

The R Journal

Volume 15/4, December 2023

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

Contents

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

Contributed Research Articles

SIMEXBoost: An R package for Analysis of High-Dimensional Error-Prone Data Based on Boosting Method	5
[binGroup2](https://CRAN.R-project.org/package=binGroup2): Statistical Tools for Infection Identification via Group Testing	21
multiocc: An R Package for Spatio-Temporal Occupancy Models for Multiple Species .	37
Accessible Computation of Tight Symbolic Bounds on Causal Effects using an Intuitive Graphical Interface	52
singR: An R Package for Simultaneous Non-Gaussian Component Analysis for Data Integration	68
RobustCalibration: Robust Calibration of Computer Models in R	83
glmmPen: High Dimensional Penalized Generalized Linear Mixed Models	105
Unified ROC Curve Estimator for Diagnosis and Prognosis Studies: The sMSROC Package	128
Sparse Model Matrices for Multidimensional Hierarchical Aggregation	149
openalexR: An R-Tool for Collecting Bibliometric Data from OpenAlex	166
Computer Algebra in R Bridges a Gap Between Symbolic Mathematics and Data in the Teaching of Statistics and Data Science.	180
A Comparison of R Tools for Nonlinear Least Squares Modeling	197
exvatoools: Value Added in Exports and Other Input-Output Table Analysis Tools. .	215
PLreg: An R Package for Modeling Bounded Continuous Data	235
Inference for Network Count Time Series with the R Package PNAR	254
SURvival Control Chart EStimation Software in R: the success Package	269

News and Notes

Changes in R	291
Changes on CRAN	294
News from the Forwards Taskforce	296
R Foundation News	298

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:

Simon Urbanek, University of Auckland, New Zealand

Executive editors:

Catherine Hurley, Maynooth University, Ireland

Mark van der Loo, Statistics Netherlands, The Netherlands

Rob Hyndman, Monash University, Australia

Emi Tanaka, Australian National University, Australia

R Journal Homepage:

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

Email of editors and editorial board:

r-journal@R-project.org

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

Editorial

by Simon Urbanek

On behalf of the editorial board, I am pleased to present Volume 15 Issue 4 of the R Journal.

This is my last issue as Editor-in-Chief. Mark van der Loo takes over as Editor-in-Chief for 2024, having served as an Executive Editor since 2022 and as Associate Editor since 2021.

Catherine Hurley recently finished her Editorial board term. She has been leading the expansion to four issues a year as the first Editor-in-Chief to oversee the process, and worked tirelessly to reduce the turn-around times despite an increasing number of submissions.

During the last year, Gavin Simpson stepped down from his editorial board role, having served two years. Beth Atkinson and Earo Wang have completed their terms as Associate Editors. On behalf of the board, I would like to thank Gavin, Beth and Earo for their hard work on behalf of the Journal. New additions are Rob Hyndman who has joined the Editorial board, and Vincent Arel-Bundock who has just joined the Associate Editor team.

The articles in this issue have been carefully copy edited by Adam Bartonicek, Chase Robertson and Taylor Lee.

In this issue

News from CRAN, the R core Development Team, Forwards Taskforce and the R Foundation are included in this issue.

This issue features 16 contributed research articles the majority of which relate to R packages on a diverse range of topics. All packages are available on CRAN. Supplementary material with fully reproducible code is available for download from the Journal website.

- SIMEXBoost: An R package for Analysis of High-Dimensional Error-Prone Data Based on Boosting Method
- binGroup2: Statistical Tools for Infection Identification via Group Testing
- multiocc: An R Package for Spatio-Temporal Occupancy Models for Multiple Species
- Accessible Computation of Tight Symbolic Bounds on Causal Effects using an Intuitive Graphical Interface
- singR: An R Package for Simultaneous Non-Gaussian Component Analysis for Data Integration
- RobustCalibration: Robust Calibration of Computer Models in R
- glmmPen: High Dimensional Penalized Generalized Linear Mixed Models
- Unified ROC Curve Estimator for Diagnosis and Prognosis Studies: The sMSROC Package
- Sparse Model Matrices for Multidimensional Hierarchical Aggregation
- openalexR: An R-Tool for Collecting Bibliometric Data from OpenAlex
- Computer Algebra in R Bridges a Gap Between Symbolic Mathematics and Data in the Teaching of Statistics and Data Science
- A Comparison of R Tools for Nonlinear Least Squares Modeling
- exvatoools: Value Added in Exports and Other Input-Output Table Analysis Tools
- PLreg: An R Package for Modeling Bounded Continuous Data
- Inference for Network Count Time Series with the R Package PNAR
- SURvival Control Chart EStimation Software in R: the success Package

Simon Urbanek
University of Auckland

<https://journal.r-project.org>
ORCID: 0000-0003-2297-1732
r-journal@r-project.org

SIMEXBoost: An R package for Analysis of High-Dimensional Error-Prone Data Based on Boosting Method

by Li-Pang Chen and Bangxu Qiu

Abstract Boosting is a powerful statistical learning method. Its key feature is the ability to derive a strong learner from simple yet weak learners by iteratively updating the learning results. Moreover, boosting algorithms have been employed to do variable selection and estimation for regression models. However, measurement error usually appears in covariates. Ignoring measurement error can lead to biased estimates and wrong inferences. To the best of our knowledge, few packages have been developed to address measurement error and variable selection simultaneously by using boosting algorithms. In this paper, we introduce an R package **SIMEXBoost**, which covers some widely used regression models and applies the simulation and extrapolation method to deal with measurement error effects. Moreover, the package **SIMEXBoost** enables us to do variable selection and estimation for high-dimensional data under various regression models. To assess the performance and illustrate the features of the package, we conduct numerical studies.

1 Introduction

In statistical analysis, regression models are important methods for characterizing the relationship between response and the covariates. When the response follows exponential family distributions, generalized linear models (GLM) are commonly used to link the response and the covariates. If the response is taken as failure time and is incomplete due to censoring (e.g., Lawless, 2003), the accelerated failure time model (AFT) might be one of useful strategies to characterize the survival outcome in survival analysis. In recent years, complex modeling structures have been explored when building GLM or survival models, including semi-parametric or mixed-effects structures. To address these challenges, several statistical learning methods, including the boosting approaches (e.g., Hastie et al., 2008), have been developed.

In the contemporary statistical analysis, researchers may frequently encounter high-dimensionality in variables. In particular, high-dimensional data may contain many irrelevant covariates that may affect analysis results. Therefore, it is crucial to do variable selection. In the development of statistical methods, some useful strategies have been proposed, such as regularization approaches (e.g., Tibshirani, 1996; Zou, 2006; Zou and Hastie, 2005) or feature screening methods (e.g., Chen, 2021; Chen, 2023b). In addition, Wolfson (2011) and Brown et al. (2017) proposed the boosting method to do variable selection, which avoids having to deal with non-differentiable penalty functions. The other important feature is measurement error in variables, which is ubiquitous in applications. Moreover, ignoring measurement error effects may affect the estimation results (e.g., Chen and Yi, 2021). Therefore, it is necessary to correct for measurement error effects. A large body of methods has been well established to address variable selection, correction of measurement error, or both. Recently, Chen (2023c) developed the SIMEX method and the regression calibration method with the boosting algorithm accommodated to handle variable selection and measurement error correction for GLMs. Chen and Qiu (2023) considered the AFT model to fit time-to-event responses and proposed the SIMEX method to address measurement error. Chen (2023d) derived the corrected estimating function based on the logistic regression or probit models and applied the boosting method to do variable selection for binary outcomes.

In applications, several commonly used packages associated with existing methods have been developed for public use. A detailed list is summarized in Table 1. Specifically, with variable selection and measurement error ignored, most packages related to boosting methods, including **bst** and **adabag**, can handle classification with binary or multi-class responses. Some packages can deal with different response families. For example, **xgboost** and **lightgbm** can deal with continuous responses; **gbm** and **gamboostLSS** can be used to model survival data under the Cox model and the AFT model, respectively; **GMMBoost** is useful for handling mixed-effects models. In the presence of high-dimensional but precisely measured variables, **glmnet** and **SIS** are two popular packages for variable selection or feature screening, respectively. On the contrary, if variables are subject to measurement error, the two packages **GLSME** and **mecor** focus on linear models and aim to adjust for measurement error effects in the response and/or covariates. Moreover, the simulation and extrapolation (SIMEX) method (e.g., Chen and Yi, 2021; Carroll et al., 2006) has been a powerful strategy to correct for measurement error effects, and has been widely used under several types of regression models in

existing R packages, including **simex** for GLM, **augSIMEX** for GLM with error-prone continuous and discrete variables, and **simexaft** for the AFT model. In addition to the R software, [Chen \(2023a\)](#) developed a Python package **BOOME** to handle variable selection and measurement error for binary outcomes.

Table 1: Comparisons among existing and proposed packages. This table summarizes three categories of packages: (i) variable selection without measurement error correction (**glmnet**, **SIS**), (ii) measurement error correction without variable selection (**GLSME**, **mecor**, **augSIMEX**, **simex**, **simexaft**), (iii) statistical learning approaches that handle estimation without consideration of measurement error and variable selection (**bst**, **xgboost**, **gbm**, **adabag**, **lightgbm**, **GMMBoost**, **gamboostLSS**). The proposed package **SIMEXBoost** is included in these three categories. In the Usage heading, ‘LM’ denotes the linear model; ‘Class’ is the classification; ‘Pois’ is the Poisson regression, ‘Cox’ is the Cox model; ‘AFT’ is the AFT model; ‘SL’ is statistical learning; ‘ME’ represents measurement error correction; ‘VS’ is variable selection; and ‘Col’ represents collinearity.

Packages	Usage								
	LM	Class ¹	Pois	Cox	AFT	SL ²	ME	VS	Col
SIMEXBoost	✓	✓	✓	✗	✓	✓	✓	✓	✓
Qiu and Chen (2023)	✓	✓	✓	✓	✗	✗	✗	✓	✓
glmnet	✓	✓	✓	✓	✗	✗	✗	✓	✓
Friedman et al. (2023)	✓	✓	✓	✓	✗	✗	✗	✓	✗
SIS	✓	✓	✓	✓	✗	✗	✗	✓	✗
Feng et al. (2020)	✓	✗	✗	✗	✗	✗	✓	✗	✗
GLSME	✓	✗	✗	✗	✗	✗	✓	✗	✗
Bartoszek (2019)	✓	✗	✗	✗	✗	✗	✓	✗	✗
mecor	✓	✗	✗	✗	✗	✗	✓	✗	✗
Nab (2021)	✓	✓	✓	✗	✗	✗	✓	✗	✗
augSIMEX	✓	✓	✓	✗	✗	✗	✓	✗	✗
Zhang and Yi (2020)	✓	✓	✓	✓	✗	✗	✓	✗	✗
simex	✓	✓	✓	✓	✗	✗	✓	✗	✗
Lederer et al. (2019)	✓	✓	✓	✓	✓	✗	✓	✗	✗
simexaft	✗	✗	✗	✗	✓	✗	✓	✗	✗
Xiong et al. (2019)	✗	✓	✗	✗	✗	✓	✗	✗	✗
bst	✗	✓	✗	✗	✗	✓	✗	✗	✗
Wang and Hothorn (2023)	✓	✓	✗	✗	✗	✓	✗	✗	✗
xgboost	✓	✓	✗	✗	✗	✓	✗	✗	✗
Chen et al. (2023)	✓	✓	✓	✓	✗	✓	✗	✗	✗
gbm	✓	✓	✓	✓	✓	✓	✗	✗	✗
Greenwell et al. (2022)	✓	✓	✓	✓	✓	✓	✗	✗	✗
adabag	✗	✓	✗	✗	✗	✓	✗	✗	✗
Alfaro et al. (2023)	✓	✓	✗	✗	✗	✓	✗	✗	✗
lightgbm	✓	✓	✗	✗	✗	✓	✗	✗	✗
Shi et al. (2023)	✓	✓	✗	✗	✗	✓	✗	✗	✗
GMMBoost	✓	✓	✗	✗	✗	✓	✗	✗	✗
Groll (2020)	✓	✓	✗	✗	✗	✓	✗	✗	✗
gamboostLSS	✓	✗	✓	✗	✓	✓	✗	✗	✗
Hofner et al. (2023)	✗	✓	✗	✗	✗	✓	✗	✗	✗
BOOME (in Python)	✗	✓	✗	✗	✗	✓	✓	✓	✓
Chen (2023a)	✗	✓	✗	✗	✗	✓	✓	✓	✓

¹ “Class” includes binary or multiclass classification, and the construction of logistic regression models.

² “SL” contains several estimation methods based on machine learning approaches, such as tree and random forest. In addition, the corresponding packages may handle complex structures, including semi-parametric models, mixed-effects models, and generalized additive models.

While many packages have been available to handle either variable selection or measurement error correction, few packages deal with these two features simultaneously. To address those concerns, we develop the R package **SIMEXBoost** ([Qiu and Chen, 2023](#)) by extending the method in [Chen \(2023c\)](#) and [Chen and Qiu \(2023\)](#), which covers commonly used GLM and AFT models. Motivated by the idea of the boosting algorithm (see e.g., [Hastie et al., 2008](#), Section 16.2), **SIMEXBoost** aims to use estimating functions to iteratively retain informative covariates and exclude unimportant ones, yielding variable selection result. In addition, to deal with measurement error effects, the package **SIMEXBoost** primarily employs the SIMEX method to efficiently correct for measurement error effects for different types of regression models. There are several advantages of **SIMEXBoost** over the existing packages. Specifically, as summarized in Table 1, while **glmnet** and **SIS** are able to handle variable selection, they fail to deal with measurement error effects. In addition, the two packages **GLSME** and **mecor** focus on linear models and aim to adjust for measurement error effects in the response and/or covariates, but they cannot deal with variable selection. On the contrary, the contribution of the package **SIMEXBoost** is able handle measurement error in variables, and do variable selection and estimation simultaneously. Moreover, boosting iteration may reduce the possibility of falsely excluding important covariates and enhance the accuracy of the estimator. Most importantly, **SIMEXBoost** is

able to deal with collinearity of variables by using the L_2 -norm penalty function.

The remainder is organized as follows. In the second section, we introduce the data structure and the corresponding regression models. In addition, the boosting algorithm is outlined. In the third section, we introduce the measurement error model and extend the correction of measurement error effects to the boosting algorithm. In the fourth section, we introduce functions and their arguments in the R package **SIMEXBoost**. In the fifth section, we demonstrate the application of the R package **SIMEXBoost** and conduct simulation studies to assess the performance of the boosting estimators. Moreover, we also implement **SIMEXBoost** in a real dataset. A general discussion is presented in the last section. The supporting information, including a real dataset, programming code, and numerical results in csv files, are placed in the corresponding author's GitHub, whose link is given by <https://github.com/lchen723/SIMEXBoost.git>.

2 Notation, Models, and Boosting Procedure

2.1 Model

Let Y denote the response, and let \mathbf{X} be the p -dimensional vector of covariates. Suppose that we have a sample of n subjects and for $i = 1, \dots, n$, $\{Y_i, \mathbf{X}_i\}$ has the same distribution as $\{Y, \mathbf{X}\}$.

Let β be a p -dimensional vector of (unknown) parameters associated with the covariates \mathbf{X} , and write $\mathbf{X}^\top \beta$ as the linear predictor. In the framework of statistical learning, to characterize the relationship between Y and \mathbf{X} , a commonly used approach is to link Y and $\mathbf{X}^\top \beta$ through the convex loss function $L : S \times \mathbb{R} \rightarrow \mathbb{R}$, where S is the support of Y . Let the risk function be defined as the expectation of the loss function, i.e., $R(\beta) \triangleq E\{L(Y, \mathbf{X}^\top \beta)\}$. Under the finite sample size n , the empirical version of $R(\beta)$ is given by

$$\frac{1}{n} \sum_{i=1}^n L(Y_i, \mathbf{X}_i^\top \beta).$$

Our goal is to estimate β by minimizing the risk function, and the resulting estimator is given by

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \left\{ \frac{1}{n} \sum_{i=1}^n L(Y_i, \mathbf{X}_i^\top \beta) \right\}.$$

Equivalently, $\hat{\beta}$ satisfies the estimating equation $\mathbf{g}(\mathbf{X}, \beta) = 0$, where $\mathbf{g}(\mathbf{X}, \beta)$ is the estimating function of β , defined as the first order derivative of $\frac{1}{n} \sum_{i=1}^n L(Y_i, \mathbf{X}_i^\top \beta)$ with respect to β .

2.2 Boosting Procedure

High-dimensionality and sparsity of β are crucial concerns, which reflect the idea that some covariates are not informative with respect to Y . To address these issues and provide a reliable estimator of β , we employ the boosting procedure to perform variable selection and estimation (e.g., [Hastie et al., 2008](#)). This version of the boosting algorithm is motivated by [Wolfson \(2011\)](#) and [Brown et al. \(2017\)](#), and is applied to handle GLM ([Chen, 2023c](#)) as well as AFT models ([Chen and Qiu, 2023](#)). An overall procedure is presented in Algorithm 1 with three key steps. Specifically, Steps 1 and 2 in Algorithm 1 treat the estimating function evaluated at an iterated value as the signal, and use it to determine informative indexes of covariates and parameters. Noting that there is a parameter τ in Step 2 that is used to control the number of selected covariates in each iteration. In our numerical studies, $\tau = 0.9$ seems to be a suitable choice and has a satisfactory performance.

After that, Step 3 in Algorithm 1 updates β using the informative indexes determined in Step 2 by the sign of signals with increment κ . This approach follows the steepest descent method (see e.g., [Boyd and Vandenberghe, 2004](#), Section 9.4.2) and can be used to deal with the L_1 -norm for variable selection. In addition, as discussed in Section 16.2.1 of [Hastie et al. \(2008\)](#), the value κ has the opposite relationship with the number of iteration M that smaller κ requires larger M . In our consideration, we specify $\kappa = 0.05$. Finally, repeating the iteration M times gives the desired estimator. With M time iterations, we can obtain the final set \mathcal{J}_M containing informative covariates and the corresponding $\beta^{(M)} = (\beta_1^{(M)}, \dots, \beta_p^{(M)})^\top$, accomplishing variable selection.

Algorithm 1: Boost_VSE Algorithm

```

Let  $\beta^{(0)} = 0$  denote an initial value;
for iteration  $m$  with  $m = 0, 1, 2, \dots, M$  do
    Step 1: calculate  $\Delta^{(m-1)} = \mathbf{g}(\mathbf{X}, \beta)|_{\beta=\beta^{(m-1)}}$ ;
    Step 2: determine  $\mathcal{J}_m = \left\{ j : |\Delta_j^{(m-1)}| \geq \tau \max_{j'} |\Delta_{j'}^{(m-1)}| \right\}$ ;
    Step 3: update  $\beta_j^{(m)} \leftarrow \beta_j^{(m-1)} + \kappa \cdot \text{sign}(\Delta_j^{(m-1)})$  for all  $j \in \mathcal{J}_m$ , and define
     $\beta^{(m)} = (\beta_1^{(m)}, \dots, \beta_p^{(m)})^\top$ ;

```

Finally, for the application of Algorithm 1, we consider some specific models and the corresponding regression models listed below. With models specified, we can further determine the estimating function $\mathbf{g}(\mathbf{X}, \beta)$.

Linear regression models:

Given the dataset $\{\{Y_i, \mathbf{X}_i\} : i = 1, \dots, n\}$ with Y_i being a continuous and univariate random variable, linear models are characterized as

$$Y_i = \mathbf{X}_i^\top \beta + \epsilon_i \quad (1)$$

where ϵ_i is the noise term with $E(\epsilon_i) = 0$ and $\text{var}(\epsilon_i) = \sigma_\epsilon^2$. The estimating function is defined as the first order derivative of the least squares function:

$$\mathbf{g}(\mathbf{X}, \beta) = \sum_{i=1}^n -\mathbf{X}_i(Y_i - \mathbf{X}_i^\top \beta). \quad (2)$$

Logistic regression models:

If Y_i is a binary and univariate random variable, then Y_i and \mathbf{X}_i are usually characterized by a logistic regression model:

$$\pi_i = \frac{\exp(\mathbf{X}_i^\top \beta)}{1 + \exp(\mathbf{X}_i^\top \beta)}, \quad (3)$$

where $\pi_i \triangleq P(Y_i = 1 | \mathbf{X}_i)$. Following the idea in Agresti (2012), we can construct the likelihood function based on (3). Therefore, the resulting estimating function is given by the first order derivative of the likelihood function:

$$\mathbf{g}(\mathbf{X}, \beta) = - \sum_{i=1}^n \mathbf{X}_i \left\{ Y_i - \frac{\exp(\mathbf{X}_i^\top \beta)}{1 + \exp(\mathbf{X}_i^\top \beta)} \right\}. \quad (4)$$

Poisson regression models:

When Y_i is a count and univariate random variable, one can adopt the Poisson regression model to fit Y_i and \mathbf{X}_i :

$$\log \lambda_i = \mathbf{X}_i^\top \beta \quad (5)$$

where λ_i is the parameter of the Poisson distribution. Following the framework of generalized linear models, the likelihood function based on (5) can be determined. Therefore, the first order derivative of the likelihood function under (5) yields the corresponding estimating function, which is given by

$$\mathbf{g}(\mathbf{X}, \beta) = - \sum_{i=1}^n \mathbf{X}_i \left\{ Y_i - \exp(\mathbf{X}_i^\top \beta) \right\}. \quad (6)$$

Accelerated failure time models:

In survival analysis, the response is known as the failure time, denoted $\tilde{T}_i > 0$, and the accelerated failure time (AFT) model is a commonly used model for characterizing the relationship between the survival time and the covariates (e.g., Lawless, 2003). Specifically, the AFT model is formulated as

$$\log \tilde{T}_i = \mathbf{X}_i^\top \beta + \eta_i, \quad (7)$$

where η_i is the noise term of (7). In the framework of survival analysis, the main challenge is that \tilde{T}_i is usually incomplete due to how the observations are collected. In particular, in this study, the failure time may suffer from *length-biased* and *interval-censoring*, which cause the data

to be biased and incomplete.

Specifically, for the length-biased sampling, it is common to assume that the incidence rate of the initial event is constant over calendar time, and the truncation time, denoted \tilde{A}_i , is uniformly distributed in $[0, \xi]$, where ξ is the maximum support of \tilde{T}_i (e.g., [Chen and Qiu, 2023](#)). For the length-biased data, we can observe $(\tilde{A}_i, \tilde{T}_i)$ only if $\tilde{T}_i \geq \tilde{A}_i$, and thus, we denote $(T_i, A_i) \equiv (\tilde{T}_i, \tilde{A}_i) | \tilde{T}_i \geq \tilde{A}_i$ as the observed version of $(\tilde{T}_i, \tilde{A}_i)$.

On the other hand, for the observed T_i , we may encounter the interval-censoring. Suppose that T_i is not exactly observed but only determined at a sequence of examination times, denoted as $A_i = U_0 < U_1 < \dots < U_N \leq \xi$ for some constant $N > 0$. The failure time is then known to lie in the interval (L, R) , where $L_i = \max\{U_k : U_k < T_i, k = 0, \dots, N\}$ and $R_i = \min\{U_k : U_k \geq T_i, k = 1, \dots, N+1\}$ with $U_{N+1} \triangleq \infty$. Moreover, if T_i occurs before the first examination time, then $(L_i, R_i) \triangleq (A_i, U_1)$; if the failure has not occurred at the last examination time, then $(L_i, R_i) \triangleq (U_N, \infty)$. Finally, let Δ_i denote the indicator, where a value 1 indicates that T_i is observed and zero otherwise. As a consequence, for a sample with size n , the length-biased and interval-censored survival data is given by $\{\{A_i, \Delta_i, Y_i, X_i\} : i = 1, \dots, n\}$ with $Y_i \triangleq \{\Delta_i T_i, (1 - \Delta_i)L_i, (1 - \Delta_i)R_i\}$.

Based on the length-biased and interval-censored data, we can construct the estimating function (e.g., [Chen and Qiu, 2023](#))

$$\mathbf{g}(\mathbf{X}, \boldsymbol{\beta}) = \sum_{i=1}^n \mathbf{X}_i \left\{ \Delta_i \frac{Y_{\beta,i}}{\exp(Y_{\beta,i})} + (1 - \Delta_i) \frac{\int_{L_{i,0}}^{R_{i,0}} u^{-1} \log u dF_0(u)}{F_0(R_{i,0}) - F_0(L_{i,0})} \right\}, \quad (8)$$

where $Y_{\beta,i} = \log T_i - \mathbf{X}_i^\top \boldsymbol{\beta}$, $R_{i,0} = R_i \exp(-\mathbf{X}_i^\top \boldsymbol{\beta})$, $L_{i,0} = L_i \exp(-\mathbf{X}_i^\top \boldsymbol{\beta})$, and F_0 is the cumulative distribution function of η_i .

3 A Modified Boosting Method with the Presence of Covariate Measurement Error

3.1 Measurement Error Models

For $i = 1, \dots, n$, let \mathbf{X}_i^* denote the surrogate, or observed covariate, of \mathbf{X}_i . Let Σ_{X^*} and Σ_X be the $p \times p$ covariance matrices of \mathbf{X}_i^* and \mathbf{X}_i , respectively. In our development, we focus on the classical measurement error model (e.g., [Carroll et al., 2006](#), Chapter 1):

$$\mathbf{X}_i^* = \mathbf{X}_i + \mathbf{e}_i, \quad (9)$$

where \mathbf{e}_i is independent of $\{\mathbf{X}_i, Y_i\}$ and \mathbf{e}_i in (1), \mathbf{e}_i follows a normal distribution with mean zero and the covariance matrix Σ_e , say $N(0, \Sigma_e)$. Noting that the covariance matrix Σ_e is usually unknown. To determine it, we can either employ sensitivity analyses to reasonably specify values, or directly estimate it if additional information, such as repeated measurements or validation sample is available (see e.g., [Chen and Yi, 2021](#)). As a result, in the presence of measurement error, the *observed* dataset is now given by $\{\{Y_i, \mathbf{X}_i^*\} : i = 1, \dots, n\}$.

3.2 Boosting with Measurement Error Correction

In the presence of measurement error, it is known that directly using \mathbf{X}_i^* in the estimating procedure without the correction of measurement error effects may incur biased estimate and wrong conclusion (see e.g., [Carroll et al., 2006](#)). Therefore, even though Algorithm 1 is valid to estimate $\boldsymbol{\beta}$ for regression models in the ‘Boosting Procedure’ subsection, it is insufficient in the presence of measurement error effects.

To deal with measurement error in covariates, we extend Algorithm 1 by adopting the SIMEX method to eliminate the impact of measurement error (e.g., [Chen and Yi, 2021](#)). The modified algorithm, called *SIMEXBoost*, is summarized in Algorithm 2.

The idea of the SIMEX method is to first establish the trend of measurement error-induced biases as a function of the variance of measurement error by artificially creating a sequence of surrogate measurements, and then extrapolate this trend back to the case without measurement error. Specifically, in Step 1 of Algorithm 2, we artificially create a sequence of error-contaminated surrogate measurements by introducing different degrees of measurement error. After that, as shown in Step 2 of Algorithm 2, we apply those surrogate measurements to the boosting procedure, and use a new function $\mathbf{g}_{\text{SIM}}(\boldsymbol{\beta}; b, \zeta)$, which is defined as (2), (4), (6), or (8) with \mathbf{X}_i replaced by $\mathbf{W}_i(b, \zeta)$ defined in

(10), to obtain biased estimates by running an estimation method developed for error-free settings. Finally, Step 3 in Algorithm 2 traces the pattern of biased estimates against varying magnitudes of measurement error and then does extrapolation based on linear or quadratic regression models.

Noting that there are several parameters in Algorithm 2. The parameters M , κ , and τ in Step 2 are the same as those in Algorithm 1. On the other hand, Step 1 contains a value B and a sequence of \mathcal{Z} that are usually user-specified. Typically, \mathcal{Z} is usually defined as K equal-width cutpoints in an interval $[0, 1]$ for some positive constant K . A value B is used to perform the Monte Carlo computation in (11) and make the estimator more stable. A larger value of B may implicitly incur longer computational time. Our numerical experiments show that $B = 50$ gives satisfactory performance.

Algorithm 2: SIMEXBoost Algorithm

Step 1: Generate the working data $\mathbf{W}_i(b, \zeta)$ by

$$\mathbf{W}_i(b, \zeta) = \mathbf{X}_i^* + \sqrt{\zeta} \mathbf{e}_{i,b} \quad (10)$$

for $b = 1, \dots, B$ and $\zeta \in \mathcal{Z}$, where $\mathbf{e}_{i,b} \sim N(0, \Sigma_e)$ independently.

Step 2: Boosting estimation.

Perform the following boosting procedure with M iterations.

for $b = 1, \dots, B$ and $\zeta \in \mathcal{Z}$ **do**

Let $\beta^{(0)}(b, \zeta) = \mathbf{0}$ denote an initial value;

for iteration m with $m = 0, 1, 2, \dots, M$ **do**

Step 2.1: calculate $\Delta^{(m-1)}(b, \zeta) = g_{\text{SIM}}(\beta; b, \zeta)|_{\beta=\beta^{(m-1)}}$;

Step 2.2: determine $\mathcal{J}_m(b, \zeta) = \left\{ j : |\Delta_j^{(m-1)}(b, \zeta)| \geq \tau \max_j |\Delta_j^{(m-1)}(b, \zeta)| \right\}$;

Step 2.3: update $\beta_j^{(m)}(b, \zeta) \leftarrow \beta_j^{(m-1)}(b, \zeta) + \kappa \cdot \text{sign}(\Delta_j^{(m-1)}(b, \zeta))$ for all $j \in \mathcal{J}_m(b, \zeta)$;

Write $\beta^{(M)}(b, \zeta) = (\beta_1^{(M)}(b, \zeta), \dots, \beta_p^{(M)}(b, \zeta))^T$;

When $\beta^{(M)}(b, \zeta)$ is obtained for $b = 1, \dots, B$ and $\zeta \in \mathcal{Z}$, compute an average

$$\beta^{(M)}(\zeta) = \frac{1}{B} \sum_{b=1}^B \beta^{(M)}(b, \zeta) \text{ for } \zeta \in \mathcal{Z}. \quad (11)$$

Step 3: Extrapolation.

Fit a sequence $\{(\zeta, \beta^{(M)}(\zeta)) : \zeta \in \mathcal{Z}\}$ by a regression model, and the final value is given by the extrapolated value at $\zeta = -1$.

4 Description and Implementation of **SIMEXBoost**

We develop an R package, called **SIMEXBoost**, to implement the variable selection and estimation with measurement error correction described in the preceding section. This package depends on the **MASS** package only. The package **SIMEXBoost** contains three functions: **ME_Data**, **Boost_VSE**, and **SIMEXBoost**. The function **ME_Data** aims to generate artificial data under specific models listed in ‘Boosting Procedure’ subsection and error-prone covariates. The function **Boost_VSE** implements the boosting procedure in Algorithm 1, and the function **SIMEXBoost** implements the error-eliminated boosting procedure as displayed in Algorithm 2. We now describe the details of these three functions.

ME_Data

We use the following command to obtain the artificial data:

```
ME_Data(X, beta, type="normal", sigmae, pr0=0.5)
```

where the meaning of each argument is described as follows:

- **X:** An $n \times p$ matrix with components generated by random variables. It is provided by the user.
- **beta:** A p -dimensional vector of parameters, which is specified by the user.
- **type:** A regression model that is specified to generate the response. Some choices listed in ‘Boosting Procedure’ subsection are provided in this argument. **normal** means the linear

regression model (1) with the error term generated by the standard normal distribution; binary means the logistic regression model (3); poisson means the Poisson regression model (5). In addition, the accelerated failure time (AFT) model is considered to fit length-biased and interval-censored survival data. Specifically, AFT-normal generates the length-biased and interval-censored survival data under the AFT model (7) with the error term being normal distributions; AFT-loggamma generates the length-biased and interval-censored survival data under the AFT model with the error term being log-gamma distributions.

- **sigmae:** A $p \times p$ covariance matrix Σ_e in the measurement error model (9). Given Σ_e with non-zero entries, by (9), one can generate the error-prone covariates \mathbf{X}_i^* . Moreover, if Σ_e is given by the zero matrix, then \mathbf{e}_i is generated as zero values, yielding that \mathbf{X}_i^* is equal to \mathbf{X}_i , and thus, the resulting covariate is the original input given by users.
- **pr0:** A numerical value in an interval $(0, 1)$. It is used to determine the censoring rate for the length-biased and interval-censored data.

The function ME_Data returns a list of components:

- **response:** It gives the response generated by a specific regression model. type="normal" gives a n-dimensional continuous vector; type="binary" gives a n-dimensional vector with binary entries; type="poisson" gives a n-dimensional vector with entries being counting numbers. In addition, type="AFT-normal" and type="AFT-loggamma" generates a $n \times 2$ matrix of length-biased and interval-censored responses, where the first column is the lower bound of an interval-censored response and the second column is the upper bound of an interval-censored response.
- **ME_covariate:** This output gives a $n \times p$ matrix of "error-prone" or "precisely measured" covariates. Specifically, as discussed in an argument sigmae, if Σ_e is a non-zero covariance matrix, then the result of ME_covariate is given by \mathbf{X}_i^* ; if Σ_e is a zero matrix, then the result of ME_covariate is the original input, say \mathbf{X}_i .

Boost_VSE

We use the following function to perform Algorithm 1:

```
Boost_VSE(Y,Xstar,type="normal",Iter=200,Lambda=0)
```

where the meaning of each argument is described as follows:

- **Y:** The response variable. If type is specified as normal, binary, or poisson, then Y should be a n-dimensional vector; if type is given by AFT-normal or AFT-loggamma, then Y should be a $n \times 2$ matrix of interval-censored responses, where the first column is the lower bound of an interval-censored response and the second column is the upper bound of an interval-censored response.
- **Xstar:** This argument needs a $n \times p$ matrix of covariates. It can be error-prone or precisely measured.
- **type:** This argument specifies the regression models as previously described in the function ME_Data as well as their corresponding estimating functions given by (2), (4), (6), and (8).
- **Iter:** The number of iterations M for the boosting procedure in Algorithm 1.
- **Lambda:** A tuning parameter that aims to deal with the collinearity of covariates. Lambda=0 means that no L_2 -norm is involved, and it is the default value.

The function Boost_VSE returns a list of components:

- **BetaHat:** The vector of estimated coefficient obtained by Algorithm 1. In particular, if the covariate Xstar is generated by ME_Data with the argument sigmae being the zero matrix, then the resulting vector is the "ordinary" estimator based on the boosting procedure; if the covariate Xstar is generated by ME_Data with the argument sigmae being a non-zero covariance matrix, then we call the resulting vector as the *naive* estimator due to involvement of measurement error effects without correction.

SIMEXBoost

Basically, some arguments in this function are the same as Boost_VSE, except for some slightly different requirements and additional arguments that are related to the SIMEX method.

We use the following function to perform Algorithm 2:

```
SIMEXBoost(Y,Xstar,zeta=c(0,0.25,0.5,0.75,1),B=500,type="normal",sigmae,
Iter=100, Lambda=0, Extrapolation="linear")
```

where the meanings of arguments Y , $Xstar$, $type$, $Iter$, and $Lambda$ are the same as those in the function `Boost_VSE`; additional arguments $zeta$, B , $sigmae$, and $Extrapolation$, which are used to implement the SIMEX method to correct for measurement error effects, are described as follows:

- $zeta$: The user-specific sequence of values described as \mathcal{Z} in Step 1 of Algorithm 2.
- B : The user-specific positive number of repetition described as B in Step 1 of Algorithm 2.
- $sigmae$: An $p \times p$ covariance matrix Σ_e in the measurement error model (9). In practical applications, if auxiliary information is unavailable, sensitivity analyses can be adopted to reasonably specify values of Σ_e . If additional information, such as repeated measurements or validation samples, is available, one can directly estimate Σ_e .
- $Extrapolation$: A extrapolation function for the SIMEX method implemented to Step 3 of Algorithm 2. In the framework of the SIMEX method, quadratic and linear functions are common. Therefore, in this argument, we provide two choices of the extrapolation functions, linear and quadratic.

The function `SIMEXBoost` returns a list of components:

- `BetaHatCorrect`: The resulting vector of corrected estimates obtained by Algorithm 2.

5 Numerical Studies and Demonstration of Programming Code

In this section, we demonstrate the usage of the functions in the package `SIMEXBoost`. There are two parts in this section: we first illustrate simulation studies for linear regression, Poisson regression, and AFT models. After that, we apply the package to analyze a real-world dataset with binary responses based on a logistic regression model.

5.1 Simulation Studies

In this section, we use simulation studies to demonstrate applications of functions in the package `SIMEXBoost` and assess the performance of the estimators derived by two functions `Boost_VSE` and `SIMEXBoost`.

We consider the dimension of covariates $p = 200$ or 500 , and let the sample size $n = 400$. Let the true value $\beta_0 = (1, 1, 1, \mathbf{0}_{p-3}^\top)^\top$, where $\mathbf{0}_q$ is a q -dimensional zero vector. The unobserved covariate \mathbf{X}_i is generated by the standard normal distribution, and it can be used to generate the response Y_i based on (1), (5), and (7). For the error-prone covariate \mathbf{X}_i^* , it can be generated by (9), where Σ_e is a diagonal matrix with common entries σ_e being 0.1 , 0.3 , and 0.5 .

Based on the artificial data $\{(Y_i, \mathbf{X}_i^*) : i = 1, \dots, n\}$, we first use the function `Boost_VSE` to obtain the estimator *without* measurement error correction, which is called the *naive* estimator. Next, we apply the function `SIMEXBoost` to derive the *corrected* estimator. To assess the performance of variable selection, we examine the specificity (Spe) and the sensitivity (Sen), where the specificity is defined as the proportion of zero coefficients that are correctly estimated to be zero, and the sensitivity is defined as the proportion of non-zero coefficients that are correctly estimated to be non-zero. In addition, to evaluate the performance of estimation, we use the L_1 and L_2 -norms to measure bias, which are respectively defined as

$$\|\hat{\beta} - \beta_0\|_1 = \sum_{j=1}^p |\hat{\beta}_j - \beta_{0,j}| \text{ and } \|\hat{\beta} - \beta_0\|_2 = \left\{ \sum_{j=1}^p (\hat{\beta}_j - \beta_{0,j})^2 \right\}^{1/2}, \quad (12)$$

where $\hat{\beta}$ is the estimator, $\hat{\beta}_j$ and $\beta_{0,j}$ are the j th component in $\hat{\beta}$ and β_0 , respectively.

We use two ‘for’ loops cover all combinations of p and σ_e ($sigmae$). For each p and $sigmae$, we first adapt the function `ME_Data` to generate the simulated data, where Y and $Xstar$ represent the response and error-prone covariates, respectively. After that, to examine the naive estimator derived by Algorithm 1, we use the function `Boost_VSE` to derive the naive estimator and denote it by `naive`. To implement Algorithm 2, we adopt the function `SIMEXBoost`, where two components \mathcal{Z} and B in Step 1 of Algorithm 2 are specified as $zeta=c(0,0.25,0.5,0.75,1)$ and $B=50$, respectively. For the `SIMEXBoost` method, we also examine linear and quadratic extrapolation functions with the argument `Extrapolation="linear"` or `Extrapolation="quadratic"`, and denote two resulting estimators as

`correctL` and `correctQ`, respectively. For the two functions `Boost_VSE` and `SIMEXBoost`, we set the number of iterations `Iter=50`. To save the space in the limited text, we simply illustrate the model (1) with the argument `type="normal"`; numerical results under other models can be reproducied by the following code with `type="normal"` replaced by `type="poisson"` or `type="AFT-normal"`.

