on real data

SIMTAP project (<https://www.simtap.eu>) aimed to develop sustainable aquaculture production system that, in particular, contributes to reduce fish feed inputs and resources consumption. We used data from an experiment in which gilthead seabream (*sparus aurata*) were stocked in 1.6 m^3 tanks at a density of $1.5 \text{ kg} \cdot \text{m}^{-3}$. Fish were reared for 46 days in a single recirculating aquaculture system composed of three rearing tanks. At the beginning of the experiment, a number $n = 200$ of the fish were individually weighed (dg), and their length at the caudal fork (mm) was measured. The nonparametric kernel estimator was applied to provide smoothed count distributions of weight and length of fish measured. As the true pmf f of real data was unknown, the performance of the nonparametric kernel estimator $\hat{f}_{K,h}$ was compared to that of the frequency estimator f_0 by using SSE previously defined in simulations. In addition, the cross-validation method was used to select the bandwidth parameter.

The following code illustrates this application on real experimental fish data with the [kernopt](#) package. The corresponding data are included in the package as an example data set and can be load through the *utils* library `data()` function as “fish_data” data set, e.g., `data('fish_data', package = 'kernopt')`. It also uses the [kernopt](#) `cv_bandwidth()` function to compute the estimate an optimal bandwidth parameter from the cross-validation procedure and then the `estim_kernel()` function in order to calculate the corresponding SSE. Empirical frequencies and discrete kernel estimates of count data of weight of fish were graphically compared (Figure 5).

```
#### Data
data("fish_data", package = "kernopt")
y <- fish_data$weight
data1 <- as.data.frame(table(y))
x <- as.numeric(as.character(data1[, 1]))
freq_weight <- as.numeric(as.character(data1[, 2]))
f0 <- freq_weight / sum(freq_weight)
n <- sum(freq_weight)

#### Optimal kernel
H <- seq((max(y) - min(y)) / 200, (max(y) - min(y)) / 2, length.out = 50)
k <- 1
hcv_opt_k1 <- cv_bandwidth(kernel = "optimal", y, H, k = 1)
fn_opt_k1 <- estim_kernel(kernel = "optimal", x, hcv_opt_k1, y, k = 1)
SSE_opt_k1 <- sum((fn_opt_k1 - f0)^2)

#### Epanechnikov
H <- seq(2, 10, 1)
hcv_epanech <- cv_bandwidth(kernel = "epanech", y, H)
fn_epanech <- estim_kernel(kernel = "epanech", x, hcv_epanech, y, k = NULL)
SSE_epanech <- sum((fn_epanech - f0)^2)

# Graph
par(mfrow = c(1, 2))
plot(x, f0,
     col = "black", axes = F, lwd = 3, ylab = "", xlab = "Weight (dg)",
     ylim = c(0, 0.06), xlim = c(41, 132), type = "h",
     main = "(a)", cex.axis = 1, cex.lab = 1
```

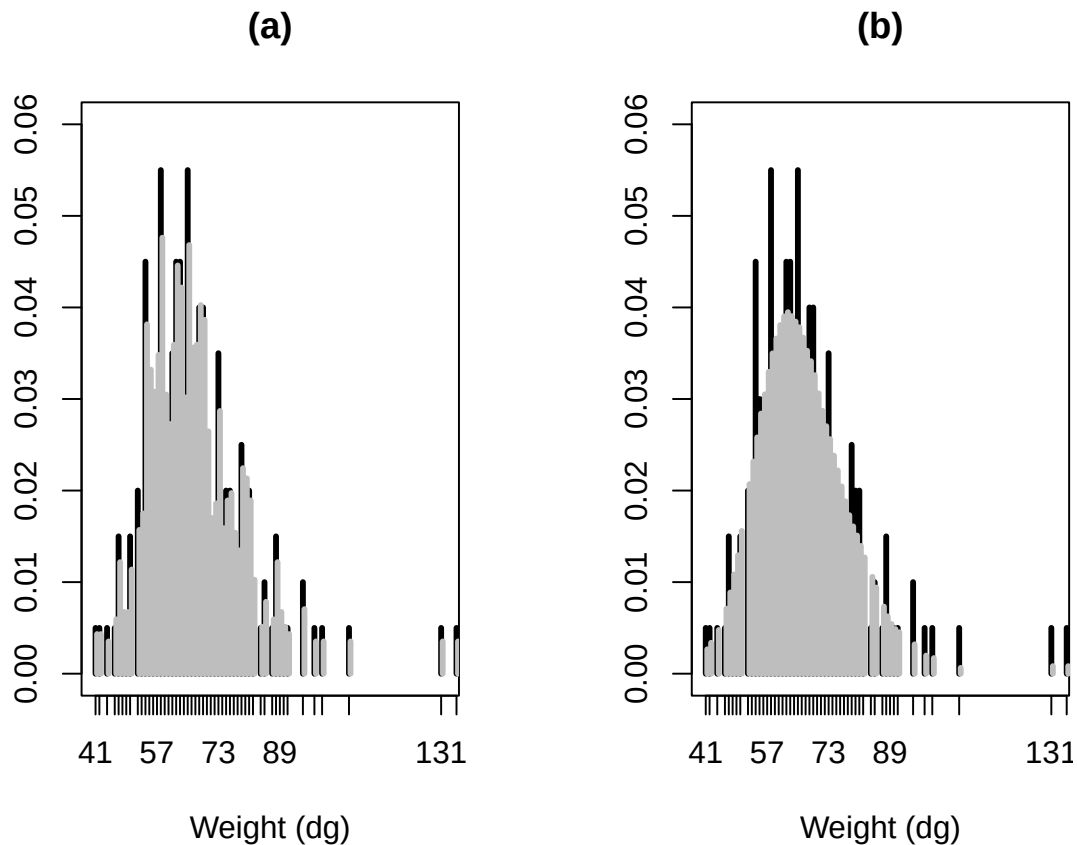


Figure 5: Density estimations (gray lines) of count distributions of weight (dg) (black lines) of fish by using optimal and Epanechnikov discrete kernels with bandwidth parameter (h_{cv}) from the cross-validation procedure. Summed Squared Error (SSE) was also calculated. (a) Optimal $k=1$: $SSE=0.0006$ ($h_{cv} = 0.47$); (b) Epanechnikov: $SSE=0.0033$ ($h_{cv} = 9$).

```
)
axis(1, at = x, cex.axis = 1, cex.lab = 1)
axis(2)
box()
points(x + 0.4, fn_opt_k1, lwd = 3, col = "grey", lty = 1, type = "h")
plot(x, f0,
      col = "black", axes = F, lwd = 3, xlab = "Weight (dg)", ylab = "",
      ylim = c(0, 0.06), xlim = c(41, 132), type = "h",
      main = "(b)", cex.axis = 1, cex.lab = 1
)
axis(1, at = x, cex.axis = 1, cex.lab = 1)
axis(2)
box()
points(x + 0.4, fn_epanech, lwd = 3, col = "grey", lty = 1, type = "h")
```

5.1 Acknowledgments

We thank the [SIMTAP](#) project (Self-sufficient Integrated Multi-Trophic AquaPonic systems) supported by the PRIMA program (Partnership for Research and Innovation in the Mediterranean Area) under grant (18110-2) for providing the data.

References

- D. Bagkavos. *NPHazardRate: Nonparametric Hazard Rate Estimation*, 2018. URL <https://CRAN.R-project.org/package=NPHazardRate>. R package version 0.1. [p319]
- S. X. Chen. A beta kernel estimator for density functions. *Computational Statistics and Data Analysis*, 31:131–145, 1999. doi: 10.1016/S0167-9473(99)00010-9. [p315]
- C.-Y. Chu, D. J. Henderson, and C. F. Parmeter. On discrete Epanechnikov kernel functions. *Computational statistics & data analysis*, 116:79–105, 2017. doi: 10.1016/j.csda.2017.07.003. [p315, 316]
- T. Duong. ks: Kernel density estimation and kernel discriminant analysis for multivariate data in R. *Journal of statistical software*, 21:1–16, 2007. doi: 10.18637/jss.v021.i07. [p315]
- V. A. Epanechnikov. Non-parametric estimation of a multivariate probability density. *Theory of Probability and its Applications*, 14:153–158, 1969. doi: 10.1137/1114019. [p315]
- A. Huang, L. Sippel, and T. Fung. Consistent second-order discrete kernel smoothing using dispersed Conway–Maxwell–Poisson kernels. *Computational Statistics*, 37(2):551–563, 2022. doi: 10.1007/s00180-021-01144-w. [p315]
- C. Kokonendji, T. Senga Kiessé, and S. S. Zocchi. Discrete triangular distributions and non-parametric estimation for probability mass function. *Journal of Nonparametric Statistics*, 19(6-8):241–254, 2007. doi: 10.1080/10485250701733747. [p317]
- C. C. Kokonendji and T. Senga Kiessé. Discrete associated kernel method and extensions. *Statistical Methodology*, 8:497–516, 2011. doi: 10.1016/j.stamet.2011.07.002. [p315, 322]
- C. C. Kokonendji, S. M. Somé, Y. Esstafa, and M. Bourguignon. On Underdispersed Count Kernels for Smoothing Probability Mass Functions. *Stats*, 6(4):1226–1240, 2023. doi: 10.3390/stats6040076. [p315]
- E. Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 09 1962. URL <http://www.jstor.org/stable/2237880>. [p315]
- J. S. Racine, Q. Li, and K. X. Yan. Kernel smoothed probability mass functions for ordered datatypes. *Journal of Nonparametric Statistics*, 32(3):563–586, 2020. doi: 10.1080/10485252.2020.1759595. [p315]
- M. Rosenblatt. Remarks on Some Nonparametric Estimates of a Density Function. *The Annals of Mathematical Statistics*, 27(3):832–837, 09 1956. URL <http://www.jstor.org/stable/2237390>. [p315]
- O. Scaillet. Density estimation using inverse and reciprocal inverse Gaussian kernels. *Nonparametric statistics*, 16(1-2):217–226, 2004. doi: 10.1080/10485250310001624819. [p315]
- T. Senga Kiessé. On finite sample properties of nonparametric discrete asymmetric kernel estimators. *Statistics*, 51(5):1046–1060, 2017. doi: 10.1080/02331888.2017.1293060. [p316]
- T. Senga Kiessé and G. Durrieu. On a discrete symmetric optimal associated kernel for estimating count data distributions. *Statistics & Probability Letters*, 208:110078, 2024. ISSN 0167-7152. doi: 10.1016/j.spl.2024.110078. [p315, 316, 322, 325]
- G. Tutz and L. Pritscher. Nonparametric estimation of discrete hazard functions. *Lifetime Data Analysis*, 2(3):291–308, 1996. doi: 10.1007/BF00128979. [p319]
- M. Wand. *KernSmooth: Functions for Kernel Smoothing Supporting Wand & Jones (1995)*, 2025. URL <https://CRAN.R-project.org/package=KernSmooth>. R package version 2.23-26. [p315]
- M. P. Wand and M. C. Jones. *Kernel smoothing*. CRC press, 1995. doi: 10.1201/b14876. [p315]

W. E. Wansouwé, S. M. Somé, and C. C. Kokonendji. Ake: An R Package for Discrete and Continuous Associated Kernel Estimations. *The R Journal*, 8(2):258–276, 2016. doi: 10.32614/RJ-2016-045. [p315, 318]

Gilles Durrieu

Univ Bretagne Sud, LMBA CNRS UMR 6205

F-56000 Vannes, France

ORCID: [0000-0003-0375-912X](https://orcid.org/0000-0003-0375-912X)

gilles.durrieu@univ-ubs.fr

Thomas Fillon

Univ Bretagne Sud, LMBA CNRS UMR 6205 & IRISA CNRS UMR 6074

F-56000 Vannes, France

ORCID: [0000-0003-1207-8414](https://orcid.org/0000-0003-1207-8414)

thomas.fillon@univ-ubs.fr

Tristan Senga Kiessé

INRAE, Institut Agro & UMR 1069 SAS

F-35000 Rennes, France

ORCID: [0000-0003-2710-5825](https://orcid.org/0000-0003-2710-5825)

tristan.senga-kiesse@inrae.fr

Clustering Binary Data Optimizing the Exclusiveness of Features

by J. Orestes Cerdeira and Tiago Monteiro-Henriques

Abstract A problem that has been studied for a long time in Vegetation Science consists in finding distinct groups among a collection of vegetation samples (*relevés*). Recently a criterion has been proposed to capture the patterns of differential species among the groups from any arbitrary M-cluster of *relevés*. The criterion optimization is quite complex (NP-hard) as it implies searching a huge set of possible combinations of *relevés* into M groups. In this paper we give an integer linear programming formulation to optimize the criterion for M=2, that can be used to solve moderately-sized data problems; we also describe a greedy randomized adaptive search procedure and a simulated annealing algorithm for arbitrary M. We combined these different approaches and prepared a collection of software functions, which are available in the open-source R package *diffval*. We illustrate the use of such functions using real-world data.

1 Introduction

Vegetation classification is a complex problem that has occupied researchers for decades. This is traditionally done by manipulating matrices of vegetation samples (*relevés*), as described in detail by [Mueller-Dombois and Ellenberg \(1974\)](#). The main purpose of this manipulation is clustering, i.e., the researcher seeks to partition the collection of *relevés* in groups (subsets of *relevés*), which can be distinguished floristically (by subsets of the present taxa). Such manipulation is even more ambitious and seeks to find, simultaneously, an optimal number of clusters to partition the *relevé* set. This process is usually called tabulation, tabular classification, or table sorting. Manual tabulation is a particularly tedious process, being impractical for large data sets.

With the advent of computers and software, several authors have proposed a plethora of different approaches to vegetation classification. In essence, all of them aim at clustering groups of *relevés* based on common taxa that are expected to differentiate those groups, ultimately pursuing a typical block-structured sorted table. The vast majority of such approaches does this heuristically, relying on numerical clustering methods based on pairwise (dis)similarity measures. Direct optimization, even if considered a natural way of approaching the issue, is hindered by the huge number of possible combinations of *relevé* groups ([Peet and Roberts, 2013](#)). However, there are some proposals trying to find directly the desired block-structured tables, e.g., Two-Way Indicator Species Analysis–TWINSPAN ([Hill, 1979](#)), analysis of concentration ([Feoli and Orlóci, 1979](#)), chi-squared optimization ([Podani and Feoli, 1991](#)), the ESPRESSO software ([Bruehlheide and Flintrop, 1994](#)), the COCKTAIL algorithm ([Bruehlheide, 2000](#)), and modified-TWINSPAN ([Roleček et al., 2009](#)). With the same objective [Monteiro-Henriques \(2025\)](#) recently proposed an optimization approach to tabulation, devising a new index specifically for that purpose: the DiffVal index. In [Monteiro-Henriques \(2025\)](#) comparisons to other clustering methods, using both synthetic data sets with known group structure, and real-world data are provided.

In general, the approaches mentioned for classifying vegetation data fall within the clustering or partitioning ([Kaufman and Rousseeuw, 1990](#)) and biclustering ([Castanho et al., 2024](#)) frameworks. Within these broader disciplines, what are here called *relevés* are usually referred to as *objects* or *samples*, and *species* as *features*, *variables*, or *attributes*.

Given a vegetation table and a partition of the *relevés*, the DiffVal index quantifies the ability of a taxon in differentiating those groups, taking into account the following characteristics:

C1) The greater the relative frequency in the group (or groups) where the taxon occurs, the higher is the corresponding DiffVal. This is a commonly sought feature among diagnostic taxa (see, e.g., the IndVal index of [Dufrêne and Legendre, 1997](#)), as frequent taxa in the group have great bioindicator and practical value in recognizing communities in the field.

C2) The greater the exclusiveness of a taxon to a group (or groups) the better, i.e., the greater the full absence from other groups, the higher is the corresponding DiffVal. This relates to fidelity (*sensu* [Mueller-Dombois and Ellenberg, 1974](#)).

C3) Conversely, a taxon occurring in all groups should present a zero DiffVal, as it is useless in distinguishing the groups with its presence.

Balancing those characteristics (C1 to C3) in a single index is challenging, however, the DiffVal index is able to incorporate them in a simple combinatorial way (i.e. counting the presences, the absences and the group sizes). Eventually, having the DiffVals for all the taxa in a vegetation table, we

can compute their sum, which is a natural criterion to portray the overall pattern of present differential taxa. Finding the desired block-structured table will be possible then, if we are able to find relevé partitions that maximize this sum.

Mathematically, this contextualizes as follows. (For an ecological interpretation see [Monteiro-Henriques \(2025\)](#), where a worked example of DiffVal calculation with a small hypothetical vegetation table is provided.) The presence of taxa in relevés is described as a, $m \times n$, matrix $A = [a_{ij}]$, with 0/1 entries

$$a_{ij} = \begin{cases} 1 & \text{if taxon } i \text{ occurs in relevé } j \\ 0 & \text{otherwise} \end{cases}$$

In what follows we assume that $\sum_{j=1}^n a_{ij} \geq 1$, i.e., any taxon occurs at least in one relevé.

A solution S_M for the vegetation classification in M groups is an M -partition of the set of columns (relevés) $C: G_1, \dots, G_M$ such that $\cup_{k=1}^M G_k = C$ and $G_k \cap G_l = \emptyset$, with $k \neq l = 1, \dots, M$.

The value of the DiffVal index for taxon i w.r.t. solution S_M is

$$\text{DiffVal}_i(S_M) = \sum_{k=1}^M \frac{\sum_{j=1}^n a_{ij} x_j^k}{\sum_{j=1}^n x_j^k} \frac{\sum_{l \neq k=1}^M \sum_{j=1}^n x_j^l g_i^l}{\sum_{l \neq k=1}^M \sum_{j=1}^n x_j^l} \frac{1}{M - \sum_{l=1}^M g_i^l} \quad (1)$$

where $x_j^k = 1$ ($x_j^k = 0$), if column j is (not) in group G_k of solution S_M and $g_i^k = 1$ ($g_i^k = 0$) if, in group G_k of S_M , (not) all entries $a_{ij} = 0$.

Thus, expressions $\sum_{j=1}^n a_{ij} x_j^k$ and $\sum_{j=1}^n x_j^k$ indicate the number of occurrences of taxon i in group G_k and the size of group G_k , respectively, and, therefore $\frac{\sum_{j=1}^n a_{ij} x_j^k}{\sum_{j=1}^n x_j^k}$ is the relative representation of taxon i in G_k , incorporating the characteristic C1, referred above, in the DiffVal index.

Expression $\frac{\sum_{l \neq k=1}^M \sum_{j=1}^n x_j^l g_i^l}{\sum_{l \neq k=1}^M \sum_{j=1}^n x_j^l}$ captures the absence of taxon i in all groups but G_k . The numerator $\sum_{l \neq k=1}^M \sum_{j=1}^n x_j^l g_i^l$ sums the sizes of all empty groups, while $\sum_{l \neq k=1}^M \sum_{j=1}^n x_j^l$ is the sum of the sizes of all groups except G_k . Thus, the ratio $\frac{\sum_{l \neq k=1}^M \sum_{j=1}^n x_j^l g_i^l}{\sum_{l \neq k=1}^M \sum_{j=1}^n x_j^l}$ is the proportion of emptiness of taxon i outside G_k group, incorporating characteristic C2 in the index. Note that if taxon i occurs in all groups, i.e., $g_i^k = 0$, for $k = 1, \dots, M$, then $\text{DiffVal}_i(S_M) = 0$ fulfilling characteristic C3.

Finally, $M - \sum_{l=1}^M g_i^l$ in the denominator of (1) is used to make the index $\text{DiffVal} \leq 1$.

Taking the sum of $\text{DiffVal}_i(S_M)$, for all taxon i , we get

$$\text{TotDiffVal}(S_M) = \sum_{i=1}^m \text{DiffVal}_i(S_M) \quad (2)$$

that quantifies the extent to which solution S_M is based on differential taxa, that we aim to maximize. Dividing $\sum_{i=1}^m \text{DiffVal}_i(S_M)$ by m also ensures a value bounded between 0 and 1, yet this is unnecessary for the maximization, thus it is disregarded here.

The following constraints ensure that variables x_j^k have the meaning given above (i.e., $x_j^k = 1$ if column j is in group G_k , and $x_j^k = 0$ otherwise) and that, in every solution that maximizes (2), variables g_i^k also have the meaning above (i.e., $g_i^k = 1$ if taxon i is absent from all relevés of group G_k , and $g_i^k = 0$, otherwise).

$$\sum_{k=1}^M x_j^k = 1, \quad j = 1, \dots, n \quad (3)$$

$$\sum_{j=1}^n x_j^k \geq 1, \quad k = 1, \dots, M \quad (4)$$

$$x_j^k \in \{0, 1\} \quad j = 1, \dots, n; k = 1, \dots, M \quad (5)$$

$$g_i^k \leq 1 - a_{ij} x_j^k, \quad j = 1, \dots, n; k = 1, \dots, M; i = 1, \dots, m \quad (6)$$

Constraints (5) define the range of variables x_j^k . Equation (3) assigns each column j to exactly one group, and inequality (4) ensures that every group has at least one column. Hence, variables x_j^k define an M -partition of the set of columns of matrix A .

Inequalities (6) define upper bounds on variable g_i^k : $g_i^k \leq 0$ when taxon i is present in some relevé j of group G_k (i.e., $a_{ij}x_j^k = 1$), and $g_i^k \leq 1$ when taxon i is absent from group G_k (i.e., $a_{ij}x_j^k = 0$, for all j). Clearly, the upper bounds will be attained in every M -partition that maximizes the value of TotDiffVal (2).

Maximizing (2) subject to (3)-(6) is an NP-hard problem (Garey and Johnson, 1979), for any fixed number of groups $M \geq 2$. We briefly sketch the proof. For the case $M = 2$, we reduce from the minimum graph bisection problem, which is known to be NP-hard (Garey et al., 1976). In that problem, the task is to divide the vertices of a graph into two equal parts while minimizing the number of edges crossing between them. This is closely related to our setting: a minimum bisection can be reformulated as finding a balanced 2-partition of the transposed incidence matrix of the graph that maximizes the TotDiffVal index. We then show that such balanced 2-partitions arise as optimal solutions of TotDiffVal maximization (for $M = 2$) on a suitably constructed larger matrix, without explicitly enforcing the balancing constraint. The full reduction is presented in the Appendix. Intuitively, the problem is computationally intractable because the number of possible partitions grows exponentially with the size of the data, and the balancing requirement further constrains the search. By induction on M , the NP-hardness extends to any fixed number of groups $M \geq 2$ (see Appendix). Hence, unless $P = NP$, no efficient algorithm exists to find an M -partition that maximizes TotDiffVal.

The rest of this paper is organized as follows. In Section 2 we address the problem of maximizing TotDiffVal in the case where $M = 2$, and give two integer linear programming formulations. In Sections 3 and 4 we present a greedy randomized adaptive search procedure and a simulated annealing algorithm to maximize the criterion TotDiffVal for arbitrary M , respectively. The approaches of Sections 2, 3 and 4 are combined in a number of software functions included in the R package `diffval` (Monteiro-Henriques and Cerdeira, 2025) which we describe in Section 5, illustrating their application on real-world data.

2 Case $M = 2$

When $M = 2$ the value of x_j^1 (x_j^2) determines the value of x_j^2 (x_j^1). We may therefore replace variables x_j^1 and x_j^2 by a unique variable x_j , where $x_j = 1$ indicates that column j is assigned to one of the two groups, say, e.g., group G_1 . Thus, $x_j = 0$ indicates that column j is assigned to group G_2 .

Constraints (3)-(6) may be rewritten as follows.

$$\sum_{j=1}^n x_j \geq 1 \quad (7)$$

$$\sum_{j=1}^n x_j \leq \left\lfloor \frac{n}{2} \right\rfloor \quad (8)$$

$$x_j \in \{0, 1\} \quad j = 1, \dots, n \quad (9)$$

$$g_i^1 \leq 1 - a_{ij}x_j, \quad j = 1, \dots, n; i = 1, \dots, m \quad (10)$$

$$g_i^2 \leq 1 - a_{ij}(1 - x_j), \quad j = 1, \dots, n; i = 1, \dots, m \quad (11)$$

Constraints (7)-(9) define a 2-partition of the set of columns: $x_j = 1$ ($x_j = 0$) if column j is assigned to group G_1 (G_2), while ensuring that none of the two groups is the empty set.

Inequalities (10) and (11) imply that $g_i^k \leq 0$ when taxon i occurs in some relevé j of group G_k , and that $g_i^k \leq 1$ when taxon i is absent from group G_k .

Expression (1), for $M = 2$, reads

$$\frac{\sum_{j=1}^n a_{ij}x_j}{\sum_{j=1}^n x_j} g_i^2 + \frac{\sum_{j=1}^n a_{ij}(1 - x_j)}{n - \sum_{j=1}^n x_j} g_i^1$$

Note that the term $(2 - (g_i^1 + g_i^2))^{-1}$ is not needed since (recall that we assumed $\sum_{j=1}^n a_{ij} \geq 1$) either $g_i^1 + g_i^2 \leq 1$, case where for every optimal solution we will have $g_i^1 + g_i^2 = 1$ (i.e., taxon i occurs in exactly one of the two groups) and therefore $(2 - (g_i^1 + g_i^2))^{-1} = 1$, or else $g_i^1, g_i^2 \leq 0$, case where for every optimal solution $g_i^1 = g_i^2 = 0$ (i.e., taxon i occurs in the two groups) and therefore

$$\frac{\sum_{j=1}^n a_{ij} x_j}{\sum_{j=1}^n x_j^1} g_i^2 + \frac{\sum_{j=1}^n a_{ij} x_j}{n - \sum_{j=1}^n x_j} g_i^1 = 0.$$

Taking into account that when $g_i^2 = 1$ ($g_i^1 = 1$) we have $\sum_{j=1}^n a_{ij} x_j = \sum_{j=1}^n a_{ij} (\sum_{j=1}^n a_{ij} (1 - x_j) = \sum_{j=1}^n a_{ij})$, if we let $\alpha_i = \sum_{j=1}^n a_{ij} \geq 1$, the formula above can be written as

$$\frac{\alpha_i}{\sum_{j=1}^n x_j} g_i^2 + \frac{\alpha_i}{n - \sum_{j=1}^n x_j} g_i^1$$

Hence, when $M = 2$ the problem of maximizing TotDiffVal (2) subject to (3)-(6) converts to maximizing

$$\sum_{i=1}^m \left(\frac{\alpha_i}{\sum_{j=1}^n x_j} g_i^2 + \frac{\alpha_i}{n - \sum_{j=1}^n x_j} g_i^1 \right) \quad (12)$$

subject to the linear constraints (7)-(11).

This is a fractional 0/1 programming problem (Borrero et al., 2017) for which we now give two linearization approaches.

2.1 Model 1

The linearization technique that we use here follows the procedure described in Borrero et al. (2017). If we let

$$y_i^1 = \frac{\alpha_i}{\sum_{j=1}^n x_j} g_i^2 \text{ and } y_i^2 = \frac{\alpha_i}{n - \sum_{j=1}^n x_j} g_i^1$$

expression (12) reads

$$\sum_{i=1}^m (y_i^1 + y_i^2) \quad (13)$$

The two equations above may be written as

$$\sum_{j=1}^n y_i^1 x_j = \alpha_i g_i^2 \text{ and } n y_i^2 - \sum_{j=1}^n y_i^2 x_j = \alpha_i g_i^1 \quad (14)$$

that can be linearized by letting $z_{ij}^1 = y_i^1 x_j$ and $z_{ij}^2 = y_i^2 x_j$, writing equations (14) as

$$\sum_{j=1}^n z_{ij}^1 = \alpha_i g_i^2, \quad i = 1, \dots, m \quad (15)$$

$$n y_i^2 - \sum_{j=1}^n z_{ij}^2 = \alpha_i g_i^1, \quad i = 1, \dots, m \quad (16)$$

and relating z_{ij}^k to y_i^k , $k = 1, 2$, and to x_j through the following inequalities

$$z_{ij}^1 \leq x_j, \quad j = 1, \dots, n; i = 1, \dots, m \quad (17)$$

$$z_{ij}^1 \leq y_i^1, \quad j = 1, \dots, n; i = 1, \dots, m \quad (18)$$

$$z_{ij}^1 \geq y_i^1 + x_j - 1, \quad j = 1, \dots, n; i = 1, \dots, m \quad (19)$$

$$z_{ij}^2 \leq x_j, \quad j = 1, \dots, n; i = 1, \dots, m \quad (20)$$

$$z_{ij}^2 \leq y_i^2, \quad j = 1, \dots, n; i = 1, \dots, m \quad (21)$$

$$z_{ij}^2 \geq y_i^2 + x_j - 1, \quad j = 1, \dots, n; i = 1, \dots, m \quad (22)$$

$$z_{ij}^1, z_{ij}^2 \geq 0, \quad j = 1, \dots, n; i = 1, \dots, m \quad (23)$$

Since x_j are 0/1 variables, inequalities (17), (23) and (18), (19) imply that $z_{ij}^1 = y_i^1 x_j$, and (15) becomes equivalent to the left equation in (14). A similar argument can be used to show that $z_{ij}^2 = y_i^2 x_j$

and (16) is equivalent to the right equation in (14).

Hence, the problem of maximizing (12) subject to (7)-(11) can be formulated as the mixed integer linear programming that consists in maximizing (13) subject to (7)-(11), (15)-(23). We call this Model 1.

2.2 Model 2

We call Model 2 the model that results from fixing the size of group G_1 , i.e., requiring that

$$\sum_{j=1}^n x_j = t \quad (24)$$

and writing the objective function as follows.

$$M2(t) = \sum_{i=1}^m \left(\frac{\alpha_i}{t} g_i^2 + \frac{\alpha_i}{n-t} g_i^1 \right) \quad (25)$$

In that way, the problem consists of finding, for $t = 1, \dots, \lfloor \frac{n}{2} \rfloor$, the maximum of $M2(t)$, subject to (24), (9)-(11).

In Section 5 we report some results on the computational performance of the two models.

3 A greedy randomized adaptive search procedure

The greedy algorithm for a combinatorial maximization problem starts with set $S = \emptyset$, and at each step chooses an element to add to S that most increases the objective function and that will not turn S infeasible. The algorithm stops when no more elements on these conditions exist.

The greedy randomized adaptive search procedure (GRASP, see [Feo and Resende, 1995](#); [Festa and Resende, 2018](#)) chooses the elements to add to current S with probabilities which are proportional to the values they add to the objective function.

Our GRASP starts by uniformly selecting M columns of matrix A (relevés), and by assigning each column to exactly one of the M groups. If $j_1, \dots, j_M$ are the indices of the selected columns, the incidence vector of the starting set S is $x_{j_1}^1 = \dots = x_{j_M}^M = 1$ and $x_j^k = 0$, for every $j \neq j_1, \dots, j_M$ and $k = 1, \dots, M$.

At each step a column is chosen to be added to a certain group of current S in the following way. For every index j of a column not in S , i.e., $x_j^k = 0$, $k = 1, \dots, M$, we define S_j^k to be the (partial) solution resulting from adding to group k of S the column of index j , i.e., the incidence vector x of S_j^k is the same as S , except that, for S_j^k , $x_j^k = 1$. We then compute $\text{TotDiffVal}(S_j^k) = \sum_{i=1}^m \text{DiffVal}_i(S_j^k)$, where $\text{DiffVal}_i(S_j^k)$ is given by expression (1), if $M - \sum_{l=1}^M g_i^l \neq 0$ and $\text{DiffVal}_i(S_j^k) = 0$, if $M - \sum_{l=1}^M g_i^l = 0$. The new current set S is randomly chosen with probabilities proportional to the values of $\text{TotDiffVal}(S_j^k)$, among the $p\%$ sets S_j^k having the largest values of $\text{TotDiffVal}(S_j^k)$.

The algorithm stops when current set S includes all columns of A , i.e., when equation (3) is satisfied. We call S the GRASP solution.

4 A simulated annealing algorithm

At each step, the simulated annealing (SA) algorithm ([Kirkpatrick et al., 1983](#); [Aarts et al., 1997](#)) randomly selects some *neighbour* solution S' of the current feasible solution S , and probabilistically decides whether to maintain S as current or to replace S by S' .

In our SA implementation the neighbourhood of a solution S is the set of all feasible solutions obtained by moving one column from one group to another group of S . At iteration i , a neighbour S' of current S is uniformly chosen and S' replaces the current solution S if $\Delta_i = \text{TotDiffVal}(S') - \text{TotDiffVal}(S) \geq 0$ or, with probability $\exp(\Delta_i / T_i)$, if $\Delta_i < 0$, where $T_i > 0$ is the *temperature* at iteration i . We start with a given initial temperature $T_1 \leq 1$, and gradually decrease the current value by $\alpha\%$ every nt iterations, so that in the last iteration the temperature reaches a given established value. The algorithm stops with the best solution found at any stage, after a given number *nit* of iterations.

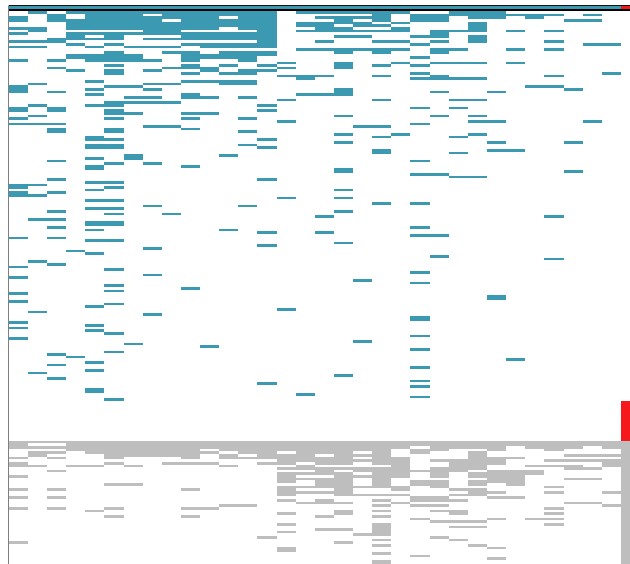


Figure 1: Image of the sorted phytosociological matrix showing the isolated relevé. Each coloured rectangle represents a presence in the matrix. Presences of taxa exclusive of group 1 are represented in blue, while those exclusive of group 2 are in red. Taxa shared by the two groups are represented in grey.

```
#> [1] 0.1513158
```

The isolated relevé stands out for the high number of taxa (62, while the mean number of taxa is 29.7), from which 15 taxa are exclusive to it. Using function `tabulation()` from [diffval](#), it is possible to visualize the sorted phytosociological table, given the obtained partition of its columns:

```
tabul <- tabulation(
  m_bin = taxus_bin,
  p = result$par,
  taxa_names = rownames(taxus_bin),
  plot_im = "normal",
  palette = "Zissou 1"
)
```

The `tabulation()` function reorders the columns of the original matrix by putting together the relevés belonging to the same group (as defined by the given partition). Additionally, it reorders the rows, by placing exclusive taxa towards the top of the table, while the taxa occurring in all groups of the partition are greyed out and placed towards the bottom of the matrix. Figure 1 shows the image generated by the `tabulation()` function using as input the 2-partition previously obtained using Gurobi. It is possible to distinguish the isolated relevé, showing its exclusive taxa in red.

For bigger matrices, where `optim_tdv_gurobi_k_2()` computation time might become prohibitive, heuristics such as `partition_tdv_grasp()` or `optim_tdv_simul_anne()`, or a combination of both, are recommended to find patterns of differential species.

Function `partition_tdv_grasp()` creates partitions of the data set using the greedy randomized adaptive search procedure described in Section 3. The input parameter `thr` (from 0 to 1) defines the $p\%$ sets defined in Section 3. For `thr = 1` the algorithm corresponds to the greedy algorithm. By default, `thr = 0.95`. Exemplifying with the *T. baccata* forests data set (using `set.seed()` only to ensure the reproducibility of the outputs):

```
set.seed(1)
partition_tdv_grasp(taxus_bin, 3)

#> [1] 3 2 3 1 2 2 2 2 2 1 1 2 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1

# [1] 3 2 3 1 2 2 2 2 2 1 1 2 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1
```

Function `optim_tdv_simul_anne()` implements the simulated annealing algorithm as presented in Section 4, searching for a k -partition (k , defined by the user), optimizing the TDV, i.e., searching

for a global maximum of TDV (by rearranging the relevés into k groups). As initial partition, the function can use any user-given partition (`p_initial`), a uniformly generated partition (`p_initial = "random"`) or (if `use_grasp = TRUE`) a partition generated by the `partition_tdv_grasp()` function.

The simulated annealing algorithm decreases the temperature according to a predefined cooling schedule, which can be controlled by the user by entering the following parameters: the initial temperature (`t_inic`), the final temperature (`t_final`), the fraction of temperature dropping (`alpha`) and the number of iterations `n_iter`. Given these inputs, the cooling schedule is obtained calculating the number of times, say `nt`, that the temperature has to drop (by multiplying the current temperature by $1 - \alpha$) in order to approximate `t_final` starting from `t_inic`. The number of times that the temperature has to drop (`nt`) is calculated by the expression: $\text{floor}(\log(t_{\text{final}} / t_{\text{inic}}) / \log(1 - \alpha))$. Finally, these decreasing stages are scattered through the desired iterations `n_iter` homogeneously, by calculating the indices of the iterations that will experience a decrease in temperature using $\text{floor}(n_{\text{iter}} / nt * (1:nt))$.

To illustrate the use of `optim_tdv_simul_anne()` and in order to reproduce the example in the original article of [Portela-Pereira et al. \(2021\)](#), we start by removing taxa occurring in only one relevé:

```
taxus_bin_wmt <- taxus_bin[rowSums(taxus_bin) > 1, ]
```

We then apply the simulated annealing algorithm to this new data, starting from a uniformly-generated initial partition and searching for three groups. We selected five runs keeping the progress of the optimization for all of them (by setting `n_sol = 5` and `full_output = TRUE`), so that we can inspect the convergence of the method. We use `set.seed()` only to ensure the repeatability of the outputs:

```
set.seed(1)
result_sa <- optim_tdv_simul_anne(
  m_bin = taxus_bin_wmt,
  k = 3,
  p_initial = "random",
  n_runs = 5,
  n_sol = 5,
  use_grasp = FALSE,
  full_output = TRUE
)
```

The maximum TDV obtained in each run can be extracted from the output in the following way:

```
sapply(result_sa$SANN, function(x) x$tdv)

#> [1] 0.2005789 0.1958471 0.1879158 0.1749407 0.1583337
```

The second highest TDV (0.1958471) corresponds to the partition in three groups (Estrela, Gerês and Galicia) from the original article of [Portela-Pereira et al. \(2021\)](#):

```
result_sa$SANN[[2]]$par

#> [1] 3 3 3 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

The initial temperature defaults to 0.3 and the final temperature to 10^{-6} . Generally, convergence failure can be spotted when final TDV values are similar to the initial ones, especially when starting from random partitions. In such cases, as a rule of thumb, it is advisable to decrease `t_final` by a factor of 10. Figure 2 shows the plots of the TDV kept in each iteration, in each of the five runs, showing that the method is converging to higher values. Such a plot can be obtained in the following way:

```
plot(
  result_sa$SANN[[1]]$current.tdv,
  type = "l",
  xlab = "Iteration number",
  ylab = "TDV"
)
for (run in 2:5) {
  lines(result_sa$SANN[[run]]$current.tdv)
}
```

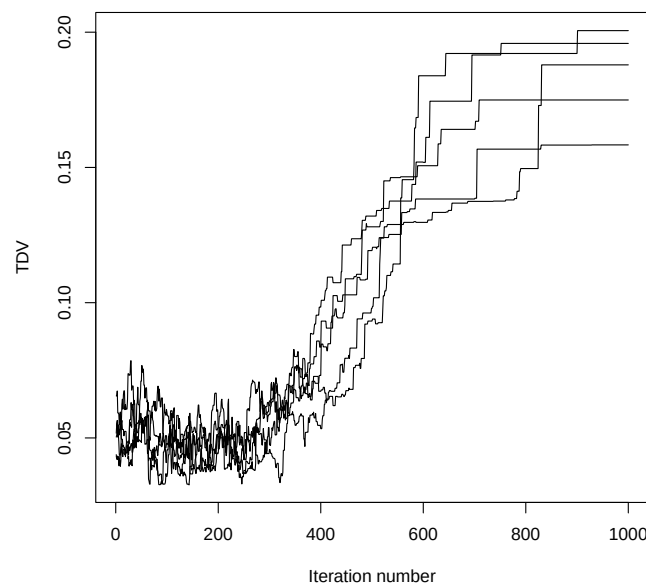


Figure 2: Plots of the TDV obtained in each of the five runs of the simulated annealing optimization.

Table 2: Computational performance of GRASP and simulated annealing (SA) for different numbers of groups k , applied to the *T. baccata* forests data set.

k	GRASP: total time	GRASP: memory allocation	GRASP: TDV	SA: total time	SA: memory allocation	SA: TDV
2	89.42ms	37.66MB	0.1188258	3.08s	1.04GB	0.1284921
3	166.9ms	58.21MB	0.1345673	3.58s	1.31GB	0.1956457
4	328.35ms	78.19MB	0.1991826	4.51s	1.58GB	0.2544701
5	508.1ms	98.01MB	0.2442543	5.1s	1.85GB	0.3183166

We can plot the sorted phytosociological table (see Figure 3), using the same partition as the original work of [Portela-Pereira et al. \(2021\)](#):