Next, we assess the performance of naive, `correctL`, and `correctQ`. Given a true vector of parameter `beta0`, we compute L_1 and L_2 -norms in (12) to examine the bias, and compute Spe and Sen to examine variable selection. Under a given `p` and `sigma`, biases and variable selection results are recorded by `EST1`, `EST2`, and `EST3` for the estimators `naive`, `correctL`, and `correctQ`, respectively. Finally, numerical results of three estimators `naive`, `correctL`, and `correctQ` under all settings are summarized by `NAIVE`, `SIMEXL`, and `SIMEXQ`, respectively.

```
library(SIMEXBoost)
library(MASS)

NAIVE = NULL # naive method
SIMEXL = NULL # simex method with linear extrapolation function
SIMEXQ = NULL # simex method with quadratic extrapolation function

for (p in c(200, 500)) {
  for (sigma in c(0.1, 0.3, 0.5)) {
    set.seed(202270)
    beta0 = c(1, 1, 1, rep(0, p - 3))

    X = matrix(rnorm((p) * 400),
               nrow = 400,
               ncol = p,
               byrow = TRUE)

    Sig = diag(sigma ^ 3, dim(X)[2])

    data = ME_Data(
      X = X,
      beta = beta0,
      type = "normal",
      sigmae = Sig
    )
    Y = data$response
    Xstar = data$ME_covariate

    naive = Boost_VSE(Y, Xstar, type = "normal", Iter = 50)$BetaHat

    correctL = SIMEXBoost(
      Y,
      Xstar,
      zeta = c(0, 0.25, 0.5, 0.75, 1),
      B = 50,
      type = "normal",
      sigmae = Sig,
      Iter = 50,
      Lambda = 0,
      Extrapolation = "linear"
    )$BetaHatCorrect
    correctL[which(abs(correctL) < 0.5)] = 0
    correctQ = SIMEXBoost(
      Y,
      Xstar,
      zeta = c(0, 0.25, 0.5, 0.75, 1),
      B = 50,
      type = "normal",
      sigmae = Sig,
      Iter = 50,
      Lambda = 0,
      Extrapolation = "quadratic"
    )$BetaHatCorrect
  }
}
```

```

correctQ[which(abs(correctQ) < 0.5)] = 0

#####
Sen = which(beta0 != 0)
Spe0 = which(beta0 == 0)

## results for the naive estimator
naive = as.numeric(naive)
L1_norm = sum(abs(naive - beta0))
L2_norm = sqrt(sum((naive - beta0) ^ 2))
Spe = length(which(naive[Spe0] == 0)) / length(Spe0)
Sen = length(which(naive[Sen0] != 0)) / length(Sen0)

## results for the error-corrected estimator based on Extrapolation="linear"
L1_norm_l = sum(abs(correctL - beta0))
L2_norm_l = sqrt(sum((correctL - beta0) ^ 2))
Spe_l = length(which(correctL[Spe0] == 0)) / length(Spe0)
Sen_l = length(which(correctL[Sen0] != 0)) / length(Sen0)

## results for the error-corrected estimator based on Extrapolation="quadratic"
L1_norm_q = sum(abs(correctQ - beta0))
L2_norm_q = sqrt(sum((correctQ - beta0) ^ 2))
Spe_q = length(which(correctQ[Spe0] == 0)) / length(Spe0)
Sen_q = length(which(correctQ[Sen0] != 0)) / length(Sen0)

#####

NAIVE = rbind(NAIVE, c(L1_norm, L2_norm, Spe, Sen))
SIMEXL = rbind(SIMEXL, c(L1_norm_l, L2_norm_l, Spe_l, Sen_l))
SIMEXQ = rbind(SIMEXQ, c(L1_norm_q, L2_norm_q, Spe_q, Sen_q))
}
}

```

Numerical results under (1), (5), and (7) are placed in Tables 2–4, respectively. We observe that the naive and corrected estimates are affected by the magnitudes of measurement error effects and the dimension p . When values of p and σ become large, the biases given by L_1 and L_2 -norms are increasing. For the comparison between the naive and corrected estimators, we can see that biases produced by the naive estimator are significantly larger than those obtained by the corrected estimator. In addition, for the variable selection result, the corrected estimator is able to correctly retain the informative covariates and exclude unimportant ones, except for some cases that one or two covariates may be falsely included. On the contrary, we can observe from the naive method that values of Spe_{naive} are always small while values of Sen_{naive} are equal to one. It indicates that the naive estimator retains the truly important covariates, and meanwhile, includes a lot of unimportant ones, which shows an evidence that the naive method fails to do variable selection. Finally, for the comparison between two extrapolation functions, $Extrapolation="linear"$ and $Extrapolation="quadratic"$, we observe that the specification of a linear extrapolation function has slightly better performance than the case under a quadratic extrapolation function, especially when σ is large.

While the proposed SIMEXBoost method can handle measurement error well, the main concern is the computational time. According to the record of the system CPU time (in seconds) from the R function `proc.time()` under the device ASUS DESKTOP-HJSD47S with the processor Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz, we find that, for each setting with fixed p and σ , the proposed SIMEXBoost method requires 14.09 and 15.04 seconds under $Extrapolation="linear"$ and $Extrapolation="quadratic"$ to derive the estimator for $p = 200$, respectively. Unsurprisingly, when the dimension increases to $p = 500$, the computational times under $Extrapolation="linear"$ and $Extrapolation="quadratic"$ are increasing to 17.70 and 19.68 seconds, respectively. On the other hand, without measurement error correction under each setting, the naive method needs 12.98 and 15.71 seconds to derive the estimator for $p = 200$ and 500, yielding the slightly faster computational time than the SIMEXBoost method. It might be due to the measurement error correction with the involvement of Z and the number of repetitions B in Step 2 of Algorithm 2. As a result, we comment that the SIMEXBoost method is able to address measurement error correction, but a slightly longer computational time is the price that the users should pay for.

Table 2: Simulation results for the model (1) with type="normal". "Naive" is the naive method obtained by the function Boost_VSE. "SIMEXBoost-Linear" and "SIMEXBoost-Quadratic" refer to the proposed method obtained by the function SIMEXBoost with the argument Extrapolation = "linear" and Extrapolation = "quadratic", respectively. L_1 -norm and L_2 -norm are given by (12). Spe and Sen are specificity and sensitivity, respectively.

p	σ_e	Methods	L_1 -norm	L_2 -norm	Spe	Sen
200	0.1	Naive	6.900	0.738	0.487	1.000
		SIMEXBoost-Linear	0.116	0.099	1.000	1.000
		SIMEXBoost-Quadratic	0.109	0.090	1.000	1.000
	0.3	Naive	6.350	0.684	0.503	1.000
		SIMEXBoost-Linear	0.101	0.072	1.000	1.000
	0.5	SIMEXBoost-Quadratic	0.106	0.075	1.000	1.000
		Naive	7.600	0.758	0.426	1.000
		SIMEXBoost-Linear	0.257	0.182	1.000	1.000
500	0.1	SIMEXBoost-Quadratic	0.281	0.188	1.000	1.000
		Naive	8.050	0.733	0.732	1.000
		SIMEXBoost-Linear	0.054	0.051	1.000	1.000
	0.3	SIMEXBoost-Quadratic	0.069	0.057	1.000	1.000
		Naive	8.900	0.778	0.710	1.000
		SIMEXBoost-Linear	0.300	0.187	1.000	1.000
	0.5	SIMEXBoost-Quadratic	0.300	0.187	1.000	1.000
		Naive	10.000	0.889	0.692	1.000
		SIMEXBoost-Linear	0.600	0.354	1.000	1.000
	0.5	SIMEXBoost-Quadratic	0.600	0.354	1.000	1.000

Table 3: Simulation results for the model (5) with type="poisson". "Naive" is the naive method obtained by the function Boost_VSE. "SIMEXBoost-Linear" and "SIMEXBoost-Quadratic" refer to the proposed method obtained by the function SIMEXBoost with the argument Extrapolation = "linear" and Extrapolation = "quadratic", respectively. L_1 -norm and L_2 -norm are given by (12). Spe and Sen are specificity and sensitivity, respectively.

p	σ_e	Methods	L_1 -norm	L_2 -norm	Spe	Sen
200	0.1	Naive	1.600	0.283	0.848	1.000
		SIMEXBoost-Linear	0.208	0.126	1.000	1.000
		SIMEXBoost-Quadratic	0.538	0.407	1.000	1.000
	0.3	Naive	1.750	0.304	0.838	1.000
		SIMEXBoost-Linear	0.178	0.129	1.000	1.000
	0.5	SIMEXBoost-Quadratic	0.520	0.301	1.000	1.000
		Naive	2.500	0.387	0.782	1.000
		SIMEXBoost-Linear	0.366	0.230	1.000	1.000
500	0.1	SIMEXBoost-Quadratic	1.371	0.877	0.995	1.000
		Naive	1.400	0.265	0.944	1.000
		SIMEXBoost-Linear	0.096	0.087	1.000	1.000
	0.3	SIMEXBoost-Quadratic	0.428	0.385	1.000	1.000
		Naive	1.850	0.304	0.930	1.000
		SIMEXBoost-Linear	0.354	0.236	1.000	1.000
	0.5	SIMEXBoost-Quadratic	0.279	0.193	1.000	1.000
		Naive	3.000	0.458	0.903	1.000
		SIMEXBoost-Linear	0.554	0.393	1.000	1.000
	0.5	SIMEXBoost-Quadratic	1.238	0.771	1.000	1.000

5.2 Real Data Example

In this section, we apply the package **SIMEXBoost** to analyze the Company Bankruptcy Prediction data, which was from the Taiwan Economic Journal during a period 1999–2009, and it is now available on the Kaggle website (<https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction>). In this dataset, there are 6819 observations and 95 continuous covariates related to customers' banking records, such as assets, liability, income, and so on. In addition, the response is a binary random variable, where the value 1 reflects that the company is bankrupt, and 0 otherwise. All variables' names and descriptions can be found on the Kaggle website. The main interest in this study is to identify covariates that are informative to the bankruptcy status, and our goal is to apply (3) to characterize the bankruptcy status and covariates.

In addition to detecting important covariates from the multivariate variables, as mentioned in Chen (2023d), measurement error is ubiquitous in variables related to customers' banking records. Hence, one should take measurement error effects into account when doing variable selection. Following the example of the simulation studies, we primarily consider two scenarios: first, without consideration of measurement error, one can directly apply the function Boost_VSE to do variable selection. Second, if we wish to implement the function SIMEXBoost and take measurement error correction into account, the covariance matrix Σ_{ϵ} is needed. Since the dataset has no additional information to estimate Σ_{ϵ} , we may conduct *sensitivity analyses*, which enable us to characterize various degrees of measurement

Table 4: Simulation results for the model (7) with type="AFT-normal". "Naive" is the naive method obtained by the function Boost_VSE. "SIMEXBoost-Linear" and "SIMEXBoost-Quadratic" refer to the proposed method obtained by the function SIMEXBoost with the argument Extrapolation = "linear" and Extrapolation = "quadratic", respectively. L_1 -norm and L_2 -norm are given by (12). Spe and Sen are specificity and sensitivity, respectively.

p	σ_e	Methods	L_1 -norm	L_2 -norm	Spe	Sen
200	0.1	Naive	0.875	0.603	1.000	1.000
		SIMEXBoost-Linear	0.378	0.232	1.000	1.000
		SIMEXBoost-Quadratic	0.411	0.258	1.000	1.000
	0.3	Naive	1.450	1.078	1.000	0.667
		SIMEXBoost-Linear	0.400	0.247	1.000	1.000
	0.5	SIMEXBoost-Quadratic	0.877	0.605	1.000	1.000
500	0.1	Naive	4.700	2.801	1.000	0.667
		SIMEXBoost-Linear	1.614	1.179	0.995	1.000
		SIMEXBoost-Quadratic	2.958	2.204	0.995	1.000
	0.3	Naive	2.750	1.521	0.998	0.333
		SIMEXBoost-Linear	0.959	0.560	1.000	1.000
	0.5	SIMEXBoost-Quadratic	2.710	2.288	0.998	1.000
0.3	Naive	8.425	5.398	1.000	0.667	
	SIMEXBoost-Linear	0.949	0.554	1.000	1.000	
	SIMEXBoost-Quadratic	0.820	0.518	1.000	1.000	
	0.5	Naive	4.350	2.743	1.000	0.333
0.5	SIMEXBoost-Linear	2.541	1.364	0.998	0.667	
	SIMEXBoost-Quadratic	4.003	2.604	0.998	1.000	

error and examine the different magnitudes of measurement error effects (e.g., Chen and Yi, 2021; Chen, 2023d). Specifically, we specify Σ_e as a diagonal matrix with common entries being $R = 0.1, 0.3$ and 0.5 , and the extrapolation function is taken as linear or quadratic functions in Algorithm 2.

To show the implementation of data analysis, we demonstrate the programming code below. For the convenience of data analysis, we export the full dataset as a csv file, which is available in <https://github.com/lchen723/SIMEXBoost.git>. One can download the dataset 'bankruptcy_data.csv'. Based on the dataset, we denote Y and X_{star} as the response and the observed covariates, respectively. For the naive method without measurement error correction, we adopt the function Boost_VSE with 50 iterations. For the implementation of sensitivity analyses, we use a 'for' loop for different values of R . For each R , we run SIMEXBoost with type="binary" and two extrapolation functions Extrapolation = "linear" and Extrapolation = "quadratic".

```

library(MASS)
library(SIMEXBoost)
R = c(0.1, 0.3, 0.5)
data = read.table("bankruptcy_data.csv", sep = ",", head = T)
data = data[, -94]
Y = data[, 95]
Y = as.numeric(Y)
Xstar = t(as.matrix(data[, -95]))
Xstar = scale(Xstar)

p = dim(Xstar)[1]
n = dim(Xstar)[2]

set.seed(202270)
##naive
EST = NULL
naive = Boost_VSE(Y,
                    t(Xstar),
                    type = "binary",
                    Iter = 50,
                    Lambda = 0)$BetaHat
EST = cbind(EST, naive)
##linear
for (i in R) {
  correctL = SIMEXBoost(
    Y,
    t(Xstar),
    zeta = c(0, 0.25, 0.5, 0.75, 1),
    B = 50,
    type = "binary",

```

```

    sigmae = diag(i, p),
    Iter = 50,
    Lambda = 0,
    Extrapolation = "linear"
  )$BetaHatCorrect

  EST = rbind(EST, t(correctL))
}
##Quadratic

for (i in R) {
  correctQ = SIMEXBoost(
    Y,
    t(Xstar),
    zeta = c(0, 0.25, 0.5, 0.75, 1),
    B = 50,
    type = "binary",
    sigmae = diag(i, p),
    Iter = 50,
    Lambda = 0,
    Extrapolation = "quadratic"
  )$BetaHatCorrect

  EST = rbind(EST, t(correctQ))
}
round(EST, 3)

```

Numerical results are summarized in Table 5, where the column “ID” is the indexes of selected covariates, and the heading “EST” is the estimates of the coefficients. According to our analysis results, we find that the covariate “Net Income to Stockholder’s Equity” (ID #90) is only one that is commonly selected from Boost_VSE and SIMEXBoost under different values of R , which indicates that the covariate #90 is an informative variable regardless of measurement error effects have been corrected or not. For the corrected estimator with different R and extrapolation functions, we observe that variables “Operating Profit Rate: Operating Income/Net Sales” (ID #6), “Pre-tax net Interest Rate: Pre-Tax Income/Net Sales” (ID #7), “Continuous interest rate (after tax): Net Income-Exclude Disposal Gain or Loss/Net Sales” (ID #10), and “Liability-Assets Flag” (ID #85) are selected, except for the scenarios “Correct-L-0.3” and “Correct-L-0.5”. Moreover, different variables are selected under different values of R . It might be due to the impact of different magnitudes of measurement error. On the other hand, without measurement error correction, we observe from the naive estimator that most selected variables are different from the proposed estimator, such as “Research and development expense rate: (Research and Development Expenses)/Net Sales” (ID #12), “Tax rate (A): Effective Tax Rate” (ID #15), “Per Share Net profit before tax (Yuan ¥): Pretax Income Per Share” (ID #23), “Total Asset Growth Rate: Total Asset Growth” (ID #29), and “Cash Reinvestment %: Cash Reinvestment Ratio” (ID #32), and those estimates are close to zero. It implies that noninformative variables are possibly selected by the naive method if measurement error is not taken into account in analysis, and it shows an impact of measurement error in data analysis.

Finally, we use the function `proc.time()` to record the system CPU time, and we find that the function `SIMEXBoost` requires 3.84 and 3.50 seconds to run, under `Extrapolation = "linear"` and `Extrapolation = "quadratic"` with a fixed R , respectively, while the function `Boost_VSE` needs 0.45 seconds to derive the estimates. Consistent with the finding in simulation studies, the `SIMEXBoost` method requires slightly longer computational time than the naive method, which is caused by the repetition of boosting procedure and SIMEX correction.

6 Discussion

The package **SIMEXBoost** provides a novel method for handling high-dimensional data subject to measurement error in covariates. It covers widely used GLM and AFT models in survival analysis, and provides a strategy to deal with variable selection and measurement error correction simultaneously. Moreover, our package is able to handle the collinearity in the covariates. As evidence by longer computational times in numerical studies, the function `SIMEXBoost` seems to be more computationally demanding compared to the naive implementation, which is basically caused by settings of B and Z for correction of measurement error. Typically, following the similar idea of the Monte Carlo simulation,

Table 5: Variable selection and estimation for the Company Bankruptcy Prediction data based on the model (3) and type="binary". “Naive” refers to the naive method based on Boost_VSE. “Correct-L-0.1”, “Correct-L-0.3”, and “Correct-L-0.5” refer to the function SIMEXBoost with Extrapolation="linear" and $R = 0.1, 0.3$ and 0.5 , respectively. “Correct-Q-0.1”, “Correct-Q-0.3”, and “Correct-Q-0.5” refer to the function SIMEXBoost with Extrapolation="quadratic" and $R = 0.1, 0.3$ and 0.5 , respectively. The label “–” indicates that the variables are not selected.

ID	EST						
	Naive	Correct-L-0.1	Correct-L-0.3	Correct-L-0.5	Correct-Q-0.1	Correct-Q-0.3	Correct-Q-0.5
6	–	0.400	–	-0.383	3.025	1.781	2.296
7	–	0.474	–	–	2.895	2.126	2.943
8	–	0.322	–	-0.267	3.143	2.628	2.122
10	–	0.396	0.216	–	1.861	1.760	1.749
12	-0.050	–	–	–	–	–	–
15	-0.050	–	–	–	–	–	–
23	0.050	–	–	–	–	–	–
25	–	–	–	–	-0.257	–	–
29	-0.100	–	–	–	–	–	–
32	-0.050	–	–	–	–	–	–
37	–	–	-0.235	–	0.766	1.452	0.365
38	-0.150	–	-0.307	–	1.604	1.679	0.404
43	0.050	–	–	–	–	–	–
46	–	-0.300	–	–	0.257	–	–
48	-0.050	–	–	–	–	–	–
55	-0.050	–	–	–	–	–	–
59	0.050	-0.300	–	–	-1.282	-1.285	–
64	-0.050	–	–	–	–	–	–
65	–	0.306	0.300	–	0.916	1.261	–
68	-0.150	–	–	–	–	–	–
74	-0.050	–	–	–	–	–	–
77	-0.050	–	–	–	–	–	–
84	0.050	–	–	–	–	–	–
85	–	-0.472	–	0.237	-6.365	-2.636	-2.956
86	-0.150	–	–	–	–	–	–
87	–	-0.300	–	–	-0.912	–	–
90	-1.350	-1.518	-2.397	-2.433	1.613	-1.602	-3.668
94	0.200	0.240	–	–	-1.553	-1.532	-0.362

larger values of B and \mathcal{Z} usually give the stable estimator but also incur longer computational time. This is a common phenomenon in measurement error analysis and the SIMEX method (e.g., Yi, 2017). In summary, users need to consider whether to take measurement error effects into account based on their data and set arguments, such as B and \mathcal{Z} , for the implementation based on their computational resources.

The current state of the package allows for measurement error in continuous covariates and parametric models for exponential family distributed responses or time-to-event outcomes. There are still many possible extensions to the methods, such as consideration of measurement error in binary covariates or measurement error in the response, variable selection for semi-parametric regression models, including the Cox model in survival analysis or the partially linear single index model.

R Software

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

Acknowledgments

The authors would like to thank the editorial team for useful comments to improve the initial manuscript. Chen's research was supported by National Science and Technology Council with grant ID 110-2118-M-004-006-MY2.

References

- A. Agresti. *Categorical Data Analysis*. Wiley, New York, 2012. [p8]
- E. Alfaro, M. Gamez, L. Garcia, N. Guo, A. Albano, M. Sciandra, and A. Plaia. *adabag: Applies Multiclass AdaBoost.M1, SAMME and Bagging*, 2023. URL <https://cran.r-project.org/package=adabag>. R package version 5.0. [p6]
- K. Bartoszek. *GLSME: Generalized Least Squares with Measurement Error*, 2019. URL <https://cran.r-project.org/package=GLSME>. R package version 1.0.5. [p6]
- S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, 2004. [p7]
- B. Brown, C. J. Miller, and J. Wolfson. Threeboost: Thresholded boosting for variable selection and prediction via estimating equations. *Journal of Computational and Graphical Statistics*, 26:579–588, 2017. [p5, 7]
- R. J. Carroll, D. Ruppert, L. A. Stefanski, and C. M. Crainiceanu. *Measurement Error in Nonlinear Model*. CRC Press, New York, 2006. [p5, 9]
- L.-P. Chen. Feature screening based on distance correlation for ultrahigh-dimensional censored data with covariate measurement error. *Computational Statistics*, 36:857–884, 2021. [p5]
- L.-P. Chen. BOOME: A python package for handling misclassified disease and ultrahigh-dimensional error-prone gene expression data. *PLOS ONE*, 17:e0276664, 2023a. [p6]
- L.-P. Chen. A note of feature screening via rank-based coefficient of correlation. *Biometrical Journal*, 65: 2100373, 2023b. [p5]
- L.-P. Chen. De-noising boosting methods for variable selection and estimation subject to error-prone variables. *Statistics and Computing*, 33:38:1–13, 2023c. [p5, 6, 7]
- L.-P. Chen. Variable selection and estimation for misclassified binary responses and multivariate error-prone predictors. *Journal of Computational and Graphical Statistics*, 2023d. URL <https://doi.org/10.1080/10618600.2023.2218428>. [p5, 15, 16]
- L.-P. Chen and B. Qiu. Analysis of length-biased and partly interval-censored survival data with mismeasured covariates. *Biometrics*, 79:3929–3940, 2023. [p5, 6, 7, 9]
- L.-P. Chen and G. Y. Yi. Analysis of noisy survival data with graphical proportional hazards measurement error models. *Biometrics*, 77:956–969, 2021. [p5, 9, 16]
- T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou, M. Li, J. Xie, M. Lin, Y. Geng, Y. Li, J. Yuan, and X. contributors. *xgboost: Extreme Gradient Boosting*, 2023. URL <https://cran.r-project.org/package=xgboost>. R package version 1.7.5.1. [p6]
- Y. Feng, J. Fan, D. F. Saldana, Y. Wu, and R. Samworth. *SIS: Sure Independence Screening*, 2020. URL <https://cran.r-project.org/package=SIS>. R package version 0.8-8. [p6]
- J. Friedman, T. Hastie, R. Tibshirani, B. Narasimhan, K. Tay, N. Simon, J. Qian, and J. Yang. *glmnet: Lasso and Elastic-Net Regularized Generalized Linear Models*, 2023. URL <https://cran.r-project.org/package=glmnet>. R package version 4.1-7. [p6]
- B. Greenwell, B. Boehmke, J. Cunningham, and G. Developers. *gbm: Generalized Boosted Regression Models*, 2022. URL <https://cran.r-project.org/package=gbm>. R package version 2.1.8.1. [p6]
- A. Groll. *GMMBoost: Likelihood-Based Boosting for Generalized Mixed Models*, 2020. URL <https://cran.r-project.org/package=GMMBoost>. R package version 1.1.3. [p6]
- T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, 2008. [p5, 6, 7]
- B. Hofner, A. Mayr, N. Fenske, J. Thomas, and M. Schmid. *gamboostLSS: Boosting Methods for 'GAMLSS'*, 2023. URL <https://cran.r-project.org/package=glmnet>. R package version 4.1-7. [p6]
- J. F. Lawless. *Statistical Models and Methods for Lifetime Data*. Wiley, New York, 2003. [p5, 8]
- W. Lederer, H. Seibold, H. Küchenhoff, C. Lawrence, and R. F. Brøndum. *simex: SIMEX- And MCSIMEX-Algorithm for Measurement Error Models*, 2019. URL <https://cran.r-project.org/package=simex>. R package version 1.8. [p6]

- L. Nab. *mecor: Measurement Error Correction in Linear Models with a Continuous Outcome*, 2021. URL <https://cran.r-project.org/package=mecor>. R package version 1.0.0. [p6]
- B. Qiu and L.-P. Chen. *SIMEXBoost: Boosting Method for High-Dimensional Error-Prone Data*, 2023. URL <https://cran.r-project.org/package=SIMEXBoost>. R package version 0.2.0. [p6]
- Y. Shi, G. Ke, D. Soukhavong, J. Lamb, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, N. Titov, Y. Yan, M. Corporation, I. Dropbox, J. Loden, D. Daeschler, G. Rodola, A. Ferreira, D. Lemire, V. Zverovich, I. Corporation, and D. Cortes. *lightgbm: Light Gradient Boosting Machine*, 2023. URL <https://cran.r-project.org/package=lightgbm>. R package version 3.3.5. [p6]
- R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of Royal Statistical Society, Series B*, 58:267–288, 1996. [p5]
- Z. Wang and T. Hothorn. *bst: Gradient Boosting*, 2023. URL <https://cran.r-project.org/package=bst>. R package version 0.3-24. [p6]
- J. Wolfson. Eeboost: a general method for prediction and variable selection based on estimating equation. *Journal of the American Statistical Association*, 106:296–305, 2011. [p5, 7]
- J. Xiong, W. He, and G. Y. Yi. *simexaft: simexaft*, 2019. URL <https://cran.r-project.org/package=simexaft>. R package version 1.0.7.1. [p6]
- G. Y. Yi. *Statistical Analysis with Measurement Error and Misclassification: Strategy, Method and Application*. Springer, New York, 2017. [p18]
- Q. Zhang and G. Y. Yi. *augSIMEX: Analysis of Data with Mixed Measurement Error and Misclassification in Covariates*, 2020. URL <https://cran.r-project.org/package=augSIMEX>. R package version 3.7.4. [p6]
- H. Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101: 1418–1429, 2006. [p5]
- H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B*, 67:301–320, 2005. [p5]

Li-Pang Chen
Department of Statistics, National Chengchi University
No. 64, Section 2, Zhinan Rd, Wenshan District, Taipei City, 116
Taiwan (R.O.C.)
ORCID: 0000-0001-5440-5036
lchen723@nccu.edu.tw

Bangxu Qiu
Department of Statistics, National Chengchi University
No. 64, Section 2, Zhinan Rd, Wenshan District, Taipei City, 116
Taiwan (R.O.C.)
1135427976@qq.com

binGroup2: Statistical Tools for Infection Identification via Group Testing

by Christopher R. Bilder, Brianna D. Hitt, Brad J. Biggerstaff, Joshua M. Tebbs, and Christopher S. McMahan

Abstract Group testing is the process of testing items as an amalgamation, rather than separately, to determine the binary status for each item. Its use was especially important during the COVID-19 pandemic through testing specimens for SARS-CoV-2. The adoption of group testing for this and many other applications is because members of a negative testing group can be declared negative with potentially only one test. This subsequently leads to significant increases in laboratory testing capacity. Whenever a group testing algorithm is put into practice, it is critical for laboratories to understand the algorithm's operating characteristics, such as the expected number of tests. Our paper presents the `binGroup2` package that provides the statistical tools for this purpose. This R package is the first to address the identification aspect of group testing for a wide variety of algorithms. We illustrate its use through COVID-19 and chlamydia/gonorrhea applications of group testing.

1 Introduction

The COVID-19 pandemic showed the need for accessible, fast, and reliable testing for severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) and for infections in general. To meet this need, many laboratories turned to the use of group testing. Group testing—also known as pooled testing, specimen pooling, batch testing, and bulk testing—involves combining portions of multiple specimens from different individuals into a “group” and testing this group as if it were a single specimen. If the group tests negative, all individuals represented within it generally can be declared negative. Thus, for a group of size 10, it takes only one test to determine whether all 10 individuals are infection free, whereas it would take 10 tests if each specimen was tested separately. Alternatively, if a group tests positive, retesting in some form is needed to determine who is positive or negative within the group. Dorfman (1943) proposed the original retesting method that involved simply retesting each group member separately using the remaining portions of their specimens. Since this seminal paper, many other group testing algorithms have been proposed. These algorithms are categorized into hierarchical and non-hierarchical approaches corresponding to whether specimens are tested in non-overlapping or overlapping groups, respectively, at each stage of the algorithm (Hitt et al. 2019).

A large number of research papers (see e.g. Abdalhamid et al. 2020; Hogan et al. 2020; Barathidasan et al. 2022), articles in the news media (see e.g. Abdelmalek 2020; Mandavilli 2020; Anthes 2022), and Emergency Use Authorizations given by the US Food and Drug Administration (LabCorp 2022; Verily Life Sciences 2022; Yale University 2022) showed that group testing was one of the important tools available to mitigate the crisis caused by COVID-19. Our new `binGroup2` package for the R software environment provides the statistical tools that researchers and laboratories need for group testing. This package is available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/package=binGroup2>. By its name, one can surmise that it is the second large-scale implementation of the package. The `binGroup` package (Bilder et al. 2010b) focused on estimation using the unique type of data that arises through group testing. Its applications ranged from estimating an overall infection prevalence to estimating a regression model for individual-specific infection probabilities as a function of risk factors. Bilder et al. (2010b) encouraged researchers to provide new functions so that there would be one main package for group testing in R. This resulted in new functions for estimation and a few functions focused on the identification aspect of group testing (i.e., determine a positive/negative outcome for each individual). Over time, these additions resulted in many different syntax styles among functions, making use of the package inconsistent and more difficult. The new `binGroup2` package was created to unify function syntax and, most importantly, to incorporate a new suite of functions for identification. These new functions provide the information that laboratories need to increase their testing capacity to its fullest potential.

Our paper focuses on these identification functions of `binGroup2` because they represent the main new contribution of the package. There are two other R packages on CRAN that also examine the identification aspect of group testing but for different settings. First, the `gtcorr` package (Lendle 2011; archived in 2022) examines hierarchical and non-hierarchical algorithms for a homogeneous population when there is a constant correlation among individuals within a group. This package implements the work of Lendle et al. (2012) that uses single-infection assays only. Second, the `mMPA` package (Liu and Xu 2018) examines three hierarchical algorithms, where two are for homogeneous populations and one is for heterogeneous populations. This package implements the work of Liu

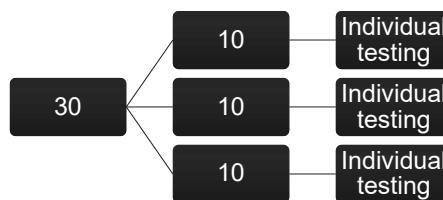


Figure 1: The hierarchical testing algorithm of Lohse et al. (2020).

et al. (2017) for single-infection assays that return an additive count outcome (e.g., viral load) when applied to a group. Both `gcorr` and `mMPA` are useful for their specialized situations. Our package encompasses a total of 27 different algorithms that include homogeneous/heterogeneous populations and hierarchical/non-hierarchical algorithms. These algorithms are constructed for the most common setting of a binary test response (positive/negative) from independent individuals within a group. We allow for the possibility of testing error and single or multiple-infection assays. Our package implements the work of more than ten papers on the identification aspect of group testing (e.g., Kim et al. 2007; McMahan et al. 2012b; Hou et al. 2020).

An outline of our paper is as follows. The next section provides a review of group testing algorithms that are commonly used by laboratories. The third section discusses how to calculate important operating characteristics, such as the expected number of tests, for a group testing algorithm by using `binGroup2`. This section also demonstrates how one can choose the most efficient implementation of an algorithm. Finally, we conclude with an overview of changes to estimation functions and future extensions of the package. Throughout our text, we phrase our discussion in the context of testing humans for infectious diseases. Applications of group testing for this purpose outside of COVID-19 include testing for influenza (Van et al. 2012), gonorrhea (Ando et al. 2021), HIV (Kim et al. 2014), and West Nile virus (American Red Cross 2023). The package can also be used in a wide variety of other areas where group testing is implemented, such as infectious disease testing for farm animals (Nebraska Veterinary Diagnostic Center 2023), detection of human exposure to pollutants (Thai et al. 2020), determination of virus presence in insect carriers (Zhao and Rosa 2020), development of new pharmaceuticals (Salzer et al. 2016), and security of computer networks (Thai 2011).

2 Algorithms

Laboratories select a group testing algorithm by examining 1) the expected number of tests needed to make a positive/negative determination for every individual, 2) the expected accuracy of the positive/negative determinations, 3) the information available on individuals tested, and 4) the ease of implementation. The `binGroup2` package provides computations needed to address 1) through 3), while laboratories need to address 4) relative to their work environment. Overall, there is not one group testing algorithm that is best for all situations, which has resulted in many different hierarchical and non-hierarchical algorithms being used in practice.

2.1 Hierarchical algorithms

Hierarchical group testing algorithms involve testing a group and splitting its members into smaller, non-overlapping sub-groups for retesting if the original group tests positive. These sub-groups are split further whenever they test positive. If a group/sub-group tests negative, its members are declared negative, so that no further testing is performed upon them. Overall, testing can continue until each group member is tested separately.

A recent example of a hierarchical algorithm comes from Lohse et al. (2020) to test for SARS-CoV-2 (see Figure 1). Individuals were put into non-overlapping groups of size 30. If a group tested positive, three non-overlapping groups of size 10 were formed from its members. If one of these sub-groups tested positive, each of its members were retested separately (i.e., individual testing). This hierarchical algorithm is referred to as a three-stage algorithm because three distinct stages are defined for it.

Hierarchical algorithms can have more or fewer stages than used by Lohse et al. (2020). For example, Abdalhamid et al. (2020) used two stages for SARS-CoV-2 detection by testing in groups of size 5 and subsequently retesting each member of a positive group separately. More than three stages are possible as well, but applications are much rarer because of logistics, larger chance for error, and delay in obtaining the positive/negative individual outcomes. With respect to the last reason, this delay can be significant for nucleic acid amplification tests which can take hours to complete.

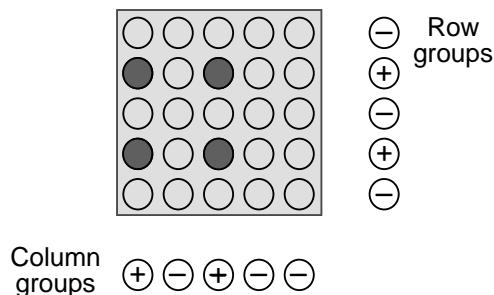


Figure 2: The array testing algorithm of [LabCorp \(2022\)](#). Circles within the square represent specimen locations. Row and column groups are formed from the specimens with two each testing positive. The intersections of these positive rows and columns indicate which specimens need to be retested (filled-in circles).

For example, the Centers for Disease Control and Prevention’s assay to detect SARS-CoV-2 takes approximately four hours to complete.

2.2 Non-hierarchical algorithms

Non-hierarchical algorithms were developed to minimize the number of retests needed after an initial stage of testing. The most prominent of these algorithms is referred to as array testing (also known as matrix pooling). For this algorithm, specimens are arranged in a grid (see Figure 2). Groups are formed by row and by column, and each of these groups are subsequently tested. Specimens that lie at the intersections of positive testing rows and positive testing columns are retested separately in a second stage to determine a positive/negative outcome for each of them. All other specimens are declared negative. Because the accuracy of infectious disease assays is not perfect, ambiguities may occur, resulting in positive rows (columns) without any positive columns (rows). In those situations, members of positive rows (columns) should be retested separately ([Kim et al. 2007](#)). A recent example of an array testing algorithm is the use of a 5×5 array by LabCorp for SARS-CoV-2 detection ([LabCorp 2022](#)).

Array testing has a number of variants in application. For example, a master group containing portions of all specimens within an array can be tested first. If this master group is negative, all individuals are quickly declared negative. If the master group is positive, row and column groups are tested as usual.

2.3 Additional considerations

The probability an individual has an infection plays a very important role in determining the number of tests needed by any group testing algorithm. For a homogeneous population of individuals tested, we can define p as this probability. Equivalently, this p is the overall infection prevalence. In general, the lower (higher) the p , the lower (higher) the expected number of tests needed for group testing (see e.g. Figure 3 of [Kim et al. \(2007\)](#) and Table 1 of [Bilder et al. \(2021a\)](#)).

In many situations, additional information is available on each individual to be tested. This information can be used to determine an individual-specific probability of an infection, say p_i , which can be incorporated into the group testing algorithm to reduce the number of tests ([Bilder et al. 2010b](#); [Lewis et al. 2012](#)). For example, testing is performed by public health laboratories across the United States for *Chlamydia trachomatis* (CT) and *Neisseria gonorrhoeae* (NG), the bacteria that lead to chlamydia and gonorrhea, respectively. Clinics collecting specimens also obtain important information on their patients, including recent sexual history and whether a patient has symptoms, and this information can be incorporated into a statistical model to estimate individual-specific probabilities of infection. In general, group testing algorithms that take advantage of this type of additional information are referred to as being “informative” hierarchical/non-hierarchical algorithms ([McMahan et al. 2012a](#); [McMahan et al. 2012b](#)).

Group testing algorithms can also be used with multiplex assays. Thus, rather than testing for only one infection, multiple infections can be detected simultaneously. For example, Roche was the first to receive an Emergency Use Authorization for their SARS-CoV-2 and influenza multiplex assay ([Roche 2020](#)). Also, the widely adopted Aptima Combo 2 Assay ([Hologic 2023](#)) is used for CT and NG detection via group testing. Algorithms with multiplex assays are implemented in the same way as for

single-infection assays, but now a group that tests positive for *at least one* infection leads to retesting for *all* infections in the next stage. For example, if a group tests positive for CT and negative for NG in a two-stage hierarchical algorithm, each group member is retested for both bacteria in the second stage. Testing is performed in this manner, rather than retesting for CT only, because of the dedicated testing platforms used.

3 Identification

Identifying infected individuals is most often the primary goal of infectious disease testing. Prior to implementing a particular group testing algorithm for this purpose, laboratories need to know the expected number of tests. This information is used for planning purposes to make sure enough resources and staff are available to implement testing. Because cost is often directly proportional to the number of tests, the expected number of tests can be used for budgeting purposes as well.

Define T as the number of tests required to determine the positive/negative outcomes for I individuals. These I individuals may be those represented within one initial group for hierarchical testing or in one array for array testing. For the simplest algorithm, two-stage hierarchical testing, suppose a single-infection assay is applied to a homogeneous population. The expected number of tests is

$$E(T) = 1 + I \left[S_e^{(1)} + (1 - S_p^{(1)} - S_e^{(1)}) (1 - p)^I \right],$$

where $S_e^{(1)}$ and $S_p^{(1)}$ are the first-stage sensitivity and specificity, respectively, of the assay. This expression includes a leading 1 because a single group test starts the testing process. The I outside of the square brackets is for the separate tests performed on each individual when the group tests positive (which has the probability given within the square brackets; see [Kim et al. \(2007\)](#) and [Bilder \(2019\)](#)). Expressions for $E(T)$ become much more complicated with other group testing algorithms. This is especially the case for informative algorithms and when multiplex assays are used. We refer interested readers to the works of [Kim et al. \(2007\)](#), [McMahan et al. \(2012a\)](#), [McMahan et al. \(2012a\)](#), [Black et al. \(2015\)](#), [Bilder et al. \(2019\)](#), and [Hou et al. \(2020\)](#) for specific expressions and calculation details.

The efficiency of a group testing algorithm is expressed typically as the expected number of tests per individual $E(T)/I$, rather than $E(T)$ alone, so that comparisons can be made for different I . The optimal testing configuration (OTC) is the group size or set of group sizes needed by a single algorithm to minimize $E(T)/I$. This testing configuration represents the most efficient process to implement group testing. In other words, it is the group size(s) that increases a laboratory's testing capacity to its fullest potential, because resources saved from its application can be used to test more specimens by the same algorithm. Unfortunately, closed form expressions for the OTC do not exist, but grid searches are sufficient to find it. Note that group testing is more efficient than initially testing each individual separately when $E(T)/I < 1$. This is simply because testing each individual separately results in 1 test per individual.

Accuracy is also important to examine for group testing algorithms. This includes the pooling sensitivity, the probability a truly positive individual is found to be positive from the group testing algorithm, and the pooling specificity, the probability a truly negative individual is found to be negative from the group testing algorithm. These probabilities are not necessarily equal to an assay's stated sensitivity and specificity because individuals are tested in one or more groups. For example, the pooling sensitivity for a two-stage hierarchical algorithm can be shown to be $S_e^{(1)}S_e^{(2)}$, where $S_e^{(2)}$ is the sensitivity of the assay at the second stage ([Johnson et al. 1991](#), [Kim et al. 2007](#), and [Hitt 2020](#)). Expressions for these accuracy measures corresponding to other algorithms are available in the previously mentioned references for $E(T)$.

One issue related to accuracy is what is known as the *dilution effect* that sometimes can reduce the sensitivity of a group test. If the same sample volume is used for a group and individual test, the dilution effect can occur because each individual is represented by a smaller portion in the group than if each individual was tested separately. For example, [Abdalhamid et al. \(2020\)](#) used a sample volume of $250\mu\text{L}$ for a group of size 5 with each individual contributing $50\mu\text{L}$. In their validation study, specimens were tested one at a time using a sample volume of $250\mu\text{L}$. Thus, there can be less pathogen present for the test performed upon the group than for a test performed upon a single specimen. [Bilder et al. \(2021b\)](#) discussed the potential dilution effect and solutions for it. In summary, the resulting outcome from the dilution effect is often known for assays ([Tan et al. 2020](#)), like for those using real-time reverse transcription polymerase chain reaction (RT-qPCR) which was widely used for SARS-CoV-2 testing. This effect can be accounted for by specifying a smaller value for a group test sensitivity when computing $E(T)$. Alternatively, the process for applying the assay can be changed so

Table 1: Values for algorithm in opChar1() and opChar2(). A five-stage informative hierarchical algorithm (ID5) is also available for two infections. Informative array testing is not included for opChar2() because closed-form expressions of operating characteristics have not been proposed in the literature.

algorithm	Description
A2	Array testing
A2M	Array testing with master group
D2	Two-stage hierarchical
D3	Three-stage hierarchical
D4	Four-stage hierarchical
IA2	Informative array testing
ID2	Two-stage, informative hierarchical
ID3	Three-stage, informative hierarchical
ID4	Four-stage, informative hierarchical

that there is not a reduction in the sensitivity (Abdalhamid et al. 2020, Sanghani et al. 2021).

3.1 Main functions

There are two main sets of functions in **binGroup2** used for identification. First, the opChar1() and opChar2() functions calculate operating characteristics for a group testing algorithm. These functions are for single-infection and two-infection assays, where the number in the name designates the number of infections. Calculations for three or more infection assays are discussed in the concluding section of the paper. The syntax for opChar1() is

```
opChar1(algorithm, p = NULL, probabilities = NULL, Se = 0.99,
       Sp = 0.99, hier.config = NULL, rowcol.sz = NULL, alpha = 2,
       a = NULL, print.time = TRUE, ...)
```

The required algorithm argument specifies the chosen group testing algorithm. Possible values for this argument are listed in Table 1. For example, a value of "D2" indicates two-stage hierarchical testing in a homogeneous population (the "D" is for its originator, Robert Dorfman). The remaining arguments are dependent on the algorithm chosen or are optional. To indicate the probability of an infection, a scalar can be given for p or a vector of potentially different probabilities for each individual can be given for probabilities. Rather than providing a specific vector of probabilities, the p and alpha arguments can be used together for informative group testing algorithms to specify a beta distribution from which the expected values of order statistics are found. For this case, p represents the expected value of a beta random variable and alpha represents a shape parameter. This type of specification is helpful when probabilities of infection can be characterized well by a beta distribution.

For hierarchical algorithms, a group membership matrix (Bilder et al. 2019) must be provided for hier.config to detail the testing of each individual. In this matrix, the rows correspond to the stages of testing, the columns correspond to each individual to be tested, and the cell values specify the group number of each individual at each stage. We provide an example of its use shortly. For non-hierarchical algorithms, the rowcol.sz argument must be provided for the row and column size of a square array.

Additional arguments include Se and Sp for the sensitivity and specificity of the assay, respectively. If a single value is given for one of these arguments, this value is used for each stage. Otherwise, a vector of values can be given in order of stage. The a argument specifies individuals for which to compute accuracy measures. By default, accuracy measures are computed for all individuals. Finally, print.time allows users to turn off information regarding the duration of calculations.

The opChar2() function follows a similar syntax so we do not provide it here. The main difference involves p becoming a vector of joint probabilities of infection. For example, p = (0.90, 0.03, 0.02, 0.05) represents $(p_{--}, p_{+-}, p_{-+}, p_{++})$, where p_{ab} is the probability of being positive/negative (+/-) for infections a and b. Also, the probabilities argument value becomes a $4 \times I$ matrix of these probabilities. Similarly, the alpha argument becomes the parameter vector for a Dirichlet distribution.

The second set of functions are OTC1() and OTC2() that find the OTC corresponding to single-infection and two-infection assays, respectively. The syntax for OTC1() is

```
OTC1(algorithm, p = NULL, probabilities = NULL, Se = 0.99, Sp = 0.99,
      group.sz, obj.fn = "ET", weights = NULL, alpha = 2, trace = TRUE,
```

```
print.time = TRUE, ...)
```

The syntax is similar to `opChar1()` as well, so we highlight the main differences only. There is no group membership matrix or row/column size specified because `OTC1()` searches for the OTC. Instead, the `group.sz` argument specifies the initial group sizes to search over. For example, a value of `3:10` searches over initial group sizes of 3 to 10 for hierarchical testing or array testing (row/column size).

The `obj.fn` argument of `OTC1()` indicates which objective function to minimize when searching for the OTC. Earlier in this section, we focused on using the expected number of tests per individual as this objective function (`obj.fn = "ET"`). While this is used most often in practice, other objective functions are possible (Hitt et al. 2019). For example, Graff and Roeloffs (1972) proposed to minimize a linear combination of the expected number of tests and the number of misclassified individuals (false positives, false negatives). This objective function is specified using "GR", and the coefficients in this linear combination are given in the `weights` argument.

The `OTC2()` function follows a similar syntax as `OTC1()` with changes like those for `opChar2()` compared to `opChar1()`. We next provide examples using these functions for group testing applications.

3.2 Operating characteristics

We illustrate the use of `opChar1()` for the three-stage hierarchical testing algorithm of Lohse et al. (2020). The group membership matrix for this application is the 3×30 matrix below.

```
> group.member <- matrix(data = c(rep(1, times = 30), rep(1:3, each = 10),
  1:30), nrow = 3, ncol = 30, byrow = TRUE)
> group.member[, 11]
[1] 1 2 11
```

For example, the 11th individual is tested initially in a group of size 30 that includes every specimen (one group overall). In the second stage, this individual is tested in the second sub-group of 10 individuals if its first-stage group is positive. In the third stage, this individual is tested separately if its second-stage group is positive.

We need to specify p or p_i for each individual. In actual practice, these values will be unknown. However, very good point estimates are usually available from past testing results in high volume clinical specimen settings where group testing is used. For the Lohse et al. (2020) application, the observed prevalence of SARS-CoV-2 was 0.0193, so we use this value here for p . By specifying "D3" for `algorithm` to represent three-stage hierarchical testing, we invoke `opChar1()` as follows.

```
> library(package = "binGroup2")
> save.Lohse <- opChar1(algorithm = "D3", p = 0.0193, Se = 1, Sp = 1,
  hier.config = group.member, print.time = FALSE)
> summary(save.Lohse)
```

Algorithm: Non-informative three-stage hierarchical testing

Testing configuration:

Stage 1: 30

Stage 2: 10,10,10

Expected number of tests: 7.64

Expected number of tests per individual: 0.2547

Accuracy for individuals:

PSe	PSp	PPPV	PNPV	Individuals
1	1.0000	1.0000	1.0000	All

Overall accuracy of the algorithm:

PSe	PSp	PPPV	PNPV
1	1.0000	1.0000	1.0000

PSe denotes the pooling sensitivity.

PSp denotes the pooling specificity.

PPPV denotes the pooling positive predictive value.

PNPV denotes the pooling negative predictive value.

The sensitivity (Se) and the specificity (Sp) are set to 1 for each stage because the assay accuracy is not stated in Lohse et al. (2020) or in the product insert of the assay (Altona Diagnostics 2023). While the algorithm's accuracy values produced will not be useful, this approach is often followed in practice to focus on the expected number of tests regardless of false positives/negatives.

The generic summary() function uses the corresponding method function for the class to summarize the operating characteristics. For example, the expected number of tests per individual is $E(T)/I = 0.2547$. A more compact summary based on the expected number of tests is available with ExpTests().

```
> ExpTests(save.Lohse)
```

	ExpTests	ExpTestsPerIndividual	PercentReductionTests	PercentIncreaseTestCap
1	7.6403	0.2547	74.53	292.66

Group testing requires $1 - E(T)/I = 75\%$ fewer tests on average than testing each specimen separately. In turn, this leads to a $100\{1/[E(T)/I] - 1\}\% = 293\%$ increase in testing capacity on average when applying the algorithm to a continuous stream of specimens. This type of large increase in testing capacity is why laboratories implemented group testing during the COVID-19 pandemic.

Our second example focuses on the Aptima Combo 2 Assay and its use by the State Hygienic Laboratory (SHL) at the University of Iowa. The SHL tests thousands of female swab specimens each year in groups of size four using a two-stage hierarchical algorithm. We focus here instead on their male urine specimens because group testing is not currently implemented. The reason is due to concern that their infection prevalence may be too large for group testing to be beneficial. Bilder et al. (2019) provided a Dirichlet distribution with parameter vector $\alpha = (10.99, 0.18, 2.04, 0.31)$ to describe a vector of the joint probabilities of infection for CT and NG that take into account risk factors, such as symptoms and exposure to an infected individual. Using a first-stage group size of 5, we want to calculate how well three-stage hierarchical testing would perform when allowing for differences among probabilities of infection for individuals. To begin, we simulate what one potential set of probabilities of infection could be using the Dirichlet distribution and the rdirichlet() function of the rBeta2009 package (Cheng et al. 2012). These probabilities are ordered by the probability of having at least one infection.

```
> library(package = "rBeta2009")
> set.seed(3789)
> p.unordered <- t(rdirichlet(n = 5, shape = c(10.99, 0.18, 2.04, 0.31)))
> p.ordered <- p.unordered[, order(1 - p.unordered[1, ])]
> round(p.ordered, 4)

 [,1]  [,2]  [,3]  [,4]  [,5]
[1,] 0.9714 0.8841 0.8356 0.8291 0.7009
[2,] 0.0001 0.0006 0.0000 0.0238 0.0000
[3,] 0.0274 0.0879 0.1643 0.1292 0.2991
[4,] 0.0011 0.0274 0.0001 0.0178 0.0000
```

Next, we create the group membership matrix for three-stage informative hierarchical testing using the OTC found in Bilder et al. (2019). Because individuals 4 and 5 are tested separately in the second stage, a NA is used for them in the third stage of the group membership matrix.

```
> group.member <- matrix(data = c(rep(1, times = 5), 1, 1, 1, 2, 3, 1, 2,
      3, NA, NA), nrow = 3, ncol = 5, byrow = TRUE)
> group.member

 [,1]  [,2]  [,3]  [,4]  [,5]
[1,]    1    1    1    1    1
[2,]    1    1    1    2    3
[3,]    1    2    3   NA   NA
```

Lastly, the opChar2() function calculates the operating characteristics for the algorithm. The Se and Sp matrices provide the sensitivity and specificity values, respectively, for each infection at each stage (ICBSS 2014). Because these accuracies are equal across the stages, the sensitivity (specificity) for CT and NG could be instead included as a single vector of length 2 for the Se (Sp) argument of opChar2(). Alternatively, if accuracies were different across stages, the full matrices provide a general way to include this information.

```
> Se <- matrix(data = rep(c(0.979, 0.985), times = 3), nrow = 2, ncol = 3,
      dimnames = list(Infestation = 1:2, Stage = 1:3))
> Sp <- matrix(data = rep(c(0.985, 0.996), times = 3), nrow = 2, ncol = 3,
```

```

dimnames = list(Inflection = 1:2, Stage = 1:3)
> save.SHL <- opChar2(algorithm = "ID3", probabilities = p.ordered, Se = Se,
  Sp = Sp, hier.config = group.member, print.time = FALSE)
> summary(save.SHL)

```

Algorithm: Informative three-stage hierarchical testing

Testing configuration:

Stage 1: 5

Stage 2: 3,1,1

Expected number of tests: 3.59

Expected number of tests per individual: 0.7182

Disease 1 accuracy for individuals:

	PSe	PSp	PPPV	PNPV	Individuals
1	0.9770	0.9958	0.2111	1.0000	1
2	0.9778	0.9961	0.8784	0.9994	2
3	0.9784	0.9958	0.0124	1.0000	3
4	0.9729	0.9915	0.8330	0.9988	4
5	0.9787	0.9913	0.0012	1.0000	5

Disease 2 accuracy for individuals:

	PSe	PSp	PPPV	PNPV	Individuals
1	0.9585	0.9990	0.9643	0.9988	1
2	0.9633	0.9992	0.9940	0.9952	2
3	0.9575	0.9994	0.9969	0.9917	3
4	0.9725	0.9979	0.9879	0.9953	4
5	0.9714	0.9984	0.9961	0.9879	5

Overall accuracy of the algorithm:

	PSe	PSp	PPPV	PNPV
1	0.9749	0.9941	0.7039	0.9996
2	0.9669	0.9988	0.9931	0.9941

PSe denotes the pooling sensitivity.

PSp denotes the pooling specificity.

PPPV denotes the pooling positive predictive value.

PNPV denotes the pooling negative predictive value.

The expected number of tests per individual is 0.7182. Because this value is less than 1, group testing would be more efficient on average than testing each individual separately. The PSe and PSp values in the output are the pooling sensitivity and specificity, respectively. Additionally, values are given for the pooling positive predictive value (PPPV) and the pooling negative predictive value (PNPV; see [Altman and Bland \(1994\)](#) and [Hitt et al. \(2019\)](#)).

Of course, not every set of 5 individuals has these 5 joint probabilities of infection. When this group testing algorithm was applied to the data from which the Dirichlet distribution was estimated, [Bilder et al. \(2019\)](#) showed that the number of tests decreased by approximately 26% on average.

3.3 Optimal testing configuration

Returning to the three-stage hierarchical algorithm of [Lohse et al. \(2020\)](#), suppose again that $p = 0.0193$. The OTC is found using the following code.

```

> OTC.Lohse <- OTC1(algorithm = "D3", p = 0.0193, Se = 1, Sp = 1,
  group.sz = 3:20, obj.fn = "ET", print.time = FALSE)

Initial Group Size = 3
Initial Group Size = 4
Initial Group Size = 5
Initial Group Size = 6
Initial Group Size = 7
Initial Group Size = 8

```

```

Initial Group Size = 9
Initial Group Size = 10
Initial Group Size = 11
Initial Group Size = 12
Initial Group Size = 13
Initial Group Size = 14
Initial Group Size = 15
Initial Group Size = 16
Initial Group Size = 17
Initial Group Size = 18
Initial Group Size = 19
Initial Group Size = 20

> summary(OTC.Lohse)

```

Algorithm: Non-informative three-stage hierarchical testing

Optimal testing configuration:

Stage 1	Stage 2
ET	16 4,4,4,4

Expected number of tests:

E(T)	Value
ET	3.27 0.2045

E(T) denotes the expected number of tests.

Value denotes the objective function value per individual.

Overall accuracy of the algorithm:

PSe	PSp	PPPV	PNPV
ET	1.0000	1.0000	1.0000

PSe denotes the pooling sensitivity.

PSp denotes the pooling specificity.

PPPV denotes the pooling positive predictive value.

PNPV denotes the pooling negative predictive value.

The summary() function summarizes the OTC. The first-stage group size is 16, the second stage has 4 groups of size 4, and the third stage uses individual testing. The expected number of tests per individual is 0.2045. Therefore, the OTC decreases the expected number of tests per individual further by almost 20% ($= (0.2547 - 0.2045) / 0.2547$) over the testing configuration used by Lohse et al. (2020). In terms of expected test capacity, one can show with ExpTests(OTC.Lohse) that the OTC leads to a 389% increase in testing capacity when compared to testing each specimen separately. Again, this is significantly better than the testing configuration chosen by Lohse et al. (2020) and helps to show the importance of choosing a testing configuration.

The OTC1() function searches for the optimal set of group sizes at *each* first-stage group size specified in the group.sz argument. By default, the function's progress is printed during its running (trace = TRUE). We take this approach to finding the OTC, rather than optimizing over all group sizes initially, because a laboratory may prefer a sub-optimal testing configuration due to their work environment. The Config() function provides information about these sub-optimal testing configurations. This function extracts each of the best testing configurations by initial group size and returns them as a data frame sorted by the value of the objective function ($E(T)/I$ because the default was used in OTC1()).

```
> Config(OTC.Lohse)
```

I	config	ET	value	PSe	PSp	PPPV	PNPV
1	16 4,4,4,4	3.2714	0.2045	1	1	1	1
2	17 5,4,4,4	3.4922	0.2054	1	1	1	1
3	15 4,4,4,3	3.0842	0.2056	1	1	1	1
4	20 4,4,4,4,4	4.1138	0.2057	1	1	1	1
5	19 4,4,4,4,3	3.9176	0.2062	1	1	1	1

Adding top.overall = TRUE to Config() provides the same information but with the top testing configurations overall rather than for each initial group size.

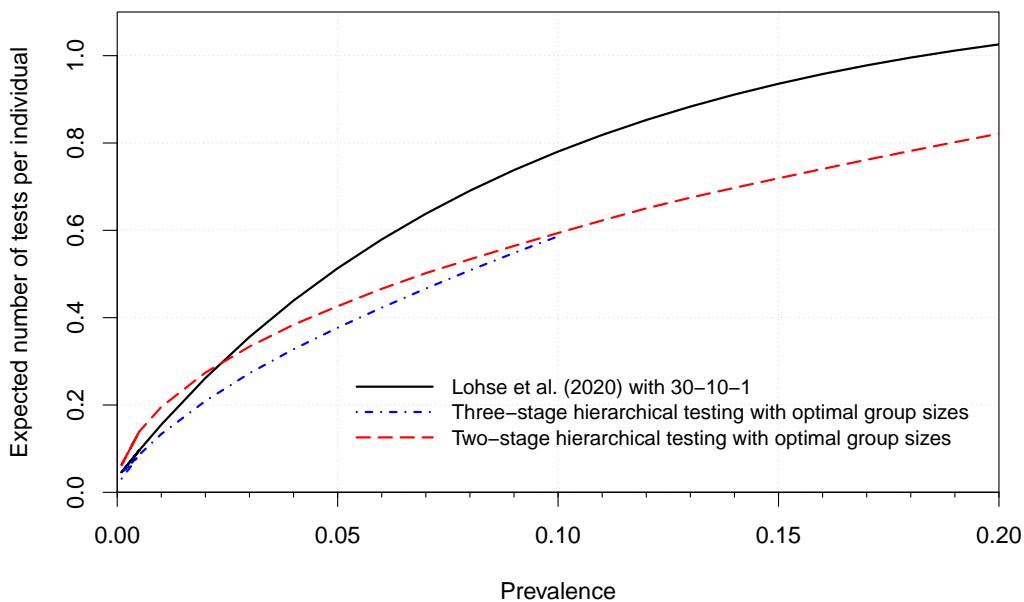


Figure 3: Expected number of tests per individual as a function of infection prevalence. Three-stage hierarchical testing using the OTC is not plotted for prevalences greater than 0.10 because two stages results in a lower expected number of tests.

The most efficient outcome from the OTC depends on the chosen value of p . As mentioned earlier in this section, very good point estimates for p are often available. Still, it would be of interest to determine what would occur for other choices of p . The `binGroup2` package provides the tools needed by experienced R users for this more in depth examination. Our Appendix A discusses additional code using `OTC1()` to produce Figure 3. The testing configuration chosen by Lohse et al. (2020) is never optimal. In fact, two-stage hierarchical group testing is more efficient than Lohse et al. (2020) for $p > 0.03$.

Our second example focuses on SARS-CoV-2 detection again, but now in the context of LabCorp's array testing. LabCorp (2022) examined the potential performance of their approach using possible prevalences between 0.001 and 0.15, so we begin with using a prevalence of 0.05. Also, LabCorp (2022) estimated that approximately 0.023 of all positives would be missed using array testing rather than individual testing. This is not quite the sensitivity of the assay, but we will use 0.977 and 1 as the sensitivity for the first and second stages, respectively, for illustration purposes. We will also use 1 as the specificity for illustration purposes. The expected number of tests per individual for the algorithm is found first using `opChar1()` with `algorithm = "A2"`.

```
> save.LabCorp <- opChar1(algorithm = "A2", p = 0.05, rowcol.sz = 5,
  Se = c(1 - 0.023, 1), Sp = 1, print.time = FALSE)
> ExpTests(save.LabCorp)

  ExpTests ExpTestsPerIndividual PercentReductionTests PercentIncreaseTestCap
1 12.0763           0.4831                  51.69                107.02
```

The expected number of tests per individual is 0.48, resulting in an expected 107% increase in testing capacity.

Could LabCorp do better? Below is the code used to find the OTC for a range of group sizes from 3 to 10 with array testing.

```
> OTC.LabCorp.Array <- OTC1(algorithm = "A2", p = 0.05, Se = c(0.977,
  1), Sp = 1, group.sz = 3:10, obj.fn = "ET", trace = FALSE,
  print.time = FALSE)
> summary(OTC.LabCorp.Array)
```

Algorithm: Non-informative array testing without master pooling

Optimal testing configuration:

	Row/column size	Array size
ET	10	100

Expected number of tests:

E(T)	Value
ET	37.20 0.3720

E(T) denotes the expected number of tests.

Value denotes the objective function value per individual.

Overall accuracy of the algorithm:

PSe	PSp	PPPV	PNPV
ET 0.9550	1.0000	1.0000	0.9976

PSe denotes the pooling sensitivity.

PSp denotes the pooling specificity.

PPPV denotes the pooling positive predictive value.

PNPV denotes the pooling negative predictive value.

> ExpTests(OTC.LabCorp.Array)

ExpTests	ExpTestsPerInd	PercentReductionTests	PercentIncreaseTestCap
opt.ET 37.1953	0.3720	62.80	168.85

The OTC for array testing is a 10×10 array that has an expected number of tests per individual of 0.37 and an increase in testing capacity of 169%. This is a significant improvement over the 5×5 array. One could perform similar calculations using two-stage hierarchical testing with algorithm = "D2" in OTC1. This results in an OTC that uses an initial group size of 5 and leads to an expected number of tests per individual of 0.42.

Because these calculations rely on $p = 0.05$, Appendix B includes additional results for different values of p and a discussion of the corresponding more advanced code. In most cases, we find that a 5×5 array is not the OTC.

3.4 Additional functions

Not all group testing algorithms fit well within a general computing framework. For this reason, we include a few additional functions outside of those discussed previously. In particular, the halving() and Sterrett() functions provide alternative ways to use hierarchical testing algorithms. The former function calculates operating characteristics when positive testing groups are split only in half (Litvak et al. 1994; Black et al. 2012). The latter function calculates operating characteristics for algorithms that retest only one specimen at a time from a positive group (Sterrett 1957; Bilder et al. 2010a). Once a positive is found, the remaining specimens are retested again in a group. The idea behind this strategy is there will likely be only one positive (or few positives) in the original group.

4 Conclusion

The **binGroup2** package provides researchers and laboratories with the statistical tools needed to implement group testing effectively. An earlier version of this package was used as well by *The New York Times* to help readers understand the benefits from group testing during the COVID-19 pandemic (Bui et al. 2020). Similar to Bilder et al. (2010b), we encourage researchers to submit their own functions to us to be included within the package. This will allow users to have one overall package rather than many packages that may duplicate the work of others.

The identification functions are for one and two-infection assays. While there are a few three- and four-infection assays for infectious disease detection that have been recently released (e.g., the BD Max CT/GC/TV assay (BD 2023) that tests for pathogens which lead to chlamydia, gonorrhea, and trichomoniasis), these assays are used currently much less than single and two-infection assays. Also, derivations for operating characteristics become much more complex for more than two infections, so there are no closed-form expressions available for these types of assays. For example, Hou et al.

(2020) needed Monte Carlo simulation for a three-infection assay. We anticipate that it will be useful to include Monte Carlo simulation-based estimates of operating characteristics in future versions of **binGroup2** as these types of multiplex assays become more widely used. New research is needed though to find efficient computational approaches when searching for the OTC because of these simulation aspects.

While not the focus of this paper, the estimation functions from **binGroup** have been simplified and expanded upon using a coherent style. The new `propCI()` function combines three previous functions into one to calculate point estimates and confidence intervals for an infection prevalence. The new `propDiffCI()` provides similar functionality for the difference of two infection prevalences. Both functions incorporate new research since Bilder et al. (2010b). In particular, the bias correction methods of Hepworth and Biggerstaff (2017) are included to account for the long-standing problem of bias for point estimates in group testing (Swallow 1985). The `gtReg()` function combines three previous **binGroup** functions used to estimate regression models with group testing data that arise from different testing algorithms. The `gtSim()` function also combines three previous **binGroup** functions to simulate group testing data with covariates that can arise from different testing algorithms.

5 Acknowledgments

The authors thank Jeffrey Benfer and Kristopher Eveland at the SHL and Peter Iwen and Baha Abdalhamid at the Nebraska Public Health Laboratory for their consultation on CT/NG and SARS-CoV-2 testing. This research was supported by Grant R01 AI121351 from the National Institutes of Health. The views expressed in this article are those of the authors and do not necessarily reflect the official policy or position of the United States Air Force Academy, the Air Force, the Department of Defense, or the U.S. Government. Approved for public release: distribution unlimited. PA#: USAFA-DF-2022-474.

6 Appendix A: Details for figure

Our separate R program provides the code to create Figure 3. This figure presents a comparison of the three-stage hierarchical testing algorithm of Lohse et al. (2020) to using the OTC for two- and three-stage hierarchical testing. We use the `OTC1()` function to find the OTCs for the prevalences of 0.001, 0.005, and 0.01 to 0.20 by 0.01 with a maximum group size of 64. The large number of prevalences and large group sizes will result in a significant amount of computational time. For readers interested in testing the code, we recommend using a few prevalence values and a maximum group size of 30 for an initial running of it.

7 Appendix B: Labcorp's array testing

LabCorp (2022) described the use of arrays for up to size 5×5 . Our separate R program provides the code to create Figure 4 that examines their use further. This figure provides evidence that a 5×5 array size is not an optimal choice. For the corresponding calculations, we found the OTC for array testing by minimizing the expected number of tests per individual for the prevalences of 0.001, 0.005, and 0.01 to 0.20 by 0.01. Because two-stage hierarchical testing is also frequently used for SARS-CoV-2 detection, we also found the OTC for this algorithm. Throughout these calculations, we conservatively limit the maximum group size to 10.

References

- B. Abdalhamid, C. Bilder, E. McCutchen, S. Hinrichs, S. Koepsell, and P. Iwen. Assessment of specimen pooling to conserve SARS CoV-2 testing resources. *American Journal of Clinical Pathology*, 153:715–718, 2020. URL <https://doi.org/10.1093/ajcp/aqaa064>. [p21, 22, 24, 25]
- M. Abdelmalek. With all eyes on coronavirus testing, some researchers say ‘group testing’ could make up the shortage. *ABC News*, 2020. URL <https://abcnews.go.com/Health/eyes-coronavirus-testing-researchers-group-testing-make-shortage/story?id=70658896>. May 13; Retrieved November 11, 2023. [p21]
- D. Altman and J. Bland. Diagnostic tests 2: Predictive values. *BMJ*, 309:102, 1994. URL <https://doi.org/10.1136/bmj.309.6947.102>. [p28]

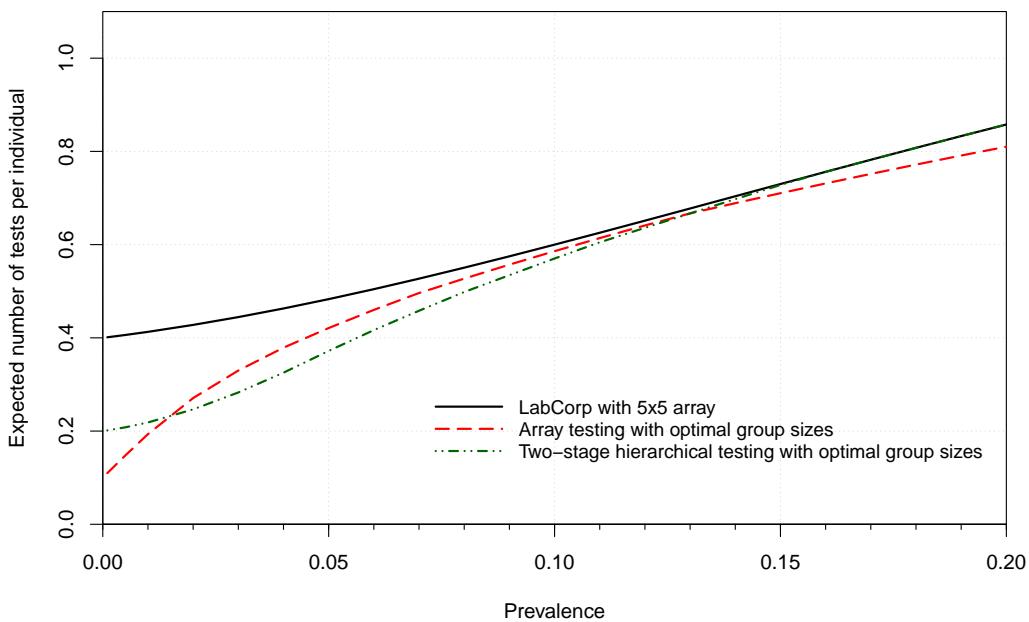


Figure 4: Expected number of tests per individual as a function of infection prevalence for LabCorp’s setting.

Altona Diagnostics. Realstar. 2023. URL <https://www.altona-diagnostics.com/en/products/reagents-140/reagents/realstar-real-time-pcr-reagents/realstar-sars-cov-2-rt-pcr-kit-ruo.html>. Retrieved November 11, 2023. [p27]

American Red Cross. Infectious disease testing. 2023. URL <https://www.redcrossblood.org/biomedical-services/blood-diagnostic-testing/blood-testing.html>. Retrieved November 11, 2023. [p22]

N. Ando, D. Mizushima, K. Watanabe, M. Takano, D. Shiojiri, H. Uemura, A. Takahiro, Y. Yanagawa, Y. Kikuchi, S. Oka, and H. Gatanaga. Modified self-obtained pooled sampling to screen for *Chlamydia trachomatis* and *Neisseria gonorrhoeae* infections in men who have sex with men. *Sexually Transmitted Infections*, 97:324–328, 2021. URL <http://dx.doi.org/10.1136/sextrans-2020-054666>. [p22]

E. Anthes. A CDC airport surveillance program found the earliest known US cases of Omicron subvariants. *The New York Times*, 2022. URL <https://www.nytimes.com/2022/03/24/health/cdc-us-ba2.html>. March 24; retrieved November 11, 2023. [p21]

R. Barathidasan, F. Sharmila, R. Raj, G. Dhanalakshmi, G. Anitha, and R. Dhodapkar. Pooled sample testing for COVID-19 diagnosis: Evaluation of bi-directional matrix pooling strategies. *Journal of Virological Methods*, page 114524, 2022. URL <https://doi.org/10.1016/j.jviromet.2022.114524>. [p21]

BD. BD Max CT/NG/TV. 2023. URL <https://moleculardiagnostics.bd.com/syndromic-solutions/womens-health-stis/CT-GC-TV>. Retrieved November 11, 2023. [p31]

C. Bilder. Group testing for identification. *Wiley StatsRef: Statistics Reference Online*, 2019. URL <https://doi.org/10.1002/9781118445112.stat08227>. [p24]

C. Bilder, J. Tebbs, and P. Chen. Informative retesting. *Journal of the American Statistical Association*, 105: 942–955, 2010a. URL <https://doi.org/10.1198/jasa.2010.ap09231>. [p31]

C. Bilder, B. Zhang, F. Schaarschmidt, and J. Tebbs. binGroup: A package for group testing. *The R Journal*, 2:56–60, 2010b. URL <https://journal.r-project.org/archive/2010-2>. [p21, 23, 31, 32]

C. Bilder, J. Tebbs, and C. McMahan. Informative group testing for multiplex assays. *Biometrics*, 75: 278–288, 2019. URL <https://doi.org/10.1111/biom.12988>. [p24, 25, 27, 28]

- C. Bilder, P. Iwen, and B. Abdalhamid. Pool size selection when testing for severe acute respiratory syndrome coronavirus 2. *Clinical Infectious Diseases*, 72:1104–1105, 2021a. URL <https://doi.org/10.1093/cid/ciaa774>. [p23]
- C. Bilder, J. Tebbs, and C. McMahan. Discussion on "Is group testing ready for prime-time in disease identification". *Statistics in Medicine*, 40:3881–3886, 2021b. URL <https://doi.org/10.1002/sim.8988>. [p24]
- M. Black, C. Bilder, and J. Tebbs. Group testing in heterogeneous populations by using halving algorithms. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 61:277–290, 2012. URL <https://doi.org/10.1111/j.1467-9876.2011.01008.x>. [p31]
- M. Black, C. Bilder, and J. Tebbs. Optimal retesting configurations for hierarchical group testing. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 64:693–710, 2015. URL <https://doi.org/10.1111/rssc.12097>. [p24]
- Q. Bui, S. Kliff, and M. Sanger-Katz. How to test more people for coronavirus without actually needing more tests. *The New York Times*, 2020. URL <https://www.nytimes.com/interactive/2020/07/27/upshot/coronavirus-pooled-testing.html>. July 27; retrieved November 11, 2023. [p31]
- C. Cheng, Y. Hung, and N. Balakrishnan. *rBeta2009: The Beta Random Number and Dirichlet Random Vector Generating Functions*, 2012. URL <https://CRAN.R-project.org/package=rBeta2009>. [p27]
- R. Dorfman. The detection of defective members of large populations. *Annals of Mathematical Statistics*, 14:436–440, 1943. URL <https://doi.org/10.1214/aoms/1177731363>. [p21]
- L. Graff and R. Roeloffs. Group testing in the presence of test error; an extension of the Dorfman procedure. *Technometrics*, 14:113–122, 1972. URL <https://doi.org/10.1080/00401706.1972.10488888>. [p26]
- G. Hepworth and B. Biggerstaff. Bias correction in estimating proportions by pooled testing. *Journal of Agricultural, Biological and Environmental Statistics*, 22:602–614, 2017. URL <https://doi.org/10.1007/s13253-017-0297-2>. [p32]
- B. Hitt. *Group Testing Identification: Objective Functions, Implementation, and Multiplex Assays*. PhD thesis, University of Nebraska-Lincoln, 2020. URL <https://digitalcommons.unl.edu/dissertations/AAI27956346>. [p24]
- B. Hitt, C. Bilder, J. Tebbs, and C. McMahan. The objective function controversy for group testing: Much ado about nothing? *Statistics in Medicine*, 38:4912–4923, 2019. URL <https://doi.org/10.1002/sim.8341>. [p21, 26, 28]
- C. Hogan, M. Sahoo, and B. Pinsky. Sample pooling as a strategy to detect community transmission of SARS-CoV-2. *Journal of the American Medical Association*, 323:1967–1969, 05 2020. URL <https://doi.org/10.1001/jama.2020.5445>. [p21]
- Hologic. Aptima STIs. 2023. URL <https://www.hologic.com/hologic-products/diagnostic-solutions/aptima-stis>. Retrieved November 11, 2023. [p23]
- P. Hou, J. Tebbs, D. Wang, C. McMahan, and C. Bilder. Array testing for multiplex assays. *Biostatistics*, 21:417–431, 2020. URL <https://doi.org/10.1093/biostatistics/kxy058>. [p22, 24, 31]
- ICBSS. *Iowa Community-Based Screening Services Procedures Manual*, 2014. URL <http://www.shl.uiowa.edu/dcd/iippmanual.pdf>. Retrieved November 11, 2023. [p27]
- N. Johnson, S. Kotz, and X. Wu. *Inspection Errors for Attributes in Quality Control*. CRC Press, 1991. URL <https://doi.org/10.1201/9781003059868>. [p24]
- H. Kim, M. Hudgens, J. Dreyfuss, D. Westreich, and C. Pilcher. Comparison of group testing algorithms for case identification in the presence of test error. *Biometrics*, 63:1152–1163, 2007. URL <https://doi.org/10.1111/j.1541-0420.2007.00817.x>. [p22, 23, 24]
- S. Kim, H. Kim, H. Kim, H. Ann, J. Kim, H. Choi, M. Kim, J. Song, J. Ahn, N. Ku, D. Oh, Y. Kim, S. Jeong, S. Han, J. Kim, D. Smith, and J. Choi. Pooled nucleic acid testing to identify antiretroviral treatment failure during HIV infection in Seoul, South Korea. *Scandinavian Journal of Infectious Diseases*, 46:136–140, 2014. URL <https://doi.org/10.3109/00365548.2013.851415>. [p22]
- LabCorp. Emergency Use Authorization summary: COVID-19 RT-PCR test. 2022. URL <https://www.fda.gov/media/136151/download>. Retrieved November 11, 2023. [p21, 23, 30, 32]