```
tabul <- tabulation(
  m_bin = taxus_bin_wmt,
  p = result_sa$SANN[[2]]$par,
  taxa_names = rownames(taxus_bin_wmt),
  plot_im = "normal",
  palette = "Zissou 1"
)
```

Simulated annealing is often seen as an exploratory technique where the temperature settings are challenging and dependent on the problem. The formulation here implemented (see Section 4) aims at mitigating such a challenge by restricting the temperature values in the interval $[0, 1]$.

This section concludes with a comparison of the computational performance of the GRASP and simulated annealing heuristics. Experiments are conducted on the *T. baccata* forests data set. The number of groups is varied from $k = 2$ to $k = 5$ in both `partition_tdv_grasp()` and `optim_tdv_simul_anne()` using default values for all optional parameters (except `p_initial = "random"` and `use_grasp = FALSE` for the simulated annealing).

Table 2 reports the running time, memory usage, and objective value (TDV) for each method. Running time and memory allocation were obtained with function `mark()` from package [bench](#) ([Hester and Vaughan, 2021](#)).

Simulated annealing consistently achieved higher TDV values for all k , although it required higher memory and running times than those of GRASP. Overall, these results indicate a trade-off between solution quality and computational efficiency, with simulated annealing providing more accurate solutions at the expense of longer computation times.

6 Concluding remarks

In this paper, we presented the mathematical approaches implemented in the R package [diffval](#) for maximizing TotDiffVal, a recently proposed criterion for assessing the quality of partitions of

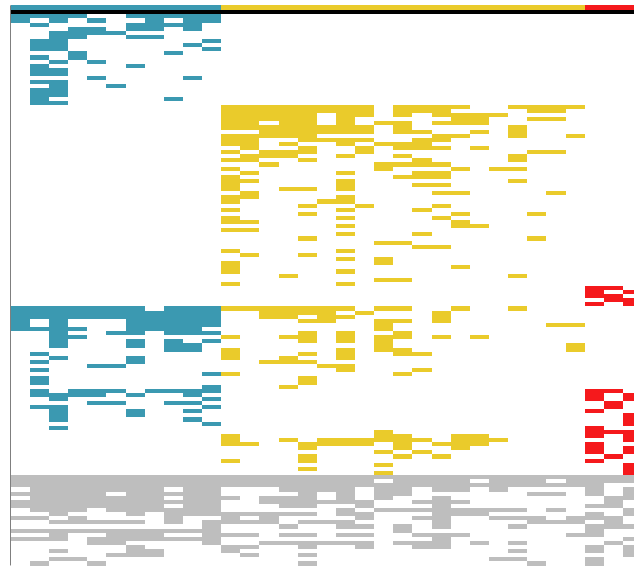


Figure 3: Image of the sorted phytosociological matrix, showing the three groups obtained by the simulated annealing optimization.

the columns (samples or objects) of a binary (0/1) matrix by capturing the exclusiveness of species (features, variables, or attributes) (Monteiro-Henriques, 2025).

We proved that finding a partition into M subsets that maximizes TotDiffVal is NP-hard for any fixed M . For the case $M = 2$, we provided two alternative integer linear programming formulations, both implemented in **diffval**, which allow optimal solutions to be obtained for moderately sized data sets. For larger instances and for $M > 2$, we developed two heuristic algorithms: a Greedy Randomized Adaptive Search Procedure (GRASP) and a Simulated Annealing approach. Good-quality partitions can often be achieved by running the Simulated Annealing algorithm initialized with solutions obtained from multiple runs of the GRASP. However, for very large instances, both heuristics may require excessive computation times. It should be noted that computing the TotDiffVal index (2) for a given M -partition requires a number of operations on the order of the matrix size ($m \times n$), and that re-evaluating the index after small modifications entails a similar computational burden. When the above procedures become computationally very expensive, an alternative is to use the function `partition_tdv_grdtp()`, which is also included in the **diffval** package. This function implements a simplified version of the Greedy algorithm that operates in the following way: Firstly, M columns are selected randomly to work as seeds for each one of the desired M groups. Secondly, one of the remaining columns is selected randomly and added to the partition group which maximizes the upcoming TDV. This second step is repeated until all columns are placed in a group of the M -partition. This function performs faster than `partition_tdv_grasp()` and `optim_tdv_simul_anne()`, but at the cost of the quality of the solutions produced.

Alternative clustering and partitioning methods especially designed for vegetation data are available, see e.g., the **vegclust** (De Cáceres, 2025) package (where k -means and partition around the medoids are modified into fuzzy clustering approaches) and the **vegan** (Oksanen et al., 2025) package (where several specific distance measures are implemented to use in agglomerative clustering). A methodological comparative analysis with 18 clustering/partitioning methods is provided in Monteiro-Henriques (2025). While most of these methods are not as computationally demanding as maximizing TDV, it should be emphasized that TDV optimization fundamentally differs from most clustering and partitioning approaches in three key aspects: i) it does not rely on hierarchical structures or geometric (distance-based) searches, ii) it treats species occurrences as not equally informative, and iii) it values a degree of exclusiveness among informative species. Encoding properties (i–iii) in a computational framework inevitably entails substantial computational costs, a challenge that the **diffval** package addresses by providing a range of exact and heuristic procedures, allowing users to balance solution quality and computation time according to their specific datasets.

7 Appendix

7.1 Computational complexity of maximizing TotDiffVal

We start by polynomially transforming the problem of finding a minimum bisection of a graph to the problem of maximizing the criterion TotDiffVal for $M = 2$.

Let $G = (V, E)$ be a graph with $|V|$ even. A bisection of G is a partition of the vertex set V into two sets of equal sizes. The size of a bisection is the number of edges going across the two sets. A minimum bisection of G is a bisection of V with minimal size. The size of a minimum bisection, denoted by $bw(G)$, is called the bisection width of graph G . The problem of finding a minimum bisection is NP-hard (Garey et al., 1976).

Let $B = [b_{ij}]$ be the transpose of the incidence matrix of graph $G = (V, E)$, i.e., $b_{ij} = 1$ (0) if vertex $v_j \in V$ and edge $e_i \in E$ are (not) incident. Add to B two columns of zeros. Let 0 and $n + 1$ denote the two new columns and denote by B_0 the resulting $m \times (n + 2)$ matrix, where $m = |E|$ and $n = |V|$ is even.

Note that if $S = (G_1, G_2)$ is an arbitrary 2-partition of the set of $n + 2$ columns of B_0 , with $|G_1| = |G_2| = \frac{n+2}{2}$, and if i is an arbitrary row, then

$$\text{DiffVal}_i(S) = \begin{cases} \frac{4}{n+2} & \text{if both vertices of edge } e_i \text{ belong to the same set of the 2-partition } S \\ 0 & \text{otherwise.} \end{cases}$$

If columns 0 and $n + 1$ are in different sets of the 2-partition S , $\text{TotDiffVal}(S) = \sum_{i=1}^m \text{DiffVal}_i(S) = \frac{4}{n+2} (m - bw(S))$, where $bw(S)$ is the size of the bisection of G corresponding to S , i.e., $(G_1 \setminus \{0, n + 1\}, G_2 \setminus \{0, n + 1\})$. If P denotes the set of all 2-partitions of the set of columns of matrix B_0 in two sets of equal size that have columns 0 and $n + 1$ in different sets, and if $bw(G)$ denotes the bisection width of graph G , we have

$$bw(G) = m - \frac{n+2}{4} \max_{S \in P} \text{TotDiffVal}(S). \quad (26)$$

Hence, we may find a minimum bisection of G by maximizing TotDiffVal(S), for 2-partitions $S \in P$.

We now create a $2n \times (n + 2)$ matrix, B' , with the same column indices as B_0 , and vertically append a number of copies of B' to B_0 to obtain a matrix which is such that the 2-partitions of the set of columns that maximize TotDiffVal are precisely the balanced partitions that have columns 0 and $n + 1$ belonging to different sets.

Matrix $B' = [b'_{ij}]$, as referred above, has the same column indices as B ; $b'_{i0} := 1, i = 1, \dots, n$; $b'_{i(n+1)} := 1, i = n + 1, \dots, 2n$; $b'_{ii} = b'_{n+ii} := 1, i = 1, \dots, n$ and all the remaining entries are equal to zero. In fact, B' is the transpose of the incidence matrix of the complete bipartite graph $K_{2,n}$ with bipartite classes $\{0, n + 1\}$ and V .

Let $\bar{B}$ denote the $(m + 2n\delta) \times (n + 2)$ matrix obtained by vertically appending to B_0 δ copies of B' .

We will prove that, for $\delta = n^3$,

- i) If $S = (G_1, G_2)$ is a 2-partition of the set of columns of $\bar{B}$ that maximizes TotDiffVal, and that has columns 0 and $n + 1$ belonging to different sets, then S is balanced (i.e. $|G_1| = |G_2| = \frac{n+2}{2}$).
- ii) If $S' = (G'_1, G'_2)$ is a 2-partition of the set of columns of $\bar{B}$ that maximizes TotDiffVal, and that has columns 0 and $n + 1$ belonging to the same set, say $0, n + 1 \in G'_1$, then $|G'_1| = n + 1$ and $|G'_2| = 1$.
- iii) $\text{TotDiffVal}(S) > \text{TotDiffVal}(S')$.

From i), ii) and iii) we can conclude that, for $\delta = n^3$, if $S = (G_1, G_2)$ is a 2-partition of the set of columns of $\bar{B}$ that maximizes TotDiffVal, then columns 0 and $n + 1$ belong to different sets, and the bisection of G corresponding to S , i.e., $(G_1 \setminus \{0, n + 1\}, G_2 \setminus \{0, n + 1\})$ is a minimum bisection of graph G . Since the size of matrix $\bar{B}$ is polynomial on the size of graph G ($\text{size}(\bar{B}) = O(n^3)$), we have the following result.

Proposition 1 Finding an M -partition that maximizes TotDiffVal is NP-hard, for $M = 2$. $\square$

We now prove i), ii) and iii).

Lemma 2 For $\delta = n^3$, if $S = (G_1, G_2)$ is a 2-partition of the set of columns of $\bar{B}$ that maximizes TotDiffVal, and that has columns 0 and $n + 1$ belonging to different sets, then S is balanced (i.e. $|G_1| = |G_2| = \frac{n+2}{2}$).

Proof. Let $X = (X_1, X_2)$ be a 2-partition of the set of columns of $\bar{B}$ with 0 and $n + 1$ belonging to

different sets. Suppose $|X_1| = t < \frac{n+2}{2}$.

$$\text{TotDiffVal}(X) = \frac{2m_1}{t} + \frac{2m_2}{n+2-t} + 2\delta \left(\frac{t-1}{t} + \frac{n+1-t}{n+2-t} \right), \quad (27)$$

where m_1 (m_2) is the the number of edges of the subgraph of graph G induced by the vertices of $X_1 \setminus \{0, n+1\}$ ($X_2 \setminus \{0, n+1\}$).

Let X' be a 2-partition obtained by moving an arbitrary column $j \neq 0, n+1$, from X_2 (the largest set) to X_1 .

$$\text{TotDiffVal}(X') = \frac{2(m_1 + |E_j(G_1)|)}{t+1} + \frac{2(m_2 - |E_j(G_2)|)}{n+1-t} + 2\delta \left(\frac{t}{t+1} + \frac{n-t}{n+1-t} \right),$$

where $E_j(G_1)$ ($E_j(G_2)$) is the set of edges linking vertex j with vertices of G_1 (G_2). We can therefore conclude that

$$\text{TotDiffVal}(X') \geq \frac{2m_1}{t+1} + \frac{2(m_2 - (n-t))}{n+1-t} + 2\delta \left(\frac{t}{t+1} + \frac{n-t}{n+1-t} \right). \quad (28)$$

From (27) and (28) we have

$$\text{TotDiffVal}(X') - \text{TotDiffVal}(X) \geq 2 \frac{-m_1 p + (m_2 - (n-t)(n+2-t))t(t+1) + \delta(p - t(t+1))}{t(t+1)p},$$

where $p = (n+1-t)(n+2-t)$.

From $t < \frac{n}{2}$, we have $p - t(t+1) > 2(t+1)$ and since $m_1 < \frac{t(t-1)}{2}$, we can conclude that

$$\text{TotDiffVal}(X') - \text{TotDiffVal}(X) > 2 \frac{-t(t-1)p/2 + (m_2 - p)t(t+1) + 2\delta(t+1)}{t(t+1)p}.$$

For $\delta = n^3$ the right hand side of the previous inequality is positive and the result follows. $\square$

Lemma 3 For $\delta = n^3$, if $S' = (G'_1, G'_2)$ is a 2-partition of the set of columns of $\bar{B}$ that maximizes TotDiffVal , and that has columns 0 and $n+1$ belonging to the same set, say $0, n+1 \in G'_1$, then $|G'_1| = n+1$ and $|G'_2| = 1$.

Proof. Let $X = (X_1, X_2)$ be a 2-partition of the set of columns of $\bar{B}$ with 0 and $n+1$ belonging to the same set. Suppose $0, n+1 \in X_1$ and $|X_1| = t < n-1$.

$$\text{TotDiffVal}(X) = \frac{2m_1}{t} + \frac{2m_2}{n+2-t} + 4\delta \frac{t-2}{t}, \quad (29)$$

where m_1 (m_2) is the the number of edges of the subgraph of graph G induced by the vertices of $X_1 \setminus \{0, n+1\}$ (X_2).

Let X' be a 2-partition obtained by moving an arbitrary column j from X_2 to X_1 .

$$\text{TotDiffVal}(X') = \frac{2(m_1 + |E_j(G'_1)|)}{t+1} + \frac{2(m_2 - |E_j(G'_2)|)}{n+1-t} + 4\delta \frac{t-1}{t+1},$$

where $E_j(G'_1)$ ($E_j(G'_2)$) is the set of edges linking vertex j with vertices of G'_1 (G'_2). We can therefore conclude that

$$\text{TotDiffVal}(X') \geq \frac{2m_1}{t+1} + \frac{2(m_2 - (n+1-t))}{n+1-t} + 4\delta \frac{t-1}{t+1}. \quad (30)$$

From (29) and (30) we have

$$\text{TotDiffVal}(X') - \text{TotDiffVal}(X) \geq 2 \frac{-m_1 p + m_2 t(t+1) - pt(t+1) + 4\delta p}{t(t+1)p},$$

where $p = (n+1-t)(n+2-t)$.

Since $m_1 \leq \frac{(t-2)(t-3)}{2}$ and $t < n-1$, for $\delta = n^3$, the right hand side of the previous inequality is positive and the result follows. $\square$

Lemma 4 For $\delta = n^3$, if $S = (G_1, G_2)$ and $S' = (G'_1, G'_2)$ are as in Lemma 2 and Lemma 3, respectively, then $\text{TotDiffVal}(S) > \text{TotDiffVal}(S')$.

Proof. From (26) we have

$$\text{TotDiffVal}(S) = \frac{4}{n+2}(m - bw(G)) + 4\delta \frac{n}{n+2},$$

where $bw(G)$ is the bisection width of graph G .

We now use the upper bound on $bw(G) \leq \frac{mn}{2(n-1)}$ (Schmidt, 2017, p. 3) to write

$$\text{TotDiffVal}(S) \geq \frac{4m}{n+2} \left(1 - \frac{n}{2(n-1)}\right) + 4\delta \frac{n}{n+2}.$$

For the 2-partition S' the following upper bound holds

$$\text{TotDiffVal}(S') \leq \frac{2m}{n+1} + 4\delta \frac{n-1}{n+1}.$$

From the two previous expressions we have

$$\text{TotDiffVal}(S) - \text{TotDiffVal}(S') \geq \frac{-4mn + 8\delta(n-1)}{(n+2)(n+1)(n-1)}.$$

Since $m \leq \frac{n(n-1)}{2}$, for $\delta = n^3$, the right hand side of the previous inequality is positive and the result follows. $\square$

We now extend Proposition 1 to an arbitrary number of groups.

Proposition 5 Finding an M -partition that maximizes TotDiffVal is NP-hard, for any fixed number of groups $M \geq 2$.

Proof. We use induction on M . We show that maximizing the criterion TotDiffVal , for arbitrary $M \geq 2$, polynomially transforms to the problem of maximizing TotDiffVal , for $M+1$ groups.

Suppose, for a given 0/1 matrix A , $m \times n$, we want to maximize TotDiffVal , for M groups. Add to A $3m$ zero rows (rows $m+1, \dots, m+3m$), and append to the resulting matrix an all one column as the rightmost column (column $n+1$). Let $\bar{A}$ be the $4m \times (n+1)$ matrix thus obtained. Let S and S' be arbitrary $(M+1)$ -partitions of $\bar{A}$ such that, in partition S , column $n+1$ is the unique column in a group, while in partition S' , column $n+1$ belongs to a group of size $t > 1$. Clearly, $\text{TotDiffVal}(S) \geq 3m$ and $\text{TotDiffVal}(S') \leq m + \frac{3}{t}m$, which implies that, for $t > 1$, $\text{TotDiffVal}(S) > \text{TotDiffVal}(S')$. Hence, column $n+1$ is isolated in a group in every optimal $(M+1)$ -partition of $\bar{A}$. Consequently, the set of the other M groups of columns, which form an optimal M -partition of the first n columns of $\bar{A}$, is also an optimal M -partition of matrix A .

The fact that, for $M = 2$, the result holds (Proposition 1), completes the proof. $\square$

Acknowledgments

JOC is funded by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the scope of the projects UID/00297/2025 (<https://doi.org/10.54499/UID/00297/2025>) and UID/PRR/00297/2025 (<https://doi.org/10.54499/UID/PRR/00297/2025>) (Center for Mathematics and Applications – NOVA Math). TMH was partially funded by the European Social Fund (POCH and NORTE 2020) and National Funds (MCTES) through Fundação para a Ciência e a Tecnologia, I.P., postdoctoral fellowship (SFRH/BPD/115057/2016), and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia, I.P., under the project UIDB/04033/2020.

References

- E. Aarts, J. Korst, and P. Laarhoven, van. Simulated annealing. In E. Aarts and J. Lenstra, editors, *Local search in combinatorial optimization*, pages 91–120. Wiley-Interscience, 1997. ISBN 0-471-94822-5. URL <https://pure.tue.nl/ws/portalfiles/portal/1332973/496713.pdf>. [p333]
- J. S. Borrero, C. Gillen, and O. A. Prokopyev. Fractional 0–1 Programming: Applications and Algorithms. *Journal of Global Optimization*, 69(1):255–282, September 2017. URL <https://doi.org/10.1007/s10898-016-0487-4>. [p332]
- H. Bruelheide. A new measure of fidelity and its application to defining species groups. *Journal of Vegetation Science*, 11:167–178, 2000. URL <https://doi.org/10.2307/3236796>. [p329]
- H. Bruelheide and T. Flintrop. Arranging phytosociological tables by species-relevé groups. *Journal of Vegetation Science*, 5(3):311–316, 1994. ISSN 11009233, 16541103. URL <http://www.jstor.org/stable/3235854>. [p329]

- E. N. Castanho, H. Aidos, and S. C. Madeira. Biclustering data analysis: a comprehensive survey. *Briefings in Bioinformatics*, 25(4):bbae342, 07 2024. ISSN 1477-4054. doi: 10.1093/bib/bbae342. URL <https://doi.org/10.1093/bib/bbae342>. [p329]
- M. De Cáceres. *vegclust: Fuzzy Clustering of Vegetation Data*, 2025. URL <https://CRAN.R-project.org/package=vegclust>. R package version 2.0.3. [p338]
- M. Dufrêne and P. Legendre. Species assemblages and indicator species: The need for a flexible asymmetrical approach. *Ecological Monographs*, 67(3):345–366, 1997. ISSN 00129615. URL <https://doi.org/10.2307/2963459>. [p329]
- T. A. Feo and M. G. Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995. URL <https://doi.org/10.1007/BF01096763>. [p333]
- E. Feoli and L. Orlóci. Analysis of concentration and detection of underlying factors in structured tables. *Vegetatio*, 40(1):49–54, 1979. ISSN 00423106. URL <http://www.jstor.org/stable/20145697>. [p329]
- P. Festa and M. G. C. Resende. Grasp. In R. Martí, P. M. Pardalos, and M. G. C. Resende, editors, *Handbook of Heuristics*, pages 465–488. Springer International Publishing, Cham, 2018. ISBN 978-3-319-07124-4. URL https://doi.org/10.1007/978-3-319-07124-4_23. [p333]
- M. Garey, D. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976. ISSN 0304-3975. URL [https://doi.org/10.1016/0304-3975\(76\)90059-1](https://doi.org/10.1016/0304-3975(76)90059-1). [p331, 339]
- M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, first edition edition, 1979. ISBN 0716710455. URL <https://www.bibsonomy.org/bibtex/2fac25cd952418baa739dd35243b3dbdc/folke>. [p331]
- Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL <https://www.gurobi.com>. [p334]
- J. O. Hanson, R. Schuster, N. Morrell, M. Strimas-Mackey, B. P. M. Edwards, M. E. Watts, P. Arcese, J. Bennett, and H. P. Possingham. *prioritizr: Systematic Conservation Prioritization in R*, 2022. URL <https://CRAN.R-project.org/package=prioritizr>. R package version 8.0.6. [p334]
- J. Hester and D. Vaughan. *bench: High Precision Timing of R Expressions*, 2021. URL <https://CRAN.R-project.org/package=bench>. R package version 1.1.2. [p334, 337]
- M. O. Hill. *TWINSPAN - A FORTRAN program for arranging multivariate data in an ordered two-way table by classification of the individuals and attributes*. Section of Ecology and Systematics, Ithaca, New York, USA, 1979. [p329]
- L. Kaufman and P. J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. Wiley series in probability and mathematical statistics. Wiley, New York, 1990. ISBN 978-0-471-87876-6. URL <https://doi.org/10.1002/9780470316801>. [p329]
- S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi: 10.1126/science.220.4598.671. URL <https://www.science.org/doi/abs/10.1126/science.220.4598.671>. [p333]
- T. Monteiro-Henriques. TDV-optimization: A novel numerical method for phytosociological tabulation. *Vegetation Classification and Survey*, 6:99–127, 2025. URL <https://doi.org/10.3897/VCS.140466>. [p329, 330, 338]
- T. Monteiro-Henriques and J. O. Cerdeira. *diffval: Vegetation Patterns*, 2025. URL <https://point-veg.gitlab.io/diffval/>. R package version 1.2.0. [p331]
- D. Mueller-Dombois and H. Ellenberg. *Aims and Methods of Vegetation Ecology*. Wiley, 1974. ISBN 978-0-471-62290-1. URL <https://books.google.pt/books?id=Gu8sAQAAIAAJ>. [p329]
- J. Oksanen, G. L. Simpson, F. G. Blanchet, R. Kindt, P. Legendre, P. R. Minchin, R. O’Hara, P. Solymos, M. H. H. Stevens, E. Szoecs, H. Wagner, M. Barbour, M. Bedward, B. Bolker, D. Borcard, T. Borman, G. Carvalho, M. Chirico, M. De Cáceres, S. Durand, H. B. A. Evangelista, R. FitzJohn, M. Friendly, B. Furneaux, G. Hannigan, M. O. Hill, L. Lahti, C. Martino, D. McGlinn, M.-H. Ouellette, E. Ribeiro Cunha, T. Smith, A. Stier, C. J. Ter Braak, and J. Weedon. *vegan: Community Ecology Package*, 2025. URL <https://CRAN.R-project.org/package=vegan>. R package version 2.7-2. [p338]

- R. K. Peet and D. W. Roberts. *Classification of Natural and Semi-natural Vegetation*, chapter 2, pages 28–70. John Wiley & Sons, Ltd, 2013. ISBN 978-1-118-45259-2. URL <https://doi.org/10.1002/9781118452592.ch2>. [p329]
- J. Podani and E. Feoli. A general strategy for the simultaneous classification of variables and objects in ecological data tables. *Journal of Vegetation Science*, 2(4):435–444, 1991. URL <https://doi.org/10.2307/3236025>. [p329]
- E. a. M. Portela-Pereira, T. Monteiro-Henriques, C. Casas, N. Forner, I. Garcia-Cabral, J. a. P. Fonseca, and C. Neto. Teixedos no noroeste da península ibérica. *Finisterra*, 56(117):127–150, Set. 2021. URL <https://doi.org/10.18055/Finis18102>. [p334, 336, 337]
- J. Roleček, L. Tichý, D. Zelený, and M. Chytrý. Modified TWINSpan classification in which the hierarchy respects cluster heterogeneity. *Journal of Vegetation Science*, 20(4):596–602, Aug. 2009. ISSN 1654-1103. doi: 10.1111/j.1654-1103.2009.01062.x. URL <http://onlinelibrary.wiley.com/doi/10.1111/j.1654-1103.2009.01062.x/abstract>. [p329]
- T. J. Schmidt. *On the minimum bisection problem in tree-like and planar graphs – structural and algorithmic results*. PhD thesis, Technische Universität München, Germany, 2017. URL <https://mediatum.ub.tum.de/doc/1338548/1338548.pdf>. [p341]

J. Orestes Cerdeira

Center for Mathematics and Applications (NOVA Math)

NOVA School of Science and Technology (NOVA FCT), Universidade NOVA de Lisboa

2829-516 Caparica, Portugal

ORCID: 0000-0002-3814-7660

jo.cerdeira@fct.unl.pt

Tiago Monteiro-Henriques

Global Change and Conservation Lab (GCC)

Faculty of Biological and Environmental Sciences, University of Helsinki

Viikinkaari 1, P.O. Box 65, 00014 University of Helsinki, Finland

ORCID: 0000-0002-4206-0699

tmh.dev@icloud.com

Priority Elastic Net: A Hierarchical Framework for Heterogeneous Data Integration

by L Musib, by H. Mourão, by E. Carrasquinha

Abstract The rapid development of high-throughput technologies has produced increasingly complex and high-dimensional biological datasets, requiring advanced statistical methods. In such contexts, where the number of variables exceeds the number of observations, variable selection is crucial to improve prediction accuracy, reduce variance, and enhance interpretability. The integration of multiple omics datasets collected from the same set of patients introduces additional challenges, including heterogeneity, multicollinearity, and the need to prioritise information across data blocks. The Priority Elastic Net method addresses these challenges by incorporating prior knowledge of block structure. The `priorityelasticnet` package, available on CRAN, provides implementations for continuous, binary, multinomial, and survival outcomes in R. It sequentially fits elastic net (or adaptive elastic net) models to predefined blocks, enabling customised penalisation and flexible feature selection. This framework is particularly useful for multi-omics integration and other structured high-dimensional applications.

1 Introduction

In the field of high-dimensional omics data analysis, regularisation techniques have become essential due to their ability to simultaneously select variables and estimate models, particularly when the number of explanatory variables greatly exceeds the sample size. A variety of computational packages have been developed to implement these techniques effectively, addressing a range of research needs in omics studies.

The least absolute shrinkage and selection operator (lasso) (Tibshirani, 1996) is a popular method for regression that uses an ℓ_1 penalty to achieve a sparse solution. This idea has been broadly applied, for example, to generalized linear models (Tibshirani, 1996) and Cox proportional hazard models for survival data (Tibshirani, 1997).

While the lasso excels at promoting sparsity, it does not account for feature ordering or structure, a limitation particularly pronounced in high-dimensional omics data. To overcome this, extensions such as the fused lasso (Tibshirani et al., 2005) incorporate penalties that encourage smoothness along ordered features. In contrast, the adaptive lasso (Zou, 2006) dynamically adjusts penalties to improve variable selection. These methods are integrated into specialised packages, such as **SQOPT** (Arnold et al., 2022) and **adalasso**.

Despite its success in many applications, the lasso method struggles with variable selection when features are highly correlated, as it tends to select only one variable from a group, a limitation in applications such as microarray data analysis, where genes often interact in networks. To address this issue, Zou and Hastie (2005) proposed the elastic net penalty, which is a weighted sum of the ℓ_1 norm and the square of the ℓ_2 norm of the coefficient vector β . The first term enforces the sparsity of the solution, whereas the second term encourages a more balanced selection of correlated variables.

The **glmnet** package, one of the most popular implementations in R and Python, provides a robust, widely used framework for applying lasso, elastic net, and related techniques. Details may be found in Friedman et al. (2010), Simon et al. (2011), Tibshirani et al. (2012). The **glmnet** R package (Friedman et al., 2010) contains efficient functions for computing the elastic net solution for a complete path-wise optimisation. The minimisation problems are solved via cyclic coordinate descent (Van der Kooij, 2007), with the core routines programmed in Fortran for computational efficiency. Earlier versions of the package contained specialised Fortran subroutines for a handful of popular GLMs and the Cox model for

right-censored survival data. The package includes functions for performing K-fold cross-validation (CV) for hyperparameter optimisation, plotting coefficient paths and CV errors, and predicting on future data. The package can also accept the predictor matrix in sparse matrix format: this is especially useful in certain applications where the predictor matrix is both large and sparse. In particular, this means that we can fit unpenalised GLMs with sparse design matrices, something that the `glm` function from the package `stats` does not support.

Version 4.0 and later of `glmnet` (Friedman et al., 2023) represent a significant release that expands the package's capabilities for survival modelling. It supports multiple events per subject, delayed entry, time-dependent covariates and/or strata, and other extensions, through the (start, stop] formulation of the survival data that specifies the start and end of each time interval. Additionally, it includes sparse X inputs, and the much-requested `survfit` method. The package computes the elastic net regularisation path for all GLMs, Cox models with (start, stop] data and strata, and includes a simplified implementation of the relaxed lasso (Hastie et al., 2020), providing greater flexibility for high-dimensional data analysis.

To address the instability often observed in methods based on the elastic net, adaptive strategies have been incorporated into regularisation frameworks. One such approach is the Adaptive Elastic Net (AENet), implemented in the R package `msaenet`. AENet refines the elastic net regularisation method to enhance feature selection and model performance, particularly in high-dimensional settings or in the presence of multicollinearity among predictors. Unlike the standard elastic net, which applies the same penalty to all coefficients, AENet introduces adaptive weighting to penalise features differently based on their importance (Zou and Zhang, 2009).

In the era of multi-omics research, integrating data across multiple omics layers is crucial for harnessing the complementary information from diverse biological systems. Single-data-type models often fail to capture the full complexity of underlying biological processes. Recent advances have underscored the importance of combining high-dimensional omics data, including clinical information, gene expression, methylation profiles, and other omics layers, to improve predictive modelling frameworks (Huang et al., 2017).

Using multi-omics data in prediction models is promising because each omics type offers unique and valuable information for predicting phenotypic outcomes. However, effectively integrating various types of omics data presents several challenges. First, there is an overlap in the predictive information each data type provides. Second, the levels of predictive details differ between the 'blocks' of variables (which can correspond to the type of data, e.g., the block of clinical variables or the block of gene expression variables) and depend on the particular outcome considered (Zhao et al., 2015). Third, the interactions between variables of different data types must be considered (Huang et al., 2017). In recent years, different methods have been proposed to address this topic. Simon et al. (2013) presented the sparse group lasso, a prediction method for grouped covariate data that automatically removes non-informative covariate groups and performs lasso-type variable selection for the remaining covariate groups. The `SGL` package (Simon et al., 2018) in R is a notable implementation that combines the strengths of group lasso and lasso for sparse solutions at both the group and individual variable levels.

One limitation of using sparse lasso in multi-omics data applications is that it does not adequately account for the varying levels of predictive information across different data blocks. In contrast, the IPF-LASSO (Boulesteix et al., 2017), a lasso-type regression method for multi-omics data, addresses this by assigning individual penalty parameters to different omics data types, allowing incorporation of prior biological knowledge or practical constraints. One common issue for all variations of LASSO, including IPF-LASSO, is instability. Even small modifications to the dataset can significantly change the selected model. The IPF-LASSO method addresses the unique challenges of multi-omics data integration, particularly the differing levels of predictive information across data blocks. The `ipflasso` package (Boulesteix and Fuchs, 2015) has been developed to implement this method, offering a tailored approach to multi-omics data analysis.

Building on the idea of representing each omics dataset as a block, Klau et al. (2018) introduced the Priority-Lasso method, implemented in the **prioritylasso** package, as a lasso-based approach designed explicitly for multi-omics data analysis. This method emphasises practical usability by allowing users to assign priority rankings to data blocks based on cost or relevance.

The package **prioritylasso**, developed for applications involving continuous, binary, and survival outcomes, was initially released on CRAN as version 0.2.4, as documented in Klau et al. (2017). Available from the CRAN repository, the package builds on the lasso regression implementation in **glmnet** (Friedman et al., 2010). The Cox-Lasso model employs the regularisation method described by Simon et al. (2011). In version 0.3.1, functionality was added to address blockwise missing data. This means that for some observations, not all blocks are observed. To address this type of missingness, **prioritylasso** offers several options for fitting a model in datasets with missing values. Nevertheless, priority-lasso struggles with multicollinearity, often arbitrarily selecting one variable from correlated groups while excluding others, resulting in unstable models. This approach applies block prioritisation to manage overlapping predictive information, ensuring that lower-priority blocks are not excluded but adjusted for previously explained variance. Although Priority-Lasso effectively promotes sparsity, it may lack stability, particularly in datasets with highly correlated variables or variable groupings of varying sizes. Furthermore, its sequential prioritisation framework does not fully exploit the hierarchical or grouped structures common in omics data, potentially limiting its applicability.

To address some of the limitations highlighted in the literature and summarised above, the Priority Elastic net method was proposed by Musib et al. (2024) as an enhancement of the Priority-Lasso approach introduced by Klau et al. (2018). By integrating the elastic net penalty, which balances ℓ_1 and ℓ_2 regularisation, Priority Elastic Net effectively mitigates issues related to multicollinearity and model instability. This method enhances the retention of predictive features from lower-priority blocks, providing greater flexibility for handling complex, high-dimensional datasets.

The **priorityelasticnet** package (Musib et al., 2025) is available on the Comprehensive R Archive Network (CRAN) at <http://CRAN.R-project.org/package=priorityelasticnet>.

2 Priority elastic net

A growing number of studies have utilised multi-omics data for various precision medicine-related machine learning tasks, including disease diagnosis, patient sub-phenotyping, and predicting disease or mortality risk. However, data from different omics modalities can be heterogeneous and noisy, with various underlying distributions and scales (Huang et al., 2017). Thus, the way these data are integrated significantly impacts subsequent analyses and interpretations.

The Priority Elastic Net (Musib et al., 2025) builds upon the Priority-Lasso framework (Klau et al., 2018), incorporating elastic net principles to address limitations in handling highly correlated predictors. The implementation of the **priorityelasticnet** package adapts several functions from the **prioritylasso** package (Klau et al., 2018, 2017). While Priority-Lasso introduced a hierarchical structure to effectively integrate multi-omics data into regression models, it struggles with datasets containing strongly correlated variables. By combining the strengths of ℓ_1 and ℓ_2 regularisation (Zou and Hastie, 2005), the Priority Elastic Net promotes a grouping effect, ensuring that correlated predictors are retained within the model. This enhancement enables the Priority Elastic Net to deliver improved robustness, interpretability, and accuracy, making it particularly suited to high-dimensional and complex omics datasets.

In the following subsections, we outline the theoretical foundations and computational implementation of Priority Elastic Net optimisation across Gaussian, multinomial, binomial, and Cox regression frameworks, demonstrating its potential in enhancing variable selection. We introduce the main functions of the **priorityelasticnet** package, which address essential

data characteristics, such as handling missing values, and include cross-validation methods. Illustrative examples, primarily based on the Gaussian distribution, are provided to demonstrate their usage. Finally, we introduce the Priority-Adaptive Elastic Net, which further improves model performance by dynamically adjusting variable priorities during optimisation.

2.1 Priority elastic net for a gaussian family

We consider the standard setup for linear regression, where the response variable $Y \in \mathbb{R}$ is continuous. Let x_{ij} be the independent variable j of individual i , where $i = 1, \dots, n$, $j = 1, \dots, p$ with n being the total number of individuals and p the number of independent variables. The vector $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{ip})^T$ represents the set of independent variables for the individual i . For simplicity, the predictor variables x_{ij} are transformed to z-scores.

In high-dimensional settings, where the number of predictors (p) may exceed the number of observations (n), or where the predictors exhibit strong correlations, traditional linear regression methods often struggle. Issues such as overfitting and multicollinearity reduce the reliability and interpretability of the estimated regression coefficients. To address these challenges, regularisation methods, such as the elastic net, have been developed. The standard elastic net method (Zou and Hastie, 2005) estimates the regression coefficients $\beta_1, \dots, \beta_p$ associated with the p regressors by minimizing the expression:

$$\hat{\beta}_{EN} = \arg \min_{\beta} \left[\frac{1}{n} \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p \left(\alpha |\beta_j| + \frac{(1-\alpha)}{2} \beta_j^2 \right) \right], \quad (1)$$

where $\lambda \geq 0$ is a tuning parameter, and $\alpha \in [0, 1]$ is a higher-level hyperparameter that controls the balance between the lasso and ridge penalties. A sequence of models is typically fitted across a range of λ values, while the value of α is predetermined based on the desired characteristics of the prediction model. For example, ridge regression (Hoerl and Kennard, 1970) is defined with $\alpha = 0$, while lasso regression (Tibshirani, 1996) is defined with $\alpha = 1$.

The final value of λ is usually chosen via cross-validation: we select the coefficients corresponding to the λ value that minimises the cross-validated error as the final model.

Coordinate descent solves the optimisation problem by iteratively optimising each coefficient β_j while keeping the others fixed (Friedman et al., 2010). The formula for updating β_j is given by:

$$\hat{\beta}_j = S \left(\frac{1}{n} \sum_{i=1}^n x_{ij} \left(y_i - \hat{y}_i^{(j)} \right), \lambda \alpha \right) \frac{1}{1 + \lambda(1 - \alpha)},$$

where

$$\hat{y}_i^{(j)} = \hat{\beta}_0 + \sum_{\ell \neq j} x_{i\ell} \hat{\beta}_\ell,$$

and the soft-thresholding operator $S(z, \gamma)$ is defined as:

$$S(z, \gamma) = \text{sign}(z) \max(|z| - \gamma, 0).$$

In this context, the notation is adapted to address the case of grouped variables, as considered in this work. The model partitions the predictors into B blocks, with the observations of the p_b variables from block b for subject i being represented as $(x_{i1}^{(b)}, \dots, x_{ip_b}^{(b)})$, for $i = 1, \dots, n$ and for $b = 1, \dots, B$. This block structure reflects the natural organisation of multi-omics data, where each omics layer (e.g., transcriptomics, proteomics, methylation) typically forms a separate block, often combined with a clinical block. More generally, it can also represent other meaningful groupings, such as biological pathways or categories of clinical variables.

Just as $x_{ij}^{(b)}$ is defined, $\beta_j^{(b)}$ is the regression coefficient of the j^{th} variable from block b , with j ranging from 1 to p_b . Accordingly, $\hat{\beta}_j^{(b)}$ denotes the respective estimated value.

Consider $\pi = (\pi_1, \dots, \pi_B)$ as a reordering of the sequence $(1, \dots, B)$, signifying the priority order of blocks, where π_1 represents the index of the block with highest significance and π_B denotes the one of least priority. Subsequently, a hierarchical method is employed to construct the predictive model, adhering to this established order of block importance. The first step is to estimate the $\beta_j^{(\pi_1)}$ coefficients, by minimising:

$$l(\beta^{(\pi_1)}) = \sum_{i=1}^n \left(y_i - \beta_0^{(\pi_1)} - \sum_{j=1}^{p_{\pi_1}} x_{ij}^{(\pi_1)} \beta_j^{(\pi_1)} \right)^2 + \lambda^{(\pi_1)} \sum_{j=1}^{p_{\pi_1}} \left(\alpha |\beta_j^{(\pi_1)}| + \frac{(1-\alpha)}{2} (\beta_j^{(\pi_1)})^2 \right). \quad (2)$$

The fitted linear predictor, $\hat{\eta}_{1,i}(\pi)$, in this step is given as:

$$\hat{\eta}_{1,i}(\pi) = \hat{\beta}_0^{(\pi_1)} + \hat{\beta}_1^{(\pi_1)} x_{i1}^{(\pi_1)} + \dots + \hat{\beta}_{p_{\pi_1}}^{(\pi_1)} x_{ip_{\pi_1}}^{(\pi_1)}. \quad (3)$$

Next, the elastic net method is applied to the block of predictors with the second highest priority, incorporating the fitted linear predictor from the first step as an offset. The goal is to determine the values of $\hat{\beta}_1^{(\pi_2)}, \dots, \hat{\beta}_{p_{\pi_2}}^{(\pi_2)}$, which are the coefficients corresponding to the predictors within the second priority block, π_2 . These coefficients are determined by minimising a specific criterion, outlined as follows:

$$l(\beta^{(\pi_2)}) = \sum_{i=1}^n \left(y_i - \hat{\eta}_{1,i}(\pi) - \beta_0^{(\pi_2)} - \sum_{j=1}^{p_{\pi_2}} x_{ij}^{(\pi_2)} \beta_j^{(\pi_2)} \right)^2 + \lambda^{(\pi_2)} \sum_{j=1}^{p_{\pi_2}} \left(\alpha |\beta_j^{(\pi_2)}| + \frac{(1-\alpha)}{2} (\beta_j^{(\pi_2)})^2 \right). \quad (4)$$

The linear predictor for the second step, $\hat{\eta}_{2,i}(\pi)$, is derived by adding the offset obtained from the first step, $\hat{\eta}_{1,i}(\pi)$, with the linear combination of the predictors in the second priority block and their respective estimated coefficients, $\hat{\beta}^{(\pi_2)}$. This relationship can be represented as:

$$\hat{\eta}_{2,i}(\pi) = \hat{\eta}_{1,i}(\pi) + \hat{\beta}_0^{(\pi_2)} + \hat{\beta}_1^{(\pi_2)} x_{i1}^{(\pi_2)} + \dots + \hat{\beta}_{p_{\pi_2}}^{(\pi_2)} x_{ip_{\pi_2}}^{(\pi_2)}. \quad (5)$$

Then, similarly, the model is fitted to the block with the third highest priority, using the linear score from the second step as an offset. All remaining blocks are treated in the same manner,

$$\hat{\eta}_{b,i}(\pi) = \hat{\eta}_{b-1,i}(\pi) + \hat{\beta}_0^{(\pi_b)} + \hat{\beta}_1^{(\pi_b)} x_{i1}^{(\pi_b)} + \dots + \hat{\beta}_{p_{\pi_b}}^{(\pi_b)} x_{ip_{\pi_b}}^{(\pi_b)}, \quad (6)$$

where $\hat{\eta}_{0,i}(\pi) = 0$ and $b = 1, \dots, B$.

The fitting procedure is carried out sequentially through the blocks up to π_B , yielding the set of coefficient estimates $\hat{\beta}^{(\pi_b)}$ for $b = 1, \dots, B$. The final fitted linear predictor becomes

$$\hat{\eta}_i(\pi) = \sum_{b=1}^B \hat{\beta}_0^{(\pi_b)} + \sum_{b=1}^B \sum_{j=1}^{p_{\pi_b}} \hat{\beta}_j^{(\pi_b)} x_{ij}^{(\pi_b)}. \quad (7)$$

As previously discussed, Priority Elastic Net assigns a priority order to blocks, allowing the model to incorporate prior knowledge. However, this hierarchical structure may negatively affect predictive accuracy by underestimating the influence of predictors in lower-priority blocks and potentially discarding informative variables. This issue occurs because the linear score in each block, $\hat{\eta}_{b,i}(\pi)$, tends to overfit the contribution of predictors in the b -th block to the response y_i . Specifically, the estimation of $\hat{\eta}_{b,i}(\pi)$ tends to overfit the effects of predictors in the b -th block on the response y_i . This overfitting arises because the coefficients $\hat{\beta}_1^{(\pi_b)}, \dots, \hat{\beta}_{p_{\pi_b}}^{(\pi_b)}$ are estimated using data that includes y_i , the response being predicted. Consequently, the overfitted score $\hat{\eta}_{b,i}(\pi)$ captures variability in y_i that might better be explained by predictors in subsequent blocks. When this overfitted estimate is used

as an offset, it limits lower-priority blocks' ability to account for unexplained variability in y_i , thereby undermining the model's overall predictive accuracy.

To mitigate this overfitting problem, cross-validated offsets, $\hat{\eta}_{b,i}(\pi)_{CV}$, previously introduced in the Priority Lasso framework, are also applied in the Priority Elastic Net to improve generalisation. In this approach, a dataset S is partitioned using K -fold cross-validation, and the cross-validated offsets are determined to effectively address the identified overfitting issues. The cross-validated offsets are computed iteratively as follows:

$$\hat{\eta}_{b,i}(\pi)_{CV} = \hat{\eta}_{b-1,i}(\pi)_{CV} + \hat{\beta}_{S \setminus S_{k,0}}^{(\pi_b)} + \hat{\beta}_{S \setminus S_{k,1}}^{(\pi_b)} x_{i1}^{(\pi_b)} + \dots + \hat{\beta}_{S \setminus S_{k,p\pi_b}}^{(\pi_b)} x_{ip\pi_b}^{(\pi_b)}, \quad b = 1, \dots, B \quad (8)$$

where the process starts with the initial condition: $\hat{\eta}_{0,i}(\pi)_{CV} = 0$.

Priority Elastic Net is implemented in the `priorityelasticnet` function in our R package of the same name. This function includes the option `cvoffset`, which allows users to choose between cross-validated offsets and standard offsets estimated without cross-validation, as outlined in the algorithm presented in this section.

To illustrate the implementation of Priority Elastic Net for the Gaussian family, a synthetic dataset was simulated to mimic the high-dimensional characteristics typical of multi-omics studies. In this example, $n = 100$ observations and $p = 500$ predictors are generated, with only the first 10 predictors contributing to the response variable.

```
set.seed(123)
n <- 100
p <- 500
X <- matrix(rnorm(n * p), n, p)
beta <- rnorm(10)
Y <- X[, 1:10] %*% beta + rnorm(n)
```

Before applying Priority Elastic Net, the user must first specify a block structure for the covariates, where each covariate belongs to exactly one of the B blocks, and second, a priority order for these blocks. In this framework, three blocks of continuous predictor variables were analysed, all of which are subject to penalisation, while adapting our notation to account for variables forming groups as considered in this work. The first entry in the blocks list specifies the indices of the variables that belong to the block assigned the highest priority (priority 1), which is the first block included in the model. As an example, if `blocks = list(1:10, 11:30, 31:500)`, the first priority block contains the first 10 variables in the data matrix. Similarly, the block assigned priority 2 includes variables 11-30, whereas the block assigned priority 3 comprises variables 31-500.

```
blocks <- list(block1 = 1:10, block2 = 11:30, block3 = 31:500)
```

The `priorityelasticnet` function fits a hierarchical elastic net model across multiple blocks of variables. In this example, the model is fitted with `family="gaussian"` for continuous response variables, with the `block1.penalization` parameter controlling how the first block (priority 1) is treated during model fitting.

```
fit_gaussian <- priorityelasticnet(
  X = X, Y = Y, family = "gaussian", blocks = blocks,
  block1.penalization = TRUE, standardize = TRUE,
  lambda.type = "lambda.min", max.coef = c(Inf, Inf, Inf),
  nfolds = 10, type.measure = "mse", alpha = 0.6,
  cvoffset = TRUE, cvoffsetnfolds = 10,
  adaptive = FALSE, initial_global_weight = FALSE
)
```

When `block1.penalization = TRUE`, the function applies a standard elastic net model to the first block, treating it similarly to other blocks. The predictions from this first block are then used as offsets in the elastic net fitting of the subsequent blocks, following their priority order. Conversely, when `block1.penalization = FALSE`, the function fits the first block without penalisation, making this setting appropriate for scenarios where clinical predictors (with $p < n$) are included. Clinical predictors in the first block are not penalised to preserve

their established relevance. In this case, the unpenalized predictions serve as offsets for the subsequent elastic net fits for lower-priority blocks.

The `standardize = TRUE` leads to a standardisation of the predictor variables. In case of an unpenalized first block, the predictor variables for the first block are not standardised. Please note that the returned coefficients are rescaled to the original scale of the predictor variables as provided in `X`. Therefore, new data in `predict.priorityelasticnet` should be on the same scale as `X`.

The `nfolds` and `lambda.type` are options referring to the cross-validation procedure in `cv.glmnet`. The `nfolds` specifies the number of folds in the procedure, while `lambda.type` handles the amount of cross-validation error. It can be set to either `lambda.min`, which is also the default, or `lambda.1se`. The `lambda.min` gives the result with minimum mean cross-validation error, whereas `lambda.1se` gives the result such that the cross-validation error is within one standard error of the minimum, and thus leads to more sparse results. Note that the latter can only be chosen in combination with no restrictions in `max.coef`.

The function used for every Priority Elastic net fit is `glmnet`, which creates a sequence of `lambda` values. The `lambda` which is chosen according to `lambda.type` is the `lambda` on position `lambda.ind` of the sequence and its value is stored in `lambda.min`. In general, the lower the value of `lambda.ind`, the higher the `lambda.min` and thus the penalisation. This leads to more sparse models. The number of `lambda` values can be chosen with an optional argument.

The values of `lambda` are automatically generated in each call of `glmnet`, based on the input data and model parameters, and may therefore differ between runs. Consequently, λ values from separate model fits cannot be directly compared unless a common sequence is explicitly defined. The corresponding mean cross-validated error is stored in the list `min.cvm`. The interpretation of its values depends on the argument `type.measure`.

The `type.measure` is the accuracy/error measure computed in cross-validation. It should be "class" (classification error) or "auc" (area under the ROC curve) if `family="binomial"`, "mse" (mean squared error) if `family="gaussian"` and "deviance" if `family="cox"`, which uses the partial-likelihood.

The `cvoffset` parameter is a logical flag that determines whether cross-validation (CV) should be used to estimate offsets during elastic net fitting. By default, its value is set to `FALSE`, meaning no CV is performed for offset estimation unless explicitly specified. If `cvoffset` is set to `TRUE`, the `cvoffsetnfolds` parameter becomes relevant, allowing the user to specify the number of folds for the CV procedure used in estimating the offsets. The default value for `cvoffsetnfolds` is 10, which means the data will be split into 10 folds during the CV process.

The adaptive argument in `priorityelasticnet` introduces an advanced layer of flexibility by enabling the adaptive elastic net, which enhances the standard elastic net through the use of data-driven adaptive weights. This methodology is explained in detail in Section 2.2.4. Additionally, the `initial_global_weight` option allows for further customisation by enabling users to apply a global weight across all predictors before fitting the adaptive elastic net. This argument is also discussed in detail in Section 2.2.4.

The main results of the fitted model from the `priorityelasticnet` are stored in an object returned by the functions provided by the package, which includes the coefficient estimates, tuning parameters, and related outputs. The estimated model parameters can be extracted using the `coef.priorityelasticnet` function, a method of the generic `coef()` function specifically designed to retrieve the coefficients from the fitted model object. It represents the relationship between the predictors and the response variable. Understanding these coefficients is essential for gaining insights into how each predictor influences the outcome, particularly in the context of penalised regression models where some coefficients may be shrunk towards zero or set to zero due to regularisation.

The `priorityelasticnet` package was developed under the assumption that practitioners have prior knowledge of the data, enabling them to specify the priorities. However, it may be that, especially in the presence of several blocks, it is not clear which block order is best.

For this reason, `cvm_priorityelasticnet` was implemented. This function allows the user to define multiple possible lists of blocks. The function selects the optimal block order based on the mean cross-validated error. Similarly, several vectors with maximal coefficients can be specified in `max.coef.list`. The following example demonstrates how to use the `cvm_priorityelasticnet` function to evaluate and compare different block configurations.