- S. Lendle. *gtcorr: Calculate Efficiencies of Group Testing Algorithms with Correlated Responses*, 2011. URL <https://CRAN.R-project.org/package=gtcorr>. Retrieved November 11, 2023. [p21]
- S. Lendle, M. Hudgens, and B. Qaqish. Group testing for case identification with correlated responses. *Biometrics*, 68:532–540, 2012. URL <https://doi.org/10.1111/j.1541-0420.2011.01674.x>. [p21]
- J. Lewis, V. Lockary, and S. Kobic. Cost savings and increased efficiency using a stratified specimen pooling strategy for *Chlamydia trachomatis* and *Neisseria gonorrhoeae*. *Sexually Transmitted Diseases*, 39: 46–48, 2012. URL <https://doi.org/10.1097/OLQ.0b013e318231cd4a>. [p23]
- E. Litvak, X. Tu, and M. Pagano. Screening for the presence of a disease by pooling sera samples. *Journal of the American Statistical Association*, 89:424–434, 1994. URL <https://doi.org/10.1080/01621459.1994.10476764>. [p31]
- T. Liu and Y. Xu. *mMPA: Implementation of Marker-Assisted Mini-Pooling with Algorithm*, 2018. URL <https://CRAN.R-project.org/package=mMPA>. Retrieved November 11, 2023. [p21]
- T. Liu, J. Hogan, M. Daniels, M. Coetzer, X. Yizhen, B. Gerald, A. DeLong, L. Ledingham, M. Orido, L. Diero, and R. Kantor. Improved HIV-1 viral load monitoring capacity using pooled testing with marker-assisted deconvolution. *Journal of Acquired Immune Deficiency Syndromes*, 75:580, 2017. URL <https://doi.org/10.1097/QAI.000000000001424>. [p21]
- S. Lohse, T. Pfuhl, B. Berkó-Göttel, J. Rissland, T. Geißler, B. Gärtner, S. Becker, S. Schneitler, and S. Smola. Pooling of samples for testing for SARS-CoV-2 in asymptomatic people. *The Lancet Infectious Diseases*, 20:1231–1232, 2020. URL [https://doi.org/10.1016/S1473-3099\(20\)30362-5](https://doi.org/10.1016/S1473-3099(20)30362-5). [p22, 26, 27, 28, 29, 30, 32]
- A. Mandavilli. Federal officials turn to a new testing strategy as infections surge. *The New York Times*, 2020. URL <https://www.nytimes.com/2020/07/01/health/coronavirus-pooled-testing.html>. July 1; retrieved November 11, 2023. [p21]
- C. McMahan, J. Tebbs, and C. Bilder. Informative Dorfman screening. *Biometrics*, 68:287–296, 2012a. URL <https://doi.org/10.1111/j.1541-0420.2011.01644.x>. [p23, 24]
- C. McMahan, J. Tebbs, and C. Bilder. Two-dimensional informative array testing. *Biometrics*, 68: 793–804, 2012b. URL <https://doi.org/10.1111/j.1541-0420.2011.01726.x>. [p22, 23]
- Nebraska Veterinary Diagnostic Center. Diagnostic tests & fees, 2023. URL <https://vbms.unl.edu/nvdc-tests-fees>. Retrieved November 11, 2023. [p22]
- Roche. Roche receives FDA Emergency Use Authorization for the cobas SARS-CoV-2 & Influenza A/B Test for use on the cobas 6800/8800 Systems. 2020. URL <https://www.roche.com/media/releases/med-cor-2020-09-04.htm>. Retrieved November 11, 2023. [p23]
- E. Salzer, E. Nixon, G. Drewes, F. Reinhard, G. Bergamini, and C. Rau. Screening pools of compounds against multiple endogenously expressed targets in a chemoproteomics binding assay. *Journal of Laboratory Automation*, 21:133–142, 2016. URL <https://journals.sagepub.com/doi/full/10.1177/2211068215595355>. [p22]
- H. Sanghani, D. Nawrot, F. Marmolejo-Cossío, J. Taylor, J. Craft, E. Kalimeris, M. Andersson, and S. Vasudevan. Concentrating pooled COVID-19 patient lysates to improve reverse transcription quantitative PCR sensitivity and efficiency. *Clinical Chemistry*, 67:797–798, 2021. URL <https://doi.org/10.1093/clinchem/hvab035>. [p25]
- A. Sterrett. On the detection of defective members of large populations. *The Annals of Mathematical Statistics*, 28:1033–1036, 1957. URL <https://doi.org/10.1214/aoms/1177706807>. [p31]
- W. Swallow. Group testing for estimating infection rates and probabilities of disease transmission. *Phytopathology*, 75:882–889, 1985. URL https://www.apsnet.org/publications/phytopathology/backissues/Documents/1985Abstracts/Phyto75_882.htm. [p32]
- J. Tan, A. Omar, W. Lee, and M. Wong. Considerations for group testing: a practical approach for the clinical laboratory. *The Clinical Biochemist Reviews*, 41:79–92, 2020. URL <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7731934>. doi:10.33176/AACB-20-00007. [p24]
- M. Thai. *Group Testing Theory in Network Security: An Advanced Solution*. Springer, 2011. URL <https://doi.org/10.1007/978-1-4614-0128-5>. [p22]

P. Thai, A. Banks, L. Toms, P. Choi, X. Wang, P. Hobson, and J. Mueller. Analysis of urinary metabolites of polycyclic aromatic hydrocarbons and cotinine in pooled urine samples to determine the exposure to PAHs in an Australian population. *Environmental Research*, 182:109048, 2020. URL <https://doi.org/10.1016/j.envres.2019.109048>. [p22]

T. Van, J. Miller, D. Warshauer, E. Reisdorf, D. Jernigan, R. Humes, and P. Shult. Pooling nasopharyngeal/throat swab specimens to increase testing capacity for influenza viruses by PCR. *Journal of Clinical Microbiology*, 50:891–896, 2012. URL <https://doi.org/10.1128/JCM.05631-11>. [p22]

Verily Life Sciences. Emergency Use Authorization summary: Verily COVID-19 RT-PCR test for use with the Verily COVID-19 Nasal Swab Kit. 2022. URL <https://www.fda.gov/media/141951/download>. Retrieved November 11, 2023. [p21]

Yale University. Emergency Use Authorization summary: SalivaDirect for use with DTC Kits assay. 2022. URL <https://www.fda.gov/media/151841/download>. Retrieved November 11, 2023. [p21]

K. Zhao and C. Rosa. Thrips as the transmission bottleneck for mixed infection of two orthopspoviruses. *Plants*, 9:509, 2020. URL <https://doi.org/10.3390/plants9040509>. [p22]

Christopher R. Bilder
University of Nebraska-Lincoln
Department of Statistics
Lincoln, NE 68583, USA
www.chrisbilder.com
chris@chrisbilder.com

Brianna D. Hitt
United States Air Force Academy
Department of Mathematical Sciences
Colorado Springs, CO 80840, USA
brianna.hitt@afacademy.af.edu

Brad J. Biggerstaff
Centers for Disease Control and Prevention
Division of Vector-Borne Diseases
Fort Collins, CO 80521, USA
bkb5@cdc.gov

Joshua M. Tebbs
University of South Carolina
Department of Statistics
Columbia, SC 29208, USA
tebbs@stat.sc.edu

Christopher S. McMahan
Clemson University
School of Mathematical and Statistical Sciences
Clemson, SC 29634, USA
mcmaha2@clemson.edu

multiocc: An R Package for Spatio-Temporal Occupancy Models for Multiple Species

by Staci Hepler and Robert Erhardt

Abstract Spatio-temporal occupancy models are used to model the presence or absence of a species at particular locations and times, while accounting for dependence in both space and time. Multivariate extensions can be used to simultaneously model multiple species, which introduces another dimension to the dependence structure in the data. In this paper we introduce multiocc, an R package for fitting multivariate spatio-temporal occupancy models. We demonstrate the use of this package fitting the multi-species spatio-temporal occupancy model to data on six species of birds from the Swiss MHB Breeding Bird Survey.

1 Introduction

Occupancy models are commonly used in ecological applications to model presence or absence of a species of interest across a geographical region over a period of study, or *season*. The goal is often to determine the likelihood that a species was present, and to study how that likelihood relates to environmental features. Occupancy models account for imperfect detection in a binary response variable Y by treating the true presence as a latent binary process Z . These methods have been used across a variety of applications including estimating the geographical distribution of species such as caribou in northern Ontario, Canada (Johnson et al., 2013) and of red fox and coyote in the eastern United States (Rota et al., 2016). Rahman et al. (2021) and Guillera-Arroita et al. (2019) used multispecies occupancy models to estimate the richness of mammal species in Bangladesh and breeding birds in Switzerland, respectively. Recently, occupancy models have been utilized in public health applications to estimate the risk or prevalence of imperfectly detected diseases such as Histoplasmosis (Hepler et al., 2022) and SARS-CoV-2 (Sanderlin et al., 2021).

A number of R packages exist for fitting specific types of occupancy models. Here, we highlight a few of the existing packages, and refer the reader to Doser et al. (2022) for a more comprehensive list. The package `unmarked` fits a variety of models using likelihood-based methods; however these models do not account for spatial dependence in the data. `stocc` fits single-species spatial occupancy models in the Bayesian framework (Johnson et al., 2013). Recent attention on joint species distribution models have highlighted the importance of jointly analyzing species that may be related. Recently, `spOccupancy` was developed to fit single-species and multi-species spatial occupancy models (Doser et al., 2022). However, the multi-species model implemented in `spOccupancy` accounts for cross-species dependence by assuming regression coefficients for different species arise from a shared community prior. While this model is reasonable when studying groups of species that are expected to respond to the environment similarly, it might not be suitable to all applications. Additionally, this package does not apply to multi-season occupancy models. `gjam` can analyze multivariate presence/absence data and assumes an unstructured covariance matrix to capture species dependence; however, it does not account for any spatio-temporal dependence (Clark et al., 2017). `Hmsc` includes a variety of joint species distribution models for analyzing presence/absence data of species within a community (Tikhonov et al., 2020; Ovaskainen et al., 2017). However, it does not account for imperfect detection in the binary observations.

In this paper, we present `multiocc`, an R package that implements the multivariate spatio-temporal occupancy model developed by Hepler and Erhardt (2021). This model can be used to analyze presence/absence data for $S \geq 1$ species across $T \geq 1$ seasons. Briefly, this model accounts for dependence across space, across seasons, and also between the species being jointly modeled. Thus, this R package overcomes limitations of the existing packages mentioned above and can be applied to more general study designs. As noted by Taylor-Rodriguez et al. (2017), modeling species independently ignores residual dependence between species and can yield misleading results. Guisan and Rahbek (2011) found independently modeling species predicted too many species per location, and Clark et al. (2014) found that prediction was improved by using a multivariate approach that exploits information from other species. Pollock et al. (2014) proposed a multivariate model for presence/absence data that captures residual correlation through a covariance matrix, Σ , and applied their method to data on frog and eucalyptus species, but the model does not account for spatio-temporal dependence or imperfect detection. The multivariate, spatio-temporal occupancy model of Hepler and Erhardt (2021) that is implemented in this package similarly accounts for residual species correlation through a matrix, Σ ,

but also accounts for spatio-temporal dependence and imperfect detection. In addition, this model has been shown to yield more accurate estimates of occupancy model parameters as compared to single-species models.

Additionally, this model and package can be used in the single-survey setting for occupancy models provided the number of seasons is “large enough”, regardless of the number of species of interest. This has important implications for study design, since previous research has stated that parameter estimation in standard occupancy models requires multiple surveys for each period of time where the occupancy status is unchanged (MacKenzie et al., 2002). The assumption of an unchanging occupancy field across surveys can be unreasonable for many applications. Further, inference can be biased if this assumption is violated, but standard occupancy models are often fit regardless (Rota et al., 2009). The model implemented here in **multiocc** allows researchers to assume that multiple, dependent seasons are observed with even just a single survey per season. This is less restrictive, as occupancy status does not need to remain static from one observation to the next, and inference is reliable provided there are enough seasons.

In the next section, we review the statistical details of the model and discuss the study designs under which this model is recommended. Then, we describe the core functions of the **multiocc** package. Lastly, we illustrate the implementation of the package’s functionality by fitting the model to data on six species of birds from the Swiss MHB Breeding Bird Survey.

2 Scientific Background

This package implements the multivariate spatio-temporal occupancy model proposed in Hepler and Erhardt (2021), which is briefly summarized here. The aim is to determine the likelihood that species s was *present* at location i during time period t , and how environmental features relate to the likelihood. However, imperfect detection implies that the observed data only confirms *detection*, and not necessarily true *presence*. Occupancy models differentiate between observed binary detection and latent true occupancy. Let $Z_{it}^{(s)}$ denote the true occupancy ($Z_{it}^{(s)} = 1$ indicates presence and $Z_{it}^{(s)} = 0$ absence) of species $s = 1, \dots, S$ at location $i \in I_t$ during season $t = 1, \dots, T$, where I_t denotes the set of locations of interest during season t with cardinality $|I_t| = n_t$. A *season* is defined to be a time period over which the true occupancy is unchanged. One feature of this R package is the ability to incorporate multiple independent observations, or *surveys* per season, but further to allow different number of surveys between sites.

Let V_{it} be the number of surveys of the occupancy field at location i during a season t . Let the observed binary detection be denoted with Y such that $Y_{itv}^{(s)} = 1$ indicates species s was detected at location i during the v th survey ($v = 1, \dots, V_{it}$) in season t and $Y_{itv}^{(s)} = 0$ indicates it was not detected. The proposed occupancy model assumes there are no false positives, so $Z_{it}^{(s)} = 0$ implies $Y_{itv}^{(s)} = 0$ for all v ; if the species is actually present ($Z_{it}^{(s)} = 1$), it may be detected during a survey ($Y_{itv}^{(s)} = 1$) or not ($Y_{itv}^{(s)} = 0$).

Royle and Dorazio (2008) introduced the hierarchical model

$$\begin{aligned} [Y_{itv}^{(s)} | Z_{it}^{(s)}, p_{itv}^{(s)}] &= \text{Bernoulli}\left(Z_{it}^{(s)} p_{itv}^{(s)}\right) \\ [Z_{it}^{(s)} | \psi_{it}^{(s)}] &= \text{Bernoulli}\left(\psi_{it}^{(s)}\right), \end{aligned} \tag{1}$$

where $\psi_{it}^{(s)}$ denotes the probability of occupancy for species s at location i during time t and $p_{itv}^{(s)}$ the probability of detection given occupancy during survey v . Hepler and Erhardt (2021) specified the detection probability as

$$p_{itv}^{(s)} = \Phi\left(\mathbf{W}'_{itv} \boldsymbol{\beta}^{(s)}\right), \tag{2}$$

where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution, \mathbf{W}_{itv} is a p_D -dimensional vector of covariates related to detectability, and $\boldsymbol{\beta}^{(s)}$ is a vector of regression coefficients. Note that the vector of covariates is assumed to be the same for all S species, but the regression coefficients are species-specific. For occupancy, Hepler and Erhardt (2021) specified the probability as

$$\psi_{it}^{(s)} = \Phi\left(\mathbf{X}'_{it} \boldsymbol{\alpha}^{(s)} + \eta_{it}^{(s)}\right), \tag{3}$$

where \mathbf{X}_{it} is a p_O -dimensional vector of covariates related to occupancy and is assumed to be the same for all species, $\boldsymbol{\alpha}^{(s)}$ is a species-specific vector of regression coefficients, and $\eta_{it}^{(s)}$ is a multivariate spatio-temporal random effect.

To reduce dimensionality and alleviate confounding with fixed effects, we use a restricted spatial regression model as in [Hughes and Haran \(2013\)](#) and [Bradley et al. \(2015\)](#). More specifically, if $\eta_t^{(s)} \equiv \{\eta_{it}^{(s)}, i \in I_t\}$ is the vector of random effects for species s and time t , then we assume $\eta_t^{(s)} = \mathbf{K}_t \gamma_t^{(s)}$, where \mathbf{K}_t is a $n_t \times q$ matrix of known spatial basis functions such that $\mathbf{K}_t' \mathbf{K}_t = \mathbf{I}_q$ and $\gamma_t^{(s)}$ is a q -dimensional multivariate random effect. Letting $\boldsymbol{\eta}_t \equiv (\eta_t^{(1)}, \dots, \eta_t^{(S)})'$ be the $n_t S$ -dimensional vector of random effects, we have $\boldsymbol{\eta}_t = \mathbf{K}_t^* \boldsymbol{\gamma}_t$, where $\mathbf{K}_t^* = \mathbf{I}_S \otimes \mathbf{K}_t$ is the $n_t S \times q S$ block diagonal matrix whose (s, s) block is \mathbf{K}_t , and $\boldsymbol{\gamma}_t = (\gamma_t^{(1)}, \dots, \gamma_t^{(S)})'$. An intrinsic multivariate conditional autoregressive (MCAR) structure is used to capture multivariate and spatial dependence, with temporal dependence captured through an autoregressive model of order 1 ([Mardia, 1988](#)). More specifically, this dependence framework is captured within the restricted spatial regression model by assuming

$$\boldsymbol{\gamma}_t \sim \begin{cases} N(\mathbf{0}, \mathbf{K}_t^{*\prime} \mathbf{Q}_t \mathbf{K}_t^*) & \text{for } t = 1 \\ N(\mathbf{M} \boldsymbol{\gamma}_{t-1}, \mathbf{K}_t^{*\prime} \mathbf{Q}_t \mathbf{K}_t^*) & \text{for } t = 2, \dots, T, \end{cases} \quad (4)$$

where $\mathbf{Q}_t = \Sigma^{-1} \otimes (\mathbf{D}_t - \mathbf{A}_t)$ is the intrinsic MCAR precision matrix, Σ is an $S \times S$ matrix that captures conditional dependence (positive or negative) among the species, \mathbf{A}_t is an $n_t \times n_t$ adjacency matrix, \mathbf{D}_t is a diagonal matrix whose (i, i) th element is the number of neighbors of location i , and $\mathbf{M} = \text{diag}(\rho_1, \dots, \rho_S) \otimes \mathbf{I}_q$ is the propagator matrix such that ρ_s accounts for temporal autocorrelation for species s . Following the work of [Bradley et al. \(2015\)](#), [Hepler and Erhardt \(2021\)](#) used spatio-temporal Moran's I (MI) basis functions for \mathbf{K}_t . More specifically, the MI operator for time t is defined as

$$\mathcal{I}(\mathbf{X}_t, \mathbf{A}_t) = \left(\mathbf{I}_{n_t} - \mathbf{X}_t (\mathbf{X}_t' \mathbf{X}_t)^{-1} \mathbf{X}_t' \right) \mathbf{A}_t \left(\mathbf{I}_{n_t} - \mathbf{X}_t (\mathbf{X}_t' \mathbf{X}_t)^{-1} \mathbf{X}_t' \right). \quad (5)$$

The q columns in \mathbf{K}_t are chosen to be the q eigenvectors of $\mathcal{I}(\mathbf{X}_t, \mathbf{A}_t)$ that correspond to the largest eigenvalues.

Since the model is fit within the Bayesian paradigm, prior distributions are chosen for all remaining hyperparameters. The prior distributions for the regression coefficients $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are improper and uniform over the real line. The temporal autocorrelation parameters ρ_1, \dots, ρ_S are assumed to be uniform over $(0, 1)$. The cross-species covariance matrix Σ has an inverse Wishart prior distribution with $S + 2$ degrees of freedom and identity scale matrix. To assist in computation, the data augmentation strategy of [Albert and Chib \(1993\)](#) is implemented in the Markov chain Monte Carlo (MCMC) algorithm. The model specification results in all known full conditional distributions, and thus a Gibbs sampling algorithm is used to simulate from the posterior distribution. Derivations of the full conditional distributions needed to implement the Gibbs sampling algorithm can be found in the supplementary material of [Hepler and Erhardt \(2021\)](#).

3 The multiocc package

The entry point for the package is the function `multioccbuild(detection = ..., occupancy = ..., coords = ..., DataNames = ..., threshold = ...)`. This function accepts raw data as inputs, and it outputs all ordered matrices and the adjacency information needed to run the MCMC algorithm and sample from the posterior distribution. This is intentionally split off as a separate function, rather than written internally as a precursor step in the MCMC algorithm. This structure allows the user to specify subsets of covariates, seasons, surveys, and/or locations for different model runs and receive informative errors (or confirmation) that the specification will lead to a valid model fit. The single output list produced by `multioccbuild()` contains an internally consistent set of ordered data frames as well as species and covariates names used, and helps the user avoid confusion by collecting all model inputs into a single list rather than requiring multiple specifications across objects which may be internally inconsistent. The five required arguments are:

- ‘detection’ is a data frame with one row for each site/season/survey combination for a total of $\sum_{t=1}^T \sum_{i \in I_t} V_{it}$ rows. The first three columns should identify the ‘site’, ‘season’, and ‘survey’, and be named as such. Site can be a character vector. For each of the S species there is a column of binary detection (1=yes, 0=no) for that particular species during the site/season/survey combination. Titles of these columns are chosen by the user, but are commonly the names of the S species. Next are the set of possible detection covariates used in \mathbf{W} . Titles of these are chosen by the user, and there is no need to include an intercept column in this data frame as one is automatically added by `multioccbuild()`. It is strongly encouraged that the quantitative explanatory variables be standardized to assist with chain mixing. Not every variable included in detection will necessarily be included in the model, as the `DataNames` argument described below is what defines the covariates to be used in \mathbf{W} .

- ‘occupancy’ is a data frame with one row for each site/season combination (note there is no survey for occupancy, as the species either is or is not present for all surveys). The number of rows is therefore $\sum_{t=1}^T n_t$, where T is the number of seasons, and n_t is the number of locations of interest (either observed or where predictions are desired) during season t . The first two columns are ‘site’ and ‘season’ and must be named as such. Site can be a character vector but must be consistent with how ‘site’ is specified in detection. Next are the set of possible occupancy covariates used in \mathbf{X} . Titles of these are chosen by the user, and there is no need to include an intercept column in this data frame as one is automatically added by `multioccbuild()`. It is strongly encouraged that the quantitative explanatory variables be standardized to assist with chain mixing. Not every variable included in occupancy will necessarily be included in the model, as the `DataNames` argument described below is what defines the covariates to be used in \mathbf{X} .
- ‘coords’ is a data frame with one column for the site name — which must be consistent with the naming of ‘site’ in occupancy and detection — and two columns for the numeric coordinates of the site location. Column names must be ‘site’, ‘x’, and ‘y’. The number of rows is equal to the number of unique locations in the study. Units are arbitrary, but must be the same in both location columns ‘x’ and ‘y’ for a valid Euclidean distance measure.
- ‘DataNames’ is a list with three elements, always titled ‘species’, ‘detection’, and ‘occupancy’. Each element is a vector specifying the precise titles of the S species, and the names of the covariates for detection and occupancy. The names in these vectors must correspond to the names of the corresponding columns in the detection and occupancy data frames. An intercept is added by default. `DataNames` is required because it allows the user to model a subset of species and/or a subset of all variables in the objects occupancy and detection, and therefore it is very easy to run multiple versions of the model with distinct covariates on the same data. If the user desires to have no covariates for detection, they can either write `detection=c()` or leave the argument `detection` out as an input. Similarly, if the user does not want any covariates to be included in occupancy, they can either write `occupancy=c()` or leave the argument `occupancy` out as an input.
- ‘threshold’ is a number which determines whether or not two sites are considered neighbors in the adjacency structure of the model. If the (Euclidean) distance between two sites is less than threshold, they are neighbors; if not, they are not neighbors. This is used to identify the non-zero entries in the adjacency matrices, A_t .

A few points of clarification are needed. First, the data frame `occupancy` should *only* have columns named ‘site’, ‘season’, and names of covariates which may appear in ‘`DataNames$occupancy`’. The data frame ‘`detection`’ should *only* have columns named ‘site’, ‘season’, ‘survey’, names for species detection data which also appear in ‘`DataNames$species`’, and names of detection covariates which may appear in ‘`DataNames$detection`’. The ordering of ‘`DataNames$species`’ will determine the ordering of all results shown by species. Internal checks in `multioccbuild()` include a scan for duplicated site/season combinations in occupancy, or site/season/survey combinations in detection. The presence of either type of duplication halts the function. The function also scans to ensure all entries for detections y are 0, 1, or NA. Also, if there are any site/season combinations in detection but not occupancy, they are removed from detection with a warning shown, but the function still runs. If there are any site/season combinations in occupancy but not detection, corresponding rows are added to detection with NAs for all variables, which enables prediction of occupancy for those site/season combinations.

There are also internal checks to identify missing covariates in either \mathbf{X} or \mathbf{W} . There are two cases of missing covariates which our model cannot accommodate. The first is any missingness in \mathbf{X} . The second is when y is observed (as either 0 or 1), but covariates for detection \mathbf{W} for the corresponding site/season/survey are missing. Both situations lead to a deletion of rows. If there are missing covariates in \mathbf{X} , then the row(s) are removed from \mathbf{X} and the V_{it} corresponding rows for the same site/season combination are removed from \mathbf{W} . If there are missing covariates for any site/season/survey combination of \mathbf{W} for cases with non-missing y , then the value of y is changed to NA which effectively removes the detection from the observed data but retains predictions of occupancy for the corresponding site/season combination.

It is possible to use this package to predict occupancy for site/season combinations where no observations were taken (meaning there are either no rows in ‘`detection`’ for that site/season, or the rows in ‘`detection`’ have y with NA), provided the values of the occupancy covariates are available. The site/season combinations of interest just need to be included as rows in the ‘`occupancy`’ and ‘`coords`’ data frames that are provided as input to `multioccbuild()`. The site/season combinations that are in ‘`occupancy`’ but not ‘`detection`’ are identified, and the Gibbs Sampling algorithm will simulate values of z from the posterior predictive distribution. We note that in other packages, prediction is a two-step procedure where the site/seasons to be predicted are provided after the model

fit. That is not the case here because computation of the Moran's I basis functions requires the full set of site/seasons at all locations - both observed and unobserved (Bradley et al., 2015).

From a modeling perspective, the primary decisions the researcher needs to make when executing the `multioccbuild()` function are: (1) Which specific covariates belong in **X** and **W** to model occupancy and detection? (2) Which species should be included? and (3) What threshold should be used to determine the definition of neighbor in the adjacency matrix? This R package does not restrict the type or number of covariates included in **X** and **W**, but it is worth noting the literature includes discussions of possible difficulty identifying regression coefficients when there is only a single survey per season. In the initial versions of occupancy models, MacKenzie et al. (2002) advised at least two surveys for a single season when the occupancy rate and detection probability are sufficiently large (0.7 and 0.3, respectively), and more surveys when these probabilities are smaller. Lele et al. (2012) investigated the single-survey scenario further and found that occupancy and detection probabilities can be estimated with a single survey when the number of spatial locations is large, provided they depend on covariates with at least one numeric covariate in detection and the two sets of covariates differ by at least one variable. Hepler et al. (2018) extended earlier work on occupancy models and proposed a spatio-temporal occupancy model for the multiple-season scenario and found the model could identify covariate effects in the single survey per season case, provided the number of seasons was large enough. For the multivariate spatio-temporal model implemented in `multiocc`, Hepler and Erhardt (2021) found the model could identify covariate effects in the single-survey case with at least 10-20 seasons. Careful investigation of the posterior distributions for dependence across the two parameters would be one way to investigate confounding and identifiability. Note that interactions between two or more variables may also be used and defined in the usual way.

The definition of the threshold governs the structure of the adjacency matrix **A**. If two sites are within this distance, then they are neighbors, and the corresponding element of the adjacency matrix is 1. When the data are collected on a regular grid, it is common to use either a rook (shared edge) or queen (shared edge or vertex) relationship to determine which grid points are neighbors. These neighborhood specifications can be implemented in this package by determining the distance between grid centroids and setting the threshold to be larger than that distance. In general, lower values of 'threshold' result in fewer neighbors and a sparser matrix **A**, while larger values result in more neighbors but a less sparse matrix **A**. There are computational advantages to a sparse adjacency matrix, and a low threshold can mimic the first-order Markov dependence commonly used when modeling aerial data (Gelfand et al., 2010).

The output of `multioccbuild()` is a list called '`model.input`'. This list contains the following elements:

- 'DataNames', the precise list with three elements, always titled 'species', 'detection', and 'occupancy' that the user specified for the `multioccbuild()` function. These are used downstream in the package.
- 'X', a matrix which serves as the design matrix for the occupancy portion of the model. The first column is a column of all 1s, and latter columns are from 'names\$occupancy'. The matrix has $\sum_{t=1}^T n_t$ rows - one for each site/season combination.
- 'W', a matrix which serves as the design matrix for the detection portion of the model. The first column is a column of all 1s, and latter columns are from 'names\$detection'. The matrix has $\sum_{t=1}^T \sum_{i \in I_t} V_{it}$ rows - one for each site/season/survey combination.
- 'y', a $\sum_{t=1}^T \sum_{i \in I_t} V_{it} \times S$ matrix of the binary values of detection for the S species. The columns are in the same order that the species are listed in in the vector 'DataNames\$species'.
- 'A', the symmetric adjacency matrix of 1s and 0s indicating which rows of 'occupancy' are neighbors. 'A' is a block diagonal matrix, where the t th block is the $n_t \times n_t$ dimensional matrix A_t .
- 'detection.info' is a data frame with the detection information containing the five columns 'siteID' (a numeric index of the site), 'site' (the factor identifier of the site), 'season', 'survey', and an indicator variable named 'observations' that indicates whether the survey result was missing (indicated by an NA in the 'y' matrix) or not.
- 'occupancy.info' is a data frame with 'siteID', 'site', and 'season' columns as well as columns for the x/y coordinates merged from 'coords'.

The function `GibbsSampler(M. iter, M. burn, M. thin, model. input, q, sv, every, WAIC, param2keep)` performs the Gibbs sampling MCMC algorithm to sample from the posterior distribution. This function is written entirely in R.

- 'M. iter' is a required input and is the number of iterations in the MCMC.
- 'M. burn' is the desired length of the burn in. Default is half of 'M. iter'.

- ‘M.thin’ is the desired thinning of the MCMC output. M.thin=1 is no thinning, M.thin=10 saves every tenth, etc. Default is 1.
- ‘model.input’ is the output from the `multioccbuild()` function described above.
- ‘q’ is the number of Moran’s I spatial basis functions used in the restricted spatial regression model. Default is 10 percent of the minimum number of locations in any season (rounded).
- ‘sv’ is a logical indicator controlling whether or not the chain should be saved at regular iterations or not. The number of iterations is set by ‘every’. Default for ‘sv’ is FALSE.
- ‘every’ a numeric value controlling the frequency that the MCMC chain results are saved. Default is 1000.
- ‘WAIC’ is a logical value indicating whether or not the MCMC should compute and save WAIC. Defaults to false. Note that computing WAIC increases the computational storage required to fit this model.
- ‘param2keep’ is a character vector that governs which outputs are saved. Permissible entries in this vector are ‘alpha’, ‘beta’, ‘gamma’, ‘rho’, ‘sigma’, ‘psi’, ‘z’, ‘p’, and ‘loglik’. The default is to save ‘alpha’, ‘beta’, ‘gamma’, ‘rho’, ‘sigma’, and ‘psi’.

A number of comments are needed to discuss issues with computational cost and computer memory. First, including more parameters in ‘param2keep’ increases the amount of storage required. In particular, saving the samples of ‘z’, ‘psi’, ‘p’, and ‘loglik’ can create storage issues depending on the amount of available RAM. This can be avoided by not saving those quantities as they can easily be computed based on their definitions provided ‘alpha’, ‘beta’, and ‘gamma’ are saved. For example, the detection probabilities ‘p’ can be computed for each stored value of β as probit ($W\beta$). However, it is also possible to store these outputs if memory is not a concern. If the user specifies `WAIC=TRUE`, ‘loglik’ will be created and computed but not stored by default; the user would additionally need to add ‘loglik’ to ‘param2keep’ to store these values. This is a higher cost to memory, but allows the user to perform other types of model assessment, e.g. leave-one-out cross-validation as in [Vehtari et al. \(2017\)](#).

The output of `GibbsSampler()` is a list. Specifically, it contains the following elements.

- ‘samples’ is a list with one element for each of the items specified in ‘param2keep’. Each individual element of this list is named according to ‘param2keep’ and is a valid MCMC object defined from the `coda` package, and contains the saved samples of that object from the MCMC.
- ‘run.time’ saves the run time for the `GibbsSampler()` command.
- ‘WAIC’ is a vector of length S which contains WAIC values by species, if this argument was true.
- ‘basis.K’ is an $\sum_{t=1}^T n_t \times q \cdot T$ matrix of the basis functions created using the `make.basis()` function.

Additionally, the list has elements ‘occupancy.info’, ‘detection.info’, ‘X’, ‘W’, and ‘y’ which are the same quantities that were outputted by `multioccbuild()` and are saved as output here as well for convenience. The ordering of the output in ‘psi’ corresponds to the site/season ordering in ‘occupancy.info’. Similarly, if ‘p’ and/or ‘loglik’ are included in ‘param2keep’ then the ordering of the values in ‘samples’ corresponds to the site/season/survey ordering in ‘detection.info’.

There are several limitations of the model implemented in the current version of the `multiocc` package. First, the code requires the same set of covariates are used for all species being modeled. Second, the package currently does not allow the user to specify their own prior distributions or initial values for model parameters. Third, the package currently requires the use of Moran’s I basis functions and assumes an intrinsic, conditional autoregressive structure where the neighborhood matrix is specific through a distance threshold. Fourth, while the use of basis functions makes the model scalable in the number of spatial locations, the computational expense could still be prohibitive if the number of species, S , or the number of seasons, T are very large. It also remains unclear how large S can be before computational and identifiability issues arise with estimating Σ . [Taylor-Rodriguez et al. \(2017\)](#) proposed a Dirichlet process approach to dimension reduction to overcome these issues in settings where the number of species is large ($\sim 10^3$). In a simulation study, they found their dimension reduced model outperformed a model using the full Σ when $S = 100$, although they noted that both models still performed well. However, their model did not include multiple seasons, which will improve estimation. It is important the user perform MCMC diagnostic checks to assess inferential stability for the elements of Σ .

4 Example

The data are taken from the Swiss MHB ("Monitoring Häufige Brutvögel") Breeding Bird Survey from the Swiss Ornithological Institute. This survey covers common bird species in Switzerland

and first began in 1999. Specifically, we consider the presence or absence of six types of Swiss Tits obtained at 267 locations over a ten year period (2004 - 2013). Each year is one season in our model. These data are freely available as ‘SwissTits’ in the R package [AHMbook](#) (Kéry et al., 2017), and are stored in our package [multiocc](#) as ‘detection’, ‘opccupancy’, and ‘coords’, which can be called with `data(detection)`, `data(occupancy)`, `data(coords)`. The locations ‘coords’ are 1km by 1km quadrats roughly evenly spread across the entire country of Switzerland. Locations are shown in Figure 1.

Locations of Sites in Switzerland

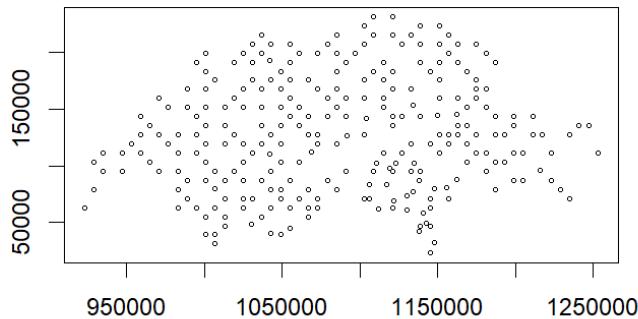


Figure 1: Locations of the 267 1km by 1km square quadrats laid out across the entirety of Switzerland in the MHB survey. In general, the spacing of locations supports the use of an areal data model with neighbors defined through an adjacency matrix A .

The response is the observed detection ($Y_{itv}^{(s)} = 1$) or not ($Y_{itv}^{(s)} = 0$) of species s at location i , time period t and survey v . There were 266 to 267 sites observed in each season, with the number of surveys per season varying from 2 to 3. Covariates on occupancy in X include the standardized percentage of forest cover at the location as well as the standardized elevation. The latent variable presence $Z_{it}^{(s)} = 1$ indicates that species s was present at location i and time period t , with detection probability $p_{it}^{(s)}$ modeled with standardized covariate duration, which measures the length of the particular survey. Histograms of the values for the covariates are shown in Figure 2. These same covariates are plotted across space in Figure 3. The left column shows raw data, and the right column shows interpolated covariates using a thin plate spline for ease of visualization only.

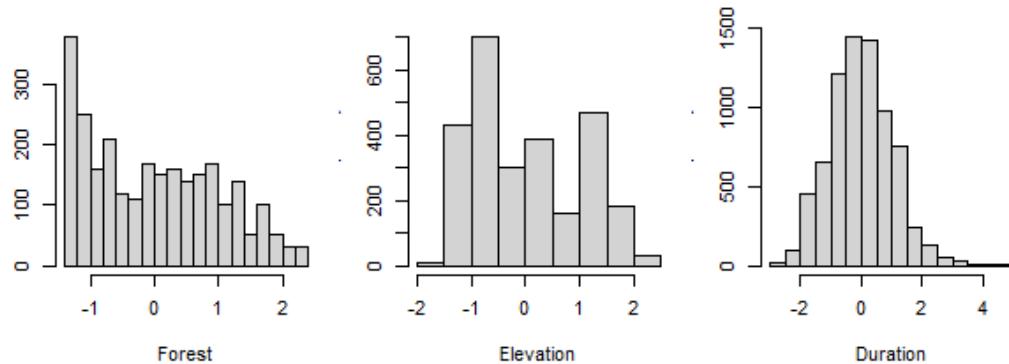


Figure 2: Histograms of the three covariates. Left: Percent forest cover across the 267 locations. Center: Elevation of the 267 sites. Right: Durations of the unique surveys. Each coavariate is standardized and therefore unitless on this scale.

To define the neighborhood structure used for the spatial random effect, we selected a threshold of 15000 (meters, as these are UTM coordinates) and defined any two locations whose distance is less than 15000 to be neighbors. This threshold gave each location somewhere between 1 and 8 neighbors, with an average of 4.23 neighbors. We assume $q = 10$ spatial basis functions in each season.

The `multioccbuild(detection, occupancy, coords, DataNames, threshold)` function takes five arguments as input, and outputs ‘model.input’ which is then an argument for `GibbsSampler()` which implements the Gibbs Sampling MCMC algorithm to simulate from the posterior distribution. Examples of the inputs ‘occupancy’, ‘detection’, ‘coords’, ‘DataNames’ and ‘threshold’ for our application are shown below:

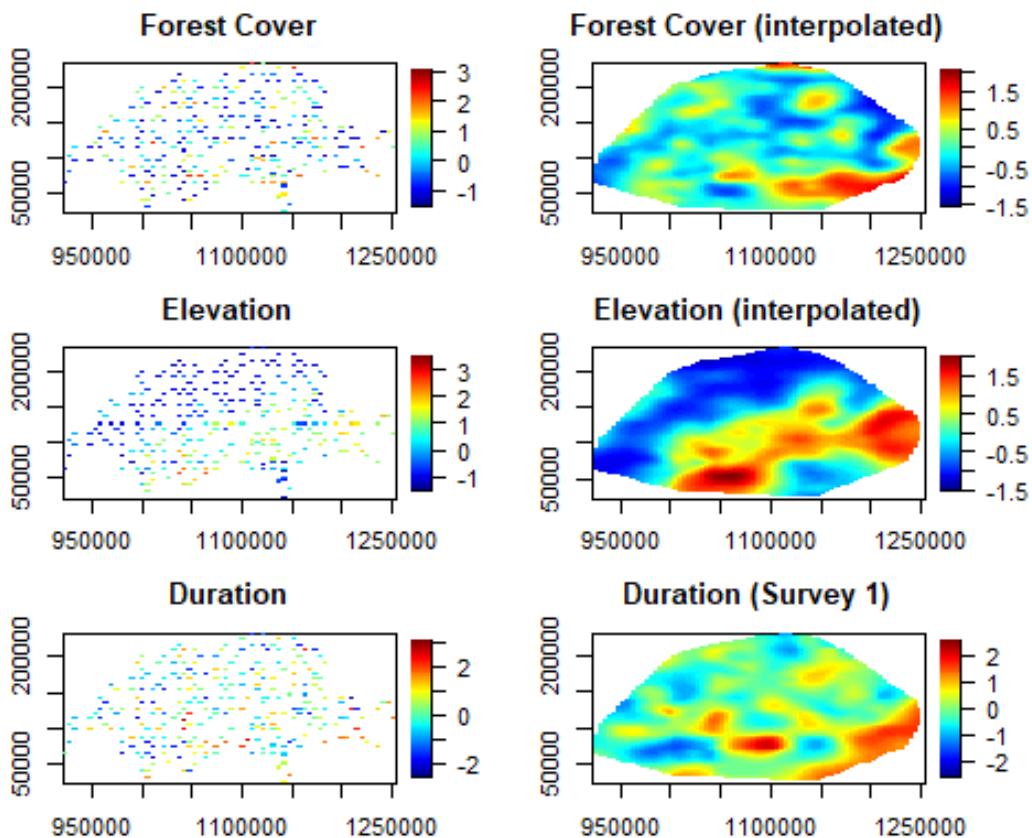


Figure 3: Spatial plots of the three covariates forest cover, elevation, and duration. The left column shows raw data, and the right column shows interpolated data using a thin plate spline for ease of visualization only. Forest and Duration are shown for season one, as these two covariates vary in time.

```
> head(occupancy, n=4L)
  site season      forest      elev
1 Q001     1 -1.148171612 -1.14938671
2 Q002     1 -0.496936229 -1.14938671
3 Q003     1 -0.098959051 -0.21468405
4 Q004     1 -0.931093151 -0.37046783

> head(detection, n=4L)
  site season survey Great.tit Blue.tit Coal.tit Crested.tit Marsh.tit Willow.tit duration
1 Q001     1      1       NA      NA      NA       NA      NA      NA  0.1563255
2 Q002     1      1       1       1       0       0       1      0 -0.9274962
3 Q003     1      1       1       0       1       0       1      0 -0.2605290
4 Q004     1      1       NA      NA      NA       NA      NA      NA -0.8441253
```

Observe that some surveys have NAs for all detections. Our model and package can easily accommodate missing values for the detections, but our model cannot handle missing values for occupancy covariates nor missing values for detection covariates when the corresponding 'y' are not missing. Missing values in the covariates will result in a warning when the `multioccbuild()` function is run, and rows corresponding to those observations in 'detection' and possibly in 'occupancy' will be removed. We suggest that all quantitative variables to be used as detection or occupancy covariates be standardized to aid with mixing in the MCMC algorithm. The remainder of the input items are:

```
> head(coords, n=4L)
  site      x      y
1 Q001 922942 63276
2 Q002 928942 79276
3 Q003 928942 103276
4 Q004 934942 95276

> DataNames = list("species"=colnames(detection)[4:9], "detection"=c("duration"),
```

```

"occupancy"=c("forest","elev"))
> DataNames
$species
[1] "Great.tit"   "Blue.tit"     "Coal.tit"     "Crested.tit" "Marsh.tit"    "Willow.tit"

$detection
[1] "duration"

$occupancy
[1] "forest" "elev"

```

With each of these inputs properly defined, we run the main data function `multioccbuild()`:

```

> model.input = multioccbuild(detection, occupancy, coords, DataNames, threshold = 15000)
Warning: Rows in detection with missing covariates have been removed for purposes of
fitting the model, but the site/season combination is retained in occupancy and therefore
predictions will be outputted.