```
fit_cvm <- cvm_priorityelasticnet(
  X = X, Y = Y, weights = rep(1, n),
  foldid = sample(rep(1:10, length.out = n)),
  family = "gaussian", type.measure = "mse",
  blocks.list = list(list(bp1 = 1:10, bp2 = 11:30, bp3 = 31:500),
    list(bp1 = 1:10, bp2 = 31:500, bp3 = 11:30)),
  max.coef.list = list(c(Inf, Inf, Inf), c(Inf, 100, Inf)),
  block1.penalization = TRUE,
  lambda.type = "lambda.min",
  standardize = TRUE,
  nfolds = 10,
  cvoffset = TRUE,
  cvoffsetnfolds = 10,
  alpha = 0.6,
  adaptive = FALSE,
  initial_global_weight = FALSE)
```

In this example, two distinct block configurations, `blocks.list`, are defined and passed to the `cvm_priorityelasticnet` function. This function performs cross-validation on each configuration, calculating the mean squared error (MSE) for each model. Comparing these MSE values enables the identification of the block configuration that yields the best predictive performance. The output from `fit_cvm` provides detailed information regarding the performance of each block configuration, including cross-validated MSE values, the optimal lambda for each configuration, and the number of non-zero coefficients selected by the model. The elements contained in the R object `fit_cvm` can be summarised as follows:

- `fit_cvm$best.blocks`: provides the block configuration that achieves the best cross-validated performance in terms of MSE;
- `fit_cvm$best.blocks.indices`: provides the position (index) of the block configuration that achieves the best cross-validated result;
- `fit_cvm$best.max.coef`: provides the blockwise maximum number of non-zero coefficients allowed in the selected optimal configuration.

Handling missing data is crucial for constructing robust models, particularly when working with real-world datasets that include missing values. The `priorityelasticnet` package provides several options for handling missing data, allowing users to select the most suitable strategy based on the dataset's characteristics and analysis goals.

The `mcontrol` argument in `priorityelasticnet` allows for specifying how missing data should be handled. This flexibility ensures the model can be effectively fitted, even with incomplete data, which might otherwise result in biased estimates or diminished predictive power.

The `priorityelasticnet` package provides methods to handle blockwise missing data, which can be specified through the `mcontrol` argument using the `missing.control()` function:

```
missing.control(handle.missingdata = c("none", "ignore", "impute.offset"),
  ),
  offset.firstblock = c("zero", "intercept"),
  impute.offset.cases = c("complete.cases", "available.cases"),
  nfolds.imputation = 10,
  lambda.imputation = c("lambda.min", "lambda.1se"),
  perc.comp.cases.warning = 0.3,
  threshold.available.cases = 30,
  select.available.cases = c("maximise.blocks", "max"))
```


`handle.missingdata`: how blockwise missing data should be treated. The default is "none", meaning that no action is taken and missing values remain as provided, "ignore" ignores the observations with missing data for the current block, "impute.offset" imputes the offset for the missing values.

When `handle.missingdata="ignore"`, the elastic net is fitted to each block using only the observations without missing values in that block. This approach avoids data imputation and ensures all available data is utilised. However, for observations with missing values in the current block, no offset can be computed for the next block. To address this issue, the following strategies are implemented:

- If observations are missing in the current block, the offset (including the intercept) from the previous block is carried forward.
- If observations are missing in the first block, the offset for the second block is set either to 0 or to the estimated intercept of the first block.

The offset of a block b for an observation i is denoted as $\delta_{b,i}$, and is formally defined as:

$$\delta_{1,i} = \begin{cases} 0 & \text{if } x_{i1}^{(1)}, \dots, x_{ip_1}^{(1)} \text{ are missing} \\ \hat{\eta}_{1,i} \text{ or } \hat{\beta}_0^{(1)} & \text{if } x_{i1}^{(1)}, \dots, x_{ip_1}^{(1)} \text{ are not missing} \end{cases}$$

$$\delta_{b,i} = \begin{cases} \delta_{b-1,i} & \text{if } x_{i1}^{(b-1)}, \dots, x_{ip_{b-1}}^{(b-1)} \text{ are missing} \\ \hat{\eta}_{b-1,i} & \text{if } x_{i1}^{(b-1)}, \dots, x_{ip_{b-1}}^{(b-1)} \text{ are not missing} \end{cases}, \quad \text{for } b = 3, \dots, B,$$

where $\hat{\eta}_{b,i}$ is the prediction from block b for the i -th observation. `offset.firstblock`: determines if the offset of the first block for missing observations is zero or the intercept of the observed values for `handle.missingdata = "ignore"`.

For `handle.missingdata="impute.offset"`, it is similar to the previous method; the elastic net models applied to the individual blocks utilise only the available data, and no covariates are imputed. However, unlike the previous approach, the offset for observations with missing values is not retained from the earlier block; instead, it is imputed. This approach offers the benefit of imputing only a single value, rather than potentially a very large number of covariates. The concept behind this method is inspired by [Hieke et al. \(2016\)](#), which only imputes an offset. In their work, they utilise just three blocks, resulting in a fixed configuration for the imputation model. Here, we extend this idea into a generalised framework, introducing two distinct strategies for selecting which blocks to use. As Priority Elastic Net is an elastic net-based method, the imputation model uses an elastic net as well. Generally, any model that can predict a continuous outcome can be used as the imputation model. In the following sections, this is represented by only referencing a general regression model $F(\cdot)$. Let $F(\cdot)$ be a general regression model that can be trained by providing the dependent variable and the independent variables.

The argument `impute.offset.cases` defines which observations are used to construct the imputation model for missing offsets. Supported options are `complete.cases`, which limits the imputation to fully observed cases (with the additional constraint that each observation may contain at most one missing block), and `all.available.cases`, which uses all observations for which data are available in the current block.

The first approach (`impute.offset.cases="complete.cases"`) utilises all blocks except the current one to impute the offset derived from the current block. For this method to work, the dataset must include some observations without missing values (complete cases). A reasonable lower limit for the number of complete cases must be established. Based on the work of [Hieke et al. \(2016\)](#), who used 26 complete cases, the default threshold is set to 30 cases. Another limitation of this approach is that each observation with missing data can have at most one missing block, as all other blocks are required for the imputation model. A more formal description is provided below.

If only the complete cases are used for the imputation model and $I = \{1, \dots, n\}$ is the set of observation indices, the observations used for the imputation model are defined as:

$$I_{\text{comp,imp}} = \{i \in I \mid x_{i1}^{(b)}, \dots, x_{ip_b}^{(b)} \text{ are not missing } \forall b\},$$

where $x_{i1}^{(b)}, \dots, x_{ip_b}^{(b)}$ are the covariates of the b -th block of the i -th observation. Let $F(\cdot)$ be a general regression model that can be trained by providing the dependent variable and the independent variables. Then $\tilde{F}(\cdot)$ represents a trained regression model that returns its prediction when provided with the independent variables.

The algorithm below describes how, for an observation j with a missing block b , the corresponding linear predictor (offset) is imputed as $\hat{\kappa}_{\text{comp},b,j}$.

1. Identify the set of complete cases $I_{\text{comp,imp}}$, as defined above.
2. For all observations $i \in I_{\text{comp,imp}}$, compute the offset (linear predictor) for block b , denoted by $\hat{\kappa}_{\text{comp},b,i}$.
3. Train the regression model $\tilde{F}(\cdot)$ using the complete cases $I_{\text{comp,imp}}$, where the dependent variable is the offset $\hat{\kappa}_{\text{comp},b,i}$ and the covariates are the predictors from all other observed blocks, collectively denoted by $x_i^{(-b)}$ (that is, all blocks except the one currently missing block b).
4. For the observation j with missing block b , use the trained regression model $\tilde{F}(\cdot)$ to predict the value of the missing offset $\hat{\kappa}_{\text{comp},b,j}$.

The offset for a given observation i in block b is defined as:

$$\delta_{1,i} = 0,$$

$$\delta_{b,i} = \begin{cases} \hat{\eta}_{b-1,i}, & \text{if } x_{i1}^{(b-1)}, \dots, x_{ip_{b-1}}^{(b-1)} \text{ are observed,} \\ \hat{\kappa}_{\text{comp},b-1,i}, & \text{if } x_{i1}^{(b-1)}, \dots, x_{ip_{b-1}}^{(b-1)} \text{ are missing,} \end{cases}$$

for $b = 2, \dots, B$, $\forall i$ such that each observation i contains at most one missing block b ,

where:

- $\delta_{b,i}$ denotes the offset for block b and observation i ;
- $\hat{\eta}_{b-1,i}$ is the prediction obtained from block $b - 1$ for observation i ;
- $\hat{\kappa}_{\text{comp},b-1,i}$ represents the predicted (compensatory) offset used when predictors in block $b - 1$ are missing.

The previous approach consistently relies on all other blocks to impute the offset for a missing block. However, in many datasets, there may be no observations with complete cases, or observations may have more than one block missing. To address these situations, a more adaptable method was developed (`impute.offset.cases="all.available.cases"`). This approach segments the observations into different subsets based on their missingness patterns. A missingness pattern indicates which specific combination of blocks is missing for a particular observation. For each unique pattern, a separate imputation model is constructed and fitted.

$$\delta_{1,i} = 0$$

$$\delta_{b,i} = \begin{cases} \hat{\eta}_{b-1,i}, & \text{if } x_{i1}^{(b-1)}, \dots, x_{ip_{b-1}}^{(b-1)} \text{ are not missing,} \\ \hat{\kappa}_{\text{comp},b-1,i}, & \text{if } x_{i1}^{(b-1)}, \dots, x_{ip_{b-1}}^{(b-1)} \text{ are missing,} \end{cases}$$

for $b = 2, \dots, B$, $\forall i$ such that every observation i misses at most 1 block b .

The remaining arguments of the `missing.control` function are described below:

- `nfolds.imputation` : nfolds for the `glmnet` of the imputation model;
- `lambda.imputation` : which lambda-value should be used for predicting the imputed offsets in `cv.glmnet`;
- `perc.comp.cases.warning`: percentage of complete cases when a warning is issued of too few cases for the imputation model.
- `threshold.available.cases`: if the number of available cases for `impute.offset.cases = available.cases` is below this threshold, `priorityelasticnet` tries to reduce the number of blocks included in the imputation model to increase the number of observations used for imputation;
- `select.available.cases`: determines how the blocks which are used for the imputation model are selected when `impute.offset.cases = available.cases`. `max` selects the blocks that maximise the number of observations, `maximise.blocks` tries to include as many blocks as possible, starting with the highest-priority ones.

Below, we demonstrate how to configure the `mcontrol` argument to handle missing data by imputing offsets.

In this example, the `handle.missingdata = "impute.offset"` option, which is part of the missing data handling mechanism, enables the `priorityelasticnet` function to impute missing values using an offset approach. This method is particularly advantageous for handling sporadic missing data, as it allows the model to utilise all available information without discarding incomplete observations.

```
mcontrol <- missing.control(handle.missingdata = "impute.offset", nfolds.imputation = 10)

fit_missing <- priorityelasticnet(
  X, Y,
  family = "gaussian",
  type.measure = "mse",
  blocks = blocks,
  mcontrol = mcontrol)
```

The output includes details on how the missing data was handled, the imputation models employed (if applicable), and the overall model fit.

The `predict.priorityelasticnet` function is used to generate predictions from a fitted model. This function can produce different types of predictions depending on the specified type parameter, including linear predictors, fitted values, or class probabilities (for classification models). The `handle.missingtestdata` parameter manages missing values in test data, with options such as `"none"`, `"omit.prediction"`, `"set.zero"`, or `"impute.block"`. In the next section, each of these options is explained in detail, including their functionality and the scenarios in which they are most appropriate. The `include.allintercepts` parameter determines whether intercepts from all blocks are included in the prediction (TRUE) or only from the first block (FALSE by default). The `use.blocks` parameter allows customising the blocks used for prediction, defaulting to `"all"` but allowing specification of block numbers. Additional customisation can be achieved through further arguments, making the function versatile for various prediction scenarios, including handling missing data and tailoring block selection to specific needs.

2.2 Priority elastic net for a multinomial logistic regression

Regularised multinomial logistic regression models are widely used in multi-class classification, particularly in high-dimensional data settings. They are instrumental in applications

such as bioinformatics and cancer research, where sparsity and interpretability are critical. However, existing regularisation methods often fail to incorporate domain-specific priorities among predictors. To address this, the Priority Elastic Net method, initially developed for binary logistic regression (Musib et al., 2024), is extended to multinomial logistic regression. This novel extension enables the classification of multiple classes by iteratively fitting blocks of predictors with specific priorities. Grounded in well-established logistic regression principles, the multinomial model reduces to standard logistic regression in the binary case ($K = 2$), where the logit function models the probability of the first class relative to the second. By preserving the block-prioritised variable selection structure, this method provides a flexible and interpretable framework for multiclass problems, making it particularly suited for high-dimensional applications.

Let $\mathbf{X}$, the predictor matrix, be an $n \times p$ matrix, where $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{ip})^T$ is the i th row of $\mathbf{X}$, i.e., the i th observation. The response vector $\mathbf{Y} = (Y_1, Y_2, \dots, Y_n)^T$ is an $n \times 1$ vector, where the response Y_i is categorical, taking a value $y_i \in \{1, 2, \dots, K\}$ for $K \geq 3$. This concept has been effectively modelled in previous research, including the works of (Zhu and Hastie, 2004) and (Friedman et al., 2010), with

$$P(Y_i = \ell \mid \mathbf{x}_i) = \frac{\exp(\beta_{0,\ell} + \mathbf{x}_i^T \boldsymbol{\beta}_\ell)}{\sum_{k=1}^K \exp(\beta_{0,k} + \mathbf{x}_i^T \boldsymbol{\beta}_k)}, \quad \ell = 1, \dots, K, \quad i = 1, \dots, n. \quad (9)$$

where $\boldsymbol{\beta}_\ell = (\beta_{1\ell}, \dots, \beta_{p\ell})^T$ are coefficients estimated by minimizing the following negative log-likelihood function for observations $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where each y_i is encoded as $\mathbf{y}_i = (y_{i1}, \dots, y_{iK})^T$ with $y_{ik} = 1_{y_i=k}$:

$$\ell(\boldsymbol{\beta}) = -\frac{1}{n} \sum_{i=1}^n \log P(Y_i = y_i \mid \mathbf{x}_i) + \lambda F_\alpha(\boldsymbol{\beta}) \quad (10)$$

where $\boldsymbol{\beta}$ denotes the matrix of dimension $(p+1) \times K$, with the parameters $\beta_{0,\ell}$ and $\boldsymbol{\beta}_\ell$ in its columns.

In Eq. (10), y_i represents the observed level of the variable Y_i and the last term represents the penalty function, i.e., the term ensuring sparsity in the model, controlled by the regularisation parameter λ .

$$F_\alpha(\boldsymbol{\beta}) = \sum_{\ell=1}^K \left(\alpha \|\boldsymbol{\beta}_\ell\|_1 + \frac{1-\alpha}{2} \|\boldsymbol{\beta}_\ell\|_2^2 \right), \quad (11)$$

where $\lambda > 0$ is a user-specified regularization parameter.

The penalised log-likelihood equation for the elastic-net penalty (Friedman et al., 2023) is given by

$$\begin{aligned} \tilde{\ell}(\boldsymbol{\beta}, \lambda, \alpha) = & -\frac{1}{n} \sum_{i=1}^n \left(\sum_{k=1}^K y_{ik} (\beta_{0,k} + \mathbf{x}_i^T \boldsymbol{\beta}_k) - \log \left(\sum_{k=1}^K \exp(\beta_{0,k} + \mathbf{x}_i^T \boldsymbol{\beta}_k) \right) \right) \\ & + \lambda \sum_{\ell=1}^K \left(\alpha \|\boldsymbol{\beta}_\ell\|_1 + \frac{1-\alpha}{2} \|\boldsymbol{\beta}_\ell\|_2^2 \right), \end{aligned} \quad (12)$$

where the response y_{ik} is given by an indicator function,

$$y_{ik} = I(y_i = k) = \begin{cases} 1, & \text{if } y_i = k \\ 0, & \text{otherwise} \end{cases}.$$

The elastic net or lasso estimate $\hat{\boldsymbol{\beta}}(\lambda, \alpha)$ can be computed in R using the `glmnet` package (Friedman et al., 2023), which uses coordinate descent to minimise the penalised negative log-likelihood.

The coordinate descent method is used to obtain the coefficient estimates in regularised multinomial regression. Following [Friedman et al. \(2010\)](#), the optimisation procedure employs partial Newton steps, obtained by forming a partial quadratic approximation, together with the coordinate descent method.

The `glmnet` package addresses the challenges introduced by regularisation in traditional multinomial logistic regression using a symmetric representation. In traditional models, coefficients for categories are modelled relative to an arbitrary base category, a choice that does not affect fitted probabilities but can lead to sensitivity under regularisation due to penalisation effects. By avoiding reliance on a specific base category, `glmnet` ensures robust and stable coefficient estimation, eliminating asymmetries arising from base-category selection. This symmetric approach is particularly valuable for methods like the Priority Elastic Net, which incorporate additional structure into the regularisation process. Adopting a symmetric representation in such methods can enhance interpretability and maintain model consistency.

Building upon these principles, the Priority Elastic Net method, an extension of the priority-lasso approach developed by [Klau et al. \(2018\)](#), applies elastic net regularisation iteratively across ordered blocks of variables, fitting each block while using the results of higher-priority blocks as offsets.

The iterative fitting of the multinomial regression model follows the same procedure as described in the previous section for the Gaussian family. Once the fitting has been carried out for all B blocks under consideration, we obtain the set of coefficient estimates $\hat{\beta}^{(\pi_b)}$ for $b = 1, \dots, B$ and the final fitted linear predictor becomes

$$\hat{\eta}_i^{(k)}(\boldsymbol{\pi}) = \sum_{b=1}^B \hat{\beta}_{0,k}^{(\pi_b)} + \sum_{b=1}^B \sum_{j=1}^{p_{\pi_b}} \hat{\beta}_{j,k}^{(\pi_b)} x_{ij}^{(\pi_b)}, \quad k = 1, \dots, K. \quad (13)$$

The final Priority Elastic Net multinomial logistic regression model, after processing all B blocks, models the probability of the outcome Y_i belonging to class k as:

$$P(Y_i = k | \mathbf{x}_i) = \frac{\exp\left(\sum_{b=1}^B \hat{\eta}_{b,i}^{(k)}(\boldsymbol{\pi})\right)}{\sum_{l=1}^K \exp\left(\sum_{b=1}^B \hat{\eta}_{b,i}^{(l)}(\boldsymbol{\pi})\right)}. \quad (14)$$

However, the linear score $\hat{\eta}_{1,i}^{(k)}(\boldsymbol{\pi})$ tends to be over-optimistic with respect to the information usable for predicting Y_i that is contained in block π_1 . The reason for this is that Y_i was part of the data used for obtaining the estimates $\hat{\beta}_{j,k}^{(\pi_1)}$ for $j = 1, \dots, p_{\pi_1}$, which are then used to calculate $\hat{\eta}_{1,i}^{(k)}(\boldsymbol{\pi})$.

When using this over-optimistic estimate $\hat{\eta}_{1,i}^{(k)}(\boldsymbol{\pi})$ as an offset in the second step, the influence of block π_2 conditional on the influence of block π_1 will tend to be underestimated. The reason for this is that by considering the over-optimistic estimate $\hat{\eta}_{1,i}^{(k)}(\boldsymbol{\pi})$ as an offset, a part of the variability in Y_i is removed that is actually not explainable by block π_1 but would possibly be explained by block π_2 . As noted above, this problem results from the fact that Y_i is contained in the training data used for estimating $\hat{\beta}_{j,k}^{(\pi_1)}$ for $j = 1, \dots, p_{\pi_1}$. As a solution to this problem we suggest estimating the offsets $\hat{\eta}_{1,i}^{(k)}(\boldsymbol{\pi})$ using cross-validation.

$$\hat{\eta}_{b,i}^{(k)}(\boldsymbol{\pi})_{CV} = \hat{\eta}_{b-1,i}^{(k)}(\boldsymbol{\pi})_{CV} + \hat{\beta}_{0,k}^{(\pi_b)} + \hat{\beta}_{1,k}^{(\pi_b)} x_{i1}^{(\pi_b)} + \dots + \hat{\beta}_{p_{\pi_b},k}^{(\pi_b)} x_{ip_{\pi_b}}^{(\pi_b)}, \quad k = 1, \dots, K, \quad (15)$$

where $\hat{\eta}_{0,i}^{(k)}(\boldsymbol{\pi})_{CV} = 0$ and $b = 1, \dots, B$.

In the following analysis, we address the challenge of fitting a multinomial regression model with predictors organised into distinct blocks. To this end, we employ the Priority Elastic Net, which adaptively tunes block-specific penalties to balance sparsity and model accuracy by combining ridge and lasso regularisation.

The process began with the simulation of synthetic data comprising 100 observations, 500 predictors, and a categorical response variable with three classes. To reflect a realistic scenario, the predictors were divided into three blocks, where block-specific penalties could be applied. This setting is particularly well suited for investigating the joint influence of groups of predictors, each containing a large number of variables, on the response variable.

```
set.seed(123)
n <- 100 # Number of observations
p <- 500 # Number of predictors
k <- 3   # Number of classes

# Simulate a matrix of predictors
x <- matrix(rnorm(n * p), n, p)

# Simulate a response vector with three classes
y <- factor(sample(1:k, n, replace = TRUE))

# Define predictor blocks
blocks <- list(block1 = 1:10, block2 = 11:30, block3 = 31:500)

# Fit a multinomial model using priorityelasticnet
fit_multinom <- priorityelasticnet(
  X = x, Y = y,
  family = "multinomial",
  alpha = 0.5,
  type.measure = "class",
  blocks = blocks,
  block1.penalization = TRUE,
  lambda.type = "lambda.min",
  standardize = TRUE,
  nfolds = 10,
  adaptive = FALSE,
  initial_global_weight = FALSE)
```

2.3 Priority elastic net for a Cox regression

Survival analysis studies the time until an event of interest occurs and is applied in various fields of science, especially in medical research. It provides statistical methods for investigating the time until an event occurs, such as death, disease recurrence, or the length of stay, defined as the time spent in the hospital during a single admission episode. However, not all event times are observed within the study period, leading to censored observations.

Let $\mathbf{y} = (y_1, \dots, y_n)$ represent the observed survival times, and $\delta = (\delta_1, \dots, \delta_n)$ indicate the censoring indicator, where $\delta_i = 1$ if the event of interest occurs for subject i and $\delta_i = 0$ if the observation is censored. Let $\mathbf{X}$ be the $n \times p$ design matrix, where p is the number of features, each row represents observations on one subject, and each column represents values of one feature across the n subjects. Our goal is to develop a predictive model for the survival times, $\mathbf{y}$, based on the design matrix $\mathbf{X}$.

The Cox regression model (Cox, 1972) is widely used to analyse survival data, mainly because of its flexibility in handling censored observations and its semi-parametric nature, which does not require specifying the form of the baseline hazard function. The hazard function $h(t)$ at time t for the i th patient is defined as:

$$h(t, \mathbf{X}_i) = h_0(t) \exp(\mathbf{X}_i^T \boldsymbol{\beta}), \quad (16)$$

where $h(t, \mathbf{X}_i)$ is the hazard for the i th observation at time t , $h_0(t)$ is an unspecified baseline hazard function, and $\boldsymbol{\beta} = (\beta_1, \beta_2, \dots, \beta_p)^T \in \mathbb{R}^p$ is the vector of the regression coefficients to be estimated (Gui and Li, 2005).

Based on the available sample data, Cox's partial likelihood (Cox, 1972) can be written as

$$L(\boldsymbol{\beta}) = \prod_{i=1}^n \left[\frac{\exp(\mathbf{X}_i^T \boldsymbol{\beta})}{\sum_{j \in R(t_i)} \exp(\mathbf{X}_j^T \boldsymbol{\beta})} \right]^{\delta_i}, \quad (17)$$

where δ_i is an indicator for censoring, t_i is the survival time (observed or censored) of the i -th sample and $R(t_i) = \{j : t_j \geq t\}$ denotes the set of individuals at risk at time t_i (Simon et al., 2011). To estimate the regression coefficients $\boldsymbol{\beta}$ in Eq. (17), we need to maximise its log partial likelihood function,

$$l(\boldsymbol{\beta}) = \log L(\boldsymbol{\beta}) = \sum_{i=1}^n \delta_i \left\{ \mathbf{X}_i^T \boldsymbol{\beta} - \log \left[\sum_{j \in R(t_i)} \exp(\mathbf{X}_j^T \boldsymbol{\beta}) \right] \right\} \quad (18)$$

In the `glmnet` package, the negative log of the partial likelihood is penalised using an elastic net regularisation term.

By adding the regularization term $P(\boldsymbol{\beta}; \lambda)$ to the negative of Eq. (18) and minimising the resulting sum, we obtain the regularised Cox proportional hazards model:

$$\tilde{\ell}(\boldsymbol{\beta}) = \arg \min_{\boldsymbol{\beta}} [-l(\boldsymbol{\beta}) + P(\boldsymbol{\beta}; \lambda)], \quad (19)$$

where $P(\boldsymbol{\beta}; \lambda)$ is the loss function and λ is the tuning parameter.

The model in Eq. (19) performs variable selection and coefficient estimation simultaneously by shrinking some components of the regression coefficient vector $\boldsymbol{\beta}$ towards zero, with some coefficients that may be set exactly to zero.

Therefore, the penalised negative log of the partial likelihood is given by:

$$\tilde{\ell}(\boldsymbol{\beta}) = - \sum_{i=1}^n \delta_i \left\{ \mathbf{X}_i^T \boldsymbol{\beta} - \log \left[\sum_{j \in R(t_i)} \exp(\mathbf{X}_j^T \boldsymbol{\beta}) \right] \right\} + \lambda \mathcal{P}(\boldsymbol{\beta}), \quad (20)$$

where,

$$\mathcal{P}(\boldsymbol{\beta}) = \alpha \|\boldsymbol{\beta}\|_1 + \frac{(1-\alpha)}{2} \|\boldsymbol{\beta}\|_2^2.$$

Building on regularised Cox regression, the method applies an elastic net penalty that allows variable- or group-specific weighting to account for their relative importance during penalisation.

In time-to-event analyses, variables can often be grouped by shared biological, clinical, or measurement characteristics. Accounting for this hierarchical structure is particularly valuable in survival models, where relationships with the outcome may vary across groups. The Priority Elastic Net accommodates such structures by assigning group-level weights, enabling blockwise prioritisation and shrinkage while maintaining within-group flexibility. This hierarchical approach aligns the model with the data's natural organisation, enhancing interpretability and predictive performance.

The detailed procedure for implementing the Priority Elastic Net for Cox regression, including the penalisation steps and coefficient selection, is provided in Algorithm 1. This algorithm outlines the step-by-step approach to ensure accurate feature selection and model fitting.

As before, a simulation of synthetic data considering $n = 200$ and $p = 600$ using `priorityelasticnet` package for the Cox regression is illustrated. In the example, the data was split into training and test sets to provide a more realistic evaluation of model performance and to reduce overoptimism that may occur when assessing the model on the same data used for fitting.

```
library(survival)
n <- 200
```



```

p <- 600
nzc <- trunc(p / 10)

x <- matrix(rnorm(n * p), n, p)
beta <- rnorm(nzc)
fx <- x[, seq(nzc)] %*% beta / 3
hx <- exp(fx)
ty <- rexp(n, hx)
tcens <- rbinom(n = n, prob = .3, size = 1)
y <- Surv(ty, 1 - tcens)

train_index <- sample(1:n, size = n * 0.7)
x_train <- x[train_index, ]
x_test <- x[-train_index, ]
y_train <- y[train_index]
y_test <- y[-train_index]

blocks <- list(bp1 = 1:20, bp2 = 21:300, bp3 = 301:600)

fit_cox <- priorityelasticnet(
  x_train, y_train,
  family = "cox",
  alpha = 0.5,
  type.measure = "deviance",
  blocks = blocks,
  block1.penalization = TRUE,
  lambda.type = "lambda.min",
  standardize = TRUE,
  nfolds = 10,
  adaptive = FALSE,
  initial_global_weight = FALSE,
  cvoffset = TRUE)

```

In this example, Kaplan–Meier curves were used to visualise and compare the survival probabilities of two distinct risk groups—High-Risk and Low-Risk—in both training and test sets (Figure 1). These groups are determined based on the results of the Priority Elastic Net Cox proportional hazards model, which incorporates variable prioritisation to enhance the interpretability and accuracy of survival predictions.

The grouping and visualisation process is facilitated through the `separate2GroupsCox` function from the `glmSparseNet` package (Veríssimo et al., 2018). This function uses the linear predictor (risk score) from the fitted Cox model to divide individuals into two risk categories, typically using the median risk score as the cutoff. It then provides a graphical comparison of survival probabilities between the resulting high- and low-risk groups, thereby illustrating the impact of the identified risk factors on survival.