```

The above message arises from cases in which we have NAs for covariates in detection \mathbf{W} , but we have actual observations of y for these same site/season/survey combinations. Our model cannot be fit using rows with missing detection covariates but non-missing detections in y . There are 544 NAs in the covariate, but only 4 of these cases have non-missing detections:

```

> sum(is.na(detection$duration))
[1] 544
> sum(!is.na(detection$Great.tit) & is.na(detection$duration))
[1] 4

```

One issue is site Q091, season 8:

```

> detection[detection$site == "Q091" & detection$season==8,]
  site season survey Great.tit Blue.tit Coal.tit Crested.tit Marsh.tit Willow.tit duration
5698 Q091     8      1       0       0       1       1       0       1 -1.177609
5965 Q091     8      2       1       0       1       1       0       1 -1.094238
6232 Q091     8      3       0       0       1       1       0       1      NA

```

The second issue is site Q125, season 2:

```

> detection[detection$site=="Q125" & detection$season==2,]
  site season survey Great.tit Blue.tit Coal.tit Crested.tit Marsh.tit Willow.tit duration
926  Q125     2      1       1       1       1       1       1       0      NA
1193 Q125     2      2       1       1       1       0       1       0      NA
1460 Q125     2      3       1       1       1       1       1       0      NA

```

The `multioccbuild()` function replaces the observed occupancy data y with NAs for these 4 cases. The output is 'model.input':

```

model.input = list("DataNames"=DataNames, "X"=X, "W"=W, "y"=y, "A"=A,
"Detection.info"=detection[,c("siteID", "site", "season", "survey", "observations")],
"Occupancy.info"=occupancy[,c("siteID", "site", "season", "x", "y")])

```

Objects saved here reflect handling the missing data for the 4 rows described above:

```

>model.input$y[(model.input$detection.info$site == "Q091" &
model.input$detection.info$season == 8),]
  Great.tit Blue.tit Coal.tit Crested.tit Marsh.tit Willow.tit
5698        0        0       1       1       0       1
5965        1        0       1       1       0       1
6232       NA       NA      NA      NA      NA      NA

> model.input$y[(model.input$detection.info$site == "Q125" &
model.input$detection.info$season == 2),]
  Great.tit Blue.tit Coal.tit Crested.tit Marsh.tit Willow.tit
926       NA       NA      NA      NA      NA      NA
1193       NA       NA      NA      NA      NA      NA
1460       NA       NA      NA      NA      NA      NA

```

The package reorders the data as site, then season, then survey, and ensures all matrices are in this order. The resulting design matrices for occupancy \mathbf{X} and detection \mathbf{W} are

```
> head(model.input$X, n=4L)
      forest      elev
1 1 -1.14817161 -1.1493867
11 1 -0.49693623 -1.1493867
21 1 -0.09895905 -0.2146840
32 1 -0.93109315 -0.3704678

> nrow(model.input$X)
[1] 2670

> head(model.input$W, n=4L)
[,1]      [,2]
[1,] 1 0.1563255
[2,] 1 1.1567763
[3,] 1 1.1567763
[4,] 1 -0.9274962

> nrow(model.input$W)
[1] 8010
```

The threshold is the Euclidean distance which defines whether or not two locations are neighbors. In our setting we set threshold = 15000 which results in the 2670×2670 block diagonal adjacency matrix

```
> model.input$A[1:5,1:5]
     1 11 21 32 41
1 0 0 0 0 0
11 0 0 0 0 0
21 0 0 0 1 1
32 0 0 1 0 0
41 0 0 1 0 0
```

Observe the detections \mathbf{y} are reordered because all components of the data were reordered according to site, season, and survey:

```
> head(model.input$y, n=4L)
Great.tit Blue.tit Coal.tit Crested.tit Marsh.tit Willow.tit
1       NA      NA      NA      NA      NA      NA
268     NA      NA      NA      NA      NA      NA
535     NA      NA      NA      NA      NA      NA
2        1       1       0       0       1       0

> head(model.input$detection.info, n=4L)
  siteID site season survey observations
1       1 Q001      1      1       0
268     1 Q001      1      2       0
535     1 Q001      1      3       0
2       2 Q002      1      1       1
> nrow(model.input$detection.info)
[1] 8010

> head(model.input$occupancy.info, n=4L)
  siteID site season      x      y
1       1 Q001      1 922942 63276
11      2 Q002      1 928942 79276
21      3 Q003      1 928942 103276
32      4 Q004      1 934942 95276
> nrow(model.input$occupancy.info)
[1] 2670
```

We ran our full analysis using the GibbsSampler(`M.iter = 50000, M.burn = 20000, M.thin = 10, model.input, q=10, sv=TRUE`). This took 4.03 hours to run on a single node on the Wake Forest University DEAC Cluster ([Information Systems and Wake Forest University, 2021](#)).

Results from the analysis are shown in Tables 1 and 2. It is important to note that we standardized all covariates to aid with MCMC mixing, and so we focus on the relative ordering of parameter estimates when comparing to other studies rather than absolute estimates. Table 1 shows the ordering of elevation's impact on occupancy as Willow Tit, Crested and Coal Tit, Great Tit, Blue Tit and Marsh Tit. This ordering largely matches the ordering published in [Tobler et al. \(2019\)](#) (Table S2), which considered the same data and six species in a somewhat similar Joint Species Distribution Model. [Chamberlain et al. \(2016\)](#) published that Willow Tits occupy higher elevations than Coal Tits in a nearby region of Italy, consistent with the ordering presented here. Turning to percent forest cover, our results show higher occupancy with higher forest cover for the Willow, Coal and Crested Tits, middlling occupancy for Marsh Tit, and lower occupancy for the Great and Blue Tits, again matching the ordering of [Tobler et al. \(2019\)](#).

Species	Intercept	Forest	Elevation
Great Tit	0.847 (0.769, 0.930)	-0.143 (-0.215, -0.072)	-0.224 (-0.300, -0.149)
Blue Tit	0.639 (0.547, 0.729)	-0.139 (-0.221, -0.058)	-0.261 (-0.349, -0.177)
Coal Tit	0.996 (0.919, 1.076)	-0.007 (-0.080, 0.066)	-0.130 (-0.204, -0.055)
Crested Tit	0.764 (0.667 , 0.867)	-0.014 (-0.097, 0.071)	-0.138 (-0.223, -0.052)
Marsh Tit	0.611 (0.509 , 0.724)	-0.124 (-0.219, -0.031)	-0.283 (-0.386, -0.182)
Willow Tit	0.383 (0.267 , 0.506)	0.108 (0.013, 0.199)	0.068 (-0.028, 0.163)

Table 1: Posterior means and 95% credible intervals for the occupancy regression coefficients α .

Species	Intercept	Duration
Great Tit	0.593 (0.553, 0.633)	-0.187 (-0.228, -0.147)
Blue Tit	0.109 (0.069, 0.149)	-0.257 (-0.299, -0.214)
Coal Tit	0.738 (0.696, 0.779)	0.303 (0.263, 0.346)
Crested Tit	-0.030 (-0.072, 0.011)	0.352 (0.309, 0.395)
Marsh Tit	-0.307 (-0.351, -0.264)	-0.090 (-0.133, -0.048)
Willow Tit	-0.482 (-0.532, -0.434)	0.430 (0.377, 0.482)

Table 2: Posterior means and 95% credible intervals for the detection regression coefficients β

Turning next to the residual dependence structure, Figure 4 shows the posterior mean of the correlation matrix computed from Σ for the six species, with notable differences across pairs of species. We see a mixture of positive and negative correlations across different pairs of species, with, for instance, a strong positive signal for the Coal and Crested Tits but a strong negative correlation for the Willow and Marsh Tits. The positive correlation estimated here for Great tit and Blue tit is consistent with [Stenseth et al. \(2015\)](#), who noted these species occupy the same geographical areas and compete for resources, which can result in negative correlation for abundance, but positive correlation for occupancy. Our estimated correlations for the six species somewhat differ from those estimated by [Tobler et al. \(2019\)](#), who showed results on residual correlation of occupancy probability for the six tit species for a single season, but computed under a latent factor model. Their results showed all positive correlations among the six species, a result they noted with surprise. However, this is reflective of the factor model structure in their assumed model which is less flexible than the multivariate random effect we used here. They also assumed different covariates and only modeled a single season, and as both model's dependence structures are residual to covariate effects, the estimates are not readily comparable.

Figures 5 and 6 show posterior means for occupancy probabilities ψ for seasons 1 and 7 for all six species. Hollow circles show the actual detections of each species in each corresponding season. For all $T = 10$ years, we observe a very strong relationship between predicted occupancy probabilities ψ and species detections for all species and all seasons.

5 Conclusion

In this article, we introduced the R package **multiocc** for implementing the multivariate spatio-temporal occupancy model proposed by [Hepler and Erhardt \(2021\)](#). This model overcomes many of the limitations of occupancy models that can be implemented with already existing R packages. More

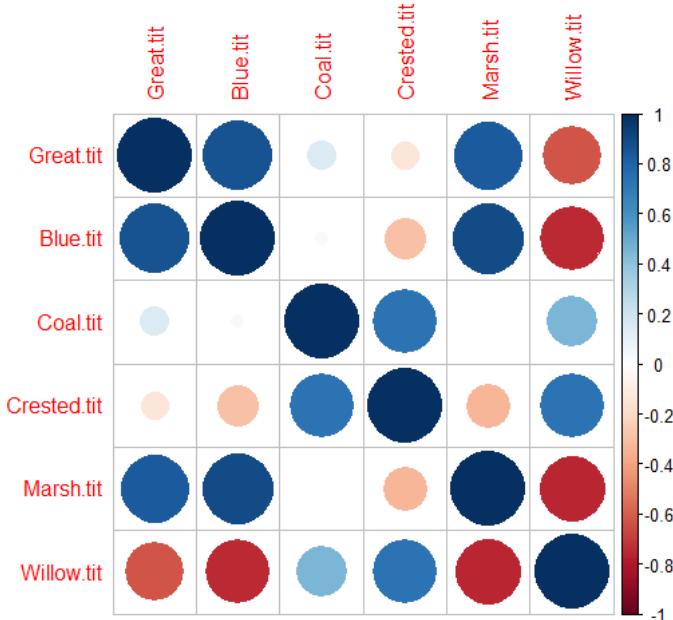


Figure 4: Posterior mean of the correlation matrix computed from Σ . Plot shows examples of pairs of strongly correlated species (e.g. Great Tit and Blue Tit), strongly negative correlated species (e.g. Marsh Tit and Willow Tit), and largely uncorrelated species (e.g. Marsh Tit and Coal Tit).

specifically, this model can jointly analyze $S \geq 1$ species over multiple seasons with a varying number of surveys at each site per season. The use of spatial basis functions makes this method feasible even with a large number of spatial locations.

Using this package to analyze imperfectly detected presence/absence data requires two main steps. First, the `multioccbuild()` function takes the raw data and outputs all matrices required to run the MCMC algorithm. Then, the `GibbsSampler()` function performs the Gibbs sampling algorithm to generate samples from the posterior distribution which can then be used to perform inference. We illustrated use of this package by analyzing occupancy data on six bird species in Switzerland from 2004 - 2013. The development of `multiocc` makes this multivariate spatio-temporal occupancy model accessible to the large community of researchers who use R for their data analysis needs.

References

- J. Albert and S. Chib. Bayesian analysis of binary and polychotomous response data. *Journal of the American Statistical Association*, 88(422):669–679, 1993. [p39]
- J. R. Bradley, S. H. Holan, and C. K. Wikle. Multivariate spatio-temporal models for high-dimensional areal data with application to longitudinal employer-household dynamics. *The Annals of Applied Statistics*, 9(4):1761–1791, 2015. [p39, 41]
- D. Chamberlain, M. Brambilla, E. Caprio, P. Pedrini, and A. Rolando. Alpine bird distributions along elevation gradients: the consistency of climate and habitat effects across geographic regions. *Oecologia*, 181(4):1139–1150, 2016. [p47]
- J. S. Clark, A. E. Gelfand, C. W. Woodall, and K. Zhu. More than the sum of the parts: forest climate response from joint species distribution models. *Ecological Applications*, 24(5):990–999, 2014. [p37]
- J. S. Clark, D. Nemergut, B. Seyednasrollah, P. J. Turner, and S. Zhang. Generalized joint attribute modeling for biodiversity analysis: Median-zero, multivariate, multifarious data. *Ecological Monographs*, 87(1):34–56, 2017. [p37]
- J. W. Doser, A. O. Finley, M. Kéry, and E. F. Zipkin. spoccupancy: An r package for single-species, multi-species, and integrated spatial occupancy models. *Methods in Ecology and Evolution*, 2022. [p37]

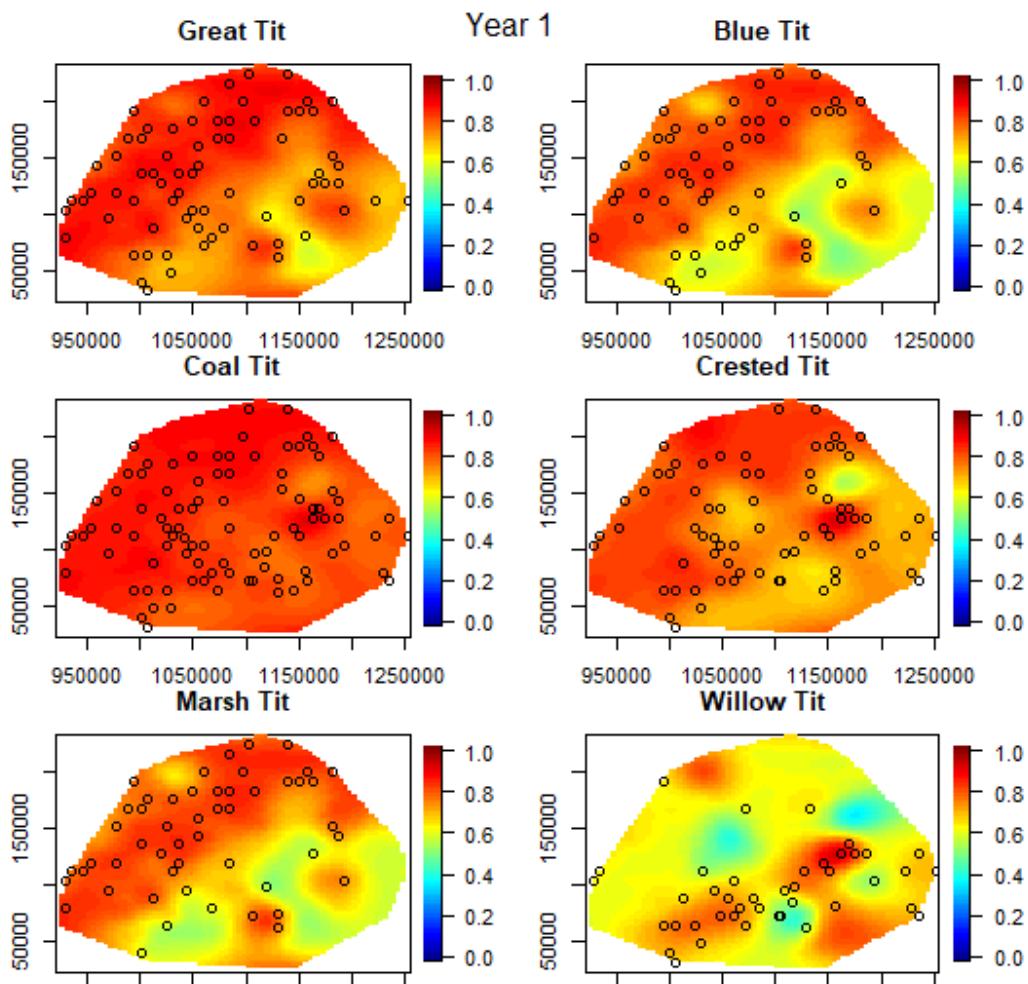


Figure 5: An example of occupancy predictions for season 1. Colors show posterior means of ψ for each location and species, interpolated across space using a thin plate spline for ease of visualization only. Hollow circles show points with detections of the species in that particular season. As expected, regions of higher posterior probability match regions with greater density of detections.

- A. E. Gelfand, P. Diggle, P. Guttorp, and M. Fuentes. *Handbook of spatial statistics*. CRC press, 2010. [p41]
- G. Guillera-Arroita, M. Kéry, and J. Jahoz-Monfort. Inferring species richness using multispecies occupancy modeling: Estimation performance and interpretation. *Ecology and Evolution*, 9(2): 780–792, 2019. [p37]
- A. Guisan and C. Rahbek. Sesam—a new framework integrating macroecological and species distribution models for predicting spatio-temporal patterns of species assemblages, 2011. [p37]
- S. Hepler, R. Erhardt, and T. Anderson. Identifying drivers of spatial variation in occupancy with limited replication camera trap data. *Ecology*, 99(10):2152–2158, 2018. [p41]
- S. Hepler, K. Kaufeld, K. Benedict, M. Toda, B. Jackson, X. Liu, and D. Kline. Integrating public health surveillance and environmental data to model presence of Histoplasma in the United States. *Epidemiology*, 2022. doi: 10.1097/EDE.0000000000001499. [p37]
- S. A. Hepler and R. J. Erhardt. A spatiotemporal model for multivariate occupancy data. *Environmetrics*, 32(2):e2657, 2021. [p37, 38, 39, 41, 47]
- J. Hughes and M. Haran. Dimension reduction and alleviation of confounding for spatial generalized linear mixed models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 75(1): 139–159, 2013. [p39]

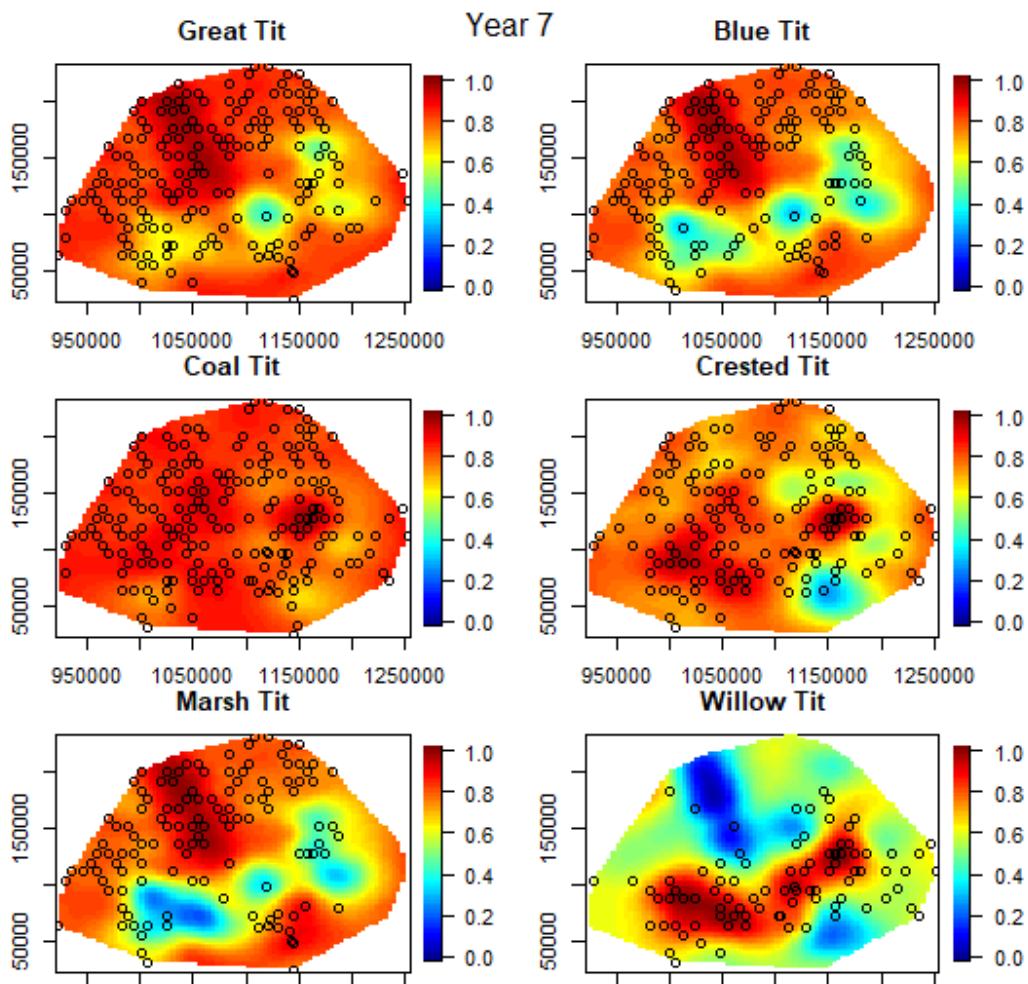


Figure 6: An example of occupancy predictions for season 7. Colors show posterior means of ψ for each location and species, interpolated across space using a thin plate spline for ease of visualization only. Hollow circles show points with detections of the species in that particular season. As expected, regions of higher posterior probability match regions with greater density of detections.

Information Systems and Wake Forest University. WFU High Performance Computing Facility, 2021.
 URL <https://hpc.wfu.edu>. [p47]

D. Johnson, P. Conn, M. Hooten, J. Ray, and B. Pond. Spatial occupancy models for large data sets. *Ecology*, 94(4):801–808, 2013. [p37]

M. Kéry, A. Royle, and M. Meredith. Ahmbook: functions and data for the book ‘applied hierarchical modeling in ecology’. *R package version 0.1, 3*, 2017. [p43]

S. Lele, M. Moreno, and E. Bayne. Dealing with detection error in site occupancy surveys: what can we do with a single survey? *Journal of Plant Ecology*, 5(1):22–31, 2012. [p41]

D. MacKenzie, J. Nichols, G. Lachman, S. Droege, J. Royle, and C. Langtimm. Estimating site occupancy rates when detection probabilities are less than one. *Ecology*, 83(8):2248–2255, 2002. [p38, 41]

K. Mardia. Multi-dimensional multivariate gaussian markov random fields with application to image processing. *Journal of Multivariate Analysis*, 24(2):265–284, 1988. [p39]

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. [p37]

- 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 (jsdm). *Methods in Ecology and Evolution*, 5(5):397–406, 2014. [p³⁷]
- H. Rahman, K. McCarthy, J. McCarthy, and M. Faisal. Application of multi-species occupancy modeling to assess mammal diversity in northeast Bangladesh. *Global Ecology and Conservation*, 25, 2021. [p³⁷]
- C. Rota, C. Wikle, R. Kays, T. Forrester, W. McShea, A. Parsons, and J. Millspaugh. A two-species occupancy model accommodating simultaneous spatial and interspecific dependence. *Ecology*, 97(1):48–53, 2016. [p³⁷]
- C. T. Rota, R. J. Fletcher Jr, R. M. Dorazio, and M. G. Betts. Occupancy estimation and the closure assumption. *Journal of Applied Ecology*, 46(6):1173–1181, 2009. [p³⁸]
- J. A. Royle and R. M. Dorazio. *Hierarchical modeling and inference in ecology: the analysis of data from populations, metapopulations and communities*. Academic Press, 2008. [p³⁸]
- J. Sanderlin, J. Golding, T. Wilcox, D. Mason, K. McKelvey, D. Pearson, and M. Schwartz. Occupancy modeling and resampling overcomes low test sensitivity to produce accurate SARS-CoV-2 prevalence estimates. *BMC Public Health*, 21(577), 2021. [p³⁷]
- N. C. Stenseth, J. M. Durant, M. S. Fowler, E. Matthysen, F. Adriaensen, N. Jonzén, K.-S. Chan, H. Liu, J. De Laet, B. C. Sheldon, et al. Testing for effects of climate change on competitive relationships and coexistence between two bird species. *Proceedings of the Royal Society B: Biological Sciences*, 282(1807):20141958, 2015. [p⁴⁷]
- D. Taylor-Rodriguez, K. Kaufeld, E. M. Schliep, J. S. Clark, and A. E. Gelfand. Joint species distribution modeling: dimension reduction using dirichlet processes. *Bayesian Analysis*, 12(4):939–967, 2017. [p^{37, 42}]
- G. Tikhonov, Ø. H. Opdal, N. Abrego, A. Lehikoinen, M. M. de Jonge, J. Oksanen, and O. Ovaskainen. Joint species distribution modelling with the r-package hmsc. *Methods in ecology and evolution*, 11(3):442–447, 2020. [p³⁷]
- M. W. Tobler, M. Kéry, F. K. Hui, G. Guillera-Arroita, P. Knaus, and T. Sattler. Joint species distribution models with species correlations and imperfect detection. *Ecology*, 100(8):e02754, 2019. [p⁴⁷]
- A. Vehtari, A. Gelman, and J. Gabry. Practical bayesian model evaluation using leave-one-out cross-validation and waic. *Statistics and computing*, 27:1413–1432, 2017. [p⁴²]

Staci Hepler
Department of Statistical Sciences, Wake Forest University
Winston-Salem NC, 27103 USA
heplersa@wfu.edu

Robert Erhardt
Department of Statistical Sciences, Wake Forest University
Winston-Salem NC, 27103 USA
erhardrj@wfu.edu

Accessible Computation of Tight Symbolic Bounds on Causal Effects using an Intuitive Graphical Interface

by Gustav Jonzon, Michael C Sachs, and Erin E Gabriel

Abstract Strong untestable assumptions are almost universal in causal point estimation. In particular settings, bounds can be derived to narrow the possible range of a causal effect. Symbolic bounds apply to all settings that can be depicted using the same directed acyclic graph and for the same effect of interest. Although the core of the methodology for deriving symbolic bounds has been previously developed, the means of implementation and computation have been lacking. Our R-package `causaloptim` aims to solve this usability problem by providing the user with a graphical interface through Shiny. This interface takes input in a form that most researchers with an interest in causal inference will be familiar: a graph drawn in the user's web browser and a causal query written in text using common counterfactual notation.

1 Introduction

A common goal in many different areas of scientific research is to determine causal relationships between one or more exposure variables and an outcome. Prior to any computation or inference, we must clearly state all assumptions made, i.e., all subject matter knowledge available, regarding the causal relationships between the involved variables as well as any additional variables, called confounders, that may not be measured but influence at least two other variables of interest. A popular tool in applied research for encoding these causal assumptions is a directed acyclic graph (DAG), in which directed edges represent direct causal influences (Greenland et al., 1999). Such a DAG not only clearly states the assumptions made by the researcher, but also comes with a sound methodology for causal inference, in the form of identification results (theorems on when an estimand is estimable) as well as derivation of expressions of causal estimands in terms of observable quantities (Pearl, 2009).

Unfortunately, point identification of a desired causal effect typically requires an assumption of no unmeasured confounders, in some form. When there are unmeasured confounders, it is sometimes still possible to derive bounds on the effect, i.e., a range of possible values for the causal effect in terms of the observed data distribution. Symbolic bounds are algebraic expressions for the bounds on the causal effect written in terms of probabilities that can be estimated using observed data. Alexander Balke and Judea Pearl first used linear programming to derive tight symbolic bounds in a simple binary instrumental variable (IV) setting (Balke and Pearl, 1997). Balke wrote a program in C++ to take a linear programming problem as text file input, perform variable reduction, conversion of equality constraints into inequality constraints, and perform the vertex enumeration algorithm of Mattheiss (1973). This program has since been used by researchers in the field of causal inference (Balke and Pearl, 1997; Cai et al., 2008; Sjölander, 2009; Sjölander et al., 2014) but it is not particularly accessible because of the technical challenge of translating the DAG plus causal query into the constrained optimization problem and determining whether it is linear. Moreover, the program is not optimized and hence does not scale well to more complex problems. Since they only cover a simple instrumental variable setting, it has also not been clear to what extent their techniques extend to more general settings, nor how to apply them to more complex queries. Thus, applications of this approach have been limited to a small number of settings and few attempts to generalize the method to more widely applicable settings have been made.

Recent developments have expanded the applicability by generalizing the techniques and the causal DAGs and effects to which they apply (Sachs et al., 2022). These new methods have been applied in novel observational and experimental settings (Gabriel et al., 2022a, 2023, 2022b). Moreover, through the R package `causaloptim` (Sachs et al., 2023), these computations are now accessible. With `causaloptim`, the user needs only to give input in a way they would usually express their causal assumptions and state their target causal estimand; through a DAG and counterfactual expression. Providing DAGs through textual input is an awkward experience for most users, as DAGs are generally communicated pictorially. Our package `causaloptim` provides a user-friendly graphical interface (GUI) through a web browser, where the user can draw their DAG in a way that is familiar to them. The methodology that underpins `causaloptim` is not universal however; some restrictions on the DAG and query are imposed. These are validated and communicated to the user through the graphical interface, which guides the user through providing the DAG and query, adding any extra conditions beyond those encoded in the DAG, computing, interpreting and exporting the bounds for various

further analyses.

There exist few other R-packages related to causal bounds and none to our knowledge for computation of symbolic bounds. **bpbounds** (Palmer et al., 2018) provides a text-based interface to compute numeric bounds for the original single instrumental variable example of Balke and Pearl and extends this by being able to compute bounds given different types of data input including a ternary rather than binary instrument. There is also a standalone program written in Java by the TETRAD Project (<https://github.com/cmu-phil/tetrad>) that includes a GUI and has a wrapper for R. Its focus, however, is on causal discovery in a given sample data set, and although it can also compute bounds, it can do so only numerically for the given data set.

In this paper we describe our R package **causaloptim**, first focusing on the graphical and programmatic user interfaces in the next 2 sections. Then we highlight some of our interesting functions and data structures that may be useful in other contexts. We provide a summary of the theoretical background and methods, while referring to the companion paper Sachs et al. (2022) for the details. We illustrate the use of the package with some numeric examples and close with a discussion and summary.

2 Graphical user interface

In the following, we will work through the binary instrumental variable example, where we have 3 observed binary variables X , Y , Z , and we want to determine the average causal effect of X on Y given by the total causal risk difference, in the presence of unmeasured confounding by U_R and an instrumental variable Z . Our causal DAG is given by $Z \rightarrow X \rightarrow Y$ and $X \leftarrow U_R \rightarrow Y$ and our causal query is $P(Y(X = 1) = 1) - P(Y(X = 0) = 1)$, where we use $Y(X = x)$ to denote the counterfactual outcome Y if X were intervened upon to have value x . The package can handle more variables and complex settings including multiple exposures, outcomes and nested counterfactuals (although only a single exposure and outcome may be set by clicking vertices in the GUI, the query may easily be modified to a more complex one), but this classic example will serve to showcase its functionality.

causaloptim includes a GUI implemented in **shiny** (Chang et al., 2022). The interface is launched in the user's default web browser by calling `specify_graph()`. Once the **shiny** app is launched, the user is presented with an interactive display as shown in Figure 1, in which they can draw their causal DAG. This display is divided into a left side \mathcal{L} and right side \mathcal{R} to classify the vertices according to the class of DAGs that the method covers. This division signifies the following: Existence of unmeasured confounding is assumed *within each* of these sides, but *not between* them, and any causal influence between the two sides *must originate* in \mathcal{L} . Generally speaking, the outcome variables of interest will typically be on the right side, things like instruments or randomized treatments will be on the left side, and confounded exposures of interest on the right side. Thus, for the example, we would want to put the instrumental variable on the left side, but the exposure and outcome on the right side. In the web version of this article an interactive version of this interface is shown at the end of this section.

The DAG that we aim to construct is depicted in Figure 2.

2.1 Specifying the setting by drawing a causal diagram and adding attributes

The DAG is drawn using a point-and-click device (e.g., a mouse) to add vertices representing variables (by Shift+click) and name them (using any valid variable name in R without underscores), and to draw edges representing direct causal influences (Shift+drag) between them. The vertices may also be moved around, renamed and deleted (as can the edges) as also described in an instruction text preceding the DAG interface. As shown in Figure 3, for the example we add a vertex Z on the left side, and vertices X and Y on the right side. Then the $Z \rightarrow X$ and $X \rightarrow Y$ edges are added by Shift+clicking on a parent vertex and dragging to the child vertex. There is no need to add the unmeasured confounder variable U_R as it is assumed and added automatically. The same would have applied to U_L , but a confounder is added to a side only if it contains at least two variables.

Importantly, the nodes may be selected and assigned additional information. In \mathcal{R} , a variable may be assigned as unobserved (click+'u'). All observed variables are assumed categorical and their cardinality (i.e., number of levels) may be set (click+'c' brings up a prompt for this this number; alternatively a short-cut click+'any digit' is provided), with the default being binary. Although the causal query (i.e., the causal effect of interest) is entered subsequently, the DAG interface provides a convenient short-cut; a node X may be assigned as an exposure (click+'e') and another Y as outcome (click+'y'), whereupon the default query is the total causal risk difference $P(Y(X = 1) = 1) - P(Y(X = 0) = 1)$. Finally, an edge may be assigned as representing an assumed monotonic influence (click+'m'), i.e., that increasing the value of the exposure can not decrease the value of the outcome, e.g., $X(Z = 0) \leq X(Z = 1)$. The nodes and edges change appearance according to their assigned

Causal Network Analysis and Optimization

How does this work?

The first step is to draw a graph, but it cannot be just any graph, there are some important rules to follow: The graph is divided into a left side and a right side. Within each side, all variables must be mutually confounded by unmeasured confounders. There must be at least 1 variable in the right side, but the left side can be empty. But you do not have to draw the unmeasured confounders – once you press ‘Analyze the graph’, the algorithm will automatically add common causes to each side. Connections between the left and right sides must originate from the left, i.e., no connections from the right to left are allowed. The left side and right side variables are unconfounded. Generally speaking, you want to put outcome variables of interest on the right side, things like instruments or randomized treatments on the left side, and exposures of interest on the right side.

Getting started

To add new variables/nodes: Hold shift and click where you want the node, then type a name for the node. Choose a short name, starting with a letter.
 To draw a directed edge connecting two nodes: Hold shift, and use the mouse to click and drag from the first node to the second.
 Click on a node to select it, it will turn salmon colored.
 If you made a mistake, select a node and press ‘d’ to remove.
 Select a node, then use the keyboard to mark it with special information (optional):

- press ‘u’ to mark it as unobserved
- press ‘v’ to mark it as the outcome of interest
- press ‘x’ to mark it as the exposure of interest
- select a node and press a digit to set that number of possible categorical values (all variables default to binary), or press ‘c’ and enter a number into the prompt.

 Click an edge and press ‘m’ to enforce monotonicity for that connection. Other constraints can be specified later.

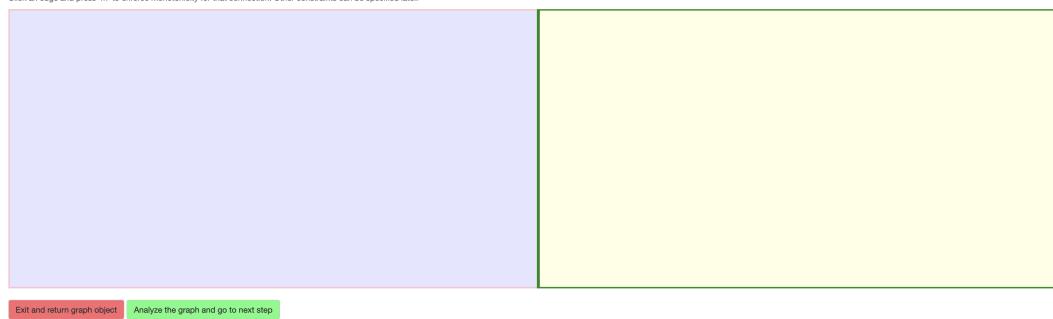


Figure 1: The Shiny web interface at launch

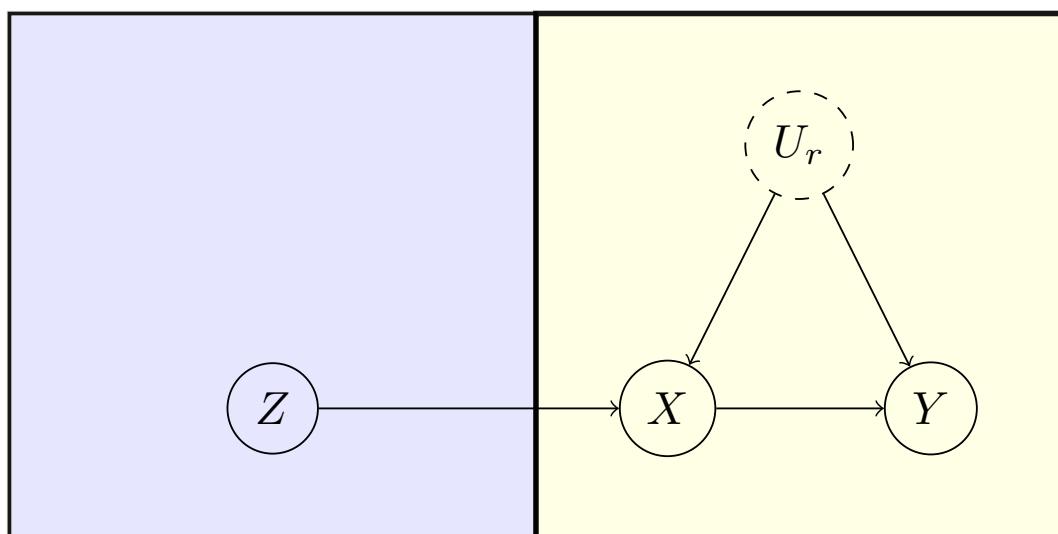
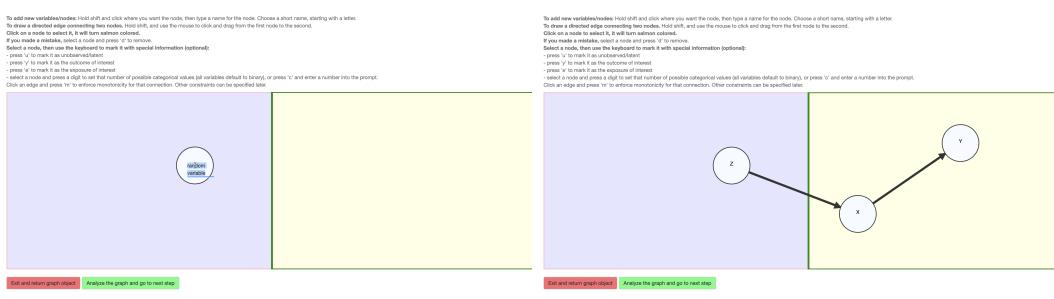


Figure 2: Causal DAG for the IV example.