```

library(glmSparseNet)
y_train_df <- data.frame(time = ty[train_index], status = 1 - tcens[train_index])

# Group patients and plot Kaplan-Meier survival curves for the training set
separate2GroupsCox(
  chosen.btas = fit_cox$coefficients,
  xdata = x_train, ydata = y_train_df,
  probs = c(0.4, 0.6),
  no.plot = FALSE,
  plot.title = "Training Set Survival Curves",
  xlim = NULL,
  ylim = NULL,
  expand.yzero = FALSE,
  legend.outside = FALSE)

```

Algorithm 1 Fitting Cox regression with Priority Elastic Net penalty

- 1: Let x_{ij} be predictor j for subject i ; $y_i = (t_i, \delta_i)$ the survival outcome, $i = 1, \dots, n$.
- 2: Partition predictors into B ordered blocks according to priority $\boldsymbol{\pi} = (\pi_1, \dots, \pi_B)$.
- 3: Choose elastic net mixing parameter $\alpha \in [0, 1]$ and number of folds K .
- 4: Construct a decreasing regularisation path $\lambda_1 > \dots > \lambda_m$.
- 5: Split $\{1, \dots, n\}$ into folds $\{S_k\}_{k=1}^K$ and initialise $\hat{\eta}_{0,i}(\boldsymbol{\pi})_{CV} = 0$.
- 6: **for** $b = 1, \dots, B$ in priority order **do**
- 7: Let $g = \pi_b$ denote the active block with predictors $X^{(g)}$.
- 8: **for** $\ell = 1, \dots, m$ **do**
- 9: For each fold k , fit the penalised Cox model on $S \setminus S_k$

$$\hat{\beta}_{-k}^{(g)}(\lambda_\ell) = \arg \min_{\beta^{(g)}} \left\{ -\ell_{\text{Cox}}^{S \setminus S_k}(\hat{\eta}_{b-1}(\boldsymbol{\pi})_{CV} + X^{(g)}\beta^{(g)}) + \lambda_\ell \sum_{j \in g} \left(\frac{1-\alpha}{2} \beta_j^2 + \alpha |\beta_j| \right) \right\}.$$

- 10: Using observations in fold S_k , compute the cross-validated deviance $CV_b(\lambda_\ell)$.
- 11: **end for**
- 12: Select the block-specific tuning parameter

$$\lambda_{b,\min} = \arg \min_{\lambda_\ell} CV_b(\lambda_\ell).$$

- 13: Update cross-validated offsets for all subjects

$$\hat{\eta}_{b,i}(\boldsymbol{\pi})_{CV} = \hat{\eta}_{b-1,i}(\boldsymbol{\pi})_{CV} + \sum_{j=1}^{p_g} x_{ij}^{(g)} \hat{\beta}_{-k(i),j}^{(g)}(\lambda_{b,\min}),$$

where $k(i)$ is the fold containing subject i .

- 14: **end for**
- 15: Final linear predictor: $\hat{\eta}_{B,i}(\boldsymbol{\pi})_{CV}$.
- 16: Estimate the hazard function for all subjects:

$$\hat{h}_i(t) = h_0(t) \exp(\hat{\eta}_{B,i}(\boldsymbol{\pi})_{CV}), \quad i = 1, \dots, n.$$

```
# Prediction on the test set
pred_test <- predict(fit_cox, newx = x_test, type = "link")
y_test_df <- data.frame(time = ty[-train_index], status = 1 - tcens[-
  train_index])

# Group patients and plot Kaplan-Meier survival curves for the test set
separate2GroupsCox(
  chosen.btas = fit_cox$coefficients,
  xdata = x_test, ydata = y_test_df,
  probs = c(0.4, 0.6),
  no.plot = FALSE,
  plot.title = "Test Set Survival Curves",
  xlim = NULL,
  ylim = NULL,
  expand.yzero = FALSE,
  legend.outside = FALSE)
```

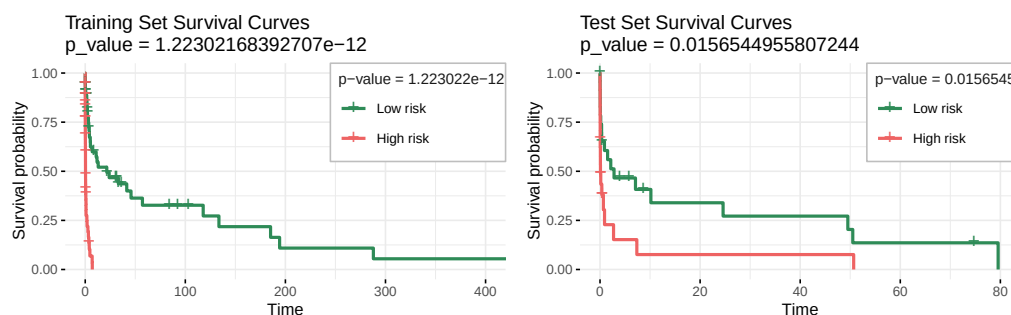


Figure 1: The Kaplan-Meier curves in this example are used to visualize and compare survival probabilities, for the train set (left) and for the test set (right), between two risk groups (High Risk and Low Risk) identified using the Priority Elastic Net Cox proportional hazards model and using function `separate2GroupsCox` from `glmSparseNet` package.

2.4 Priority-Adaptive elastic net

The potential inconsistency of the lasso method in specific scenarios has been explicitly highlighted, as demonstrated by [Zou and Hastie \(2005\)](#). In response, [Zou \(2006\)](#) proposed a modified version of lasso, known as the adaptive lasso and explicitly proved its consistency in terms of oracle properties asymptotically. Similar ideas are explored in many other papers, namely [Zhang and Lu \(2007\)](#) and [Wang et al. \(2007\)](#). Apart from lack of consistency, lasso has other drawbacks as discussed in [Zou and Hastie \(2005\)](#), for example, (a) lasso does not encourage grouped selection in case of high pairwise correlation among covariates and (b) for $p > n$ case, lasso can select at most n covariates. To overcome the above drawbacks, they introduced the elastic net, which combines both ridge ℓ_2 and lasso ℓ_1 penalties. However, the elastic net is not an oracle procedure. Notably, to solve the original elastic net problem [Zou and Hastie \(2005\)](#), transform the elastic net into an ordinary lasso-type problem in some augmented space by some algebraic manipulation. Since this is a one-to-one mapping, whenever the lasso is inconsistent in the augmented space, so is the underlying elastic net. For a more detailed discussion on this topic, please refer to [Jia and Yu \(2010\)](#), which states an explicit condition for the inconsistency of the elastic net. Instead of combining the ordinary lasso penalty, the adaptive lasso penalty was integrated with the ridge penalty. This doubly regularised approach, referred to as the adaptive elastic net, aims to leverage the strengths of both penalties.

Another direction for improvement is to correct bias in estimation by adding a weight w_j for each β_j , corresponding to different amounts of shrinkage, to the regression coefficients. The adaptive elastic net was introduced by [Zou and Zhang \(2009\)](#) and [Ghosh \(2011\)](#), combining the ℓ_2 norm regularisation with adaptive weights to enhance estimation accuracy.

In this context, the elastic net estimates $\hat{\beta}_{EN}$ are first computed as defined in (1), and then adaptive weights are constructed as follows:

$$\hat{w}_j = (|\hat{\beta}_{EN}|)^{-\gamma}, \quad j = 1, 2, \dots, p \quad (21)$$

where γ is a positive constant. Using $\gamma = 1$ for calculating adaptive weights in the Adaptive-Elastic net is a common and effective choice because it provides a balance between penalising weak predictors and retaining strong ones. Since the elastic net naturally adopts a sparse representation, the weights can also be computed as $\hat{w}_j = \left(|\hat{\beta}_{j(EN)}| + \frac{1}{n}\right)^{-\gamma}$ to avoid division by zero ([Zou and Zhang, 2009](#)).

$$\hat{\beta}_{(AENet)} = \arg \min_{\beta} \left[\frac{1}{n} \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p \hat{w}_j |\beta_j| + \frac{(1-\alpha)}{2} \sum_{j=1}^p \beta_j^2 \right]. \quad (22)$$

Building upon the expression in Equation (22), the framework for the Priority-Adaptive Elastic Net is established by calculating the adaptive weights for the variables based on the assigned priority of the block. Initially, the adaptive weights, denoted as $\hat{w}_j^{(\pi)}$ and introduced earlier in (21), are computed for each variable based on the specified priority of the block. In estimating coefficients with the priority-Adaptive Elastic Net, the adaptive weights $\hat{w}_j^{(\pi_1)}$ for the first priority block are incorporated into the ℓ_1 penalty.

The priority-Adaptive Elastic Net model is formulated by minimising the following objective function for the first block priority:

$$l(\beta^{(\pi_1)}) = \sum_{i=1}^n \left(y_i - \sum_{j=1}^{p_{\pi_1}} x_{ij}^{(\pi_1)} \beta_j^{(\pi_1)} \right)^2 + \lambda^{(\pi_1)} \left(\alpha \sum_{j=1}^{p_{\pi_1}} \hat{w}_j^{(\pi_1)} |\beta_j^{(\pi_1)}| + \frac{(1-\alpha)}{2} \sum_{j=1}^{p_{\pi_1}} \beta_j^{2(\pi_1)} \right). \quad (23)$$

Here, the objective function combines a weighted lasso penalty, controlled by α and the adaptive weights $\hat{w}_j^{(\pi_1)}$, with a ridge penalty, balancing sparsity and regularisation.

Using the fitted values from this step as an offset to fit a model considering the covariates in the block with the second highest priority:

$$\hat{\eta}_{1,i}(\pi) = \hat{\beta}_1^{(\pi_1)} x_{i1}^{(\pi_1)} + \dots + \hat{\beta}_{p_{\pi_1}}^{(\pi_1)} x_{ip_{\pi_1}}^{(\pi_1)}. \quad (24)$$

For $b = 2, \dots, B$ (where B is the number of blocks), the model fitting proceeds iteratively. Using the fitted values from Step $b - 1$ as an offset, the priority-adaptive elastic net model is then fitted to the covariates in the block with the b -th highest priority.

The implementation of the priority-Adaptive Elastic Net model can be performed using the `priorityelasticnet` function, as demonstrated in the following code chunk. In this example, the model is applied to Gaussian data with the specified parameters:

```
fit_gaussian <- priorityelasticnet(
  X = X, Y = Y,
  family = "gaussian",
  blocks = blocks,
  block1.penalization = TRUE,
  standardize = TRUE,
  lambda.type = "lambda.min",
  max.coef = c(Inf, Inf, Inf),
  nfolds = 10,
  adaptive = TRUE,
  type.measure = "mse",
```

```
initial_global_weight = FALSE,
alpha = 0.6,
cvoffset = TRUE,
cvoffsetnfolds = 10)
```

In the `priorityelasticnet` and `cvm_priorityelasticnet` functions, with the latter presented below, the `adaptive` argument controls whether adaptive weights are computed for each block of variables based on their assigned priorities and the estimated coefficient magnitudes. When `adaptive = TRUE`, the penalty term becomes data-driven, allowing stronger penalisation of less influential variables and thereby improving variable selection stability and model interpretability. Conversely, when `adaptive = FALSE`, all variables within a block are penalised equally, and the model applies fixed, non-data-dependent penalties.

The `initial_global_weight` argument determines whether global weights across all variables are computed before applying block-specific priorities. Setting `initial_global_weight = TRUE` enables this global assessment step, which is advantageous in most applications because it first derives global weights across all variables and then refines them according to block-specific priorities, capturing both overall and block-level relevance. However, when blocks differ substantially in scale or variability, setting `initial_global_weight = FALSE` may be preferable, allowing weights to be computed independently within each block and avoiding potential distortion from global weighting. An example of this situation arises when fitting a model that combines clinical, imaging, and genomic data, where it may be important for each block to determine its own internal importance independently of global patterns dominated by larger blocks.

In summary, when both arguments are set to `TRUE`, the refinement proceeds in two steps: global adaptive weights are first computed across all variables and are then rescaled according to the block-specific priorities. This two-stage adjustment ensures that both global relevance and block-level importance are reflected in the final regularisation process.

```
cvm_priorityelasticnet(
  X = matrix(rnorm(100 * 500), 100, 500),
  Y = rnorm(100),
  family = "gaussian",
  alpha = 0.6,
  type.measure = "mse",
  lambda.type = "lambda.min",
  nfolds = 10,
  block1.penalization = TRUE,
  adaptive = TRUE,
  initial_global_weight = FALSE,
  standardize = TRUE,
  cvoffset = TRUE,
  cvoffsetnfolds = 10,
  blocks.list = list(list(bp1 = 1:10, bp2 = 11:30, bp3 = 31:500),
                     list(bp1 = 1:10, bp2 = 31:500, bp3 = 11:30)),
  max.coef.list = list(c(Inf, Inf, Inf), c(Inf, 100, Inf)))
```

After running the function with the specified parameters, the resulting object contains several key outputs, including the best block configuration determined during cross-validation, which differs from the Priority Elastic Net results.

3 Summary

Priority Elastic Net is designed to address certain limitations of the Priority-Lasso. Although Priority-Lasso is effective at promoting sparsity, it can perform poorly in the presence of multicollinearity, as it tends to select a single predictor from a set of correlated variables, thereby reducing model stability. The prioritisation framework implemented in Priority Elastic Net allows different block orderings to be evaluated, thereby enabling the contribution of lower-priority predictor groups to be assessed.

The proposed Priority Elastic Net framework (Musib et al., 2025) incorporates the elastic net penalty, balancing ℓ_1 and ℓ_2 regularisation. The package also includes the adaptive elastic net formulation that improves variable selection, stable coefficient estimation, and robust performance in high-dimensional, correlated datasets. Thus, the Priority-adaptive Elastic Net may provide a useful approach for addressing the challenges of modern, complex datasets, with applications in genomics, proteomics, and large-scale predictive modelling. We have thoroughly discussed in Musib et al. (2024) the advantages of the Priority Elastic Net framework for binary logistic regression.

The sequential fitting procedure of the **Priority Elastic Net** can be computationally demanding, particularly when cross-validated offsets are used (`cvoffset = TRUE`). Although the current implementation runs sequentially, future versions of the **priorityelasticnet** package will incorporate parallel execution of cross-validation folds and explore more efficient cross-validation strategies to enhance scalability in high-dimensional, multi-block settings.

In addition, the novel offset imputation strategy, used to enable cross-validation within the sequential block-fitting procedure, may introduce statistical bias that has not yet been formally assessed. As the imputed offsets replace model predictions from higher-priority blocks, any systematic deviation between the imputed and true offsets could propagate through subsequent steps, potentially influencing coefficient estimation or model selection. This aspect will be investigated in future methodological work, both theoretically and through simulation studies. A comprehensive vignette included in the R package provides a detailed walkthrough of the main functions, examples, and use cases, helping users apply the method to their own data.

Further research will also focus on developing simulation studies to systematically compare the predictive accuracy and variable selection stability of the Priority Elastic Net, the Priority Lasso, and the standard Elastic Net across the regression contexts considered in this paper. Real-world applications will also be explored to assess the relative performance of the Priority Elastic Net in practice.

Methodological extensions under consideration include incorporating additional penalisation options, such as SCAD (Smoothly Clipped Absolute Deviation), MCP (Minimax Concave Penalty), and other non-convex penalties, to enhance flexibility and applicability across diverse datasets. These advanced regularisation techniques can mitigate some limitations of convex penalties, offering improved variable selection and estimation in complex scenarios. Finally, extending the package to support multi-response regression models will enable the simultaneous prediction of multiple dependent variables—a valuable feature for applications in genomics, proteomics, and other domains involving high-dimensional, multi-output data. Collectively, these developments aim to broaden the scope and practical utility of the package for researchers working with modern, multifaceted datasets.

Acknowledgments

This work is funded by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the project FCT Tenure program, 2023.15441.TENURE.033/CP00003/CT00010, and under under CEAUL Research Unit, UID/00006/2025, DOI: <https://doi.org/10.54499/UID/00006/2025> and by the European Union – NextGenerationEU through the project UID/PRR/00006/2025, DOI: <https://doi.org/10.54499/UID/PRR/00006/2025>.

The authors acknowledge the developers of the **prioritylasso** package (Klau et al., 2018, 2017), whose implementation served as the basis for several functions included in the **priorityelasticnet** package.

References

- T. B. Arnold, R. J. Tibshirani, M. T. Arnold, and T. ByteCompile. Package ‘genlasso’. *Statistics*, 39(3):1335–1371, 2022. doi: 10.32614/CRAN.package.genlasso. [p346]
- A. Boulesteix and M. Fuchs. ipflasso: Integrative lasso with penalty factors. *R package version 0.1.*, 2015. URL <https://CRAN.R-project.org/package=ipflasso>. [p347]
- A.-L. Boulesteix, R. De Bin, X. Jiang, and M. Fuchs. IPF-LASSO: integrative L1-penalized regression with penalty factors for prediction based on multi-omics data. *Computational and Mathematical Methods in Medicine*, 2017(1):7691937, 2017. doi: 10.1155/2017/7691937. [p347]
- D. R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(2):187–202, 1972. doi: 10.1111/j.2517-6161.1972.tb00899.x. [p359, 360]
- J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):22, 2010. doi: 10.18637/jss.v033.i01. [p346, 348, 349, 357, 358]
- J. Friedman, T. Hastie, R. Tibshirani, B. Narasimhan, and et al. glmnet: Lasso and elastic-net regularized generalized linear models. *Astrophysics Source Code Library*, pages ascl-2308, 2023. URL <https://cran.r-project.org/package=glmnet>. [p347, 357]
- S. Ghosh. On the grouped selection and model complexity of the adaptive elastic net. *Statistics and Computing*, 21:451–462, 2011. [p364]
- J. Gui and H. Li. Penalized Cox regression analysis in the high-dimensional and low-sample size settings, with applications to microarray gene expression data. *Bioinformatics*, 21(13): 3001–3008, 2005. doi: 10.1093/bioinformatics/bti422. [p359]
- T. Hastie, R. Tibshirani, and R. Tibshirani. Best subset, forward stepwise or lasso? analysis and recommendations based on extensive comparisons. *Statistical Science*, 35(4):579–592, 2020. [p347]
- S. Hieke, A. Benner, R. F. Schlenl, M. Schumacher, L. Bullinger, and H. Binder. Integrating multiple molecular sources into a clinical risk prediction signature by extracting complementary information. *BMC Bioinformatics*, 17:1–13, 2016. doi: 10.1186/s12859-016-1183-6. [p354]
- A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970. doi: 10.1080/00401706.1970.10488634. [p349]
- S. Huang, K. Chaudhary, and L. X. Garmire. More is better: recent progress in multi-omics data integration methods. *Frontiers in Genetics*, 8:84, 2017. doi: 10.3389/fgene.2017.00084. [p347, 348]
- J. Jia and B. Yu. On model selection consistency of the elastic net when $p \gg n$. *Statistica Sinica*, pages 595–611, 2010. [p363]
- S. Klau, R. Hornung, A. Bauer, and J. Hagenberg. Prioritylasso: Analyzing multiple omics data with an offset approach. R package version 0.3. 1, 2017. URL <https://doi.org/10.32614/CRAN.package.prioritylasso>. [p348, 366]
- S. Klau, V. Jurinovic, and et al. Priority-Lasso: a simple hierarchical approach to the prediction of clinical outcome using multi-omics data. *BMC Bioinformatics*, 19:1–14, 2018. doi: 10.1186/s12859-018-2344-6. [p348, 358, 366]
- L. Musib, R. Coletti, M. B. Lopes, H. Mouriño, and E. Carrasquinha. Priority-Elastic net for binary disease outcome prediction based on multi-omics data. *BioData Mining*, 17:45, 2024. doi: 10.1186/s13040-024-00401-0. [p348, 357, 366]

- L. Musib, H. Mouriño, and E. Carrasquinha. Comprehensive analysis of multi-omics data using an offset-based method, 2025. URL <https://doi.org/10.32614/CRAN.package.priorityelasticnet>. [p348, 366]
- N. Simon, J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for Cox's proportional hazards model via coordinate descent. *Journal of Statistical Software*, 39(5):1, 2011. doi: 10.18637/jss.v039.i05. URL <http://cran.r-project.org/package=glmnet>. [p346, 348, 360]
- N. Simon, J. Friedman, T. Hastie, and R. Tibshirani. A sparse-group lasso. *Journal of Computational and Graphical Statistics*, 22(2):231–245, 2013. doi: 10.1080/10618600.2012.681250. [p347]
- N. Simon, J. Friedman, T. Hastie, R. Tibshirani, and M. N. Simon. Package 'sgl'. CRAN Documentation, 2018. doi: 10.32614/CRAN.package.SGL. [p347]
- R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996. doi: 10.1111/j.2517-6161.1996.tb02080.x. [p346, 349]
- R. Tibshirani. The lasso method for variable selection in the Cox model. *Statistics in Medicine*, 16(4):385–395, 1997. [p346]
- R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67(1): 91–108, 2005. [p346]
- R. Tibshirani, J. Bien, J. Friedman, T. Hastie, N. Simon, J. Taylor, and R. J. Tibshirani. Strong rules for discarding predictors in lasso-type problems. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 74(2):245–266, 2012. doi: 10.1111/j.1467-9868.2011.01004.x. [p346]
- A. J. Van der Kooij. *Prediction accuracy and stability of regression with optimal scaling transformations*. Leiden University, 2007. [p346]
- A. Veríssimo, E. Carrasquinha, M. Lopes, A. Oliveira, M. Sagot, and S. Vinga. Network centrality metrics for elastic-net regularized models, 2018. URL <https://doi.org/doi:10.18129/B9.bioc.glmSparseNet>. [p361]
- H. Wang, G. Li, and C.-L. Tsai. Regression coefficient and autoregressive order shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 69(1):63–78, 2007. [p363]
- H. H. Zhang and W. Lu. Adaptive lasso for Cox's proportional hazards model. *Biometrika*, 94(3):691–703, 2007. doi: 10.1093/biomet/asm037. [p363]
- Q. Zhao, X. Shi, Y. Xie, J. Huang, B. Shia, and S. Ma. Combining multidimensional genomic measurements for predicting cancer prognosis: observations from TCGA. *Brief Bioinform*, 16(2):291–303, 2015. doi: 10.1093/bib/bbu003. [p347]
- J. Zhu and T. Hastie. Classification of gene microarrays by penalized logistic regression. *Biostatistics*, 5(3):427–443, 2004. doi: 10.1093/biostatistics/kxg046. [p357]
- H. Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101(476):1418–1429, 2006. doi: 10.1198/016214506000000735. [p346, 363]
- H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67(2):301–320, 2005. doi: 10.1111/j.1467-9868.2005.00503.x. [p346, 348, 349, 363]
- H. Zou and H. H. Zhang. On the adaptive elastic-net with a diverging number of parameters. *Annals of Statistics*, 37(4):1733, 2009. doi: 10.1214/08-AOS625. [p347, 364]

Laila Musib

Faculdade de Ciências da Universidade de Lisboa

Centro de Estatística e Aplicações da Universidade de Lisboa (CEAUL)

Campo Grande 016, 1749-016 Lisboa, Portugal

fc59282@alunos.ciencias.ulisboa.pt

Helena Mourão

Faculdade de Ciências da Universidade de Lisboa

Centro de Estatística e Aplicações da Universidade de Lisboa (CEAUL)

Campo Grande 016, 1749-016 Lisboa, Portugal

mhnunes@ciencias.ulisboa.pt

Eunice Carrasquinha

Faculdade de Ciências da Universidade de Lisboa

Centro de Estatística e Aplicações da Universidade de Lisboa (CEAUL)

Campo Grande 016, 1749-016 Lisboa, Portugal

eitridgeirao@ciencias.ulisboa.pt

AccSamplingDesign: An R Package for Optimizing Acceptance Sampling Plans

by Ha Truong, Victor Miranda, and Roger Kissling

Abstract This paper introduces an R package for the optimization and visualization of acceptance sampling plans. It supports classical distributions (Binomial, Poisson, Normal) and provides the first R implementation of Beta-based plans tailored for bounded, non-normal data such as proportions. The package leverages nonlinear programming to compute optimal designs significantly faster and employs less memory than traditional grid search methods, achieving over 1000× speedup for Beta-based plans and substantial improvements for Normal-based plans. The package also delivers stable performance across a wide range of sampling plan settings and constraints. Users can visualize operating characteristic curves, compare sampling plans, and apply the tools in real-world quality control settings. We outline the statistical foundation, software architecture, and key applications.

1 Introduction

Acceptance sampling (AS) is a statistical quality control method used to decide whether to accept or reject a lot based on a sample, aiming to minimize inspection effort while controlling producer's and consumer's risks (Juran and Gryna, 1988). AS procedures are traditionally divided into two categories: attributes sampling and variables sampling (Schilling and Neubauer, 2017). **Attributes sampling** classifies items as conforming or nonconforming, with decisions based on the number of defectives in the sample (ISO 2859-1, 1999). While simple to apply, it discards measurement detail and typically requires sample sizes around 50% larger than those needed for variables-based methods (Mittag, 1997). In contrast, **variables sampling** uses continuous measurements such as weight, concentration, or dimension (ISO 3951-1, 2022), leading to more efficient decision-making. However, variables sampling plans typically assume normality, which limits their effectiveness for skewed or bounded data (Brown, 1985). **Beta-based AS plans** address this limitation by accommodating asymmetry, especially near the bounds of 0 and 1, and often require smaller sample sizes (Govindaraju and Kissling, 2016). Despite these benefits, Beta-based methods remain underused in practice due to limited software support.

Optimal design in AS is crucial for efficient quality control, since it minimizes sample size while controlling error risks, directly reducing inspection time and cost. Existing tools like **AcceptanceSampling** (Andreas Kiermeier, 2008) and the methods introduced by Cano et al. (2015) focus on traditional distributions and use fixed tables or grid search for plan selection. Although nonlinear programming (NLP) has been proven to optimize Normal-based AS plans more effectively than grid search (Duarte and Saraiva, 2013), its practical adoption remains limited. Meanwhile, several packages listed in the *ExperimentalDesign* Task View—such as **OptimalDesign** (Harman and Filova, 2025) and **AlgDesign** (Wheeler and Braun, 2025)—incorporate NLP or heuristic methods for optimizing experimental designs, they do not directly support AS or the specification of risk-based design criteria.

To address this gap, we developed the **AccSamplingDesign** package (Truong et al., 2025), the first R package to support AS plans based on the Beta distribution. It leverages NLP to efficiently generate statistically optimal sampling plans under both Normal and Beta assumptions, offering a lightweight and user-friendly solution for risk-controlled design across practical constraints. The statistical foundations of this package are based on the methodologies presented in Schilling and Neubauer (2017), Wilrich (2004), and Govindaraju and Kissling (2016). This paper outlines the AS framework and models for computing the probability of acceptance, then focuses on optimization methods, the R implementation, and practical applications.

2 Acceptance sampling framework

AS assesses whether the **true quality level** p of the inspected lot meets required standards. Since p is unknown, decisions are based on a sample, and the **probability of acceptance** (P_a) represents the chance that the sample leads to acceptance under a given plan. A sampling plan controls two decision errors: the **producer's risk** (Type I error)—rejecting a good-quality lot—and the **consumer's risk** (Type II error)—accepting a poor-quality lot. These are bounded by tolerances: the **producer's risk tolerance** (α) is the maximum acceptable probability of rejecting a lot at the **producer's risk quality** (PRQ)¹, and the **consumer's risk tolerance** (β) is the maximum acceptable probability of accepting a lot at the

¹PRQ corresponds to the acceptance quality limit (AQL); see ISO 2859-1:1999.

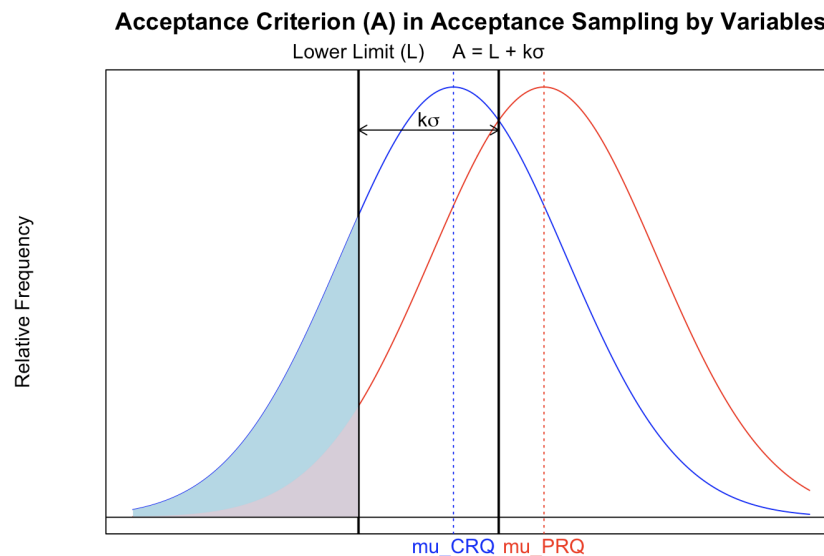


Figure 1: Visualization of the acceptance criterion (A) in variables sampling, defined relative to the lower specification limit L . The plot illustrates two process means (PRQ and CRQ), with shaded areas showing nonconforming proportions. A serves as the threshold for deciding lot acceptance based on the sample mean.

consumer's risk quality (CRQ)². PRQ reflects an acceptable low defect level, while CRQ represents an unacceptable high level (ISO 3534-2, 2006), (CXG 50-2004, 2004).

An AS plan is defined by a sample size (n) and either an acceptance number (c) or an acceptability constant (k). In attributes sampling, c sets the maximum number of nonconforming units allowed in the sample. In variables sampling, k is used to compute the **acceptance criterion** (A), a threshold based on the specification limit (SL), such as a lower (LSL or L) or upper limit (USL or U) (see Figure 1).

The performance of a plan is summarized by the **operating characteristic (OC) curves** (Dodge and Romig, 1944), which show P_a as a function of the true quality level, such as the nonconforming proportion (or proportion defective - pd) (see Figure 2).

The goal of AS design is to identify a sampling plan that satisfies specified levels of producer's risk α and consumer's risk β . This is achieved using the two-point estimation method at PRQ and CRQ:

$$P_a(\text{PRQ}) \geq 1 - \alpha \quad \text{and} \quad P_a(\text{CRQ}) \leq \beta. \quad (1)$$

This condition ensures that good-quality lots are accepted with high probability, and poor-quality lots are rejected with high probability. The probability of acceptance P_a depends on the sampling methods and the assumed distribution of the data.

3 Probability of acceptance models

This section presents the derivation of P_a under different distributional assumptions. For variables sampling, we consider both Normal and Beta distributions, each under two scenarios—known and unknown standard deviation—as classified by Lieberman and Resnikoff (1955).

3.1 Attributes sampling plans - Binomial and Poisson distributions

Attributes sampling is used to determine whether a lot should be accepted or rejected based on the number of nonconforming items. The number of nonconforming items X is modeled by either Binomial or Poisson distribution, depending on the application context.

For **Binomial model**, let $X \sim \text{Bin}(n, p)$, the probability of acceptance is:

²CRQ corresponds to the limiting quality (LQ); see ISO 3951-1:2022.

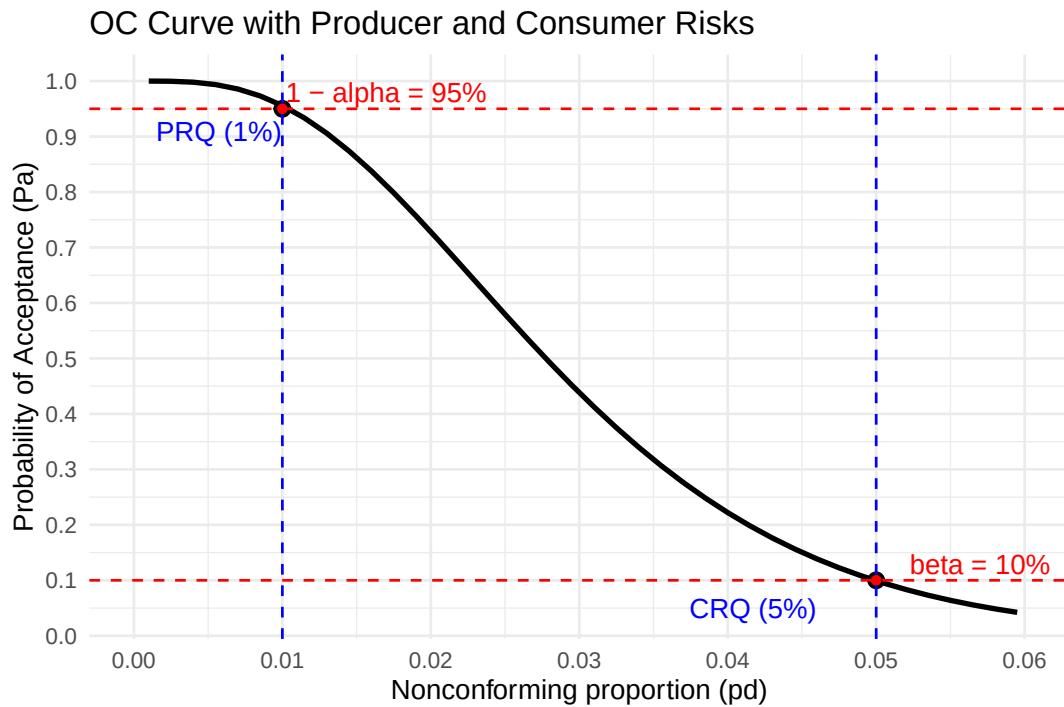


Figure 2: OC curve showing the probability of lot acceptance as a function of the nonconforming proportion. The curve includes two key points used to define the sampling plan: a 95% acceptance probability at PRQ = 0.01 and a 10% acceptance probability at CRQ = 0.05, indicating the defined producer's and consumer's risk control.

$$Pa(p) = P(X \leq c) = \sum_{i=0}^c \binom{n}{i} p^i (1-p)^{n-i}. \quad (2)$$

For **Poisson model**, let $X \sim \text{Poisson}(\lambda = np)$, the probability of acceptance is:

$$Pa(p) = P(X \leq c) = \sum_{i=0}^c \frac{(np)^i e^{-np}}{i!}. \quad (3)$$

Here, n is the sample size, c is the acceptance number, and p is the probability that a randomly selected item is defective. See [Schilling and Neubauer \(2017\)](#) for more details.

3.2 Variables sampling plans - Normal distribution with known σ

Let the quality characteristic X follow a normal distribution: $X \sim N(\mu, \sigma^2)$, where μ is the process mean and σ^2 is the process variance. The sample mean $\bar{X}$ for a sample size n_σ is then distributed as $\bar{X} \sim N(\mu, \sigma^2/n_\sigma)$. Let k_σ denote the acceptability constant. To evaluate the lot for a one-sided upper specification limit (USL), we use the test statistic:

$$Z_\sigma = \bar{X} + k_\sigma \sigma. \quad (4)$$

The lot is accepted if $Z_\sigma \leq USL$. The probability of lot acceptance is given by:

$$Pa(p) = P(Z_\sigma \leq USL) = \Phi\left(\sqrt{n_\sigma} \left(\Phi^{-1}(1-p) - k_\sigma\right)\right), \quad (5)$$

where n_σ is the sample size, p is the true nonconforming proportion and $\Phi(\cdot)$ is the standard normal cumulative distribution function (CDF). See [Wilrich \(2004\)](#) for derivation.

3.3 Variables sampling plans - Normal distribution with unknown σ

When σ is unknown, the sample standard deviation S estimates σ . The test statistic (4) becomes: $Z_s = \bar{X} + k_s S$ and the distribution of Z_s follows a **non-central t-distribution**. Then, the probability of

acceptance $P_a(p)$ is given by:

$$P_a(p) = P(Z_s \leq USL) = 1 - F_t(k_s \sqrt{n_s}, n_s - 1, -\Phi^{-1}(p) \sqrt{n_s}), \quad (6)$$

where n_s is the sample size, k_s is the acceptability constant, p is the proportion of nonconforming and $F_t(\cdot)$ is the CDF of the non-central t-distribution with $n_s - 1$ degrees of freedom (Wilrich, 2004).

3.4 Variables sampling plans - Beta distribution with known θ

For compositional data modeled as $X \sim \text{Beta}(a, b)$, Govindaraju and Kissling (2016) used a re-parameterization of the form:

$$\mu = \frac{a}{a+b}, \quad \theta = a+b, \quad \sigma^2 = \frac{\mu(1-\mu)}{\theta+1} \approx \frac{\mu(1-\mu)}{\theta} \quad (\text{for large } \theta), \quad (7)$$

where θ controls the precision of the distribution. A larger θ implies smaller variance. Thus, $X \sim \text{Beta}(\theta\mu, \theta(1-\mu))$. The nonconforming proportion p is fully determined by the Beta parameters μ and θ ; for example, $p = \Pr(X < L \mid \mu, \theta)$ for an LSL and $p = \Pr(X > U \mid \mu, \theta)$ for a USL. Consider an AS plan with sample size n , the lot is accepted if:

$$\begin{cases} \bar{X} - k\hat{\sigma} \geq L, & (\text{case of LSL}) \\ \bar{X} + k\hat{\sigma} \leq U, & (\text{case of USL}) \end{cases} \quad (8)$$

where $\bar{X}$ is the sample mean, k is the acceptability constant and $\hat{\sigma} \approx \sqrt{\frac{\bar{X}(1-\bar{X})}{\theta}}$. In both LSL and USL cases, rearranging (8) yields the same quadratic inequality:

$$(\theta + k^2) \bar{X}^2 - (2\theta L + k^2) \bar{X} + \theta L^2 \geq 0. \quad (9)$$

Solving this quadratic yields the roots:

$$z_1 = \frac{2\theta L + k^2 - \sqrt{\Delta}}{2(\theta + k^2)}, \quad z_2 = \frac{2\theta L + k^2 + \sqrt{\Delta}}{2(\theta + k^2)}, \quad (10)$$

where the discriminant is: $\Delta = (2\theta L + k^2)^2 - 4(\theta + k^2)(\theta L^2)$. Let $F_b(z)$ denote the CDF of the Beta distribution. Using Equation (8), the probability of acceptance $P_a(p)$ can then be calculated as follows:

$$P_a(p) = \begin{cases} P(\bar{X} - k\hat{\sigma} \geq L) = 1 - F_b(z_2), & (\text{for LSL case}) \\ P(\bar{X} + k\hat{\sigma} \leq U) = F_b(z_1), & (\text{for USL case}) \end{cases} \quad (11)$$

4 Optimization methods for sampling plan design

This section presents AS design and optimization methods based on the $P_a(\cdot)$ functions derived earlier. Given producer's risk α and consumer's risk β , the goal is to find a sampling plan that meets the risk conditions in Equation (1) while minimizing sample size. While closed-form solutions and search methods are available in some cases (e.g., for attributes sampling or Normal-based plans with known σ), NLP provides an effective and flexible solution when such formulas are unavailable or intractable.

For Normal-based variables sampling and when σ is known, a closed-form solution exists for the optimal plan (n_σ, k_σ) . The acceptance probability is defined in Equation (5), and the risk conditions in (1) lead to the following expressions (Wilrich, 2004):

$$n_\sigma = \left(\frac{\Phi^{-1}(1-\alpha) + \Phi^{-1}(1-\beta)}{\Phi^{-1}(1-\text{PRQ}) - \Phi^{-1}(1-\text{CRQ})} \right)^2, \quad (12)$$

$$k_\sigma = \frac{\Phi^{-1}(1-\text{PRQ}) \cdot \Phi^{-1}(\beta) + \Phi^{-1}(1-\text{CRQ}) \cdot \Phi^{-1}(1-\alpha)}{\Phi^{-1}(1-\alpha) + \Phi^{-1}(\beta)}. \quad (13)$$

For Normal-based variables sampling and when σ is unknown, the acceptance probability $P_a(\cdot)$ is defined using a non-central t -distribution, as in Equation (6). Since no closed-form solution exists for this case (though an approximation is available from Wilrich (2004)), we formulate a nonlinear optimization problem to determine the optimal sample size n_s and acceptability constant k_s :

$$\text{Objective} = \min_{n_s, k_s} (|\text{PR} - \alpha| + |\text{CR} - \beta|). \quad (14)$$

Here, the producer's risk is defined as $PR = 1 - P_a(\text{PRQ})$, where PRQ is the quality level at which producer's risk is evaluated. Similarly, the consumer's risk is $CR = P_a(\text{CRQ})$, with CRQ as the corresponding quality level. The optimization is performed using the derivative-free method from [Nelder and Mead \(1965\)](#) via the `optim()` function in R. Although numerous advanced NLP solvers are available in the *Optimization* task view, `optim()` was adequate for this setting for several reasons: First, the 2-dimensional problem (n_s, k_s) is well suited to derivative-free methods. Second, the objective function contains absolute value operations and involves non-central t -distribution computations that can introduce numerical non-smoothness. Third, initializing from the Normal-known- σ solution provides a warm start near the optimum, reducing concerns about local minima. Finally, using base R's `optim()` avoids external dependencies while providing adequate performance for this application.

For Beta-based variables sampling and when θ is known, the optimal sampling parameters (n_θ, k_θ) are found by solving the risk conditions in (1), using the P_a function from (11). To enable efficient search, we reformulate the task as a NLP problem with a penalized objective function:

$$\text{Objective} = \min_{n_\theta, k_\theta} \left[n_\theta + \lambda \cdot \left(\max(\text{PR} - \alpha, 0)^2 + \max(\text{CR} - \beta, 0)^2 \right) \right], \quad (15)$$

where $\text{PR} = 1 - P_a(\text{PRQ})$ and $\text{CR} = P_a(\text{CRQ})$. A penalty coefficient λ (i.e., $\lambda = 1e4$) is introduced to enforce these constraints by penalizing violations in the objective function. The implementation of (15) used the `optim()` function in R with the L-BFGS-B method ([Byrd et al., 1995](#)). Initial values for (n_θ, k_θ) are taken from Equations (12) and (13), placing the search close to the optimum and reducing the risk of slow convergence or suboptimal solutions. Practical search ranges for the parameters are defined around these initial values, and L-BFGS-B naturally enforces the box constraints, which improves stability and avoids unrealistic proposals.

When θ is unknown, both the sample mean $\bar{X}$ and θ must be estimated—e.g., via the `betaff()` function from the **VGAM** package ([Yee, 2025](#))—introducing additional uncertainty into the plan. To account for this, the adjusted sample size n_{adj} is computed as:

$$n_{\text{adj}} = \left(1 + \rho \cdot k_\theta^2 \right) n_\theta, \quad (16)$$

where (n_θ, k_θ) are the optimal parameters from Equation (15), and ρ reflects the added variability from estimating θ . This adjustment is similar in form to the Normal-based approximation by [Wilrich \(2004\)](#). Based on simulation by [Govindaraju and Kissling \(2016\)](#), a conservative value $\rho \approx 0.85$ is recommended to adjust the sample size when θ is estimated.

5 Package overview

The **AccSamplingDesign** package supports attributes sampling using the Binomial or Poisson distributions, as well as variables sampling based on Normal or Beta distributions. It centers on two core classes: `AttrPlan` for attributes sampling and `VarPlan` for variables sampling. The unified interface `optPlan()` selects `optAttrPlan()` or `optVarPlan()` based on the chosen distribution. The package also provides S3 methods for summary and visualization, and functions for generating OC data to compare manual and optimal designs (Table 1).

This R package is available on CRAN under the GPL-3 license. The released version can be installed as follows:

```
install.packages("AccSamplingDesign")
```

The development version is maintained at [GitHub repository](#) and can be installed through the **devtools** package ([Wickham et al., 2022](#)) as follows:

```
devtools::install_github("vietha/AccSamplingDesign")
```

5.1 Method `optPlan()` as the primary interface

The `optPlan()` function serves as the primary entry point for users to design optimal AS plans across a range of classical distributions. It provides a unified interface that automatically dispatches to either `optAttrPlan()` or `optVarPlan()`, depending on whether the specified distribution and parameters correspond to an attributes or variables plan. This design simplifies usage for most users, abstracting the underlying logic while still allowing advanced users to call lower-level methods directly for finer control. The function supports Binomial, Poisson, Normal, and Beta distributions, enabling both attributes-based and variables-based sampling schemes.

Table 1: Summary of sampling plan methods and utilities provided by the package.

Method	Description
optPlan()	Unified entry point; automatically selects between attributes or variables sampling based on inputs.
optAttrPlan()	Designs attributes sampling plans using the Binomial or Poisson distribution.
optVarPlan()	Designs variables sampling plans for Normal or Beta distributions. Supports known/unknown standard deviation and one-sided limits.
manualPlan()	Manually generate attribute or variable sampling plans based on user-specified parameters, typically used for evaluation or comparison.
OCdata()	Generates data for OC curves based on an optimal or manual plan.
summary()	Summarizes a sampling plan object, including key values such as sample size and risk levels.
plot()	Visualizes sampling plans or OC curve data. Adapts content and axis labeling based on the input object.

For attributes sampling, the optimal plan (n, c) is obtained by evaluating the Binomial case in Equation (2) or the Poisson case in Equation (3), followed by a discrete search over feasible (n, c) pairs to find the smallest sample size satisfying the risk constraints in (1). For Normal-based variables sampling and σ is known, the optimal plan (n, k) is computed using closed-form formulas in Equation (12) and (13). When σ is unknown, the plan is obtained by minimizing the objective in Equation (14) via a NLP method. For Beta-based variables sampling, when the precision parameter θ is known, the optimal plan is obtained by minimizing the objective function in Equation (15) using NLP. When θ is unknown, the approach relies on the approximation described in Equation (16).

This flexible design allows users to specify target quality thresholds—PRQ and CRQ—along with the associated risks α and β , and specification limits such as the LSL or USL for variables plans. Internally, the function manages distribution selection, parameter validation, and optimization, returning an object of class `AttrPlan` or `VarPlan`, ready for further evaluation or visualization. By streamlining the modeling workflow while supporting advanced configurations, `optPlan()` facilitates efficient design of AS plans across diverse quality control and compliance settings.

5.2 Generating operating characteristic curves with `OCdata()`

The `OCdata()` function facilitates the generation of OC curve data from either optimized sampling plans—such as those produced by `optPlan()` or user-defined parameters via `manualPlan()`. It accommodates both attributes and variables sampling plans by adapting calculations to the specified distribution, and computes acceptance probabilities across a specified range of nonconforming proportions. The output is a structured `OCdata` object that encapsulates key vectors such as `pd` (nonconforming proportions), `paccept` (acceptance probabilities), and, for variables sampling plans, `process_means` along with sample size and plan-specific parameters. This organization facilitates seamless downstream visualization and analysis.

5.3 Visualizing Sampling Plans with `plot()`

Visualization of sampling plans and their corresponding OC curves is supported via a unified `S3 plot()` method. The method adapts automatically based on the class of the input object—`AttrPlan`, `VarPlan`, or `OCdata`—and sets appropriate axis labels, reference lines, and graphical elements for each context. The x-axis scale can be controlled using the `by` argument, allowing users to plot against the proportion nonconforming or, where applicable, the process mean. The function uses base R graphics and supports further customization through standard graphical parameters.

6 Practical examples

This section demonstrates how to use the `AccSamplingDesign` package in R to design AS plans. Examples cover both attributes and variables sampling plans. We also compare user-defined and optimal plans, evaluate the performance of Normal vs. Beta-based AS methods, and demonstrate

how to use `OCdata()` to create custom plans and plot OC curves by nonconforming proportion (pd) or process mean. Before running the examples, ensure the package is loaded into the R environment.

```
library(AccSamplingDesign)
```

6.1 Sampling plan design for defective rates in packaged goods

This example illustrates an attributes sampling plan for a packaged goods manufacturer aiming to control the defect rate. Because the data are discrete (e.g., counts of defective units), a Binomial distribution assumption is appropriate. The plan is designed to ensure a low probability of rejecting acceptable batches with a defect rate of 1% ($PRQ = 0.01$)—reflecting the producer’s risk—and a low probability of accepting unacceptable batches with a defect rate of 5% ($CRQ = 0.05$)—reflecting the consumer’s risk. The risk constraints are set to $\alpha = 0.02$ and $\beta = 0.15$. The optimal plan is obtained using the generic `optPlan()` function.

```
optPlan(distribution = "binomial", PRQ = 0.01, CRQ = 0.05, alpha = 0.02, beta = 0.15)
```

```
#> AttrPlan object:
#> Distribution: binomial
#> Sample size (n): 144
#> Acceptance number (c): 4
#>
#> NOTE: summary(plan) for detail report,
#>       plot(plan) for quick OC visualization,
#>       OCdata(plan) to extract data for evaluation and custom plots.
```

This plan recommends inspecting 144 items from each batch and accepting the batch if no more than 4 defective items are found. It balances the producer’s and consumer’s risks, offering a practical guideline for quality control in production. Users can explore the plan further using `summary()`, `plot()`, or `OCdata()` to evaluate performance under other defect rates or to create customized visualizations.

6.2 Sampling plan for moisture composition in milk products

This example illustrates a variables sampling plan for monitoring moisture content in milk products, where the data are continuous and represent proportions. Such fractional data can be modeled using either a Normal or a Beta distribution. We use this example to compare the performance of both approaches. The upper specification limit (USL) is set at 0.05. The plan targets acceptance of high-quality batches with a defect rate below 0.5% ($PRQ = 0.005$), and rejection of low-quality batches with a defect rate above 3% ($CRQ = 0.03$), while controlling producer’s risk at $\alpha = 0.05$ and consumer’s risk at $\beta = 0.10$. We first design a Normal-based plan, followed by a Beta-based plan assuming a known precision parameter $\theta = 500$.

```
normal_plan <- optPlan(PRQ = 0.005, CRQ = 0.03, alpha = 0.05, beta = 0.10,
                      distribution = "normal", sigma_type = "known")
```

```
summary(normal_plan)
```

```
#> Variables Acceptance Sampling Plan
#> Distribution: normal
#> Sample Size (n, rounded up): 18 [raw n = 17.72779 ]
#> Acceptability Constant (k): 2.185
#> Population Standard Deviation: known
#> Producer's Risk (PR = 0.05 ) at PRQ = 0.005
#> Consumer's Risk (CR = 0.1 ) at CRQ = 0.03
```

```
beta_plan <- optPlan(PRQ = 0.005, CRQ = 0.03, alpha = 0.05, beta = 0.10, USL = 0.05,
                    distribution = "beta", theta = 500, theta_type = "known")
```

```
summary(beta_plan)
```

```
#> Variables Acceptance Sampling Plan
#> Distribution: beta
#> Sample Size (n, rounded up): 16 [raw n = 15.22157 ]
#> Acceptability Constant (k): 2.485
```

```
#> Population Precision Parameter (theta): known
#> Producer's Risk (PR = 0.05072676 ) at PRQ = 0.005
#> Consumer's Risk (CR = 0.1004133 ) at CRQ = 0.03
#> Upper Specification Limit (USL): 0.05
```

It is worth noting that, in this case, the Beta-based sampling plan requires a smaller sample size ($n = 16$) compared to the Normal-based sampling plan ($n = 18$). This aligns with the findings of Govindaraju and Kissling (2016).

6.3 Sampling plan for Vitamin-A in a food product

This example designs a variables sampling plan for Vitamin-A content, where data are heavily skewed and close to zero—characteristics well handled by the Beta distribution. Using a known precision parameter $\theta = 6.6 \times 10^8$ and a lower specification limit (LSL) of 5.65×10^{-6} , the plan controls producer's risk at $\alpha = 0.05$ for a preferred quality level (PRQ = 2.5%) and consumer's risk at $\beta = 0.10$ for a worse quality level (CRQ = 10%). We begin by computing the optimal Beta-based sampling plan under these conditions:

```
vita_plan <- optPlan(PRQ = 0.025, CRQ = 0.10, alpha = 0.05, beta = 0.10,
                    distribution = "beta", theta_type = "known",
                    theta = 6.6e8 , LSL = 5.65e-6)
summary(vita_plan)

#> Variables Acceptance Sampling Plan
#> Distribution: beta
#> Sample Size (n, rounded up): 19 [raw n = 18.66251 ]
#> Acceptability Constant (k): 1.571
#> Population Precision Parameter (theta): known
#> Producer's Risk (PR = 0.05087403 ) at PRQ = 0.025
#> Consumer's Risk (CR = 0.1005144 ) at CRQ = 0.1
#> Lower Specification Limit (LSL): 5.65e-06
```

The `plot()` method visualizes Beta-based AS plans. By default, it shows the acceptance probability against the nonconforming proportion pd (Figure 3).

```
plot(vita_plan)
```

Setting `by = "mean"` changes the x-axis to the process mean (Figure 4).

```
plot(vita_plan, by = "mean")
```

Sensitivity analysis can be performed by varying the sample size n or the acceptability constant k using `manualPlan()` and `OCdata()`, based on the optimal plan generated by `optPlan()`. The evaluated plans are then compared to the optimal plan using OC curves. The visualization was produced using `ggplot2` (Wickham, 2016), and the interactive version was generated with `plotly` (Sievert, 2020) (see Figure 5).

```
# Sequence of defect rates for OC curve
pd <- seq(0, 0.2, by = 0.001)
# Create manual plans to compare
plan_n_up <- manualPlan(n = vita_plan$sample_size + 10, k = vita_plan$k,
                        distribution = "beta", LSL = 5.65e-6, theta = 6.6e8)
plan_k_up <- manualPlan(n = vita_plan$sample_size, k = vita_plan$k + 0.1,
                        distribution = "beta", LSL = 5.65e-6, theta = 6.6e8)
# Combine OC data into a data frame
oc_df <- bind_rows(
  as.data.frame(OCdata(vita_plan, pd = pd)) |> mutate(Plan = "Optimal"),
  as.data.frame(OCdata(plan_n_up, pd = pd)) |> mutate(Plan = "n + 10"),
  as.data.frame(OCdata(plan_k_up, pd = pd)) |> mutate(Plan = "k + 0.1")
) |> mutate(Plan = factor(Plan, levels = c("Optimal", "n + 10", "k + 0.1")))
# Plot OC curves
plt_plan_compare <- ggplot(oc_df, aes(x = pd, y = paccept, color = Plan)) +
```

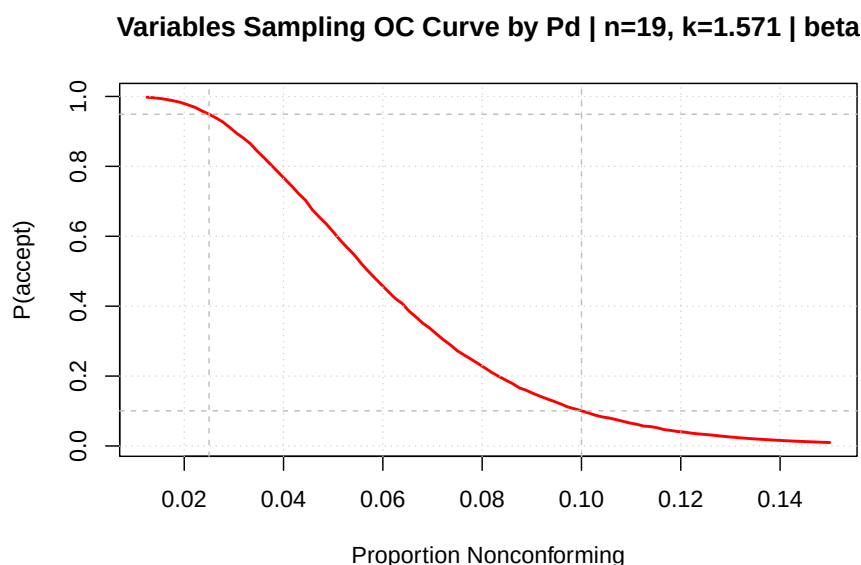


Figure 3: OC curve for Vitamin A Beta-based sampling showing the probability of acceptance across proportions nonconforming. Vertical dashed lines mark the producer's risk quality (PRQ = 2.5%) and consumer's risk quality (CRQ = 10%) levels. Horizontal dashed lines represent the corresponding acceptance probability constraints: at least 95% at PRQ (producer's risk $\leq 5\%$) and at most 10% at CRQ (consumer's risk $\leq 10\%$).

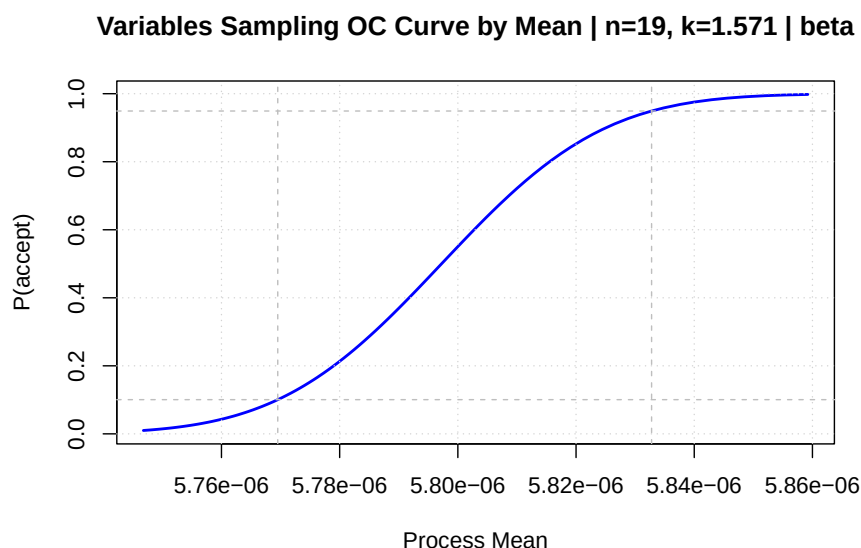


Figure 4: OC curve for Vitamin A Beta-based sampling plotted against the process mean, showing the probability of acceptance across different process mean values. Vertical dashed lines mark the mean levels corresponding to the producer's risk quality (PRQ) and consumer's risk quality (CRQ). Horizontal dashed lines represent the corresponding acceptance probability constraints: at least 95% at PRQ (producer's risk $\leq 5\%$) and at most 10% at CRQ (consumer's risk $\leq 10\%$).

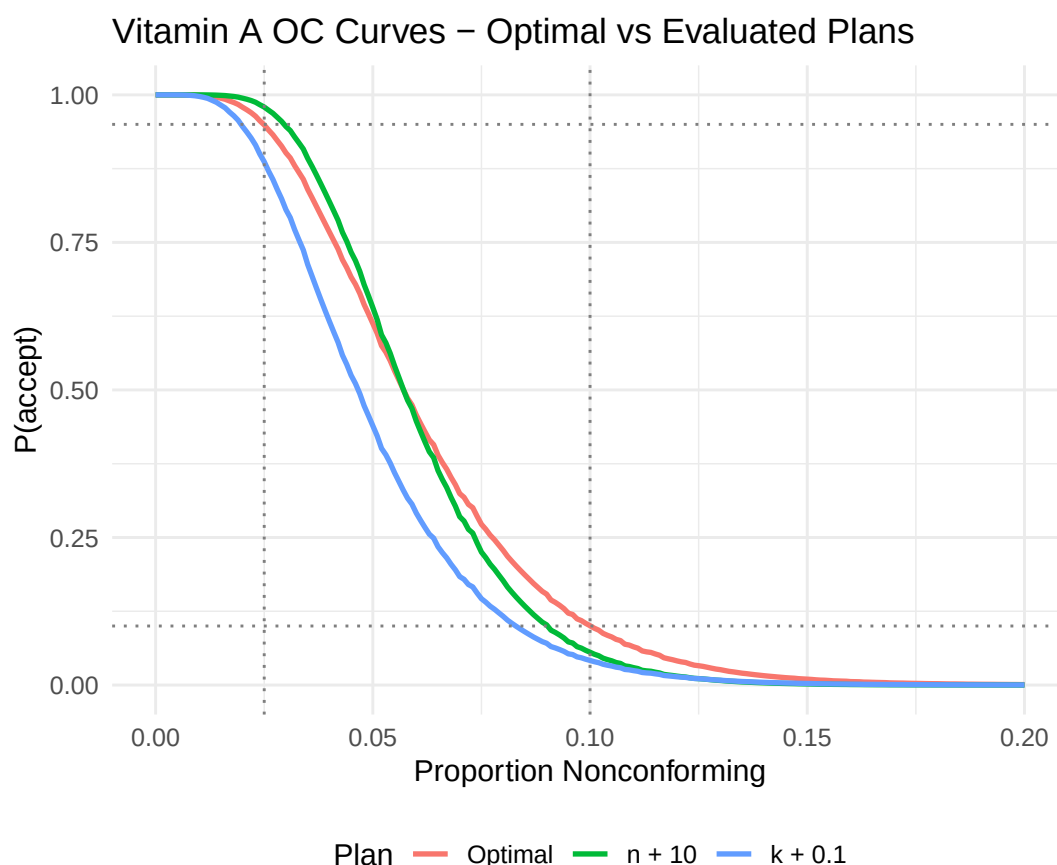


Figure 5: OC curves for Vitamin A Beta-based acceptance sampling. The optimal plan is compared with two evaluated plans: one with increased sample size ($n + 10$) and another with a higher acceptability constant ($k + 0.1$). The plot illustrates how changes in n or k affect the shape and slope of the curve, thereby influencing the producer's and consumer's risks.