(a) Adding and naming variables

(b) Adding directed edges

Figure 3: Constructing the DAG

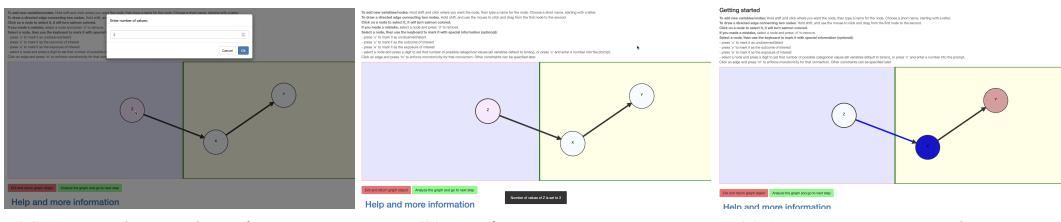


Figure 4: Setting attributes

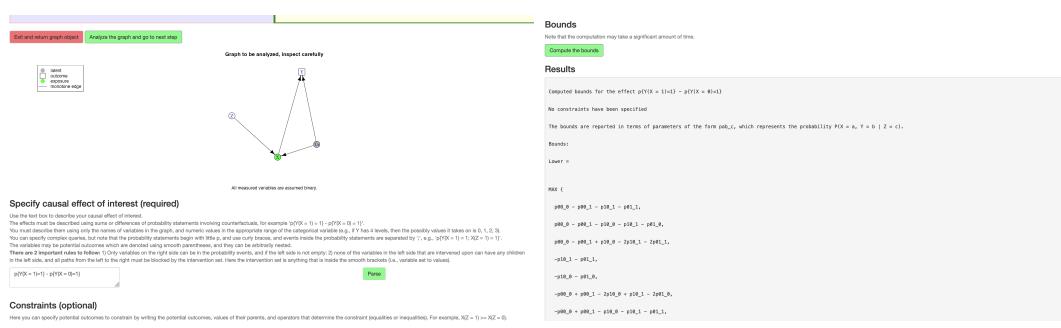


Figure 5: The causal DAG and bounds

characteristics (Figure 4) and violations to the restrictions characterizing the class of DAGs are detected and communicated to the user.

Once the DAG has been drawn, the user may click the button “Analyze the graph”, upon which the DAG is interpreted and converted into an annotated *igraph*-object (Csardi and Nepusz, 2006) as described in the implementation details below, and the results are displayed in graphical form to the user (Figure 5). The addition of U_R , the common unmeasured cause of X and Y , is added and displayed in this static plot.

2.2 Specifying the causal query

Next, the user is asked to specify the causal query, i.e., causal effect of interest. If no outcome variable has been assigned in the DAG then the input field for the causal query is left blank and a query needs to be specified. In our example, since we have assigned an exposure and outcome using the DAG interface, the total causal risk difference $P(Y(X = 1) = 1) - P(Y(X = 0) = 1)$ is suggested.

2.3 Specifying optional additional constraints

The user is also given the option to provide additional constraints besides those imposed by the DAG. This may be considered an optional advanced feature where, e.g., monotonicity of a certain direct influence of Z on X may be assumed by entering $X(Z = 1) \geq X(Z = 0)$, with any such extra constraints separated by line breaks. If this feature is used, the input is followed by clicking the button “Parse”, which identifies and fixes them.

2.4 Computing the symbolic tight bounds on the query under the given constraints

As the final step, the button “Compute the bounds” is clicked, whereupon the constraints and objective are compiled into an optimization problem which is then solved for tight causal bounds on the query symbolically in terms of observational quantities (conditional probabilities of the observed variables in the DAG) and the expressions are displayed alongside information on how the parameters are to be interpreted in terms of the given variable names (Figure 5). During computation, a progress indicator is shown, and the user should be aware that complex and/or high-dimensional problems may take significant time. The interface also provides a feature to subsequently convert the bounds to *LATEX*-code using standard probabilistic notation for publication purposes.

Once done, clicking “Exit and return objects to R” stops the `shiny` app and returns all information about the DAG, query and computed bounds to the R-session. This information is bundled in a list containing the graph, query, parameters and their interpretation, the symbolic tight bounds as expressions as well as implementations as R-functions and further log information about the formulation and optimization procedures.

3 Programmatic user interface

Interaction may also be done entirely programmatically as we illustrate with the same binary instrumental variable example. First we create the `igraph` object using the `graph_from_literal` function. Once the basic graph is created, the necessary vertex and edge attributes are added. The risk difference is defined as a character object. The `analyze_graph` function is the workhorse of `causaloptim`; it translates the causal graph, constraints, and causal effect of interest into a linear programming problem. This linear programming object, stored in `obj` in the code below, gets passed to `optimize_effect_2` which performs vertex enumeration to obtain the bounds as symbolic expressions in terms of observable probabilities. Note that when we textually enter the DAG, we also manually add the unmeasured confounding. Although the unmeasured confounder is designated as binary in the code, this is *not* assumed in the the computations where it is treated as possibly continuous, e.g.

```
graph <- igraph::graph_from_literal(Z -+ X, X -+ Y,
                                      Ur --+ X, Ur --+ Y)
V(graph)$leftside <- c(1, 0, 0, 0)
V(graph)$latent <- c(0, 0, 0, 1)
V(graph)$nvals <- c(2, 2, 2, 2)
E(graph)$rlconnect <- c(0, 0, 0, 0)
E(graph)$edge.monotone <- c(0, 0, 0, 0)

riskdiff <- "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}"
obj <- analyze_graph(graph, constraints = NULL, effectt = riskdiff)
bounds <- optimize_effect_2(obj)
bounds

#> lower bound =
#> MAX {
#>   p00_0 - p00_1 - p10_1 - p01_1,
#>   p00_0 - p00_1 - p10_0 - p10_1 - p01_0,
#>   p00_0 - p00_1 + p10_0 - 2p10_1 - 2p01_1,
#>   -p10_1 - p01_1,
#>   -p10_0 - p01_0,
#>   -p00_0 + p00_1 - 2p10_0 + p10_1 - 2p01_0,
#>   -p00_0 + p00_1 - p10_0 - p10_1 - p01_1,
#>   -p00_0 + p00_1 - p10_0 - p01_0
#> }
#> -----
#> upper bound =
#> MIN {
#>   1 - p10_1 - p01_0,
#>   1 + p00_0 + p10_0 - 2p10_1 - p01_1,
#>   2 - p00_1 - p10_0 - p10_1 - 2p01_0,
#>   1 - p10_1 - p01_1,
#>   1 - p10_0 - p01_0,
#>   1 + p00_1 - 2p10_0 + p10_1 - p01_0,
#>   2 - p00_0 - p10_0 - p10_1 - 2p01_1,
#>   1 - p10_0 - p01_1
#> }
```

The resulting `bounds` object contains character strings representing the bounds and logs containing detailed information from the vertex enumeration algorithm. The bounds are printed to the console but more features are available to facilitate their use. The `interpret_bounds` function takes the bounds and parameter names as input and returns an R function implementing vectorized forms of the symbolic expressions for the bounds.

```
bounds_function <- interpret_bounds(bounds$bounds, obj$parameters)
str(bounds_function)
```

```
#> function (p00_0 = NULL, p00_1 = NULL, p10_0 = NULL, p10_1 = NULL, p01_0 = NULL,
#>     p01_1 = NULL, p11_0 = NULL, p11_1 = NULL)
```

The results can also be used for numerical simulation using `simulate_bounds`. This function randomly generates counterfactuals and probability distributions that satisfy the constraints implied by the DAG and optional constraints. It then computes and returns the bounds as well as the true causal effect.

If one wants to bound a different effect using the same causal graph, the `update_effect` function can be used to save some computation time. It takes the object returned by `analyze_graph` and the new effect string and returns an object of class `linearcausalproblem` that can be optimized:

```
obj2 <- update_effect(obj, "p{Y(X = 1) = 1}")
```

Finally, L^AT_EX-code may also be generated using the function `latex_bounds` as in `latex_bounds(bounds$bounds, obj$parameters)` yielding

$$\begin{aligned} \text{Lower bound} = \max & \left\{ \begin{array}{l} P(X = 0, Y = 0|Z = 0) - P(X = 0, Y = 0|Z = 1) - P(X = 1, Y = 0|Z = 1) - P(X = 0, Y = 1|Z = 1), \\ P(X = 0, Y = 0|Z = 0) - P(X = 0, Y = 0|Z = 1) - P(X = 1, Y = 0|Z = 0) - P(X = 1, Y = 0|Z = 1) - P(X = 0, Y = 1|Z = 0), \\ P(X = 0, Y = 0|Z = 0) - P(X = 0, Y = 0|Z = 1) + P(X = 1, Y = 0|Z = 0) - 2P(X = 1, Y = 0|Z = 1) - 2P(X = 0, Y = 1|Z = 1), \\ -P(X = 1, Y = 0|Z = 1) - P(X = 0, Y = 1|Z = 1), \\ -P(X = 1, Y = 0|Z = 0) - P(X = 0, Y = 1|Z = 0), \\ -P(X = 0, Y = 0|Z = 0) + P(X = 0, Y = 0|Z = 1) - 2P(X = 1, Y = 0|Z = 0) + P(X = 1, Y = 0|Z = 1) - 2P(X = 0, Y = 1|Z = 0), \\ -P(X = 0, Y = 0|Z = 0) + P(X = 0, Y = 0|Z = 1) - P(X = 1, Y = 0|Z = 0) - P(X = 1, Y = 0|Z = 1) - P(X = 0, Y = 1|Z = 1), \\ -P(X = 0, Y = 0|Z = 0) + P(X = 0, Y = 0|Z = 1) - P(X = 1, Y = 0|Z = 0) - P(X = 1, Y = 0|Z = 1) - P(X = 0, Y = 1|Z = 0) \end{array} \right\} \\ \text{Upper bound} = \min & \left\{ \begin{array}{l} 1 - P(X = 1, Y = 0|Z = 1) - P(X = 0, Y = 1|Z = 0), \\ 1 + P(X = 0, Y = 0|Z = 0) + P(X = 1, Y = 0|Z = 0) - 2P(X = 1, Y = 0|Z = 1) - P(X = 0, Y = 1|Z = 1), \\ 2 - P(X = 0, Y = 0|Z = 1) - P(X = 1, Y = 0|Z = 0) - P(X = 1, Y = 0|Z = 1) - 2P(X = 0, Y = 1|Z = 0), \\ 1 - P(X = 1, Y = 0|Z = 1) - P(X = 0, Y = 1|Z = 1), \\ 1 - P(X = 1, Y = 0|Z = 0) - P(X = 0, Y = 1|Z = 0), \\ 1 + P(X = 0, Y = 0|Z = 1) - 2P(X = 1, Y = 0|Z = 0) + P(X = 1, Y = 0|Z = 1) - P(X = 0, Y = 1|Z = 0), \\ 2 - P(X = 0, Y = 0|Z = 0) - P(X = 1, Y = 0|Z = 0) - P(X = 1, Y = 0|Z = 1) - 2P(X = 0, Y = 1|Z = 1), \\ (1 - P(X = 1, Y = 0|Z = 0)) - P(X = 0, Y = 1|Z = 1) \end{array} \right\}. \end{aligned}$$

4 Implementation and program overview

An overview of the main functions and their relations is depicted as a flow chart in Figure 6. All functions may be called individually by the user in the R-console and all input, output and interaction available through the `shiny` app is available through the R-console as well.

Sachs et al. (2022) define the following class of problems for which the query in general is not identifiable, but for which a methodology to derive symbolic tight bounds on the query is provided. The causal DAG consists of a finite set $\mathcal{W} = \{W_1, \dots, W_n\} = \mathcal{W}_{\mathcal{L}} \cup \mathcal{W}_{\mathcal{R}}$ of categorical variables with $\mathcal{W}_{\mathcal{L}} \cap \mathcal{W}_{\mathcal{R}} = \emptyset$, no edges going from $\mathcal{W}_{\mathcal{R}}$ to $\mathcal{W}_{\mathcal{L}}$ and no external common parent between $\mathcal{W}_{\mathcal{L}}$ and $\mathcal{W}_{\mathcal{R}}$, but importantly external common parents $\mathcal{U}_{\mathcal{L}}$ and $\mathcal{U}_{\mathcal{R}}$ of variables within $\mathcal{W}_{\mathcal{R}}$ and $\mathcal{W}_{\mathcal{L}}$ may not be ruled out. Nothing is assumed about any characteristics of these confounding variables $\mathcal{U}_{\mathcal{L}}$ and $\mathcal{U}_{\mathcal{R}}$.

The causal query may in principle be any linear combination (although `causaloptim` version 0.9.8 implements contrasts with unit coefficients only) of joint probabilities of factual and counterfactual outcomes expressed in terms of the variables in \mathcal{W} and may always be expressed as a sum of probabilities of response function variables of the DAG. It is subject to the restriction that each outcome variable is in $\mathcal{W}_{\mathcal{R}}$ and if $\mathcal{W}_{\mathcal{L}} \neq \emptyset$ it is also subject to a few regularity conditions as detailed in Sachs et al. (2022). Tight bounds on the query may then be derived symbolically in terms of conditional probabilities of the observable variables \mathcal{W} .

Algorithms 1 and 2 in Sachs et al. (2022) construct the constraint space and causal query in terms of the joint probabilities of the response function variables and in `causaloptim` are implemented in the functions `create_R_matrix` and `create_effect_vector` respectively. Both are called as sub-procedures of the function `analyze_graph` to translate the causal problem to that of optimizing an objective function over a constraint space. The implementation of Algorithm 1 involves constructing the response functions themselves as actual R-functions. Evaluating these correspond to evaluating the structural equations of the causal DAG.

The conditions on the DAG suffice to ensure that the causal query will depend only on the response functions corresponding to the variables in $\mathcal{W}_{\mathcal{R}}$ and that the exhaustive set of constraints on their probabilities are linear in a subset of conditional probabilities of observable variables (Proposition 2 in Sachs et al. (2022)), and the conditions on the query in turn ensure that it may be expressed as a linear combination of joint probabilities of the response functions of the variables in $\mathcal{W}_{\mathcal{R}}$ (Proposition 3 in Sachs et al. (2022)). Once this formulation

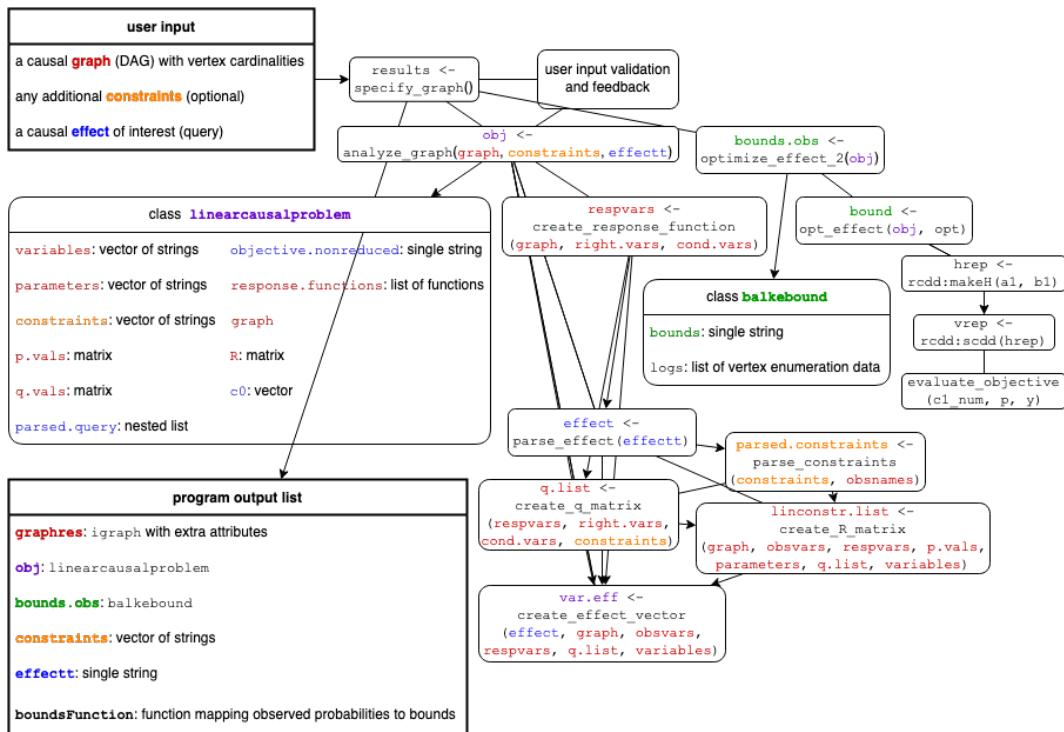


Figure 6: A function overview flow chart, listing user input, program output and overall structure. The function `specify_graph` launches the GUI. Undirected edges correspond to function calls and arrow heads to direct input. Objects are color coded according to their sources of information, with red corresponding to the DAG, blue to the query and purple to a mixture of them, as in the optimization problem of class `linearcausalproblem`. Finally, green corresponds to the subsequent optimization process producing the bounds of class `balkebound`.

of the causal problem as a linear program has been set up, a vertex enumeration method is employed to compute the extrema symbolically in terms of conditional probabilities of the observable variables.

The main and interesting functions will be described in some detail below. We begin however with an overview of how they are tied together by the `shiny` app.

specify_graph

The graphical interface is launched by `specify_graph()`, or preferably `results <- specify_graph()`. Once the `shiny` app is stopped, the input, output and other useful information is returned by the function, so we recommend assigning it to a variable so they are saved in the R-session and may easily be further analyzed and processed. All further function calls will take place automatically as the user interacts with the web interface. Thus, from a basic user perspective, `specify_graph` is the main function. The core functionality however is implemented in the functions `analyze_graph`, `optimize_effect_2` and their subroutines.

The JavaScript that handles the communication between the `shiny` server and the input as the user draws a DAG through the web interface uses on the project `directed-graph-creator`, an interactive tool for creating directed graphs, created using `d3.js` and hosted at <https://github.com/cjrd/directed-graph-creator>, which has been modified for the purpose of causal diagrams. The modification binds the user inputs as they interact with the graph to `shiny` so that the directed graph and its attributes set by the user are reactively converted into an `igraph-object` for further processing. Since directed graphs are common in many computational and statistical problems, this `shiny` interface may also be valuable to many other R-package authors and maintainers who may wish to provide their users with an accessible and intuitive way to interact with their software.

The server listens to a reactive function that, as the user draws the DAG, collects information.

mation about the current edges, collects and annotates vertices, adds left- and right-side confounding, and returns an annotated `igraph`-object, comprising information about the connectivity along with some additional attributes; for each variable, its name, cardinality, latency-indicator and side-indicator, and for each edge, a monotonicity-indicator and (to detect and communicate violations on direction) a right-to-left-indicator. The server meanwhile also monitors the DAG for any violation of the restriction that each edge between \mathcal{L} and \mathcal{R} must go from \mathcal{L} to \mathcal{R} , and if detected directly communicates this to the user through a text message in the `shiny` app. The app makes use of `shiny` modules which makes testing the interface easier, and `causaloptim` includes many automated tests to make sure that the GUI functionality works as intended.

`analyze_graph`

The function `analyze_graph` takes a DAG (in the form of an `igraph` object), optional constraints, and a string representing the causal effect of interest and proceeds to construct and return a linear optimization problem (a `linearcausalproblem`-object) from these inputs.

First, some basic data structures are created to keep track of the observed variables, their possible values, the latent variables, and whether they are in \mathcal{L} or \mathcal{R} . Once these basic data-structures have been created, the first task of the algorithm is to create the response function variables (for each variable, observed or not, except $U_{\mathcal{L}}$ and $U_{\mathcal{R}}$). Probabilities of these will be the entities \mathbf{q} in which the objective function (representing the target causal effect) is expressed and will constitute the points in the space it is optimized over, where this space itself is constrained by the relationships between them and observed conditional probabilities \mathbf{p} .

`create_response_function`

The function `create_response_function` returns a list `respvars` that has a named entry for each observed variable, containing its response function variable and response function. If X is an observed variable with n response functions, then they are enumerated by $\{0, \dots, n - 1\}$. Its entry `respvars$X` contains the response function variable R_X of X , and is a list with two entries. The first, `respvarsXindex`, is a vector containing all the possible values of R_X , i.e., the integers $(0, \dots, n - 1)$. The second, `respvarsXvalues` is itself a list with n entries; each containing the particular response function of X corresponding to its index. Each such response function is an actual R-function and may be evaluated by passing it any possible values of the parents of X as arguments.

Next, the response function variables are used in the creation of a matrix of unobserved probabilities. Specifically the joint probabilities $P(\mathbf{R}_{\mathcal{R}} = \mathbf{r}_{\mathcal{R}})$ for each possible value-combination $\mathbf{r}_{\mathcal{R}}$ of the response function variables $\mathbf{R}_{\mathcal{R}}$ of the right-side-variables $\mathbf{W}_{\mathcal{R}}$. In [Sachs et al. \(2022\)](#), the possible value-combinations $\mathbf{r}_{\mathcal{R}}$ are enumerated by $\gamma \in \{1, \dots, N_{\mathcal{R}}\}$ with corresponding probabilities $q_{\gamma} := P(\mathbf{R}_{\mathcal{R}} = \mathbf{r}_{\gamma})$ being components of the vector $\mathbf{q} \in [0, 1]^{N_{\mathcal{R}}}$.

`create_R_matrix`

The constraints that the DAG and observed conditional probabilities \mathbf{p} (in `p.vals`) impose on the unobserved probabilities \mathbf{q} (represented by `variables`) are linear. Specifically, there exists a matrix whose entries are the coefficients relating `p.vals` to `variables`. This matrix is called P in [Sachs et al. \(2022\)](#), where its existence is guaranteed by Proposition 2 and its construction is detailed in Algorithm 1, which is implemented in the function `create_R_matrix`. This function returns back a list with two entries; a vector of strings representing the linear constraints on the unobserved $\mathbf{q} \in [0, 1]^{N_{\mathcal{R}}}$ imposed by and in terms of the observed $\mathbf{p} \in [0, 1]^B$ and the numeric matrix $R \in \{0, 1\}^{(B+1) \times N_{\mathcal{R}}}$ of coefficients corresponding to these constraints as well as the probabilistic ones and given by $R = \begin{pmatrix} \mathbf{1} \\ P \end{pmatrix}$ where $P \in \{0, 1\}^{B \times N_{\mathcal{R}}} : \mathbf{p} = P\mathbf{q}$, so $R\mathbf{q} = \begin{pmatrix} \mathbf{1} \\ \mathbf{p} \end{pmatrix}$.

This determines the constraint space as a compact convex polytope in \mathbf{q} -space, i.e., in $\mathbb{R}^{|\mathcal{R}|}$. To create the matrix, we define a recursive function `gee_r` that takes two arguments; a positive integer i being the index $i \in \{1, \dots, n\}$ of a variable $W_i \in \mathcal{W}$ (i.e. the i^{th} component of \mathbf{W} or, equivalently, the i^{th} entry of `obsvars`) and a vector r being a value $\mathbf{r} \in \nu(\mathbf{R})$ in the set $\nu(\mathbf{R})$ of all possible value-vectors of the joint response function variable \mathbf{R} . This recursive function is called for each variable in `obsvars` and for each possible value of the response function variable vector. The base case is reached if the variable has no parents, in which case the list corresponding to the response function variable R_{W_i} of W_i is extracted from `respvars`. From this list, the entry whose index matches the i^{th} index of r (i.e. the one corresponding to the response function variable value $r_i = r[i]$) is extracted and finally its value, i.e., the corresponding response function itself, is extracted and is evaluated on an empty list of arguments, since it is a constant function and determined only by the value r_i .

The recursive case is encountered when `parents` is non-empty. If so, then for each parent in `parents`, its index in `obsvars` is determined and `gee_r` is recursively called with the same vector r as first argument but now with this particular index (i.e. that of the current parent) as second argument. The numeric values returned by these recursive calls are then sequentially stored in a vector `lookin`, whose entries are named by those in `parents`. Just as in the base case, the response function corresponding to the particular value r_i of the response function variable R_{W_i} (i.e. the response function of the variable `obsvars[i]` that has the index $r[i]$) is extracted from `respvars` and is now evaluated with arguments given by the list `lookin`. Note that $\text{gee}_r(r, i)$ corresponds to the value $w_i = g_{W_i}^*(\mathbf{r})$ in [Sachs et al. \(2022\)](#).

Then the values that match the observed probabilities are recorded, the corresponding entries in the current row of the matrix \mathbf{R} are set to 1 and a string representing the corresponding equation is constructed and added to the vector of constraints.

`parse_effect`

Now that the constraint space has been determined, the objective function representing the causal query needs to be specified as a linear function of the components of \mathbf{q} , i.e., variables. First, the causal query that has been provided by the user as a text-string in `effectt` is passed to the function `parse_effect`, which identifies its components including nested counterfactuals and creates a data structure representing the causal query. This structure includes nested lists which represent all interventional paths to each outcome variable in the query.

Once the nested list `effect` is returned back to `analyze_graph`, it checks that the requirements (see Proposition 3 in [Sachs et al. \(2022\)](#)) on the query are fulfilled before creating the linear objective function. Despite these regularity conditions, a large set of possible queries may be entered using standard counterfactual notation, using syntax described in the accompanying instruction text along with examples such as $P(Y(M(X=0), X=1) = 1) - P(Y(M(X=0), X=0) = 1)$; the natural direct effect ([Pearl, 2001](#)) of a binary exposure X at level $M=0$ on a binary outcome Y *not* going through the mediator M , in the presence of unmeasured confounding between M and Y ([Sjölander, 2009](#)).

`create_effect_vector`

Now that the required characteristics of the query have been established, the corresponding objective function will be constructed by the function `create_effect_vector` which returns a list `var.eff` of string-vectors; one for each term in the query. Each such vector contains the names (strings in `variables`) of the response function variables of the right-side (i.e. the components of \mathbf{q}) whose sum corresponds to the that particular term. The function `create_effect_vector` implements Algorithm 2 of [Sachs et al. \(2022\)](#) with the additional feature that if the user has entered a query that is incomplete in the sense that there are omitted mediating variables on paths from base/intervention variables to the outcome variable, then this is interpreted as the user intending the effects of the base/intervention variables to be propagated through the mediators, so that they are set to their “natural” values under this intervention. These mediators are detected and their values are set accordingly.

We define a recursive function `gee_rA` that takes three arguments; a positive integer i (the index i of a variable $W_i \in \mathcal{W} = \text{obsvars}$), a vector r (a value $\mathbf{r} \in \nu(\mathbf{R})$ in the set $\nu(\mathbf{R})$ of all possible value-vectors of the joint response function variable \mathbf{R}) and a string path that represents an interventional path and is of the form “ $X \rightarrow \dots \rightarrow Y$ ” if not `NULL`. The base case is reached either if path is `NULL` and corresponds to a path to the intervention set or if parents is empty. In the former case, the corresponding numeric intervention-value is returned, and in the latter case, the value of the corresponding response function called on the empty list of arguments is returned just as in the base case of `gee_r`. The recursive case is encountered when path is `NULL` and parents is non-empty. This recursion proceeds just as in `gee_r`, but now rather with a recursive call to `gee_rA`, whose third argument is now `path = paste(gu, "→", path)` where the string in `gu` is the name of the parent variable in parents whose index i in `obsvars` is the second argument of this recursive call. This construction traces the full path taken from the outcome of interest to the variable being intervened upon. Note that $\text{gee}_r\mathbf{A}(r, i, \text{path})$ corresponds to the value $w_i = h_{W_i}^{A_i}(\mathbf{r}, W_i)$ in [Sachs et al. \(2022\)](#). A matrix is now created just as in the observational case, but this time using `gee_rA` instead of `gee_r`.

`optimize_effect_2`

Once the constraints on \mathbf{q} as well as the effect of interest in terms of \mathbf{q} have been established, it remains only to optimize this expression over the constraint space. Here, \mathbf{c} denotes the constant gradient vector of the linear objective function and P denotes the coefficient matrix of the linear restrictions on \mathbf{q} in terms of \mathbf{p} imposed by the causal DAG. By adding the probabilistic constraints on \mathbf{q} we have arrived at e.g. the following linear program giving a tight lower bound on the average causal effect $\theta_{\mathbf{q}} = P\{Y(X=1)=1\} - P\{Y(X=0)=1\}$ in the simple instrumental variable problem of the introductory section:

$$\begin{aligned}\min_{\mathbf{q}} \theta_{\mathbf{q}} &= \min\{\mathbf{c}^\top \mathbf{q} \mid \mathbf{q} \in \mathbb{R}^{16}, \mathbf{q} \geq \mathbf{0}_{16 \times 1}, \mathbf{1}_{1 \times 16} \mathbf{q} = 1, P\mathbf{q} = \mathbf{q}\} \\ &= \max\{(1 - \mathbf{q}^\top) \mathbf{y} \mid \mathbf{y} \in \mathbb{R}^9, \mathbf{y} \geq \mathbf{0}_{9 \times 1}, (\mathbf{1}_{16 \times 1} - P^\top) \leq \mathbf{c}\} \\ &= \max\{(1 - \mathbf{q}^\top) \bar{\mathbf{y}} \mid \bar{\mathbf{y}} \text{ is a vertex of } \{\mathbf{y} \in \mathbb{R}^9 \mid \mathbf{y} \geq \mathbf{0}_{9 \times 1}, R^\top \leq \mathbf{c}\}\}\end{aligned}$$

Since we allow the user to provide additional linear inequality constraints (e.g. it may be quite reasonable to assume the proportion of “defiers” in the study population of our example to be quite low), the actual primal and dual linear programs may look slightly more complicated, but this small example still captures the essentials.

In general, given the matrix of linear constraints on the observable probabilities implied by the DAG and an optional user-provided matrix inequality, we construct the coefficient matrix and right hand side vector of the dual polytope.

The optimization via vertex enumeration step in `causaloptim` is implemented in the function `optimize_effect_2` which uses the double description method for vertex enumeration, as implemented in the `rcdd` package ([Geyer et al., 2021](#)). This step of vertex enumeration has previously been the major computational bottleneck. The approach is now based on `cddlib` (https://people.inf.ethz.ch/fukudak/cdd_home/), which has an implementation of the Double Description Method (dd). Any convex polytope can be dually described as either an intersection of half-planes (which is the form we get our dual constraint space in) or as a minimal set of vertices of which it is the convex hull (which is the form we want it in) and the dd algorithm efficiently converts between these two descriptions. `cddlib` also uses exact rational arithmetic, so there is no need to worry about any numerical instability issues. The vertices of the dual polytope are obtained and stored as rows of a matrix with `hrep <- rcdd::makeH(a1, b1); vrep <- rcdd::scdd(hrep); vertices <- vrep$output[vrep$output[, 1] == 0 & vrep$output[, 2] == 1, -c(1, 2), drop=FALSE]`.

The rest is simply a matter of plugging them into the dual objective function, evaluating the expression and presenting the results. The first part of this is done by `apply(vertices, 1, function(y) evaluate_objective(c1_num, p, y))` (here $(c1_num, p) = ((b_\ell^\top \mathbf{1}), p)$ separates the dual objective gradient into its numeric and symbolic parts).

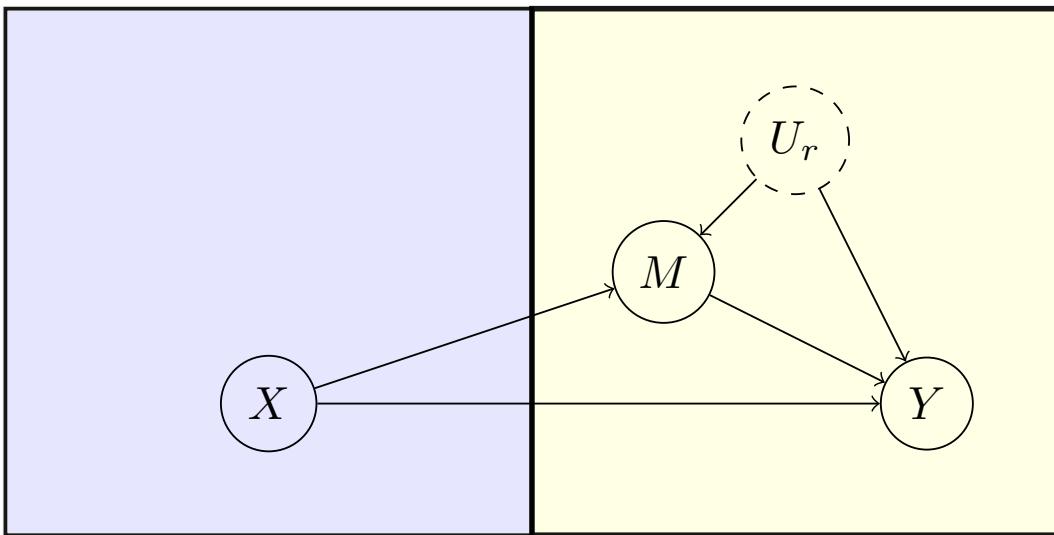


Figure 7: Causal DAG for mediation example

`causaloptim` also contains a precursor to `optimize_effect_2`, called `optimize_effect`. This legacy function uses the original optimization procedure written in C++ by Alexander Balke and involves linear program formulation followed by the vertex enumeration algorithm of Mattheiss (1973). This has worked well for very simple settings but has been found to struggle on more complex problems and thus been insufficient for the ambitions of `causaloptim`. The efficiency gains of `optimize_effect_2` over the legacy code have reduced the computation time for several setting from hours to milliseconds.

5 Numeric examples

We illustrate with a couple of applications. Although the problem formulations are entered via the textual interface below, they may of course just as well be entered via the graphical interface prior to numeric evaluation. This is done by launching the GUI and assigning its results to a variable as in `results <- specify_graph()`, following steps similar to the initial example in the section “Graphical User Interface” and clicking “Exit and return objects to R”, whereupon all information about the DAG, query and bounds are accessible via `results`. Regardless if drawn via the GUI or entered via code, the DAG may be subsequently visualized using the `plot` function.

5.1 A Mediation Analysis

In Sjölander (2009), the author derives bounds on natural direct effects in the presence of confounded intermediate variables and applies them to data from the Lipid Research Clinics Coronary Primary Prevention Trial (Freedman et al., 1992), where subjects were randomized to cholestyramine treatment and presence of coronary heart disease events as well as levels of cholesterol were recorded after a 1-year follow-up period. We let X be a binary treatment indicator, with $X = 0$ indicating actual cholestyramine treatment and $X = 1$ indicating placebo. We further let Y be an indicator of the occurrence of coronary heart disease events within follow-up, with $Y = 0$ indicating event-free follow-up and $Y = 1$ indicating an event. We finally let M be a dichotomized (cut-off at 280 mg/dl) cholesterol level indicator, with $M = 0$ indicating levels $< 280 \text{ mg/dl}$ and $M = 1$ indicating levels $\geq 280 \text{ mg/dl}$. The causal assumptions are summarized in the DAG shown in Figure 7, where U_r is unmeasured and confounds the effect of M on Y .

```

b <- igraph::graph_from_literal(X -+ Y, X -+ M, M -+ Y,
                                 Ur -+ Y, Ur -+ M)
V(b)$leftside <- c(1, 0, 0, 0)
  
```

```
V(b)$latent    <- c(0, 0, 0, 1)
V(b)$nvals      <- c(2, 2, 2, 2)
E(b)$rlconnect   <- c(0, 0, 0, 0, 0)
E(b)$edge.monotone <- c(0, 0, 0, 0, 0)
```

Note that although the unmeasured confounder Ur appears to be designated as binary in this code, this is *not* assumed in the the computations, where nothing is assumed about its distribution.

Using the data from Table IV of Sjölander (2009), we compute the observed conditional probabilities.

```
# parameters of the form pab_c, which represents
# the probability P(Y = a, M = b | X = c)
p00_0 <- 1426/1888 # P(Y=0,M=0|X=0)
p10_0 <- 97/1888 # P(Y=1,M=0|X=0)
p01_0 <- 332/1888 # P(Y=0,M=1|X=0)
p11_0 <- 33/1888 # P(Y=1,M=1|X=0)
p00_1 <- 1081/1918 # P(Y=0,M=0|X=1)
p10_1 <- 86/1918 # P(Y=1,M=0|X=1)
p01_1 <- 669/1918 # P(Y=0,M=1|X=1)
p11_1 <- 82/1918 # P(Y=1,M=1|X=1)
```

We proceed to compute bounds on the controlled direct effect $CDE(0) = P(Y(M = 0, X = 1) = 1) - P(Y(M = 0, X = 0) = 1)$ of X on Y not passing through M at level $M = 0$, the controlled direct effect $CDE(1) = P(Y(M = 1, X = 1) = 1) - P(Y(M = 1, X = 0) = 1)$ at level $M = 1$, the natural direct effect $NDE(0) = P(Y(M(X = 0), X = 1) = 1) - P(Y(M(X = 0), X = 0) = 1)$ of X on Y at level $X = 0$ and the natural direct effect $NDE(1) = P(Y(M(X = 1), X = 1) = 1) - P(Y(M(X = 1), X = 0) = 1)$ at level $X = 1$.

```
CDE0_query <- "p{Y(M = 0, X = 1) = 1} - p{Y(M = 0, X = 0) = 1}"
CDE0_obj <- analyze_graph(b, constraints = NULL, effectt = CDE0_query)
CDE0_bounds <- optimize_effect_2(CDE0_obj)
CDE0_boundsfunction <- interpret_bounds(bounds = CDE0_bounds$bounds,
                                             parameters = CDE0_obj$parameters)
CDE0_numericbounds <- CDE0_boundsfunction(p00_0 = p00_0, p00_1 = p00_1,
                                             p10_0 = p10_0, p10_1 = p10_1,
                                             p01_0 = p01_0, p01_1 = p01_1,
                                             p11_0 = p11_0, p11_1 = p11_1)

CDE1_query <- "p{Y(M = 1, X = 1) = 1} - p{Y(M = 1, X = 0) = 1}"
CDE1_obj <- update_effect(CDE0_obj, effectt = CDE1_query)
CDE1_bounds <- optimize_effect_2(CDE1_obj)
CDE1_boundsfunction <- interpret_bounds(bounds = CDE1_bounds$bounds,
                                             parameters = CDE1_obj$parameters)
CDE1_numericbounds <- CDE1_boundsfunction(p00_0 = p00_0, p00_1 = p00_1,
                                             p10_0 = p10_0, p10_1 = p10_1,
                                             p01_0 = p01_0, p01_1 = p01_1,
                                             p11_0 = p11_0, p11_1 = p11_1)

NDE0_query <- "p{Y(M(X = 0), X = 1) = 1} - p{Y(M(X = 0), X = 0) = 1}"
NDE0_obj <- update_effect(CDE0_obj, effectt = NDE0_query)
NDE0_bounds <- optimize_effect_2(NDE0_obj)
NDE0_boundsfunction <- interpret_bounds(bounds = NDE0_bounds$bounds,
                                             parameters = NDE0_obj$parameters)
NDE0_numericbounds <- NDE0_boundsfunction(p00_0 = p00_0, p00_1 = p00_1,
                                             p10_0 = p10_0, p10_1 = p10_1,
                                             p01_0 = p01_0, p01_1 = p01_1,
                                             p11_0 = p11_0, p11_1 = p11_1)

NDE1_query <- "p{Y(M(X = 1), X = 1) = 1} - p{Y(M(X = 1), X = 0) = 1}"
```

Table 1: Bounds on the controlled and natural direct effects.

	lower	upper
CDE(0)	-0.20	0.39
CDE(1)	-0.78	0.63
NDE(0)	-0.07	0.56
NDE(1)	-0.55	0.09

```

NDE1_obj <- update_effect(CDE0_obj, effectt = NDE1_query)
NDE1_bounds <- optimize_effect_2(NDE1_obj)
NDE1_boundsfunction <- interpret_bounds(bounds = NDE1_bounds$bounds,
                                             parameters = NDE1_obj$parameters)
NDE1_numericbounds <- NDE1_boundsfunction(p00_0 = p00_0, p00_1 = p00_1,
                                             p10_0 = p10_0, p10_1 = p10_1,
                                             p01_0 = p01_0, p01_1 = p01_1,
                                             p11_0 = p11_0, p11_1 = p11_1)

```

We obtain the same symbolic bounds as [Sjölander \(2009\)](#) and the resulting numeric bounds are given in Table 1 which of course agree with those of Table V in [Sjölander \(2009\)](#).