```
geom_line(linewidth = 1) +
geom_vline(xintercept = c(0.025, 0.10), linetype = "dotted", color = "gray50") +
geom_hline(yintercept = c(0.95, 0.10), linetype = "dotted", color = "gray50") +
labs(title = "Vitamin A OC Curves – Optimal vs Evaluated Plans",
      x = "Proportion Nonconforming", y = "P(accept)") +
theme_minimal(base_size = 12) + theme(legend.position = "bottom")
```

For case of unknown θ , the sampling plan is designed with the following commands:

```
vita_plan2 <- optPlan(PRQ = 0.025, CRQ = 0.10, alpha = 0.05, beta = 0.10,
                     distribution = "beta", LSL = 5.65e-6,
                     theta = 6.6e8, theta_type = "unknown")
summary(vita_plan2)
```

```
#> Variables Acceptance Sampling Plan
#> Distribution: beta
#> Sample Size (n, rounded up): 59 [raw n = 58.84879 ]
#> Acceptability Constant (k): 1.571
#> Population Precision Parameter (theta): unknown
#> Producer's Risk (PR = 0.04934987 ) at PRQ = 0.025
#> Consumer's Risk (CR = 0.0985057 ) at CRQ = 0.1
#> Lower Specification Limit (LSL): 5.65e-06
```

7 Computational performance and benchmarking

All computations were performed in R version 4.5.1 (R Core Team, 2025) on a Mac system with an Apple M2 chip, 8 GB RAM, and macOS 15.4.1. To assess computational efficiency, we benchmarked

Table 2: Comparison of AS plans under the Normal model with unknown standard deviation. For each PRQ/CRQ pair, the table reports AS plan (n, k), and computation time from two R packages: *AccSamplingDesign* (using NLP) and *AcceptanceSampling* (grid search). The sample size (n) is rounded up to the next integer for practical application.

PRQ	CRQ	Package	Time (ms)	n	k
0.0050	0.03	<i>AccSamplingDesign</i>	0.87	62	2.192
0.0050	0.03	<i>AcceptanceSampling</i>	11.26	62	2.194
0.0075	0.03	<i>AccSamplingDesign</i>	0.81	93	2.127
0.0075	0.03	<i>AcceptanceSampling</i>	18.22	93	2.127
0.0100	0.03	<i>AccSamplingDesign</i>	1.69	138	2.079
0.0100	0.03	<i>AcceptanceSampling</i>	27.87	138	2.079
0.0125	0.03	<i>AccSamplingDesign</i>	1.60	205	2.041
0.0125	0.03	<i>AcceptanceSampling</i>	46.51	205	2.041
0.0150	0.03	<i>AccSamplingDesign</i>	1.59	309	2.009
0.0150	0.03	<i>AcceptanceSampling</i>	70.54	309	2.009
0.0175	0.03	<i>AccSamplingDesign</i>	1.71	491	1.982
0.0175	0.03	<i>AcceptanceSampling</i>	83.41	491	1.982
0.0200	0.03	<i>AccSamplingDesign</i>	1.85	836	1.957
0.0200	0.03	<i>AcceptanceSampling</i>	100.24	836	1.957

AccSamplingDesign version 0.0.8 against existing tools for designing variables AS plans, focusing on models based on the Normal and Beta distributions.

For Normal-based AS plans with unknown standard deviation, we reanalyzed the *moisture sampling plan* example and compared results between our NLP-based method, implemented in *AccSamplingDesign*, and the grid search approach used by the *AcceptanceSampling* package (version 1.0-10), across varying PRQ values (see Table 2). As shown by Duarte and Saraiva (2013), NLP outperforms grid search in both feasibility and efficiency. Our findings support this observation: while both approaches produced nearly identical plans (n,k), the NLP method was significantly faster, with speed improvements of approximately 50× when PRQ approached CRQ. It also demonstrated consistent processing time and precision across repeated runs, regardless of the PRQ value (see Figure 6).

To the best of our knowledge, no existing R package provides automated support for the design of Beta-based AS plans. We evaluated the performance of *AccSamplingDesign* against the original R scripts developed by Govindaraju and Kissling (2016). In the case study on *vitamin A sampling*, their grid search procedure required over 20 seconds to identify an optimal plan, whereas our implementation achieved the same result in under 0.02 seconds, operating over 1000 times faster and with substantially reduced memory usage. Benchmarking was performed using the *bench* package (Hester et al., 2025). Complete results and reproducible code are provided in the supplementary materials (see *beta_plans_benchmarking.Rmd*).

8 Conclusion

The *AccSamplingDesign* package makes several important contributions to AS design and evaluation within R. First, it introduces support for Beta-based variables sampling plans, which are particularly well suited for bounded quality characteristics such as proportions and concentrations commonly encountered in food, pharmaceutical, and biomedical industries. Compared to Normal-based plans under equivalent settings, Beta-based plans typically require smaller sample sizes, thereby enhancing inspection efficiency. Second, it uses a closed-form expression for the acceptance probability, avoiding computationally intensive Monte Carlo simulation (Govindaraju and Kissling, 2016), and successfully applies NLP for AS plan optimization. This approach outperforms traditional grid search for both Normal- and Beta-based plans, achieving over 1000-fold speed gains for Beta-based plans, making them feasible for real-time use in interactive tools, dashboards, and automated quality control pipelines. Third, the package supports ISO standards and offers a user-friendly design that facilitates practical application of AS plan development in industry settings. It includes functions for visualizing and comparing OC curves, helping users evaluate and communicate plan performance effectively.

Overall, *AccSamplingDesign* fills a notable gap in the R ecosystem by offering a flexible, risk-based framework for AS plan design that aligns with ISO standards. It caters to statisticians, quality control

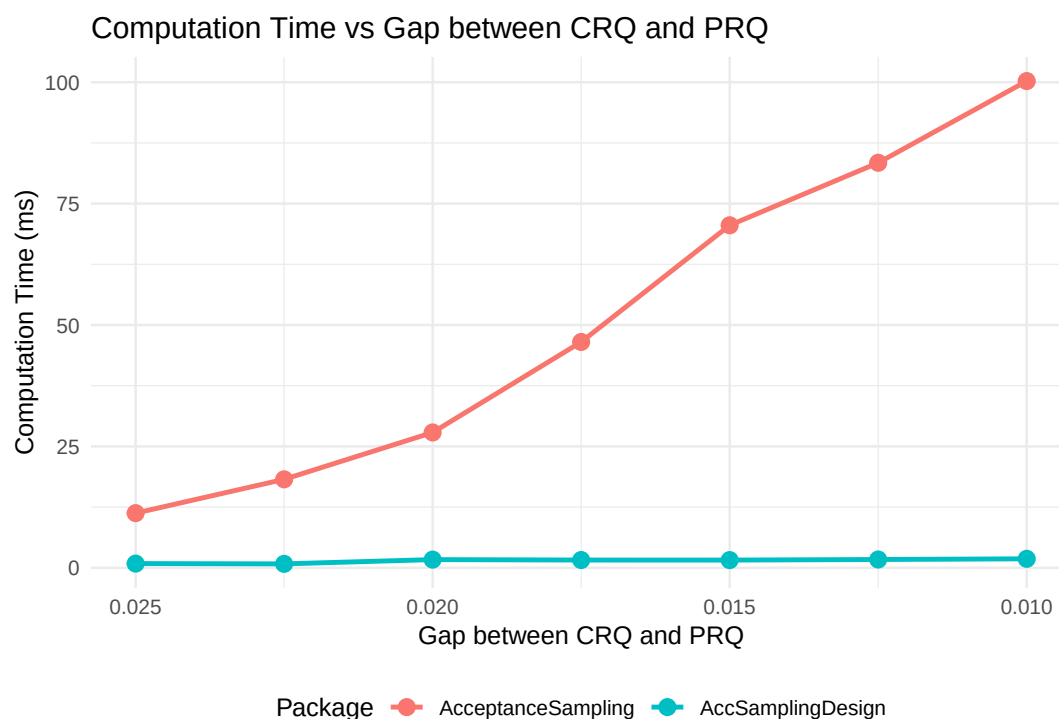


Figure 6: Computation time comparison between `AccSamplingDesign` and `AcceptanceSampling` for the Normal model with unknown standard deviation, as the CRQ–PRQ gap narrows (harder cases on the right). `AccSamplingDesign` is consistently faster and maintains stable performance, while `AcceptanceSampling` slows down.

practitioners, and industry users, promoting accessibility and practical adoption. Future development plans include extending the package to support multivariate sampling plans and measurement error adjustment for compositional data, thereby expanding its utility in advanced quality control settings.

References

- Andreas Kiermeier. Visualizing and assessing acceptance sampling plans: The R package `AcceptanceSampling`. *Journal of Statistical Software*, 26(6):1–20, 2008. doi: 10.18637/jss.v026.i06. [p368]
- G. H. Brown. Acceptance sampling plans for material in bulk form using a Beta model. *Journal of Quality Technology*, 17:134–139, 1985. doi: 10.1080/00224065.1985.11978952. [p368]
- R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995. doi: 10.1137/0916069. [p372]
- E. L. Cano, J. M. Moguerza, and M. P. Corcoba. *Quality Control with R: An ISO Standards Approach*. Springer, 2015. ISBN 978-3-319-23454-1. doi: 10.1007/978-3-319-24046-6. [p368]
- CXG 50-2004. General guidelines on sampling (CXG 50-2004), 2004. URL <https://www.fao.org/fao-who-codexalimentarius/en/>. Rome: FAO/WHO. [p369]
- H. F. Dodge and H. G. Romig. *Sampling Inspection Tables: Single and Double Sampling*. John Wiley & Sons, 1944. URL <https://archive.org/details/samplinginspecti00dodg>. [p369]
- B. P. M. Duarte and P. M. Saraiva. Optimal design of acceptance sampling plans by variables for nonconforming proportions when the standard deviation is unknown. *Communications in Statistics - Simulation and Computation*, 42(6):1318–1342, 2013. doi: 10.1080/03610918.2012.665548. [p368, 378]
- K. Govindaraju and R. C. Kissling. Sampling plans for Beta distributed compositional fractions. *Chemometrics and Intelligent Laboratory Systems*, 151:103–107, 2016. doi: 10.1016/j.chemolab.2015.12.009. [p368, 371, 372, 375, 378]
- R. Harman and L. Filova. *OptimalDesign: A Toolbox for Computing Efficient Designs of Experiments*, 2025. URL <http://www.iam.fmph.uniba.sk/design/>. R package version 1.0.2.1. [p368]

- J. Hester, D. Vaughan, and D. Schmidt. *bench: High Precision Timing of R Expressions*, 2025. URL <https://CRAN.R-project.org/package=bench>. R package version 1.1.4. [p378]
- ISO 2859-1. *Sampling Procedures for Inspection by Attributes — Part 1: Sampling Schemes Indexed by Acceptance Quality Limit (AQL) for Lot-by-Lot Inspection*. International Organization for Standardization, Geneva, Switzerland, 1999. URL <https://www.iso.org/standard/1141.html>. The primary standard for attribute-based acceptance sampling plans. [p368]
- ISO 3534-2. *Statistics – vocabulary and symbols – part 2: Applied statistics*, 2006. URL <https://www.iso.org/standard/40189.html>. [p369]
- ISO 3951-1. *Sampling Procedures for Inspection by Variables — Part 1: Specification for Single Sampling Plans Indexed by Acceptance Quality Limit (AQL) for Lot-by-Lot Inspection*. International Organization for Standardization, Geneva, Switzerland, 2022. URL <https://www.iso.org/standard/74706.html>. Complementary to ISO 2859 for variable-based acceptance sampling. [p368]
- J. M. Juran and F. M. Gryna. *Juran's Quality Control Handbook*. McGraw-Hill, 4th edition, 1988. URL <https://search.worldcat.org/en/title/17546189>. [p368]
- G. J. Lieberman and G. J. Resnikoff. Sampling plans for inspection by variables. *Journal of Quality Technology*, July 15 1955. doi: 10.2307/2280972. [p369]
- H. J. Mittag. Measurement error effects on the performance of process capability indices. In H. J. Lenz and P. T. Wilrich, editors, *Frontiers in Statistical Quality Control*, volume 5. Physica, Heidelberg, 1997. doi: 10.1007/978-3-642-59239-3_15. [p368]
- J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4): 308–313, 1965. doi: 10.1093/comjnl/7.4.308. [p372]
- 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/>. [p377]
- E. G. Schilling and D. V. Neubauer. *Acceptance Sampling in Quality Control*. CRC Press, 3rd edition, 2017. doi: 10.4324/9781315120744. [p368, 370]
- C. Sievert. *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman and Hall/CRC, 2020. ISBN 9781138331457. URL <https://plotly-r.com>. [p375]
- H. Truong, V. Miranda, and R. Kissling. *AccSamplingDesign: Acceptance Sampling Plans Design*, 2025. URL <https://CRAN.R-project.org/package=AccSamplingDesign>. R package version 0.0.8. [p368]
- B. Wheeler and J. Braun. *AlgDesign: Algorithmic Experimental Design*, 2025. URL <https://CRAN.R-project.org/package=AlgDesign>. R package version 1.2.1.2. [p368]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p375]
- H. Wickham, J. Hester, W. Chang, and J. Bryan. *devtools: Tools to Make Developing R Packages Easier*, 2022. URL <https://CRAN.R-project.org/package=devtools>. R package version 2.4.5. [p372]
- P. T. Wilrich. Single sampling plans for inspection by variables under a variance component situation. In H. J. Lenz and P. T. Wilrich, editors, *Frontiers in Statistical Quality Control 7*, volume 7 of *Frontiers in Statistical Quality Control*. Physica, Heidelberg, 2004. doi: 10.1007/978-3-7908-2674-6_4. [p368, 370, 371, 372]
- T. W. Yee. *VGAM: Vector Generalized Linear and Additive Models*, 2025. URL <https://CRAN.R-project.org/package=VGAM>. R package version 1.1-13. [p372]

Ha Truong
Auckland University of Technology
55 Wellesley Street East, Auckland City, New Zealand
<https://github.com/vietha>
ORCID: 0009-0008-2986-1425
ha.truong@aut.ac.nz

Victor Miranda
Auckland University of Technology
55 Wellesley Street East, Auckland City, New Zealand

<https://academics.aut.ac.nz/victor.miranda>
ORCID: 0000-0001-8920-3573
victor.miranda@aut.ac.nz

Roger Kissling
Fonterra Co-operative
New Zealand
roger.kissling@fonterra.com

SurvDNN: Survival Deep Learning Models for Tabular Data

by Imad EL-BADISY

Abstract Deep learning is increasingly applied in survival analysis for high-dimensional data with complex effects. However, most implementations in R rely on Python backends, limiting accessibility and integration with R workflows. We present *survdnn*, a fully native R package for deep neural survival modeling, built on the *torch* library. The package offers a formula interface consistent with R conventions, supports multiple loss functions (Cox, Cox with L2 penalty, Accelerated Failure Time, Cox-Time), and provides utilities for prediction, model evaluation, cross-validation, and hyperparameter tuning. Benchmarks on real data show that *survdnn* performs competitively against classical models such as Cox regression and Random Survival Forests. This package extends the scope of survival analysis to deep learning within the R ecosystem.

1 Introduction

1.1 Background and motivation

Survival analysis is a foundational methodology in biomedical and clinical research, constituting the methodological groundwork for applications such as oncology prognosis, cardiovascular risk stratification, and reliability engineering (Wiegbe et al., 2024). Classical statistical models, most notably the Cox proportional hazards (PH) model, have long been favored for their interpretability and solid theoretical foundation (Cox, 1972). However, these models impose restrictive assumptions, including linear covariate effects and PH, which are frequently violated in real-world applications (Heagerty and Zheng, 2005; Batson et al., 2016).

To address these limitations, machine learning algorithms have been increasingly adopted as flexible modeling approaches that relax restrictive assumptions and often yield improved predictive accuracy. Among these methods, deep learning has emerged as a particularly powerful paradigm, capable of capturing complex non-linear relationships and high-dimensional feature interactions that are difficult to capture with classical approaches (Wiegbe et al., 2024; Bengio et al., 2021).

1.2 Advances in deep survival models

The evolution of deep survival models began with early efforts such as that of Faraggi and Simon (Faraggi and Simon, 1995), who replaced the linear component of Cox regression with a shallow neural network. However, these early models failed to consistently outperform classical models in terms of discrimination (Xiang et al., 2000; Sargent, 2001). A significant advancement came with DeepSurv (Katzman et al., 2018), which introduced modern neural optimization techniques to improve predictive accuracy under the PH setting.

Recent deep learning methods have achieved state-of-the-art performance in survival analysis, particularly through models such as DeepSurv, DeepHit (Lee et al., 2018), and Cox-Time (Kvamme et al., 2019), which have been successfully applied to real-world clinical datasets. These methods are summarized in recent reviews (Wiegbe et al., 2024).

To model non-PH and flexible survival distributions, DeepHit (Lee et al., 2018) introduced a discrete-time neural approach with time-varying, non-linear effects. The *pycox* library unified DeepSurv, DeepHit, and Cox-Time using PyTorch (Paszke et al., 2019), while TorchSurv (Monod et al., 2024) more recently offered a lower-level API for neural Cox and AFT models with customizable losses.

1.3 Statement of need

Despite the growing adoption of deep learning methods for survival analysis, the R ecosystem lacks native tools for training and deploying deep survival models. Existing solutions, such as the [survivalmodels](#) package (Sonabend, 2024), rely on Python backends accessed via [reticulate](#) (Ushey et al., 2025), which introduces installation complexity and versioning issues. This dependency has limited the accessibility of deep survival modeling for biostatisticians and clinical researchers who rely primarily on R.

1.4 Contribution

To address this gap, we developed [survdnn](#), a fully native R package for deep neural network-based survival analysis. Built on the [torch](#) backend (Falbel and Luraschi, 2025), [survdnn](#) provides a high-level formula interface, supports multiple loss functions including Cox, Cox-L2, AFT, and Cox-Time, and integrates seamlessly with modern R workflows. The package includes training and prediction utilities, built-in evaluation metrics such as the concordance index (C-index), time-dependent Brier score, and Integrated Brier Score (IBS), as well as functions for cross-validation, hyperparameter tuning, and regularization mechanisms including dropout, batch normalization, and early-stopping callbacks. [survdnn](#) complements the rich foundational survival modeling tools in R by extending them into the deep learning era without leaving the R ecosystem.

2 Implementation

2.1 Package overview

At its core, [survdnn](#) is built around the `survdnn()` function, which fits a deep neural network for right-censored survival data using R's formula interface. Users specify the network architecture through an integer vector of hidden layer sizes and select the activation function from a set of supported options, including "relu", "leaky_relu", "tanh", "sigmoid", "gelu", "elu", and "softplus". The training objective is defined via the `loss` argument, with support for "cox", "cox_l2", "aft", and "coxtime", while core hyperparameters such as the learning rate and number of epochs remain fully user-configurable. The training pipeline further incorporates regularization and optimization mechanisms such as dropout, batch normalization, and user-defined callbacks (including early stopping based on training loss dynamics) to improve stability and generalization.

Model predictions are computed using `predict.survdnn()`, which supports three output types: the raw linear predictor ("lp"), survival probabilities at specified time points ("survival"), and cumulative risk at a given time ("risk"). Predictions can be returned either as numeric vectors or as data frames with one column per evaluation time.

The package includes built-in utilities for evaluation and resampling. The `evaluate.survdnn()` function computes common and built-in performance metrics. The `cv.survdnn()` function performs stratified k-fold cross-validation to preserve event censoring balance across folds, with results summarized using `summarize_cv.survdnn()`.

Hyperparameter tuning is supported through two complementary functions. The `tune.survdnn()` function performs k-fold cross-validation over a user-defined hyperparameter grid and selects the best configuration according to a chosen evaluation metric (the first metric is used for selection), optionally refitting the final model on the full dataset. In contrast, `gridsearch.survdnn()` evaluates the same type of hyperparameter grid under a fixed training-validation split: each candidate model is fitted on the training set and assessed on a user-supplied validation set, without internal resampling or automatic refitting.

Missing data are handled explicitly through a user-controlled policy (`na_action = "omit"` or `"fail"`), with informative reporting of excluded observations when applicable.

Finally, `plot.survdmn()` provides a visualization method for predicted survival curves using `ggplot2` style (Wickham, 2016), with options to group curves by a categorical variable and to display either individual or averaged curves.

2.2 Model architecture

The model architecture in `survdmn` is defined as a sequence of k transformation layers applied to the input features $\mathbf{x}$:

$$f_{\theta}(\mathbf{x}) = L_k \circ \phi_{k-1} \circ \dots \circ \phi_1 \circ L_1(\mathbf{x})$$

Each stage consists of a linear transformation followed by non-linear activation and regularization components. The linear transformation at layer k is expressed as:

$$L_k(\cdot) = W_k \mathbf{h}^{(k-1)} + b_k$$

where W_k is the weight matrix and b_k the bias vector. The transformed output is then passed through a composition of non-linear operations. Specifically, each layer includes an activation function ϕ , batch normalization to stabilize learning, and dropout to prevent overfitting. Together, a hidden layer is computed as:

$$\mathbf{h}^{(k)} = \text{Dropout} \left(\phi \left(\text{BatchNorm} \left(W_k \mathbf{h}^{(k-1)} + b_k \right) \right) \right)$$

The final layer outputs a scalar prediction $f_{\theta}(\mathbf{x})$, interpreted based on the selected loss function: as log-risk under Cox-based losses, log-event time under the AFT loss, or time-dependent log-risk under the Cox-Time loss.

Before training, a model matrix is created from the user-specified formula interface. This step expands factor variables into dummy indicators and extracts the design matrix.

2.3 Loss functions

In survival machine learning, the loss function defines the model's training objective and dictates how it handles censoring, time-to-event outcomes, and individual risk dynamics (Wiegbe et al., 2024). This choice critically influences convergence, calibration, and generalization (Katzman et al., 2018).

The `survdmn` package implements four core loss functions tailored for right-censored survival data, covering both PH and non-PH settings:

Cox Partial Likelihood Loss ("cox"): The standard Cox loss implements the partial likelihood from the Cox PH model (Cox, 1972). It is the basis of classical methods and modern neural-based implementations like DeepSurv (Katzman et al., 2018) and pycox (Kvamme et al., 2019).

The model assumes a multiplicative hazard function:

$$\lambda(t | \mathbf{x}) = \lambda_0(t) \cdot \exp(f_{\theta}(\mathbf{x})) = \lambda_0(t) \cdot e^{\eta}$$

where $f_{\theta}(\mathbf{x}) = \eta$ is a learned log-risk score and $\lambda_0(t)$ is an unspecified baseline hazard. To avoid estimating $\lambda_0(t)$, Cox proposed the partial likelihood:

$$\mathcal{L}_{\text{cox}}(\theta) = \prod_{i: \delta_i=1} \frac{e^{\eta_i}}{\sum_{j \in \mathcal{R}(t_i)} e^{\eta_j}}$$

The negative log of this expression is minimized during training. In `survdmn`, this is implemented using `torch_logcumsumexp()` to ensure numerical stability and efficient batch-wise computation.

The Cox loss learns relative risks, not calibrated survival probabilities. However, post-processing via Breslow estimation is performed to obtain full survival curve estimates (Breslow, 1972).

L2-Regularized Cox Loss ("cox_l2"): This variant augments the Cox loss with an L2 penalty applied directly to the predicted log-risk scores:

$$\mathcal{L}_{\text{cox-l2}}(\theta) = \mathcal{L}_{\text{cox}}(\theta) + \lambda \cdot \frac{1}{n} \sum_{i=1}^n \eta_i^2$$

This ridge-type regularization stabilizes optimization and reduces overfitting, particularly in small-sample or high-dimensional settings (Friedman et al., 2010).

Unlike classical weight decay, the penalty acts on model outputs rather than internal parameters, preserving architectural flexibility while directly constraining extreme risk predictions.

Accelerated Failure Time Loss ("aft"): The AFT option in `survdnn` implements a log-normal AFT model with right-censoring, trained by minimizing the censored negative log-likelihood:

$$\log(T_i) = \mu_i + \sigma Z_i, \quad Z_i \sim \mathcal{N}(0, 1)$$

where $\mu_i = f_{\theta}(\mathbf{x}_i) + \text{aft\_loc}$ is a subject-specific location parameter predicted by the neural network, $\sigma > 0$ is a global scale parameter, and aft_loc is an optional centering constant used to improve numerical stability.

For an observed time t_i and event indicator δ_i , the contribution to the negative log-likelihood is:

$$\ell_i(\theta) = \begin{cases} \log t_i + \log \sigma + \frac{1}{2} z_i^2 & \delta_i = 1 \\ -\log S(z_i) & \delta_i = 0 \end{cases}$$

where

$$z_i = \frac{\log t_i - \text{aft\_loc} - f_{\theta}(\mathbf{x}_i)}{\sigma}, \quad S(z) = 1 - \Phi(z)$$

and $\Phi(\cdot)$ denotes the standard normal cumulative distribution function.

The overall loss minimized during training is the average censored negative log-likelihood:

$$\mathcal{L}_{\text{aft}}(\theta) = \frac{1}{n} \sum_{i=1}^n \ell_i(\theta)$$

In `survdnn`, the scale parameter σ can either be fixed or learned jointly with network parameters through a differentiable reparameterization ($\sigma = \exp(\log \sigma)$). This formulation enables the AFT model to fully use both event and censored observations while preserving the direct time-scale interpretation of AFT models.

Cox-Time Loss ("coxtime"): The Cox-Time model extends the Cox framework by allowing covariate effects to vary with time (Kvamme et al., 2019). The log-relative hazard is modeled as:

$$g_{\theta}(t, \mathbf{x}) = \log \left(\frac{h(t | \mathbf{x})}{h_0(t)} \right)$$

yielding the hazard:

$$h(t | \mathbf{x}) = h_0(t) \exp(g_{\theta}(t, \mathbf{x}))$$

The corresponding loss is the time-dependent Cox partial likelihood:

$$\mathcal{L}_{\text{coxtime}}(\theta) = - \sum_{i: \delta_i=1} \left[g_{\theta}(t_i, \mathbf{x}_i) - \log \sum_{j \in \mathcal{R}(t_i)} \exp(g_{\theta}(t_i, \mathbf{x}_j)) \right]$$

In `survdnn`, this loss is implemented using batched evaluation over event times and risk sets. While Cox-Time offers the greatest modeling flexibility among the available losses, it is also the most computationally demanding due to repeated evaluation of risk sets across time.

2.4 Prediction interface

Once a `survdnn` model has been trained, predictions can be obtained using the generic `predict()` method. This function supports multiple output types to serve different goals in survival analysis:

- *Linear predictors* (type = "lp"): returns the log-risk score $\eta = f_{\theta}(\mathbf{x})$, used for risk ranking and concordance index evaluation.
- *Survival probabilities* (type = "survival"): returns $\hat{S}(t | \mathbf{x})$, the estimated survival function evaluated at user-specified time points.
- *Cumulative risk* (type = "risk"): returns the cumulative incidence $\hat{F}(t) = 1 - \hat{S}(t)$, the probability of experiencing the event by time t .

Predictions are returned either as numeric vectors for single time points or as data frames with one column per evaluation time.

The underlying computation depends on the loss function used during training. For models trained with "cox" and "cox_l2", the baseline cumulative hazard $\hat{H}_0(t)$ is estimated using the Breslow estimator.

Survival probabilities are then computed as:

$$\hat{S}(t | \mathbf{x}) = \exp(-\hat{H}_0(t) \exp(\eta))$$

For "aft", predictions are based on the fitted log-normal accelerated failure time model, using the estimated scale parameter $\hat{\sigma}$:

$$\hat{S}(t | \mathbf{x}) = 1 - \Phi\left(\frac{\log t - f_{\theta}(\mathbf{x})}{\hat{\sigma}}\right)$$

where Φ denotes the standard normal cumulative distribution function.

For "coxtime", predictions follow the approach proposed by (Kvamme et al., 2019), using a time-dependent neural log-risk function $g(t, \mathbf{x})$. Survival probabilities are obtained by accumulating predicted hazards over a discrete time grid:

$$\hat{S}(t | \mathbf{x}) = \prod_{s \leq t} \exp(-\hat{\lambda}(s | \mathbf{x}))$$

Survival prediction under Cox-Time requires numerical integration of the time-dependent hazard over a user-defined time grid. While training relies only on observed event times, prediction involves evaluating the neural risk function across grid points for each individual, which can become computationally expensive for large datasets or fine time grids.

2.5 Training workflow

The `survdnn()` function implements a complete training workflow for deep neural networks on right-censored survival data. It relies on R's formula interface to construct design matrices, applies Z-score standardization for numerical stability, and converts inputs into `torch_tensor()` objects for efficient automatic differentiation using the `torch` backend (Keydana, 2023).

Model architectures are defined via `build_dnn()`, which constructs a fully connected multilayer perceptron with user-configurable depth and width. Each hidden layer may optionally include batch normalization and dropout, enabling regularization and improved optimization stability. Non-linear activations are applied uniformly across layers and selected from a predefined set of commonly used functions.

The training objective is specified via the `loss` argument ("`cox`", "`cox_l2`", "`aft`", or "`coxtime`"), each implemented as a differentiable loss function compatible with backpropagation. Optimization is performed using full-batch gradient descent with a choice of optimizers, including Adam, AdamW, SGD, RMSprop, and Adagrad, all exposed through a unified interface.

The empirical risk minimized during training is given by:

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(f_{\theta}(\tilde{\mathbf{x}}_i), t_i, \delta_i)$$

where $\ell(\cdot)$ denotes the selected survival loss and $\tilde{\mathbf{x}}_i$ are standardized covariates.

To improve training robustness and prevent overfitting, `survdnn` supports user-defined callback functions, including early stopping based on the training loss trajectory. Callbacks are evaluated at each epoch and may interrupt training once predefined criteria are met. Reproducibility is ensured by a unified `.seed` argument that synchronizes R and `torch` random number generators.

The fitted model is returned as an S3 object of class "`survdnn`", containing the trained neural network, preprocessing metadata, optimizer configuration, and the full loss history. This object integrates seamlessly with the `predict()` method to compute predictions.

2.6 Evaluation metrics

`survdnn` supports standard survival metrics for assessing both discrimination and calibration at user-defined evaluation time points. Discrimination is evaluated using the C-index (Harrell et al., 1982), which measures the model's ability to correctly rank individuals by risk and is maximized during evaluation and hyperparameter tuning.

Calibration is assessed using the time-dependent Brier score (Graf et al., 1999) and the IBS, both computed with inverse probability of censoring weighting to account for right-censoring (Zhou et al., 2023). These metrics quantify the accuracy of predicted survival probabilities and are minimized during evaluation and hyperparameter tuning.

2.7 Cross-validation

Cross-validation is performed using the `cv_survdnn()` function, which implements stratified k -fold partitioning based on the event indicator to preserve the proportion of observed events and censored observations across folds. Time-to-event values are not stratified, in accordance with standard practice in survival resampling.

Within each fold, models are trained using `survdnn()` on the training split and evaluated on the held-out data using `evaluate_survdnn()` at user-specified time points and metrics. Results are returned in a tidy tabular format, enabling transparent benchmarking, hyperparameter selection, and performance comparison while reducing optimistic bias (Varma and Simon, 2006).

2.8 Tuning

Hyperparameter tuning is crucial in deep survival modeling, where censoring can destabilize optimization and introduce noisy gradients, and small datasets with few events further increase overfitting risk (Gensheimer and Narasimhan, 2019).

`survdnn` provides `tune_survdnn()` to perform grid-based k -fold cross-validation across key hyperparameters, including:

- **Architecture:** hidden layer sizes (hidden)
- **Activation:** "relu", "leaky_relu", "tanh", "sigmoid", "gelu", "elu", "softplus"
- **Optimization:** learning rate (lr), number of epochs (epochs)
- **Loss function:** "cox", "cox_l2", "aft", "covertime"

This design follows best practices in regularized learning (Friedman et al., 2010) and deep neural optimization (Bengio, 2012). An optional `refit = TRUE` automatically retrains the best configuration and fits the model on the full dataset in an AutoML style.

2.9 Sensitivity to censoring

Because censoring directly affects the effective sample size and the information available for risk estimation, we investigated the sensitivity of `survdnn` models to increasing levels of right censoring. Using a controlled simulation design, we evaluated predictive discrimination (C-index) and calibration (IBS) across censoring rates ranging from 10% to 90%.

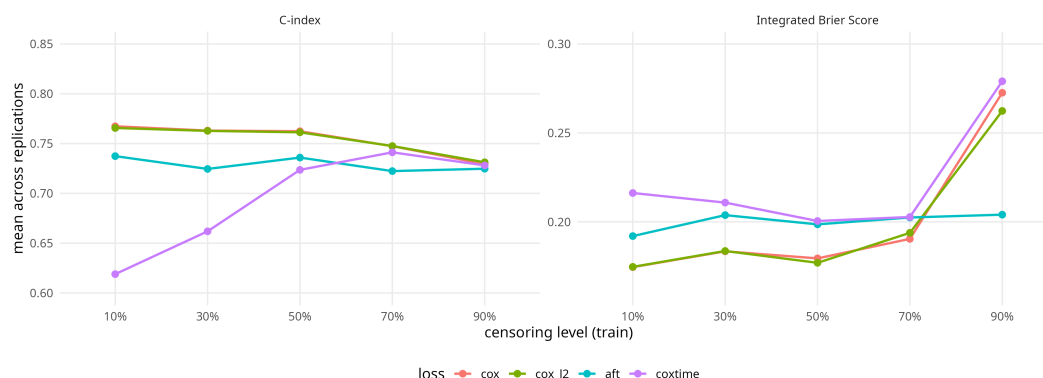


Figure 1: Censoring sensitivity analysis. C-index and IBS of `survdnn` models across increasing right-censoring rates (10-90%) for the four implemented loss functions.

Across losses, discrimination (C-index) decreases only moderately with increasing censoring, whereas calibration (IBS) is most affected at extreme censoring (Figure 1). Cox-based losses ("cox" and "cox_l2") show stable C-index up to $\approx 50\%$ censoring, followed by a gradual decline, while IBS remains low up to $\approx 70\%$ but increases sharply at 90% censoring. The AFT loss exhibits comparatively stable performance across censoring levels for both C-index and IBS, and yields the lowest IBS under very high censoring (90%). In contrast, the Cox-Time loss shows poor discrimination at low censoring but improves markedly up to $\approx 70\%$, while maintaining higher IBS than the other losses and deteriorating strongly at 90% censoring, consistent with increased variance under limited effective sample size.

These results suggest that, in practice, Cox-based losses are preferable under low to moderate censoring, offering a favorable balance between discrimination and calibration. The AFT loss demonstrates robust calibration across censoring levels, including under very high censoring, but yields comparatively lower discrimination, indicating that its use should be guided by the relative importance of ranking versus probability accuracy. Highly

flexible models such as Cox-Time require larger effective sample sizes and careful calibration assessment, as their apparent gains in discrimination at moderate censoring are offset by substantially poorer calibration under extreme censoring.

2.10 Computation backend

Since `survdnn` is built on the `torch` backend, it supports both CPU and GPU execution. The computation device can be selected via `.device = c("auto", "cpu", "cuda")`, where "auto" uses CUDA when available and otherwise falls back to CPU. On CPU, performance benefits from multithreaded linear algebra provided by the underlying BLAS/LAPACK implementation.

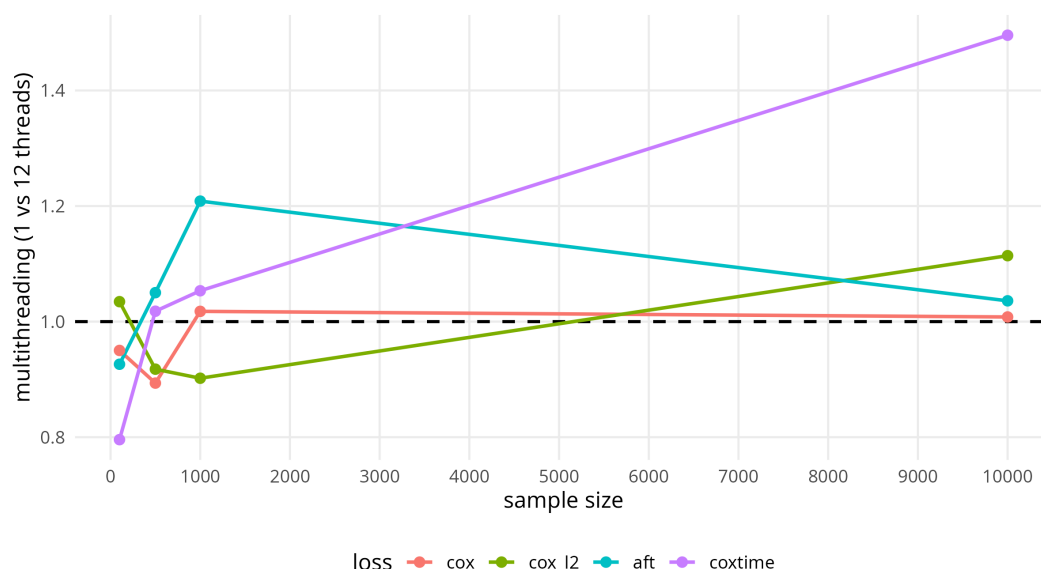


Figure 2: CPU multithreading speedup across loss functions. Relative training-time speedup comparing multi-threaded and single-thread CPU execution as a function of sample size for Cox, Cox-L2, AFT, and Cox-Time losses.

To characterize computational behavior, we evaluated training time across increasing sample sizes and compared single-thread and multi-thread CPU execution for different loss functions (Figure 2). Multithreading benefits vary markedly across losses and depend strongly on sample size. Cox-based losses exhibit little to no speedup, with performance remaining close to single-thread execution even at large sample sizes, reflecting the dominance of global, sequential risk-set operations. The AFT loss shows moderate speedup already at small to moderate sample sizes, but scaling gains diminish as sample size increases, consistent with dense likelihood computations that limit parallel efficiency. In contrast, the Cox-Time loss demonstrates substantial and increasing speedup with sample size, achieving the largest gains under multithreading, indicating favorable parallelization of its time-dependent risk evaluations despite higher overhead at small sample sizes.

2.11 Reproducibility

Reproducibility in `survdnn` requires controlling both R-level and `torch`-level sources of randomness. While `set.seed()` controls randomness in R, neural network training relies on `torch`, which maintains its own random number generator for operations such as weight initialization and dropout.

To simplify reproducible workflows, `survdnn` exposes a `.seed` argument in its main fitting and resampling functions (including `survdnn()`, `cv_survdnn()`, and `tune_survdnn()`). When provided, this argument internally synchronizes both R and `torch` random number generators to ensure fully reproducible model training and evaluation.

3 Usage examples

We illustrate the use of `survdnn` through two case studies: a complete modeling pipeline on the veteran's lung cancer dataset and a benchmark on the real-world METABRIC breast cancer dataset.

3.1 Example 1: API demonstration

This example uses the Veteran's lung cancer dataset from the `survival` package (Terry M. Therneau and Patricia M. Grambsch, 2000) to illustrate a typical `survdnn` workflow: fit a model, inspect the training configuration, compute survival predictions at user-defined time points, evaluate with standard metrics (C-index, IBS, Brier), perform cross-validation, tune hyperparameters, and visualize training dynamics and survival curves.

First, we fit a Cox-loss model with a modest multilayer architecture:

```
pacman::p_load(survdnn, survival, dplyr, ggplot2, tibble, randomForestSRC, partykit, party, pec, rsample)

veteran <- survival::veteran

mod <- survdnn(
  Surv(time, status) ~ age + karno + celltype,
  data = veteran,
  hidden = c(32, 16),
  epochs = 300,
  loss = "cox",
  verbose = TRUE,
  .seed = 123
)
```

We can summarise the model to inspect the architecture, preprocessing, and training setup:

```
summary(mod)

#>
#> Formula:
#>   Surv(time, status) ~ age + karno + celltype
#> <environment: 0x16b867538>
#>
#> Model architecture:
#>   Hidden layers: 32 : 16
#>   Activation: relu
#>   Dropout: 0.3
#>   Batch norm: TRUE
#>   Final loss: 3.807069
#>
#> Training summary:
#>   Epochs: 300
#>   Learning rate: 1e-04
#>   Loss function: cox
#>   Optimizer: adam
#>   Device: cpu
#>   CPU threads: default
#>   NA action: omit
#>
#> Data summary:
```

```
#> Observations (used/total): 137 / 137
#> Predictors (5): age, karno, celltypesmallcell, celltypeadeno, celltypelarge
#> Time range: [ 1, 999 ]
#> Events / censored: 128 / 9
#> Event rate: 93.4%
#> Predictors standardized: yes
```

Next, we can obtain survival probabilities at clinically meaningful horizons (30, 90, 180 days):

```
times_eval <- c(30, 90, 180)
pred_surv <- predict(mod, newdata = veteran, type = "survival", times = times_eval)
head(pred_surv)
```

```
#>      t=30      t=90      t=180
#> 1 0.8500299 0.6484395 0.3320248
#> 2 0.8908178 0.7347433 0.4563402
#> 3 0.8837166 0.7192317 0.4322180
#> 4 0.8764462 0.7035644 0.4086562
#> 5 0.8869652 0.7263022 0.4431147
#> 6 0.7840704 0.5228142 0.1919280
```

Model performance can be assessed via the C-index and the IBS:

```
eval_res <- evaluate_survdnn(mod, metrics = c("cindex", "ibs"), times = times_eval)
print(eval_res)
```

```
#> # A tibble: 2 x 2
#>   metric value
#>   <chr>   <dbl>
#> 1 cindex 0.745
#> 2 ibs    0.169
```

The Brier score can also be computed at specific time points (or over a grid, if desired):

```
times_eval <- seq(min(veteran$time), max(veteran$time), 10)
eval_res <- evaluate_survdnn(mod, metrics = "brier", times = times_eval)
print(eval_res)
```

```
#> # A tibble: 100 x 3
#>   metric time    value
#>   <chr>   <dbl>   <dbl>
#> 1 brier     1 0.000073
#> 2 brier    11 0.0880
#> 3 brier    21 0.150
#> 4 brier    31 0.181
#> 5 brier    41 0.192
#> 6 brier    51 0.194
#> 7 brier    61 0.199
#> 8 brier    71 0.198
#> 9 brier    81 0.193
#> 10 brier   91 0.188
#> # i 90 more rows
```

To assess stability, we can run k -fold cross-validation with predefined metrics and time points:

```

cv_results <- cv_survdnn(
  Surv(time, status) ~ age + karno + celltype, data = veteran,
  times = c(30, 90, 180),
  metrics = c("ibs"),
  folds = 3,
  hidden = c(16, 8),
  loss = "cox",
  epochs = 100,
  .seed = 123
)
print(cv_results)

#> # A tibble: 3 x 3
#>   fold metric value
#>   <int> <chr>   <dbl>
#> 1     1 ibs     0.230
#> 2     2 ibs     0.226
#> 3     3 ibs     0.230

```

Hyperparameter tuning can be automated over a small grid. Below we vary architecture, optimiser settings, and loss:

```

grid <- list(
  hidden = list(c(16, 32, 64, 16)),
  lr = c(1e-3),
  activation = c("relu", "softplus"),
  epochs = c(100),
  loss = c("cox_l2", "aft")
)

tune_res <- tune_survdnn(
  Surv(time, status) ~ age + karno + celltype,
  data = veteran,
  times = c(30, 90, 180),
  metrics = "cindex",
  param_grid = grid,
  folds = 3,
  refit = FALSE,
  return = "all"
)

benchmark_results <- summarize_tune_survdnn(tune_res, by_time = TRUE)

tuning_results_clean <- benchmark_results %>%
  mutate(
    hidden = sapply(hidden, function(x) paste(x, collapse = "-")),
    across(where(is.numeric), ~ round(.x, 3))
  )

tuning_results_clean

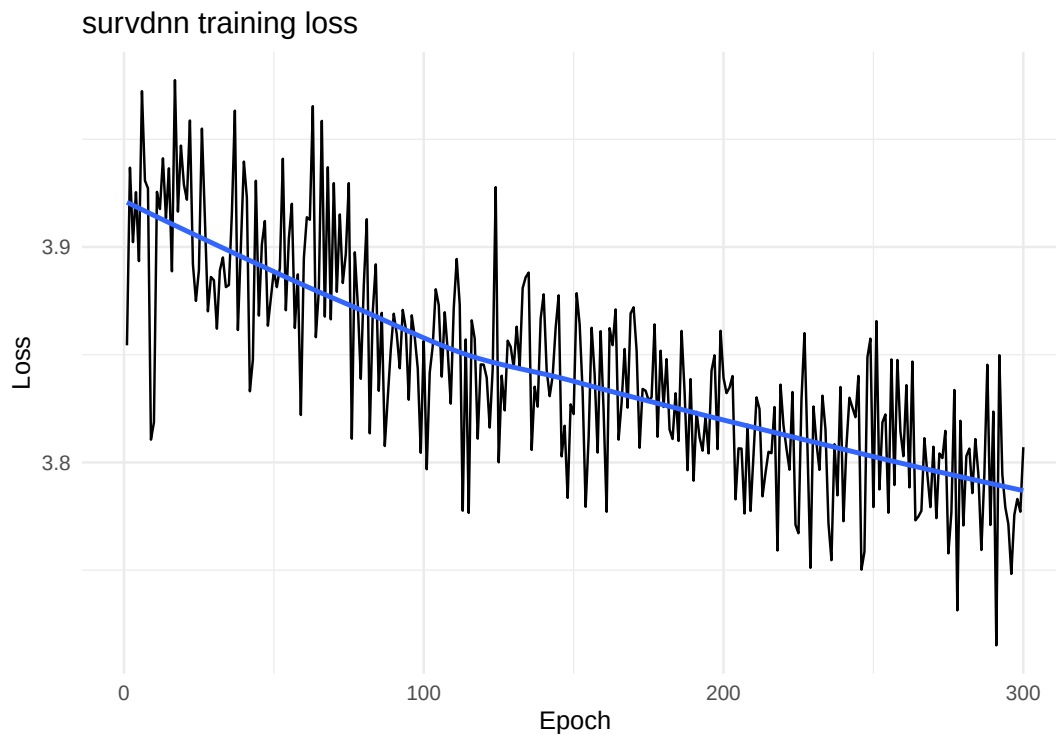
#> # A tibble: 4 x 8
#>   hidden      lr activation epochs loss  metric mean  sd
#>   <chr>      <dbl> <chr>      <dbl> <chr> <chr> <dbl> <dbl>
#> 1 16-32-64-16 0.001 softplus    100 aft  cindex 0.737 0.072
#> 2 16-32-64-16 0.001 softplus    100 cox_l2 cindex 0.736 0.074

```

```
#> 3 16-32-64-16 0.001 relu      100 aft      cindex 0.723 0.079
#> 4 16-32-64-16 0.001 relu      100 cox_l2 cindex 0.715 0.06
```

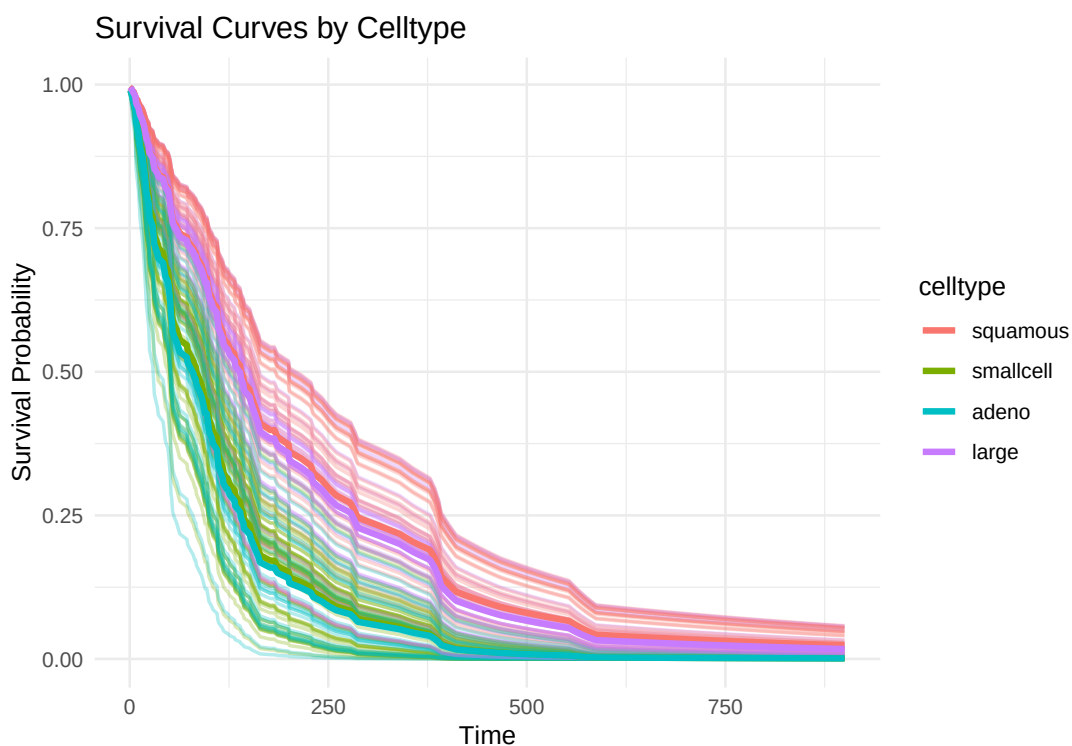
Training dynamics can be visualised using the built-in `plot_loss()` helper:

```
plot_loss(mod, smooth = TRUE)
```

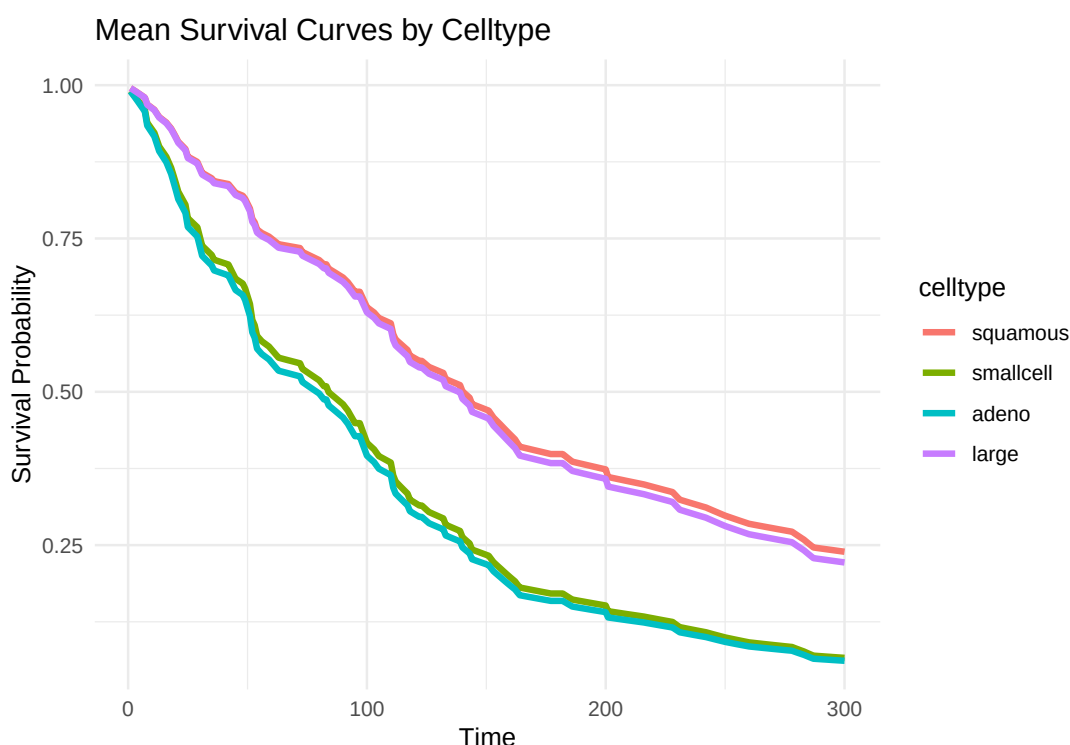


Finally, we visualise survival curves by cancer cell type, showing both individual and mean summaries:

```
plot(mod, group_by = "celltype", times = 1:900) +
  ggtitle("Survival Curves by Celltype")
```




```
plot(mod, group_by = "celltype", times = 1:300, plot_mean_only = TRUE) +
  ggtitle("Mean Survival Curves by Celltype")
```



3.2 Example 2: METABRIC benchmark

Here we benchmark [survdnn](#) against Cox PH, Random Survival Forests (RSF), and Conditional Inference Forests (CForest) using five-fold cross-validation on the METABRIC breast cancer dataset ([Mukherjee et al., 2018](#)). All models are trained on identical splits and evaluated with the C-index at predefined time points.

```
#> # A tibble: 4 x 3
#>   model    mean_cindex sd_cindex
#>   <chr>      <dbl>      <dbl>
#> 1 SurvDNN    0.611      0.029
#> 2 CoxPH      0.604      0.029
#> 3 RSF        0.554      0.033
#> 4 CForest    0.52       0.044
```

4 Discussion

The [survdnn](#) package fills an important gap in the R ecosystem by providing a fully native, [torch](#)-based framework for deep survival modeling ([Falbel and Luraschi, 2025](#)), without relying on Python backends. This design choice improves installation reliability and facilitates seamless integration with established R workflows for statistical modeling and reproducible research.

Across multiple benchmarks and sensitivity analyses, [survdnn](#) demonstrates competitive predictive performance relative to established survival models, including Cox PH models ([Cox, 1972](#)), RSF ([Ishwaran et al., 2008](#)), and CForest ([Hothorn et al., 2006](#)). The availability of multiple likelihood-based loss functions enables analysts to adapt modeling assumptions to PH, non-PH, non-linear covariate effects, and time-varying effects within a unified deep learning framework. All implemented losses explicitly account for right-censoring through

either partial likelihoods (Cox, L2-penalised Cox, Cox-Time) or fully specified censored likelihoods (AFT).

Additional investigations highlight practical trade-offs in both statistical robustness and computational behavior. Sensitivity analyses across increasing censoring levels (Figure 1) show that discrimination and calibration degrade as expected with reduced effective sample size, with differences across loss functions. Complementary runtime experiments (Figure 2) demonstrate that computational behavior varies substantially by loss function: Cox-based losses exhibit low computational overhead but derive little benefit from multithreaded execution, Cox-Time incurs higher per-iteration cost due to repeated evaluation of time-dependent risk scores yet scales efficiently with parallel execution at larger sample sizes, and AFT likelihoods involve dense computations that benefit from parallelism at moderate sizes but show limited scalability as sample size increases.

The current scope of `survdnn` is intentionally focused on tabular data, using multilayer perceptron architectures (Bengio et al., 2021) that are well aligned with the structure of most survival analysis applications. This design choice prioritizes stability and compatibility with classical survival modeling workflows. Training is performed using full-batch optimization, which favors reproducibility and stable convergence but may limit scalability to very large datasets.

Ongoing work maps `survdnn` to the `mlr3` ecosystem (Lang et al., 2019) via a dedicated survival learner, providing users with additional access to standardized benchmarking, resampling, and tuning workflows.

Planned developments include expanded support for time-dependent evaluation metrics (Uno et al., 2007; Blanche et al., 2013) and the incorporation of additional survival loss functions. These directions aim to broaden the applicability of `survdnn` while preserving its core emphasis on transparent and statistically principled modeling.

Overall, `survdnn` provides a practical and extensible foundation for deep survival modeling in R, combining modern deep learning methodology with the transparency and reproducibility expected in statistical computing environments.

5 Acknowledgements

The author thanks the anonymous referees for their valuable comments and suggestions, which substantially improved the quality and clarity of this manuscript.

References

- S. Batson, G. Greenall, and P. Hudson. Review of the reporting of survival analyses within randomised controlled trials and the implications for meta-analysis. *PLoS One*, 11(5): e0154870, 2016. [p382]
- Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade: Second edition*, pages 437–478. Springer, 2012. [p388]
- Y. Bengio, Y. Lecun, and G. Hinton. Deep learning for AI. *Communications of the ACM*, 64(7): 58–65, 2021. [p382, 395]
- P. Blanche, J.-F. Dartigues, and H. Jacqmin-Gadda. Estimating and comparing time-dependent areas under receiver operating characteristic curves for censored event times with competing risks. *Statistics in Medicine*, 32(30):5381–5397, 2013. [p395]
- N. E. Breslow. Contribution to discussion of paper by DR Cox. *Journal of the Royal Statistical Society, Series B*, 34:216–217, 1972. [p385]
- D. R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(2):187–202, 1972. [p382, 384, 394]

- D. Falbel and J. Luraschi. *torch: Tensors and Neural Networks with 'GPU' Acceleration*, 2025. URL <https://torch.mlverse.org/docs>. R package version 0.15.0. [p383, 394]
- D. Faraggi and R. Simon. A neural network model for survival data. *Statistics in Medicine*, 14(1):73–82, 1995. [p382]
- J. H. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010. [p385, 388]
- M. F. Gensheimer and B. Narasimhan. A scalable discrete-time survival model for neural networks. *PeerJ*, 7:e6257, 2019. [p388]
- E. Graf, C. Schmoor, W. Sauerbrei, and M. Schumacher. Assessment and comparison of prognostic classification schemes for survival data. *Statistics in Medicine*, 18(17-18): 2529–2545, 1999. [p387]
- F. E. Harrell, R. M. Califf, D. B. Pryor, K. L. Lee, and R. A. Rosati. Evaluating the yield of medical tests. *JAMA*, 247(18):2543–2546, 1982. [p387]
- P. J. Heagerty and Y. Zheng. Survival model predictive accuracy and ROC curves. *Biometrics*, 61(1):92–105, 2005. [p382]
- T. Hothorn, P. Bühlmann, S. Dudoit, A. Molinaro, and M. J. Van Der Laan. Survival ensembles. *Biostatistics*, 7(3):355–373, 2006. [p394]
- H. Ishwaran, U. B. Kogalur, E. H. Blackstone, and M. S. Lauer. Random survival forests. 2008. [p394]
- J. L. Katzman, U. Shaham, A. Cloninger, J. Bates, T. Jiang, and Y. Kluger. DeepSurv: personalized treatment recommender system using a Cox proportional hazards deep neural network. *BMC Medical Research Methodology*, 18(1):24, 2018. [p382, 384]
- S. Keydana. *Deep learning and scientific computing with R torch*. Chapman and Hall/CRC, 2023. [p387]
- H. Kvamme, Ø. Borgan, and I. Scheel. Time-to-event prediction with neural networks and Cox regression. *Journal of Machine Learning Research*, 20(129):1–30, 2019. [p382, 384, 385, 386]
- M. Lang, M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, Q. Au, G. Casalicchio, L. Kotthoff, and B. Bischl. mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*, dec 2019. doi: 10.21105/joss.01903. URL <https://joss.theoj.org/papers/10.21105/joss.01903>. [p395]
- C. Lee, W. Zame, J. Yoon, and M. Van Der Schaar. Deephit: A deep learning approach to survival analysis with competing risks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. [p382]
- M. Monod, P. Krusche, Q. Cao, B. Sahiner, N. Petrick, D. Ohlssen, and T. Coroller. Torchsurv: A lightweight package for deep survival analysis. *arXiv preprint arXiv:2404.10761*, 2024. [p382]
- A. Mukherjee, R. Russell, S.-F. Chin, B. Liu, O. Rueda, H. Ali, G. Turashvili, B. Mahler-Araujo, I. Ellis, S. Aparicio, et al. Associations between genomic stratification of breast cancer and centrally reviewed tumour pathology in the METABRIC cohort. *npj Breast Cancer*, 4(1):5, 2018. [p394]
- 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. [p382]

- D. J. Sargent. Comparison of artificial neural networks with other statistical approaches: results from medical data sets. *Cancer: Interdisciplinary International Journal of the American Cancer Society*, 91(S8):1636–1642, 2001. [p382]
- R. Sonabend. *survivalmodels: Models for Survival Analysis*, 2024. URL <https://CRAN.R-project.org/package=survivalmodels>. R package version 0.1.191. [p383]
- Terry M. Therneau and Patricia M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer, New York, 2000. ISBN 0-387-98784-3. [p390]
- H. Uno, T. Cai, L. Tian, and L.-J. Wei. Evaluating prediction rules for t-year survivors with censored regression models. *Journal of the American Statistical Association*, 102(478):527–537, 2007. [p395]
- K. Ushey, J. Allaire, and Y. Tang. *reticulate: Interface to ‘Python’*, 2025. URL <https://github.com/rstudio/reticulate>. R package version 1.42.0.9000, commit 3e1fa25e27d151a269172379e678b59c100221f9. [p383]
- S. Varma and R. Simon. Bias in error estimation when using cross-validation for model selection. *BMC Bioinformatics*, 7(1):91, 2006. [p387]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p384]
- S. Wiegrefe, P. Kopper, R. Sonabend, B. Bischl, and A. Bender. Deep learning for survival analysis: a review. *Artificial Intelligence Review*, 57(3):65, 2024. [p382, 384]
- A. Xiang, P. Lapuerta, A. Ryutov, J. Buckley, and S. Azen. Comparison of the performance of neural network methods and Cox regression for censored survival data. *Computational Statistics & Data Analysis*, 34(2):243–257, 2000. [p382]
- H. Zhou, H. Wang, S. Wang, and Y. Zou. SurvMetrics: An R package for predictive evaluation metrics in survival analysis. *R Journal*, 14(4):252–263, 2023. [p387]

Imad EL-BADISY

AI and Data Science Service, Mohammed VI Center for Research and Innovation (CM6RI)

Rabat, Morocco

ORCID: [0000-0003-4848-4895](https://orcid.org/0000-0003-4848-4895)

elbadisyimad@gmail.com

R Foundation News

by *Torsten Hothorn*

1 Donations and members

Membership fees and donations received between 2026-01-05 and 2026-04-10.

2 Donations

b-data GmbH (Switzerland); Gilberto Camara (Brazil); Jeevaka Dassanayake (United States); One Health Collaborating Center Universitas Gadjah Mada (Indonesia); Rudolph Martin (United States); The University of Auckland, Statistics Department (New Zealand); Dr. Alfred Wagner (Germany).

3 Supporting institutions

Ngo Thaiduong (Vietnam); Ef-prime, Inc., Tokyo (Japan); Institute of Botany of the Czech Academy of Sciences, Pruhonice (Czechia); Universitätsspital Basel, Basel (Switzerland).

4 Supporting members

Matthew Bhagat-Conway (United States); Michael Blanks (United States); Emmanuel Blondel (France); Tom Boulay (United States); James Briggs (United States); Pina Brinker (Germany); Cédric Chambru (Switzerland); John Chandler (United States); Chao Cheng (China); Gerard Conaghan (United Kingdom); Alistair Cullum (United States); James Curran (New Zealand); Brandon Dahl (United States); Martin Elff (Germany); Isaac Florence (United Kingdom); Bernd Fröhlich (Germany); Jan Marvin Garbuszus (Germany); Eduardo García Galea (Spain); Sujoy Ghosh (United States); Frank Hafner (United States); Philippe Heymans Smith (Costa Rica); Delia Hol (Paraguay); Alexander Huelle (Germany); Heidi Imker (United States); Ben Jefferies (United States); Sebastian Jentschke (Norway); Ziyad Knio (United States); Sebastian Koehler (Germany); Maryna Kołeczek (Poland); Chris Kutty (United States); Luca La Rocca (Italy); Caleb Lareau (United States); Eric Lim (United Kingdom); Baoxiao Liu (Netherlands); Philippe Michel (France); Ernst Molitor (Germany); David Monderde (Spain); Stefan Moog (Germany); Steffen Moritz (Germany); Andrew Morrison (Canada); Tsubasa Narihiro (Japan); Maciej Nasinski (Poland); Tan Duc Nguyen (Vietnam); Ana Ns (Paraguay); Bastián Olea Herrera (Chile); Antonio Paez (Canada); Brynjar Snær Pálsson (Iceland); PierGianLuca Porta Mana (Norway); Joyce Robbins (United States); Ingo Ruczinski (United States); Erik Ruzek (United States); Dejan Schuster (Germany); Harald Sterly (Germany); Robert van den Berg (Austria); Fredrik Wartenberg (Sweden); Nan Xiao (United States); Yihui Xie (China); Shinya Suzuki (Japan).

Torsten Hothorn

Universität Zürich

Switzerland

ORCID: [0000-0001-8301-0471](https://orcid.org/0000-0001-8301-0471)

Torsten.Hothorn@R-project.org