5.2 A Mendelian Randomization Study of the Effect of Homocysteine on Cardiovascular Disease

Mendelian randomization ([Davey Smith and Ebrahim, 2003](#)) assumes certain genotypes may serve as suitable instrumental variables for investigating the causal effect of an associated phenotype on some disease outcome.

In [Palmer \(2011\)](#), the authors investigate the effect of homocysteine on cardiovascular disease using the 677CT polymorphism (rs1801133) in the Methylenetetrahydrofolate Reductase gene as an instrument. They use observational data from [Meleady et al. \(2003\)](#) in which the outcome is binary, the treatment has been made binary by a suitably chosen cut-off at $15\mu\text{mol/L}$, and the instrument is ternary (this polymorphism can take three possible genotype values).

With X denoting the treatment, Y the outcome and Z the instrument, the DAG is the one in Figure 2 (although now with a ternary instrument) and the conditional probabilities are given as follows.

```

params <- list(p00_0 = 0.83, p00_1 = 0.88, p00_2 = 0.72,
                p10_0 = 0.11, p10_1 = 0.05, p10_2 = 0.20,
                p01_0 = 0.05, p01_1 = 0.06, p01_2 = 0.05,
                p11_0 = 0.01, p11_1 = 0.01, p11_2 = 0.03)

```

The computation using [causaloptim](#) is done using the following code.

```

# Input causal DAG
b <- graph_from_literal(Z -> X, X -> Y, Ur -> X, Ur -> Y)
V(b)$leftside <- c(1, 0, 0, 0)
V(b)$latent <- c(0, 0, 0, 1)
V(b)$nvals <- c(3, 2, 2, 2)
E(b)$rlconnect <- c(0, 0, 0, 0)
E(b)$edge.monotone <- c(0, 0, 0, 0)
# Construct causal problem
obj <- analyze_graph(b, constraints = NULL,
                      effectt = "p{Y(X = 1) = 1} - p{Y(X = 0) = 1}")
# Compute bounds on query
bounds <- optimize_effect_2(obj)
# Construct bounds as function of parameters
boundsfunction <- interpret_bounds(bounds = bounds$bounds,

```

```

parameters = obj$parameters)
# Insert observed conditional probabilities
numericbounds <- do.call(boundsfunction, as.list(params))
round(numericbounds, 2)

#>   lower upper
#> 1 -0.09  0.74

```

Our computed bounds agree with those computed using `bpbounds` as well as those estimated using Theorem 2 of Richardson and Robins (2014), who independently derived expressions for tight bounds that are applicable to this setting.

6 Summary and discussion

The methods and algorithms described in Sachs et al. (2022) to compute symbolic expressions for bounds on non-identifiable causal effects are implemented in the package `causaloptim`. Our aim was to provide a user-friendly interface to these methods with a graphical interface to draw DAGs, specify causal effects with commonly used notation for counterfactuals, and take advantage of an efficient implementation of vertex enumeration to reduce computation times. These methods are applicable to a wide variety of causal inference problems which appear in biomedical research, economics, social sciences and more. Aside from the graphical interface, programming with the package is encouraged to promote reproducibility and advanced use. Our package includes automated unit tests and also tests for correctness by comparing the symbolic bounds derived using our program to independently derived bounds in particular settings.

Our implementation uses a novel approach to draw DAGs using JavaScript in a web browser that can then be passed to R using `shiny`. This graphical approach can be adapted and used in other settings where graphs need to be specified and computed on, such as other causal inference settings, networks, and multi-state models. Other algorithms and data structures that could be more broadly useful include the representation of structural equations as R functions, recursive evaluation of response functions, and parsing of string equations for causal effects and constraints.

Note that `causaloptim` computes bounds only in settings with tightness guarantees. Although the set of available causal queries is quite large, the set of admissible DAGs can appear restrictive. The usefulness of the methodology and implementation extends to a much larger class of problems however. Tight bounds for restricted settings can be used to derive potentially not tight but narrow valid bounds in less restrictive settings. In Cai et al. (2007) and Cui and Tchetgen (2021) the authors make use of the tight Balke-Pearl bounds to derive narrow (but not necessarily tight) valid bounds on causal effects subject to causal DAGs that fall outside the scope for tightness guarantees using this methodology. We believe the tight bounds computed by `causaloptim` hold great potential for such applications.

References

- A. Balke and J. Pearl. Bounds on treatment effects from studies with imperfect compliance. *Journal of the American Statistical Association*, 92(439):1171–1176, 1997. [p52]
- Z. Cai, M. Kuroki, and T. Sato. Non-parametric bounds on treatment effects with non-compliance by covariate adjustment. *Statistics in Medicine*, 26(16):3188–3204, 2007. doi: <https://doi.org/10.1002/sim.2766>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.2766>. [p65]
- Z. Cai, M. Kuroki, J. Pearl, and J. Tian. Bounds on direct effects in the presence of confounded intermediate variables. *Biometrics*, 64(3):695–701, 2008. [p52]
- W. Chang, J. Cheng, J. Allaire, C. Sievert, B. Schloerke, Y. Xie, J. Allen, J. McPherson, A. Dipert, and B. Borges. *shiny: Web Application Framework for R*, 2022. URL <https://CRAN.R-project.org/package=shiny>. R package version 1.7.4. [p53]

- G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*:1695, 2006. URL <https://igraph.org>. [p55]
- Y. Cui and E. T. Tchetgen. A semiparametric instrumental variable approach to optimal treatment regimes under endogeneity. *Journal of the American Statistical Association*, 116(533):162–173, 2021. doi: 10.1080/01621459.2020.1783272. URL <https://doi.org/10.1080/01621459.2020.1783272>. PMID: 33994604. [p65]
- G. Davey Smith and S. Ebrahim. ‘Mendelian randomization’: can genetic epidemiology contribute to understanding environmental determinants of disease? *International Journal of Epidemiology*, 32(1):1–22, 02 2003. ISSN 0300-5771. doi: 10.1093/ije/dyg070. URL <https://doi.org/10.1093/ije/dyg070>. [p64]
- L. S. Freedman, B. I. Graubard, and A. Schatzkin. Statistical validation of intermediate endpoints for chronic diseases. *Statistics in Medicine*, 11(2):167–178, 1992. doi: <https://doi.org/10.1002/sim.4780110204>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.4780110204>. [p62]
- E. E. Gabriel, M. C. Sachs, and A. Sjölander. Causal bounds for outcome-dependent sampling in observational studies. *Journal of the American Statistical Association*, 117(538):939–950, 2022a. [p52]
- E. E. Gabriel, M. C. Sachs, and A. Sjölander. Sharp nonparametric bounds for decomposition effects with two binary mediators. *Journal of the American Statistical Association*, page In press, 2022b. [p52]
- E. E. Gabriel, A. Sjölander, and M. C. Sachs. Nonparametric bounds for causal effects in imperfect randomized experiments. *Journal of the American Statistical Association*, 118(541):684–692, 2023. [p52]
- C. J. Geyer, G. D. Meeden, and incorporates code from cddlib written by Komei Fukuda. *rcdd: Computational Geometry*, 2021. URL <https://CRAN.R-project.org/package=rcdd>. R package version 1.5. [p61]
- S. Greenland, J. Pearl, and J. M. Robins. Causal diagrams for epidemiologic research. *Epidemiology*, pages 37–48, 1999. [p52]
- T. H. Mattheiss. An algorithm for determining irrelevant constraints and all vertices in systems of linear inequalities. *Operations Research*, 21(1):247–260, 1973. [p52, 62]
- R. Meleady, P. M. Ueland, H. Blom, A. S. Whitehead, H. Refsum, L. E. Daly, S. E. Vollset, C. Donohue, B. Giesendorf, I. M. Graham, A. Ulvik, Y. Zhang, and A.-L. Bjørke Monsen. Thermolabile methylenetetrahydrofolate reductase, homocysteine, and cardiovascular disease risk: the European Concerted Action Project. *The American journal of clinical nutrition*, 77(1):63–70, Jan. 2003. ISSN 0002-9165. doi: 10.1093/ajcn/77.1.63. Place: United States. [p64]
- T. Palmer, R. Ramsahai, V. Didelez, and N. Sheehan. *bpbounds: R package implementing Balke-Pearl bounds for the average causal effect*, 2018. URL <https://github.com/remlapmot/bpbounds>. [p53]
- T. M. Palmer. Nonparametric bounds for the causal effect in a binary instrumental-variable model. *Stata Journal*, 11(3):345–367(23), 2011. URL <https://journals.sagepub.com/doi/pdf/10.1177/1536867X1101100302>. [p64]
- J. Pearl. Direct and indirect effects. In *Proceedings of the Seventeenth Conference on Uncertainty and Artificial Intelligence*, 2001, pages 411–420. Morgan Kaufman, 2001. [p60]
- J. Pearl. *Causality*. Cambridge university press, 2009. [p52]
- T. S. Richardson and J. M. Robins. Ace bounds; sems with equilibrium conditions. *Statistical Science*, 29(3):363–366, 2014. [p65]

- M. C. Sachs, G. Jonzon, A. Sjölander, and E. E. Gabriel. A general method for deriving tight symbolic bounds on causal effects. *Journal of Computational and Graphical Statistics*, 0(0):1–10, 2022. doi: 10.1080/10618600.2022.2071905. URL <https://doi.org/10.1080/10618600.2022.2071905>. [p52, 53, 57, 59, 60, 61, 65]
- M. C. Sachs, G. Jonzon, A. Sjölander, and E. E. Gabriel. *causaloptim: An Interface to Specify Causal Graphs and Compute Bounds on Causal Effects*, 2023. URL <https://github.com/sachsmc/causaloptim>. R package version 0.9.8. [p52]
- A. Sjölander. Bounds on natural direct effects in the presence of confounded intermediate variables. *Statistics in Medicine*, 28(4):558–571, 2009. [p52, 60]
- A. Sjölander, W. Lee, H. Källberg, and Y. Pawitan. Bounds on causal interactions for binary outcomes. *Biometrics*, 70(3):500–505, 2014. [p52]
- A. Sjölander. Bounds on natural direct effects in the presence of confounded intermediate variables. *Statistics in Medicine*, 28(4):558–571, 2009. doi: <https://doi.org/10.1002/sim.3493>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.3493>. [p62, 63, 64]

Gustav Jonzon

Department of Medical Epidemiology and Biostatistics, Karolinska Institutet, Sweden

<https://ki.se/meb>
gustav.jonzon@ki.se

Michael C Sachs

Department of Public Health, University of Copenhagen, Denmark

Supported in part by Novo Nordisk Fonden Grant NNF22OC0076595

<https://biostat.ku.dk/>
ORCID: 0000-0002-1279-8676
michael.sachs@sund.ku.dk

Erin E Gabriel

Department of Public Health, University of Copenhagen, Denmark

Supported in part by Novo Nordisk Fonden Grant NNF22OC0076595

<https://biostat.ku.dk/>
ORCID: 0000-0002-0504-8404
erin.gabriel@sund.ku.dk

singR: An R Package for Simultaneous Non-Gaussian Component Analysis for Data Integration

by Liangkang Wang, Irina Gaynanova, and Benjamin Risk

Abstract This paper introduces an R package singR that implements Simultaneous non-Gaussian Component Analysis (SING) for data integration. SING uses a non-Gaussian measure of information to extract feature loadings and scores (latent variables) that are shared across multiple datasets. We describe the functions implemented in singR and showcase their use on two examples. The first example is a toy example working with images. The second example is a simulated study integrating functional connectivity estimates from a resting-state functional magnetic resonance imaging dataset and task activation maps from a working memory functional magnetic resonance imaging dataset. The SING model can produce joint components that accurately reflect information shared by multiple datasets, particularly for datasets with non-Gaussian features such as neuroimaging.

1 Introduction

Combining information across different datasets collected on the same individuals is an important task, especially in biology and medicine. For example, in neuroimaging research, combining information across different modalities, or types of imaging data, can lead to a more comprehensive picture of human brain activity. Commonly used modalities to investigate brain function include functional magnetic resonance imaging (fMRI), resting-state fMRI (rs-fMRI), diffusion MRI, structural images, electroencephalography, and positron emission tomography. Combining data from multiple modalities can result in a better understanding of the underlying biology than analyzing each modality separately (Calhoun and Sui 2016). In previous studies, researchers use data fusion to define a set of joint components that are composed of subject scores (a vector in \mathbb{R}^n , where n is the number of subjects) and loadings (a vector in \mathbb{R}^{p_k} , where p_k is the number of variables in the k th dataset). For a particular component, the subject scores are equal or strongly correlated across datasets. The loadings represent the relative importance of each variable to each component. A higher subject score implies the vector of loadings is more important in that individual.

Data integration approaches for neuroimaging should accommodate the distinct statistical properties of imaging data. Imaging features characterizing brain activation and intrinsic functional connectivity are substantially more non-Gaussian compared to noise. Methods employing principal component analysis (PCA) for dimension reduction and independent component analysis (ICA) have been widely used in neuroscience (Calhoun and Sui 2016; J. Sui et al. 2012; Zhou et al. 2016). ICA maximizes the non-Gaussianity of components, which is useful for extracting interesting features from imaging data. ICA is commonly used to estimate resting-state networks (Beckmann et al. 2005), estimate task-activated components (Jing Sui et al. 2010), and has been used to derive network structure in resting-state correlation matrices (Amico et al. 2017). Risk and Gaynanova (2021) proposed simultaneous non-Gaussian component analysis (SING) for analyzing two datasets, which uses an objective function that maximizes the skewness and kurtosis of latent components with a penalty to enhance the similarity between subject scores. Unlike previous methods, SING does not use PCA for dimension reduction, but rather uses non-Gaussianity measured by skewness and kurtosis, which can improve feature extraction.

Some useful software have been developed in this area. In multimodal analysis and multitask fMRI data fusion, **FIT** is a Matlab toolbox that implements jointICA, parallel ICA (Vergara et al. 2014), and multimodal/multiset CCA (J. Sui et al. 2011). **GIFT** provides functions for conducting group ICA on fMRI data from multiple subjects from a single modality (Calhoun, Liu, and Adali 2009; Calhoun, Adali, Giuliani, et al. 2006; Calhoun, Adali, Kiehl, et al. 2006). On the Comprehensive R Archive Network (CRAN), there are several R packages for ICA functions, including **steadyICA** (Risk, James, and Matteson 2015), **ica** (Helwig 2018), **fastICA** (Marchini, Heaton, and Ripley 2021), **JADE** (Miettinen, Nordhausen, and Taskinen 2017), and **templateICAr** (Mejia et al. 2020). These R packages use different algorithms to extract non-Gaussian features but are designed for decomposing a single modality. For data integration, **r.jive** (O'Connell and Lock 2020) and **ajive** (Carmichael 2022; Feng et al. 2018) capture the joint variation, or variance shared between datasets, and individual variation, or variance unique to a dataset. JIVE methods use singular value decompositions, which are related to maximizing variance instead of non-Gaussianity. In this way, there exists a need for freely available software for extracting joint structure from multiple datasets using non-Gaussian measures of information.

This paper introduces **singR**, an R package to implement Risk and Gaynanova (2021). This paper is structured as follows. In Section 2, we review the Simultaneous non-Gaussian component analysis (SING) model. In Sections 3 and 4, we present the main functions in the **singR** package and show how to utilize it for joint components estimation in two example datasets. Finally, Section 5 summarizes our conclusions.

2 Methods

2.1 Linear non-Gaussian Component Analysis

Matrix decomposition for one dataset. Based on linear non-Gaussian component analysis (LNGCA), we first summarize a matrix decomposition for a single dataset $X \in \mathbb{R}^{n \times p_x}$ (n subjects and p_x features) into a non-Gaussian subspace and a Gaussian subspace. Each row of X is a vector of features from the i th subject. Let X_c denote the double-centered data matrix such that $1^T X_c = 0^T$ and $X_c 1 = 0$ where 1 denotes the vector of ones of appropriate dimension, which has rank $n - 1$ when $p_x > n$. Let I_{r_x} denote the $r_x \times r_x$ identity matrix. Then define the matrix decomposition

$$X_c = M_x S_x + M_{N_x} N_x. \quad (1)$$

Here, $M_x \in \mathbb{R}^{n \times r_x}$, and the columns of M_x are called subject scores. $M_{N_x} \in \mathbb{R}^{n \times (n-r_x-1)}$, and the matrix $[M_x, M_{N_x}]$ is called the mixing matrix and has rank $n - 1$. $S_x \in \mathbb{R}^{r_x \times p_x}$ and $N_x \in \mathbb{R}^{(n-r_x-1) \times p_x}$. $S_x S_x^T = p_x I_{r_x}$, $N_x N_x^T = 0_{(n-r_x-1) \times r_x}$. The rows of S_x are the non-Gaussian components, and elements of S_x are called variable loadings because $\frac{1}{p_x} X_c S_x^T = M_x$. The rows of N_x are the Gaussian components. The rows of S_x have the largest non-Gaussianity, as described below.

This decomposition may be meaningful in neuroimaging studies because: 1) vectorized components like brain activation maps and resting-state networks have highly non-Gaussian distributions, and 2) it is often the situation that $p_x \gg n$, i.e., the number subjects is smaller than the number of voxels or edges.

To achieve the matrix decomposition in (1), we need to find a separating matrix A_x that maximizes non-Gaussianity and satisfies the constraint $A_x X_c X_c^T A_x^T = S_x S_x^T = p_x I_{r_x}$. We utilize a prewhitening matrix and then reparameterize the model with a semiorthogonal separating matrix to enforce this constraint. Let $\widehat{\Sigma}_x = X_c X_c^\top / p_x$. Then define the eigenvalue decomposition $\widehat{\Sigma}_x = V_x \Lambda_x V_x^\top$ and the prewhitening matrix $\widehat{L}_x = V_x \Lambda_x^{-1/2} V_x^\top$ (for double-centered X_c , this is understood to be the square root of the generalized inverse from the non-zero eigenvalues). Note $A_x = U_x \widehat{L}_x$, and it follows that $\widehat{M}_x = \widehat{L}_x^{-1} U_x^\top$, with \widehat{L}_x^{-1} denoting the generalized inverse. Let $f()$ be a measure of non-Gaussianity. Then (1) is estimated using

$$\begin{aligned} \underset{U_x}{\text{minimize}} \quad & - \sum_{l=1}^{r_x} f(u_{xl}^\top \widehat{L}_x X_c), \\ \text{subject to} \quad & U_x U_x^\top = I_{r_x}, \end{aligned} \quad (2)$$

where u_{xl}^\top is the l th row of the $r_x \times n$ matrix U_x .

We measure non-Gaussianity with the **Jarque-Bera (JB) statistic**, a combination of squared skewness and kurtosis. For a vector $s \in \mathbb{R}^p$, the JB statistic is

$$f(s) = 0.8 \left(\frac{1}{p} \sum_j s_j^3 \right)^2 + 0.2 \left(\frac{1}{p} \sum_j s_j^4 - 3 \right)^2. \quad (3)$$

2.2 Simultaneous non-Gaussian component analysis model

Matrix decomposition for two datasets. We now decompose $X \in \mathbb{R}^{n \times p_x}$ and $Y \in \mathbb{R}^{n \times p_y}$ into a joint non-Gaussian subspace defined by shared subject score directions, individual non-Gaussian subspaces, and Gaussian subspaces. Let r_j denote the rank of the joint non-Gaussian subspace and r_x, r_y denote the rank of the non-Gaussian subspaces for X and Y , respectively. Define the double-centered X_c and Y_c , i.e., $1^T X_c = 0^T$ and $X_c 1 = 0$. In data applications, we also recommend standardizing each feature to have unit variance, as is common in PCA. The double centering with standardization requires an iterative algorithm that standardizes each feature (mean 0 variance 1 across subjects), then centers the

features for a given subject (the mean of the features for a subject is 0), and repeats (typically < 10 iterations on real data suffices). The function `standard` is described in Section [3].

The SING matrix decomposition is

$$\begin{aligned} X_c &= M_J D_x S_{Jx} + M_{Ix} S_{Ix} + M_{Nx} N_x, \\ Y_c &= M_J D_y S_{Jy} + M_{Iy} S_{Iy} + M_{Ny} N_y. \end{aligned} \quad (4)$$

Here, $M_J \in \mathbb{R}^{n \times r_J}$, $M_{Ix} \in \mathbb{R}^{n \times (r_x - r_J)}$, $M_{Iy} \in \mathbb{R}^{n \times (r_y - r_J)}$, $M_{Nx} \in \mathbb{R}^{n \times (n - r_x - 1)}$, and $M_{Ny} \in \mathbb{R}^{n \times (n - r_y - 1)}$. D_x and D_y are diagonal and allow the scaling of M_J to vary between datasets. S_{Jx} are the joint non-Gaussian components, S_{Ix} are the individual non-Gaussian components, and N_x are the individual Gaussian components, respectively, for X , with constraints $S_{Jx} S_{Jx}^T = p_x I_{r_J}$, $S_{Ix} S_{Ix}^T = p_x I_{r_x - r_J}$, $S_{Jx} S_{Ix}^T = 0_{r_J \times (r_x - r_J)}$, $N_x S_{Jx}^T = 0_{(n - r_x - 1) \times r_J}$, $N_x S_{Ix}^T = 0_{(n - r_x - 1) \times (r_x - r_J)}$, and similarly define the components of Y .

Simultaneous non-Gaussian Component Analysis fitting and algorithm. Recall the whitening matrix for X_c is \hat{L}_x , and define its generalized inverse $\hat{L}_x^- = (X_c X_c^T / p_x)^{1/2} = V_x \Lambda_x^{1/2} V_x^T$. We will estimate a semiorthogonal unmixing matrix \hat{U}_x such that $\hat{M}_x = \hat{L}_x^- \hat{U}_x^T$. Similarly define the whitening matrix \hat{L}_y and \hat{M}_y for Y_c . Let f be the JB statistic, as defined in (3). We consider

$$\begin{aligned} \text{minimize}_{U_x, U_y} \quad & - \sum_{l=1}^{r_x} f(u_{xl}^\top \hat{L}_x X_c) - \sum_{l=1}^{r_y} f(u_{yl}^\top \hat{L}_y Y_c) + \rho \sum_{l=1}^{r_J} d(\hat{L}_x^- u_{xl}, \hat{L}_y^- u_{yl}) \\ \text{subject to} \quad & U_x U_x^\top = I_{r_x}, \quad U_y U_y^\top = I_{r_y}, \end{aligned} \quad (5)$$

where $d(x, y)$ is the chosen distance metric between vectors x and y , calculated using the chordal distance: $d(x, y) = \left\| \frac{xx^\top}{\|x\|_2^2} - \frac{yy^\top}{\|y\|_2^2} \right\|_F^2$. When columns are mean zero, the chordal distance between joint scores in the SING objective function is equal to zero when their correlation is equal to one. Larger values of the tuning parameter ρ result in common M_J , but smaller values result in highly correlated joint structure and could also be considered. In our examples, we match components from the separate LNGCA, determine candidate joint components from a permutation test, and then set ρ equal to the sum of the JB statistics of all candidate joint loadings divided by 10, which results in $\hat{L}_x^- u_{xl} \approx \hat{L}_y^- u_{yl}$, i.e., a shared M_J .

Let \hat{U}_x and \hat{U}_y be the estimated value of U_x and U_y in (5). The corresponding estimated non-Gaussian components are defined as $\hat{S}_x = \hat{U}_x X_w$ and $\hat{S}_y = \hat{U}_y Y_w$, where $X_w = \hat{L}_x X_c$, and $Y_w = \hat{L}_y Y_c$, respectively. Then the first r_J columns of $\hat{M}_x = \hat{L}_x^- \hat{U}_x^T$, scaled to unit norm, define \hat{M}_{Jx} . We can similarly define \hat{M}_{Jy} . For sufficiently large ρ , $\hat{M}_{Jx} = \hat{M}_{Jy} = \hat{M}_J$, and more generally, \hat{M}_J is defined from their average. Additionally, the first r_J rows of \hat{S}_x correspond to \hat{S}_{Jx} .

3 Overview of functions

The R package `singR` implements simultaneous non-Gaussian component analysis for data integration in neuroimaging. Highlighted below are key functions:

Ingca: This function estimates non-Gaussian components for a single dataset. Non-Gaussian components can be estimated using the Jarque-Bera test statistic, which is the non-Gaussian measure used in SING, or using likelihood component analysis, which achieves dimension reduction while estimating the densities of non-Gaussian components (Risk, Matteson, and Ruppert 2019). It returns \hat{U}_x and \hat{S}_x from decomposing X_c through (2). It also returns the non-Gaussianity of each estimated component.

standard: This function is an iterative algorithm that standardizes each feature (mean 0 variance 1 across subjects), then centers the features for a given subject (the mean of the features for a subject is 0) for the original datasets ($\mathbb{R}^{n \times p}$), and repeats until the variance is approximately equal to 1 (typically < 10 iterations on real data suffices).

est.M.ols: This function returns \hat{M}_x with input \hat{S}_x and X_c .

greedymatch: This function reorders the columns in \hat{U}_x to match the columns (subject scores) in \hat{U}_y based on the chordal distances between corresponding \hat{M}_x and \hat{M}_y .

permTestJointRank: This function tests whether the correlation between matched columns (subject scores) is significant and returns the family wise error rate corrected p-values.

%^%: Calculates the matrix exponential. For example, `A%^%0.5` returns a matrix square root. Used during prewhitening.

calculateJB: This function calculates the sum of the JB statistics across components and is useful for determining the size of the penalty parameter ρ (sufficiently large ρ results in the chordal distance between M_{Jx} and M_{Jy} equal to 0). Assumes the variance of each row of the input S is equal to 1 and mean of each row is equal to 0.

curvilinear: This function gives the final estimates of \hat{U}_x and \hat{U}_y using the curvilinear algorithm derived from (5). This is a pure R implementation but is slow.

curvilinear_c: This implements the curvilinear algorithm in C++, which is faster.

NG_number: This is a wrapper function for `FOBIasymp` from `ICtest` (Nordhausen et al. 2022) that can be used to estimate the number of non-Gaussian components in a single dataset.

signchange: This function makes the skewness of each row of \hat{S}_x positive, which is useful for visualizing non-Gaussian component loadings.

singR: This function integrates all the functions above. We can use this function to estimate joint scores and loadings from two datasets X and Y and optionally return the individual scores and loadings.

4 Examples

To illustrate the use of `singR`, we provide two examples.

4.1 Example 1. The toy datasets decomposition

The tutorial dataset `exampledadata` are included in the `singR` package. We generate the SING model in (4) as follows. We generate joint subject scores $M_J = [m_{J1}, m_{J2}] \in \mathbb{R}^{n \times 2}$ with $m_{J1} \sim N(\mu_1, I_n)$, $m_{J2} \sim N(\mu_2, I_n)$, $\mu_1 = (1_{24}^\top, -1_{24}^\top)^\top$ and $\mu_2 = (-1_{24}^\top, 1_{24}^\top)^\top$. We set $D_x = I$ and $D_y = diag(-5, 2)$ to have differences in both sign and scale between the two datasets. We generate M_{Ix} and M_{Iy} similar to M_J using iid unit variance Gaussian entries with means equal to $\mu_{3y} = (-1_6^\top, 1_6^\top, -1_6^\top, 1_6^\top, -1_6^\top, 1_6^\top - 1_6^\top, -1_6^\top)^\top$, $\mu_{4y} = (1_{24}^\top, -1_{24}^\top)^\top$, $\mu_{3x} = (-1_{12}^\top, 1_{12}^\top, -1_{12}^\top, 1_{12}^\top)^\top$ and $\mu_{4x} = (1_{12}^\top, -1_{12}^\top, 1_{12}^\top, -1_{12}^\top)^\top$. These means result in various degrees of correlation between the columns of the mixing matrices. For the Gaussian noise, we generate M_{Nx} , M_{Ny} , N_x and N_y using iid standard Gaussian mean zero entries.

Each row of S_{Jx} and S_{Ix} is a vectorized image. We can reshape the loadings back to their image dimensions for visualization. The loadings S_{Jx} are inspired by activation patterns found in functional MRI, and similar simulations were considered in (Risk, Matteson, and Ruppert 2019). The rows of S_{Jy} and S_{Iy} are formed from the lower diagonal of a symmetric matrix, which are inspired by ICA of correlation matrices (Amico et al. 2017), and we can visualize the loadings by reshaping the vectors back to the symmetric matrix. The true loadings of latent non-Gaussian components are plotted in figure 1.

```
library(singR)
data(exampledadata)
data <- exempledata

lgrid = 33
par(mfrow = c(2, 4))
# Components for X
image(matrix(data$sjX[1, ], lgrid, lgrid), col = heat.colors(12),
      xaxt = "n", yaxt = "n", main = expression("True S"["Jx"] * ",1"))
image(matrix(data$sjX[2, ], lgrid, lgrid), col = heat.colors(12),
      xaxt = "n", yaxt = "n", main = expression("True S"["Jx"] * ", 2"))
image(matrix(data$siX[1, ], lgrid, lgrid), col = heat.colors(12),
      xaxt = "n", yaxt = "n", main = expression("True S"["Ix"] * ", 1"))
image(matrix(data$siX[2, ], lgrid, lgrid), col = heat.colors(12),
      xaxt = "n", yaxt = "n", main = expression("True S"["Ix"] * ", 2"))

# Components for Y
image(vec2net(data$sjY[1, ]), col = heat.colors(12), xaxt = "n", yaxt = "n",
      main = expression("True S"["Jy"] * ", 1"))
image(vec2net(data$sjY[2, ]), col = heat.colors(12), xaxt = "n", yaxt = "n",
```

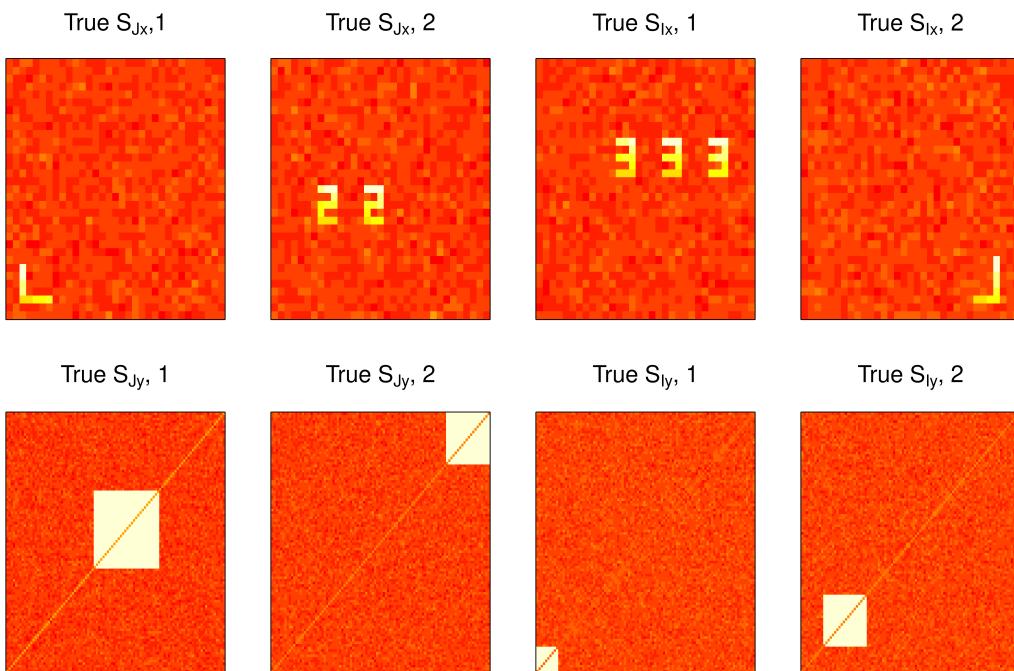


Figure 1: True joint and individual loadings in example 1.

```

main = expression("True S\"[\"Jy\"] * ", 2"))
image(vec2net(data$siY[1, ]), col = heat.colors(12), xaxt = "n", yaxt = "n",
      main = expression("True S\"[\"Iy\"] * ", 1"))
image(vec2net(data$siY[2, ]), col = heat.colors(12), xaxt = "n", yaxt = "n",
      main = expression("True S\"[\"Iy\"] * ", 2"))

```

Function singR performs all steps in the SING pipeline as a single function

We first illustrate the use of the wrapper function `singR` using the default settings. We will describe optional arguments in more detail in example 2.

```
example1 = singR(dX = data$dX, dY = data$dY, individual = T)
```

Details of the SING pipeline

We next explain each of the steps involved in SING estimation. Using these individual functions in place of the high-level `singR` function allows additional fine-tuning and can be helpful for large datasets.

Estimate the number of non-Gaussian components in datasets `dX` and `dY` using `FOBIasymp` from **ICtest**:

```
n.comp.X = NG_number(data$dX)
n.comp.Y = NG_number(data$dY)
```

Apply `lncga` separately to each dataset using the JB statistic as the measure of non-Gaussianity:

```
# JB on X
estX_JB = lncga(xData = data$dX, n.comp = n.comp.X, whiten = "sqrtprec",
                  restarts.pbyd = 20, distribution = "JB")
Uxfull <- estX_JB$U
Mx_JB = est.M.ols(sData = estX_JB$$, xData = data$dX)

# JB on Y
estY_JB = lncga(xData = data$dY, n.comp = n.comp.Y, whiten = "sqrtprec",
                  restarts.pbyd = 20, distribution = "JB")
Uyfull <- estY_JB$U
My_JB = est.M.ols(sData = estY_JB$$, xData = data$dY)
```

Use greedymatch to reorder \hat{U}_x and \hat{U}_y by descending matched correlations and use permTestJointRank to estimate the number of joint components:

```
matchMxMy = greedymatch(scale(Mx_JB, scale = F), scale(My_JB, scale = F),
    Ux = Uxfull, Uy = Uyfull)
permJoint <- permTestJointRank(matchMxMy$Mx, matchMxMy$My)
joint_rank = permJoint$rj
```

For preparing input to curvilinear_c, manually prewhiten dX and dY to get \hat{L}_x^{-1} and \hat{L}_y^{-1} :

```
# Center X and Y
dX = data$dX
dY = data$dY
n = nrow(dX)
pX = ncol(dX)
pY = ncol(dY)
dXcentered <- dX - matrix(rowMeans(dX), n, pX, byrow = F)
dYcentered <- dY - matrix(rowMeans(dY), n, pY, byrow = F)

# For X Scale rowwise
est.sigmaXA = tcrossprod(dXcentered)/(pX - 1)
whitenerXA = est.sigmaXA %^% (-0.5)
xDataA = whitenerXA %*% dXcentered
invLx = est.sigmaXA %^% (0.5)

# For Y Scale rowwise
est.sigmaYA = tcrossprod(dYcentered)/(pY - 1)
whitenerYA = est.sigmaYA %^% (-0.5)
yDataA = whitenerYA %*% dYcentered
invLy = est.sigmaYA %^% (0.5)
```

Obtain a reasonable value for the penalty ρ by calculating the JB statistics for all the joint components:

```
# Calculate the Sx and Sy.
Sx = matchMxMy$Ux[1:joint_rank, ] %*% xDataA
Sy = matchMxMy$Uy[1:joint_rank, ] %*% yDataA

# Calculate total JB
JBall = calculateJB(Sx) + calculateJB(Sy)

# Penalty used in curvilinear algorithm:
rho = JBall/10
```

Estimate \hat{U}_x and \hat{U}_y with curvilinear_c:

```
# alpha=0.8 corresponds to JB weighting of skewness and kurtosis
# (can customize to use different weighting):
alpha = 0.8
# tolerance:
tol = 1e-10

out <- curvilinear_c(invLx = invLx, invLy = invLy, xData = xDataA,
    yData = yDataA, Ux = matchMxMy$Ux, Uy = matchMxMy$Uy, rho = rho,
    tol = tol, alpha = alpha, maxiter = 1500, rj = joint_rank)
```

Obtain the final result:

```
# Estimate Sx and Sy and true S matrix using rotation matrices
# of Ux and Uy
Sjx = out$Ux[1:joint_rank, ] %*% xDataA
Six = out$Ux[(joint_rank + 1):n.comp.X, ] %*% xDataA
Sjy = out$Uy[1:joint_rank, ] %*% yDataA
```

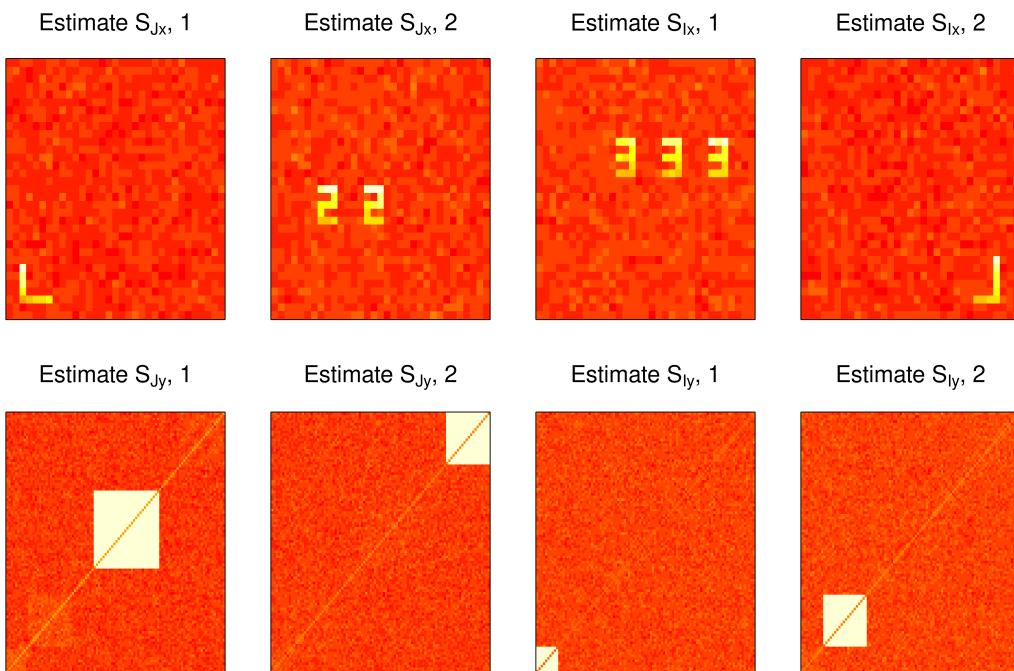


Figure 2: Estimated joint and individual loadings in example 1.

```

SiY = out$Uy[(joint_rank + 1):n.comp.Y, ] %*% yDataA

# Estimate Mj
Mxjoint = tcrossprod(invLx, out$Ux[1:joint_rank, ])
Mxindiv = tcrossprod(invLx, out$Ux[(joint_rank + 1):n.comp.X, ])
Myjoint = tcrossprod(invLy, out$Uy[1:joint_rank, ])
Myindiv = tcrossprod(invLy, out$Uy[(joint_rank + 1):n.comp.Y, ])

# signchange to make the skewness of the rows of S positive
Sjx_sign = signchange(Sjx, Mxjoint)
Sjy_sign = signchange(Sjy, Myjoint)
Six_sign = signchange(Six, Mxindiv)
SiY_sign = signchange(SiY, Myindiv)

Sjx = Sjx_sign$$
Sjy = Sjy_sign$$
Six = Six_sign$$
SiY = SiY_sign$$

Mxjoint = Sjx_sign$M
Myjoint = Sjy_sign$M
Mxindiv = Six_sign$M
Myindiv = SiY_sign$M

est.Mj = aveM(Mxjoint, Myjoint)

trueMj <- data.frame(mj1 = data$mj[, 1], mj2 = data$mj[, 2], number = 1:48)
SINGMj <- data.frame(mj1 = est.Mj[, 1], mj2 = est.Mj[, 2], number = 1:48)

```

Plot \hat{S}_{Jx} , \hat{S}_{Jy} , \hat{S}_{Ix} , and \hat{S}_{Iy} in figure 2.

Plot \hat{M}_J in figure 3.

```

library(tidyverse)
library(ggpubr)

```

```
# true Mj
```

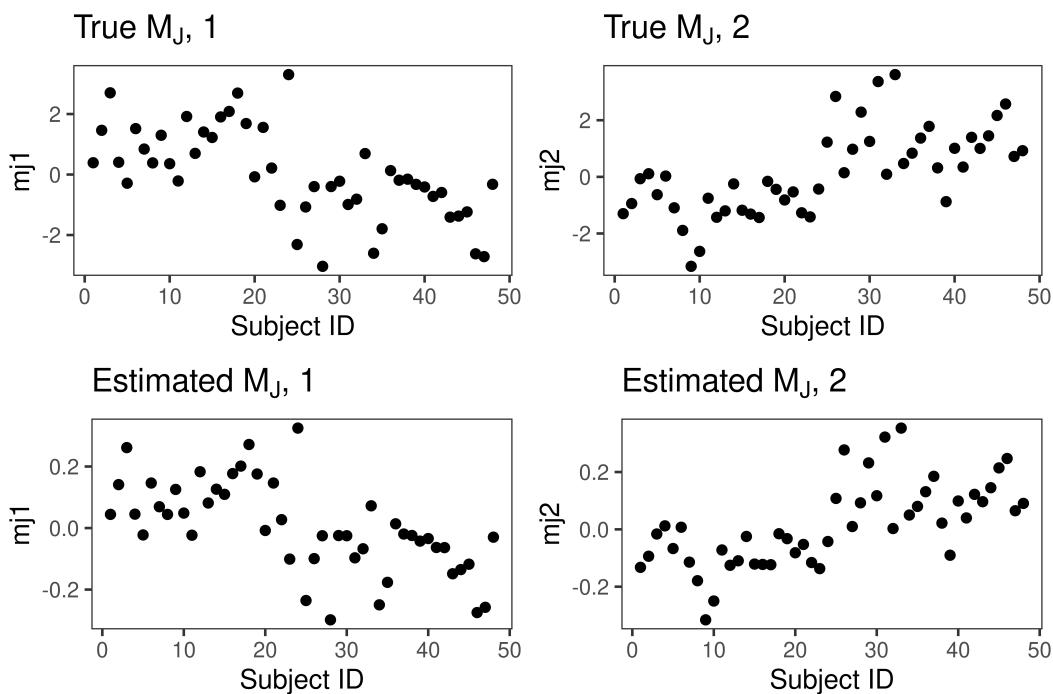


Figure 3: Estimated joint subject scores in example 1.

```
t1 <- ggplot(data = trueMj) + xlab("Subject ID") + geom_point(mapping = aes(y = mj1,
  x = number)) + ggtitle(expression("True M"[J"] * ", 1")) + theme_bw() +
  theme(panel.grid = element_blank())

t2 <- ggplot(data = trueMj) + xlab("Subject ID") + geom_point(mapping = aes(y = mj2,
  x = number)) + ggtitle(expression("True M"[J"] * ", 2")) + theme_bw() +
  theme(panel.grid = element_blank())

# SING estimated Mj
S1 <- ggplot(data = SINGMj) + xlab("Subject ID") + geom_point(mapping = aes(y = mj1,
  x = number)) + ggtitle(expression("Estimated M"[J"] * ", 1")) +
  theme_bw() + theme(panel.grid = element_blank())

S2 <- ggplot(data = SINGMj) + xlab("Subject ID") + geom_point(mapping = aes(y = mj2,
  x = number)) + ggtitle(expression("Estimated M"[J"] * ", 2")) +
  theme_bw() + theme(panel.grid = element_blank())

ggarrange(t1, t2, S1, S2, ncol = 2, nrow = 2)
```

4.2 Example 2. MRI data simulation

This example is a simulation inspired by the real data analysis of the Human Connectome Project from Risk and Gaynanova (2021). X are generated from \hat{S}_X from working memory task maps and Y are generated from \hat{S}_Y from resting-state correlations from a previous SING analysis of the Human Connectome Project. The working memory loadings are defined on the cortical surface, which is the highly folded ribbon of gray matter forming the outer layer of the brain containing billions of neural bodies and dendrites. Large working memory loadings indicate locations in the brain that tend to work together during memory tasks. The resting-state correlation loadings are defined using a brain parcellation from (Glasser et al. 2016) and (Akiki and Abdallah 2019). Large resting-state loadings are related to large correlations between brain regions occurring when a participant is lying in a scanner performing no task. Additional details are in (Risk and Gaynanova 2021). For the purposes of this example, we reduce computation time by lowering the resolution of the working memory loadings in dataset X from 60,000 to 2,000. For the resting-state correlation loadings, we subset from the 360 x 360 loadings matrices formed from the 360 regions in the multimodal parcellation (MMP) to 100 x

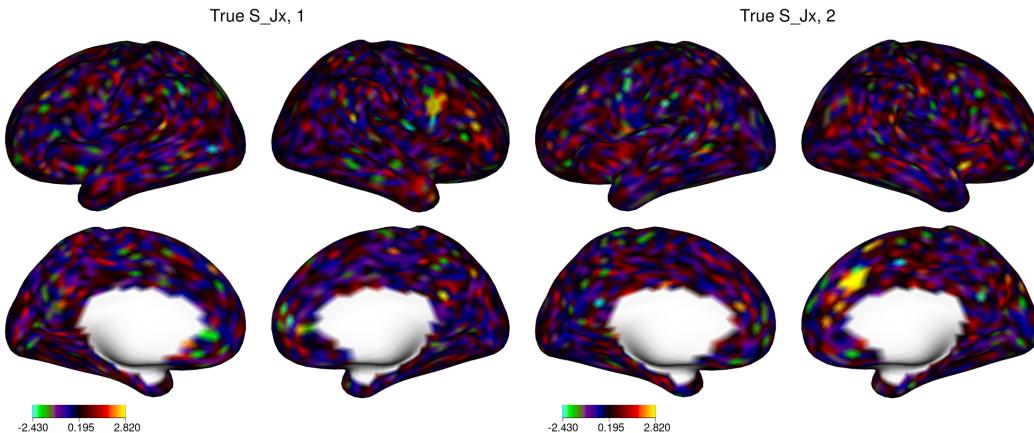


Figure 4: True joint loadings in dataset X in example 2.

100. To run this example, download the files in the folder `extdata` from the github repository (Wang, Gaynanova, and Risk 2022).

```
# Load the package
library(singR)

# Read and visualize data
load("extdata/simdata.rda")

# sign change makes the skewness positive, which makes the
# region of 'activation' yellow in the plots that follow
Sxtrue = signchange(simdata$sjx)$S
Sytrue = signchange(simdata$sjy)$S
```

The `simdata.rda` have already been resampled from 32k to 2k resolution to reduce computation time. Next, we resample the background surface (i.e., template found on (Wang, Gaynanova, and Risk 2022)) to the same resolution, which will allow us to plot the loadings on the cortical surface. This step uses `ciftiTools` (Pham, Muschelli, and Mejia 2021) and connectome workbench (Marcus et al. 2011). To run this code, one needs to install connectome workbench, as described in (<https://github.com/mandymejia/ciftiTools>).

```
library(ciftiTools)
ciftiTools.setOption("wb_path", "C:/Software/workbench")

## the template cifti file is on
## https://github.com/thebrisklab/singR/tree/main/extdata.
## here, resample to 2k resolution.
xii_template <- read_cifti("extdata/template.dtseries.nii", brainstructures = c("left",
  "right"), resamp_res = 2000)

xii_new <- newdata_xifti(xii_template, t(Sxtrue))
view_xifti_surface(select_xifti(xii_new, 1), zlim = c(-2.43, 2.82)) ## true S_JX1
view_xifti_surface(select_xifti(xii_new, 2), zlim = c(-2.43, 2.82)) ## true S_JX2
```

In figure 4, the yellow regions indicate locations with large loadings. Similar plots can be created for the two individual components (not shown). When applied to fMRI activation maps, SING tends to identify small patches of cortex, similar to this figure.

Next, we convert the rows of S_{Jy} to symmetric matrices and create plots. The nodes are organized into communities (i.e., modules) to aid visualization. SING tends to identify a single node and the connections with this node, which result in a cross-like pattern in the matrix representation. The joint loadings are plotted in figure 5 with `plotNetwork_change`, which is defined below. Similar plots can be created for the loadings from the two individual components.

```
# define plotNetwork_change
plotNetwork_change = function(component, title = "", qmin = 0.005,
```

```

qmax = 0.995, path = "mmpplus.csv", make.diag = NA) {
  # component: vectorized network of length choose(n,2)
  require(ggplot2)
  require(grid)
  require(scales)

  # load communities for plotting:
  mmp_modules = read.csv(path, header = TRUE)
  mmp_order = order(mmp_modules$Community_Vector)

  zmin = quantile(component, qmin)
  zmax = quantile(component, qmax)

  netmat = vec2net(component, make.diag)

  meltsub = create.graph.long(netmat, mmp_order)

  g2 = ggplot(meltsub, aes(X1, X2, fill = value)) + geom_tile() +
    scale_fill_gradient2(low = "blue", high = "red", limits = c(zmin,
      zmax), oob = squish) + labs(title = title, x = "Node 1",
      y = "Node 2") + coord_cartesian(clip = "off", xlim = c(-0,
      100))

  loadingsummary = apply(abs(netmat), 1, sum, na.rm = TRUE)
  loadingsum2 = loadingsummary[mmp_order]

  Community = factor(mmp_modules$Community_Label)[mmp_order]

  g3 = qplot(c(1:100), loadingsum2, col = Community, size = I(3)) +
    xlab("MMP Index") + ylab("L1 Norm of the Rows")

  return(list(netmatfig = g2, loadingsfig = g3, netmat = netmat,
    loadingsummary = loadingsummary))
}

library(cowplot)
# plot for the true component of Y
path = "extdata/new_mmp.csv"
out_true1 = plotNetwork_change(Sytrue[1, ], title = expression("True S"["Jy"] * 
  ", 1"), qmin = 0.005, qmax = 0.995, path = path)
out_true2 = plotNetwork_change(Sytrue[2, ], title = expression("True S"["Jy"] * 
  ", 2"), qmin = 0.005, qmax = 0.995, path = path)

p1 = out_true1$netmatfig
p2 = out_true1$loadingsfig
p3 = out_true2$netmatfig
p4 = out_true2$loadingsfig

plot_grid(p1, p2, p3, p4, nrow = 2)

```

In figure 5, the left plots depict the loadings in the network space, where each element represents the strength of the connection between two nodes (i.e., regions). Then if the subject score corresponding to the component is large, the loadings make a large contribution to the subject's functional connectivity. In our previous work, we found that the loadings for a component tend to be structured such that a single node is prominent, resulting in a cross-like pattern. To easily identify which node or nodes are prominent, the right plots depict the L1-norms of the rows of the loadings matrices. In this example, a single node stands out for each of the components. For additional interpretation, see (Risk and Gaynanova 2021). In (Risk and Gaynanova 2021), it was discovered that for a given joint component, the patch of cortex with largest loadings in the working memory task tended to be located in the same area as the node whose loadings have the largest L1-norm in the resting-state dataset.

Function `singR` performs all steps in the SING pipeline as a single function

In example 1, we introduced the pipeline of the SING method. We will use example 2 to explain the `singR` function in detail. The default output of `singR` is a list of \hat{S}_{Jx} , \hat{S}_{Jy} , \hat{M}_J , \hat{M}_{Jx} , and \hat{M}_{Jy} . By default, it will center the data such that the mean of each row is equal to zero. In our simulated dataset, all

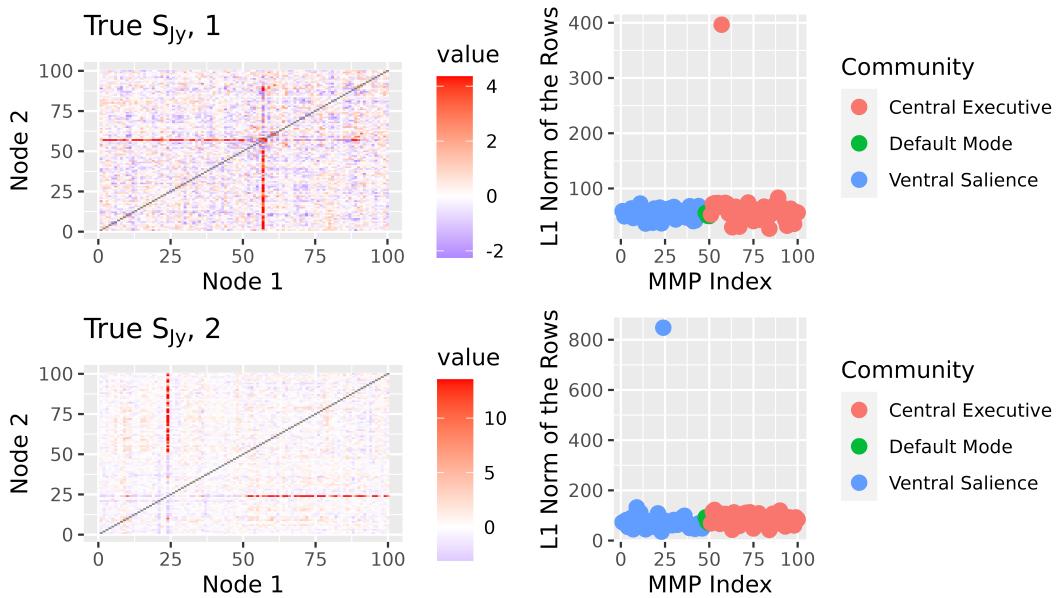


Figure 5: True joint loadings in dataset Y in example 2.

variables are on the same scale, and consequently we do not perform standardization (stand=FALSE). When stand=TRUE, the data are additionally standardized to have the mean of each column equal to zero and variance of each column equal to one, which is the standardization commonly used in PCA. If n.comp.X and n.comp.Y are not specified, singR will use FOB1asymp from [ICtest](#) to estimate the number of non-Gaussian components in each dataset, which requires additional computational expense. Other tests of the number of non-Gaussian components accounting for spatial smoothing/autocorrelation can be found in (Zhao et al. 2022). These may be more effective for spatially correlated data but are generally slower.

When individual = TRUE, the singR will additionally output \hat{M}_{Ix} , \hat{M}_{Iy} , \hat{S}_{Ix} and \hat{S}_{Iy} . When distribution = "tiltedgaussian", non-Gaussian components will be estimated through lngca using likelihood component analysis, which is slower but can be more accurate. By default distribution = "JB", and lncga will use the Jarque-Bera test statistic as the measure of non-Gaussianity of each component.

The Cplus argument determines whether to use curvilinear_c or curvilinear in singR.curvilinear is implemented with pure R but is slow while curvilinear_c uses C++. The parameter rho_extent can be one of c("small", "medium", "large") or a number. This determines the penalty ρ in curvilinear or curvilinear_c that results in equal or highly correlated \hat{M}_{Jx} and \hat{M}_{Jy} . Additionally, we can use pmse() to evaluate the distance between two subject score matrices. With larger ρ , the $pmse(\hat{M}_{Jx}, \hat{M}_{Jy})$ value will be smaller. Usually, "small" ρ is sufficient for approximately equal \hat{M}_{Jx} and \hat{M}_{Jy} . We have observed that very large ρ can adversely impact the accuracy of the loadings. Our recommendation is to use "small" ρ and check if it results in equal scores, and if not, then try other settings. The code below took approximately 20 seconds to run on a 2.8 GHz processor.

```
example2 = singR(dX = simdata$dX, dY = simdata$dY, rho_extent = "small",
Cplus = TRUE, stand = FALSE, individual = TRUE, distribution = "JB")
```

The joint loadings \hat{S}_{Jx} are depicted in figure 6.

```
xii_new <- newdata_xifti(xii_template, t(example2$Sjx))
```

```
view_xifti_surface(select_xifti(xii_new, 1), zlim = c(-2.43, 2.82)) ## component1 small rho
view_xifti_surface(select_xifti(xii_new, 2), zlim = c(-2.43, 2.82)) ## component2 small rho
```

The joint loadings \hat{S}_{Jy} are depicted in figure 7.

```
library(cowplot)
path = "extdata/new_mmp.csv"
out_rhoSmall1 = plotNetwork_change(example2$Sjy[1, ], title = expression("Estimate S"["Jy"] *
```

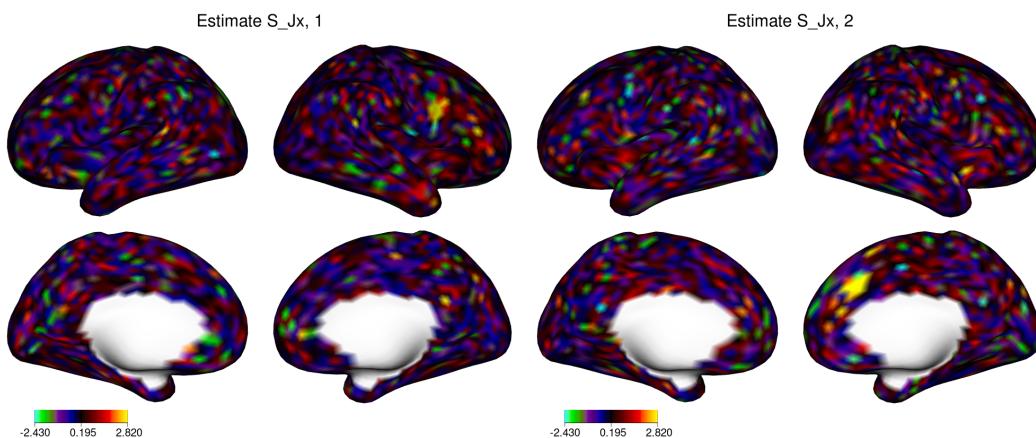


Figure 6: Estimated joint loadings in dataset X in example 2.

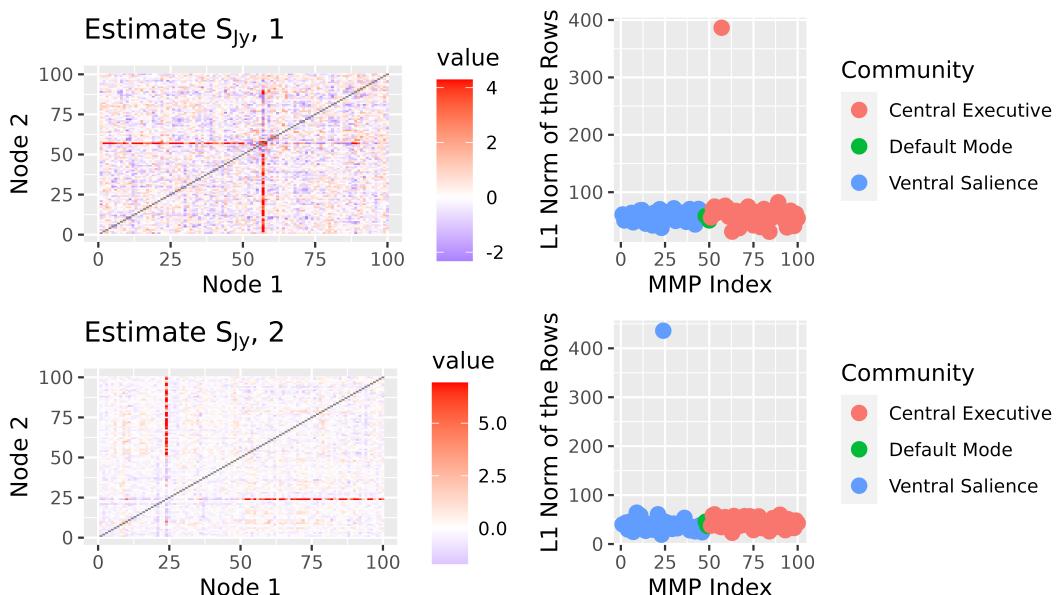


Figure 7: Estimated joint loadings in dataset Y in example 2.

```

", 1"), qmin = 0.005, qmax = 0.995, path = path)
out_rhoSmall12 = plotNetwork_change(example2$jy[2, ], title = expression("Estimate S"["Jy"] *
", 2"), qmin = 0.005, qmax = 0.995, path = path)

p5 = out_rhoSmall1$netmatfig
p6 = out_rhoSmall1$loadingsfig
p7 = out_rhoSmall2$netmatfig
p8 = out_rhoSmall2$loadingsfig

plot_grid(p5, p6, p7, p8, nrow = 2)

```

5 Summary

This paper introduces the [singR](#) package and demonstrates how simultaneous non-Gaussian component analysis can be used to extract shared features from two datasets using R. The main contribution of the R package [singR](#) is to provide easy code for data integration in neuroscience. We introduce the function [singR](#), which combines the SING pipeline into one function that performs data standardization, estimates the number of non-Gaussian components and estimates the number of joint components. Previous analyses indicate the joint structure estimated by SING can improve upon other neuroimaging data integration methods. SING can reveal new insights by using non-Gaussianity for

both dimension reduction and latent variable extraction, whereas ICA methods involve an initial PCA step that tends to aggregate features and can remove information.

6 Acknowledgments

Research reported in this publication was supported by the National Institute of Mental Health of the National Institutes of Health under award number R01MH129855 to BBR. The research was also supported by the Division of Mathematical Sciences of the National Science Foundation under award number DMS-2044823 to IG. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health and National Science Foundation.

Simulated data were based on a previous analysis of data from the Human Connectome Project. These data were provided [in part] by the Human Connectome Project, WU-Minn Consortium (Principal Investigators: David Van Essen and Kamil Ugurbil; 1U54MH091657) funded by the 16 NIH Institutes and Centers that support the NIH Blueprint for Neuroscience Research; and by the McDonnell Center for Systems Neuroscience at Washington University.

References

- Akiki, Teddy J, and Chadi G Abdallah. 2019. "Determining the Hierarchical Architecture of the Human Brain Using Subject-Level Clustering of Functional Networks." *Scientific Reports* 9 (1): 1–15. <https://doi.org/10.1038/s41598-019-55738-y>.
- Amico, Enrico, Daniele Marinazzo, Carol Di Perri, Lizette Heine, Jitka Annen, Charlotte Martial, Mario Dzemidzic, et al. 2017. "Mapping the Functional Connectome Traits of Levels of Consciousness." *NeuroImage* 148: 201–11. <https://doi.org/https://doi.org/10.1016/j.neuroimage.2017.01.020>.
- Beckmann, Christian F, Marilena DeLuca, Joseph T Devlin, and Stephen M Smith. 2005. "Investigations into Resting-State Connectivity Using Independent Component Analysis." *Philosophical Transactions of the Royal Society B: Biological Sciences* 360 (1457): 1001–13. <https://doi.org/10.1098/rstb.2005.1634>.
- Calhoun, V. D., T. Adali, N. R. Giuliani, J. J. Pekar, K. A. Kiehl, and G. D. Pearlson. 2006. "Method for Multimodal Analysis of Independent Source Differences in Schizophrenia: Combining Gray Matter Structural and Auditory Oddball Functional Data." Journal Article. *Hum Brain Mapp* 27 (1): 47–62. <https://doi.org/10.1002/hbm.20166>.
- Calhoun, V. D., T. Adali, K. A. Kiehl, R. Astur, J. J. Pekar, and G. D. Pearlson. 2006. "A Method for Multitask fMRI Data Fusion Applied to Schizophrenia." Journal Article. *Hum Brain Mapp* 27 (7): 598–610. <https://doi.org/10.1002/hbm.20204>.
- Calhoun, V. D., J. Liu, and T. Adali. 2009. "A Review of Group ICA for fMRI Data and ICA for Joint Inference of Imaging, Genetic, and ERP Data." Journal Article. *Neuroimage* 45 (1 Suppl): S163–72. <https://doi.org/10.1016/j.neuroimage.2008.10.057>.
- Calhoun, V. D., and J. Sui. 2016. "Multimodal Fusion of Brain Imaging Data: A Key to Finding the Missing Link(s) in Complex Mental Illness." Journal Article. *Biol Psychiatry Cogn Neurosci Neuroimaging* 1 (3): 230–44. <https://doi.org/10.1016/j.bpsc.2015.12.005>.
- Carmichael, Iain. 2022. *Ajive: Angle Based Joint and Individual Variation Explained*. <https://cran.r-project.org/web/packages/RaJIVE/index.html>.
- Feng, Qing, Meilei Jiang, Jan Hannig, and JS Marron. 2018. "Angle-Based Joint and Individual Variation Explained." *Journal of Multivariate Analysis* 166: 241–65. <https://doi.org/10.1016/j.jmva.2018.03.008>.
- Glasser, Matthew F, Timothy S Coalson, Emma C Robinson, Carl D Hacker, John Harwell, Essa Yacoub, Kamil Ugurbil, et al. 2016. "A Multi-Modal Parcellation of Human Cerebral Cortex." *Nature* 536 (7615): 171–78. <https://doi.org/10.1038/nature18933>.
- Helwig, Nathaniel E. 2018. *Ica: Independent Component Analysis*. <https://CRAN.R-project.org/package=ica>.
- Marchini, J L, C Heaton, and B D Ripley. 2021. *fastICA: FastICA Algorithms to Perform ICA and Projection Pursuit*. <https://CRAN.R-project.org/package=fastICA>.
- Marcus, Daniel S, John Harwell, Timothy Olsen, Michael Hodge, Matthew F Glasser, Fred Prior, Mark Jenkinson, Timothy Laumann, Sandra W Curtiss, and David C Van Essen. 2011. "Informatics and Data Mining Tools and Strategies for the Human Connectome Project." *Frontiers in Neuroinformatics* 5: 4. <https://doi.org/10.3389/fninf.2011.00004/full>.
- Mejia, Amanda F, Mary Beth Nebel, Tikai Wang, Brian S Caffo, and Ying Guo. 2020. "templateICAr." <https://doi.org/10.1080/01621459.2019.1679638>.

- Miettinen, Jari, Klaus Nordhausen, and Sara Taskinen. 2017. "Blind Source Separation Based on Joint Diagonalization in R: The Packages JADE and BSSasymp." *Journal of Statistical Software* 76 (2): 1–31. <https://doi.org/10.18637/jss.v076.i02>.
- Nordhausen, Klaus, Hannu Oja, David E. Tyler, and Joni Virta. 2022. *ICtest: Estimating and Testing the Number of Interesting Components in Linear Dimension Reduction*. <https://CRAN.R-project.org/package=ICtest>.
- O'Connell, Michael J., and Eric F. Lock. 2020. *R.jive: Perform JIVE Decomposition for Multi-Source Data*. <https://CRAN.R-project.org/package=r.jive>.
- Pham, Damon D, John Muschelli, and Amanda F Mejia. 2021. "ciftiTools: A Package for Reading, Writing, Visualizing and Manipulating CIFTI Files in r." <https://arxiv.org/abs/2106.11338>.
- Risk, Benjamin B., and Irina Gaynanova. 2021. "Simultaneous Non-Gaussian Component Analysis (SING) for Data Integration in Neuroimaging." Journal Article. *The Annals of Applied Statistics* 15 (3): 1431–54, 24. <https://doi.org/10.1214/21-AOAS1466>.
- Risk, Benjamin B., Nicholas A. James, and David S. Matteson. 2015. *steadyICA: ICA and Tests of Independence via Multivariate Distance Covariance*. <https://CRAN.R-project.org/package=steadyICA>.
- Risk, Benjamin B., David S. Matteson, and David Ruppert. 2019. "Linear Non-Gaussian Component Analysis via Maximum Likelihood." *Journal of the American Statistical Association* 114 (525): 332–43. <https://doi.org/10.1080/01621459.2017.1407772>.
- Sui, J., T. Adali, Q. Yu, J. Chen, and V. D. Calhoun. 2012. "A Review of Multivariate Methods for Multimodal Fusion of Brain Imaging Data." Journal Article. *J Neurosci Methods* 204 (1): 68–81. <https://doi.org/10.1016/j.jneumeth.2011.10.031>.
- Sui, Jing, Tülay Adali, Godfrey Pearlson, Honghui Yang, Scott R. Sponheim, Tonya White, and Vince D. Calhoun. 2010. "A CCA+ICA Based Model for Multi-Task Brain Imaging Data Fusion and Its Application to Schizophrenia." *NeuroImage* 51 (1): 123–34. <https://doi.org/10.1016/j.neuroimage.2010.01.069>.
- Sui, J., G. Pearlson, A. Caprihan, T. Adali, K. A. Kiehl, J. Liu, J. Yamamoto, and V. D. Calhoun. 2011. "Discriminating Schizophrenia and Bipolar Disorder by Fusing fMRI and DTI in a Multimodal CCA+ Joint ICA Model." Journal Article. *NeuroImage* 57 (3): 839–55. <https://doi.org/10.1016/j.neuroimage.2011.05.055>.
- Vergara, V. M., A. Ulloa, V. D. Calhoun, D. Boutte, J. Chen, and J. Liu. 2014. "A Three-Way Parallel ICA Approach to Analyze Links Among Genetics, Brain Structure and Brain Function." Journal Article. *NeuroImage* 98: 386–94. <https://doi.org/10.1016/j.neuroimage.2014.04.060>.
- Wang, Liangkang, Irina Gaynanova, and Benjamin B. Risk. 2022. "singR." <https://github.com/thebrisklab/singR>.
- Zhao, Yuxuan, David S Matteson, Stewart H Mostofsky, Mary Beth Nebel, and Benjamin B Risk. 2022. "Group Linear Non-Gaussian Component Analysis with Applications to Neuroimaging." *Computational Statistics & Data Analysis* 171: 107454. <https://doi.org/10.1016/j.csda.2022.107454>.
- Zhou, Guoxu, Qibin Zhao, Yu Zhang, Tülay Adali, Shengli Xie, and Andrzej Cichocki. 2016. "Linked Component Analysis from Matrices to High-Order Tensors: Applications to Biomedical Data." *Proceedings of the IEEE* 104 (2): 310–31. <https://doi.org/10.1109/JPROC.2015.2474704>.

Liangkang Wang
 Brown University
 Department of Biostatistics
 Providence, Rhode Island, US
 ORCID: 0000-0003-3393-243X
liangkang_wang@brown.edu

Irina Gaynanova
 University of Michigan
 Department of Biostatistics
 Ann Arbor, MI, US
<https://irinagain.github.io/>
 ORCID: 0000-0002-4116-0268
irinagn@umich.edu

Benjamin Risk
 Emory University
 Department of Biostatistics and Bioinformatics
 Atlanta, Georgia, US
<https://github.com/thebrisklab/>
 ORCID: 0000-0003-1090-0777

benjamin.risk@emory.edu

RobustCalibration: Robust Calibration of Computer Models in R

by Mengyang Gu

Abstract Two fundamental research tasks in science and engineering are forward predictions and data inversion. This article introduces a new R package [RobustCalibration](#) for Bayesian data inversion and model calibration using experiments and field observations. Mathematical models for forward predictions are often written in computer code, and they can be computationally expensive to run. To overcome the computational bottleneck from the simulator, we implemented a statistical emulator from the [RobustGaSP](#) package for emulating both scalar-valued or vector-valued computer model outputs. Both posterior sampling and maximum likelihood approach are implemented in the [RobustCalibration](#) package for parameter estimation. For imperfect computer models, we implement the Gaussian stochastic process and scaled Gaussian stochastic process for modeling the discrepancy function between the reality and mathematical model. This package is applicable to various other types of field observations and models, such as repeated experiments, multiple sources of measurements and correlated measurement bias. We discuss numerical examples of calibrating mathematical models that have closed-form expressions, and differential equations solved by numerical methods.

1 Introduction

Complex processes are often represented as mathematical models, implemented in computer code. These mathematical models are often called *computer models* or *simulators*, and are widely used to simulate different processes given a set of parameters and initial conditions ([Sacks et al., 1989](#)). The initial conditions and model parameters may be unknown or uncertain in practice. Calibrating the computer models based on real experiments or observations is one of the fundamental tasks in science and engineering, widely known as *data inversion* or *model calibration*.

We follow the notation in [Bayarri et al. \(2007a\)](#) for defining the model calibration problem. Let $y^F(\mathbf{x})$ be the real-valued field observation with a p_x -dimensional observable input vector \mathbf{x} . The field observation is often decomposed by $y^F(\mathbf{x}) = y^R(\mathbf{x}) + \epsilon$, where $y^R(\cdot)$ denotes a function of unknown reality, and ϵ denotes the noise, with the superscript 'F' and 'R' denoting the field observations and reality, respectively. Scientists often model the unknown reality by a computer model, denoted by $f^M(\mathbf{x}, \theta)$ at the p_x -dimensional observable input \mathbf{x} , and p_θ -dimensional calibration parameters θ , unobservable from the experiments, with superscript 'M' denoting the computer model.

When the computer model describes the reality perfectly, a field observation can be written as:

$$y^F(\mathbf{x}) = f^M(\mathbf{x}, \theta) + \epsilon, \quad (1)$$

where ϵ is a zero-mean Gaussian noise. We call the method by Equation (1) the *no-discrepancy calibration*.

When computer models are imperfect, the statistical calibration model below is often used:

$$y^F(\mathbf{x}) = f^M(\mathbf{x}, \theta) + \delta(\mathbf{x}) + \epsilon, \quad (2)$$

where $\delta(\cdot)$ is the unobservable discrepancy between the mathematical model and reality. Equation (2) implies that the reality can be written as $y^R(\mathbf{x}) = f^M(\mathbf{x}, \theta) + \delta(\mathbf{x})$ at any observable input $\mathbf{x} \in \mathcal{X}$. The model calibration framework has been studied extensively. In [Kennedy and O'Hagan \(2001\)](#), the discrepancy function $\delta(\cdot)$ is modeled via a Gaussian stochastic process (GaSP), leading to more accurate prediction based on the joint model of calibrated computer model and discrepancy compared to either using the computer model or discrepancy model for prediction alone. We call this method the *GaSP Calibration*. The GaSP calibration has been applied in a wide range of studies of continuous and categorical outputs ([Bayarri et al., 2007a; Higdon et al., 2008; Paulo et al., 2012; Chang et al., 2016, 2022](#)). Both no-discrepancy calibration and GaSP calibration are implemented in the [RobustCalibration](#) package.

The calibrated computer model output can be far from the observations in terms of L_2 distance when the discrepancy function is modeled by the GaSP, ([Arendt et al., 2012](#)), since a large proportion of the variability in the data can be explained by discrepancy function. To solve this problem, we implemented the scaled Gaussian stochastic process (S-GaSP) calibration, or *S-GaSP Calibration* in the [RobustCalibration](#) package. The S-GaSP model of discrepancy function was first introduced in [Gu and Wang \(2018\)](#), where more prior probability mass of the L_2 loss is placed on small values. Consequently, the calibrated computer model by the S-GaSP fits the reality better in terms of L_2 loss than the GaSP calibration. Furthermore, historical data may be used for reducing the parameter space

(Williamson et al., 2013).

A few recent approaches aim to minimize the L_2 distance between the reality and computer model (Tuo and Wu, 2015; Wong et al., 2017), where the reality and discrepancy function are often estimated in two steps separately. The S-GaSP calibration bridges the two-step L_2 calibration with the GaSP calibration. Compared with two-step approaches, parameters and discrepancy ($\theta, \delta(\cdot)$) are estimated in GaSP calibration and S-GaSP calibration jointly, and the uncertainty of these estimation can be obtained based on the likelihood function of the sampling model. Furthermore, compared with the orthogonal calibration model (Plumlee, 2017), the S-GaSP process places more prior mass on inputs leading to small values of the random L_2 loss, instead of more prior mass on inputs close to the critical points of the loss function.

Another concern is the computational cost of model calibration, as computer models may involve numerical solutions of differential equations, which can be expensive to run. We utilize a statistical emulator for approximating computationally expensive computer models. The GaSP emulator is a well-developed framework for emulating computationally expensive computer models and it was implemented in a few R packages, such as **DiceKriging** (Roustant et al., 2012), **GPfit** (MacDonald et al., 2015) and **RobustGaSP** (Gu et al., 2019). Here we specify the functions `rgasp`, `ppgasp` and `predict` in **RobustGaSP** for emulating computer models with scalar-valued and vector-valued outputs, respectively.

Building packages for Bayesian model calibration is much more complicated than packages of statistical emulator, as both field observations and computer models need to be integrated. A few statistical packages are available for Bayesian model calibration, such as **BACCO** (Hankin, 2005), **SAVE** package (Palomo et al., 2015), and **CaliCo** (Carmassi et al., 2018). The **CaliCo** package, for example, integrates **DiceKriging** for emulating computational expensive simulations. It offers different types of diagnostic plots based on **ggplot2** (Wickham, 2011), and distinct prior choices of the parameters. In the **BACCO** and **SAVE**, the GaSP model of the discrepancy function (Kennedy and O'Hagan, 2001) is assumed, and GaSP emulators can be used to approximate expensive computer models.

Although a large number of studies were developed for Bayesian model calibration, to the authors' best knowledge, many methods implemented in **RobustCalibration** have not been implemented in another software package previously. We highlight a few unique features of the **RobustCalibration** package. First of all, we allow users to specify different types of output of field observations, for different scenarios through the `output` argument in the `rcalibration` function. The simplest scenario is to have a vector of fields observations. We also allow users to input a matrix or a list of fields observations with the same or different number of replications, respectively. Efficient computation from sufficient statistics was implemented for model calibration with replications, which can improve computation up to k^3 times, where k is the number of repeated experiments. Second, distinct models can be calibrated with different sets of parameters for data from multiple sources by the `calibration_MS` function. Third, we implemented emulators for approximating expensive computer models with both scalar-valued and vector-valued outputs. Emulators for vector-valued outputs, capable of handling a large number of temporal or spatial coordinates, were not implemented in other packages for model calibration. Here we use the parallel partial Gaussian stochastic process (PP-GaSP) emulator from **RobustGaSP** package (Gu et al., 2019) as a computationally scalable emulator of vector-valued output. The **RobustGaSP** package has been applied to various applications, such as emulating geophysical models of ground deformation (Anderson et al., 2019), and storm surge simulation (Ma et al., 2022). Furthermore, users can choose no-discrepancy calibration, GaSP or S-GaSP models of discrepancy functions for different scenarios. Statistical inferences by both posterior samples and the maximum likelihood estimator (MLE) are made available for these calibration approaches by the `method` argument in the `rcalibration` function.

The rest of the paper is organized below. We first give an overview of the **RobustCalibration** package to introduce the main functions and their usage. We further introduce methodology and numerical examples to illustrate the implemented methods in the **RobustCalibration** package. Closed form expressions of likelihood functions, derivatives, posterior distributions and computational algorithms are outlined in the Appendix.

2 An overview of **RobustCalibration**

Figure 1 gives a schematic overview of the **RobustCalibration** package. We consider predicting the reality in two ways: using both calibrated computer model and discrepancy (predictive accuracy), and the calibrated computer model alone (calibration accuracy), both evaluated in terms of the L_2 loss. The left panel gives comparison of accuracy and computational cost between the implemented methods. The no-discrepancy calibration is faster than GaSP and S-GaSP calibration, as one does not need to compute the inversion and log determinant of the covariance matrix of the field data,

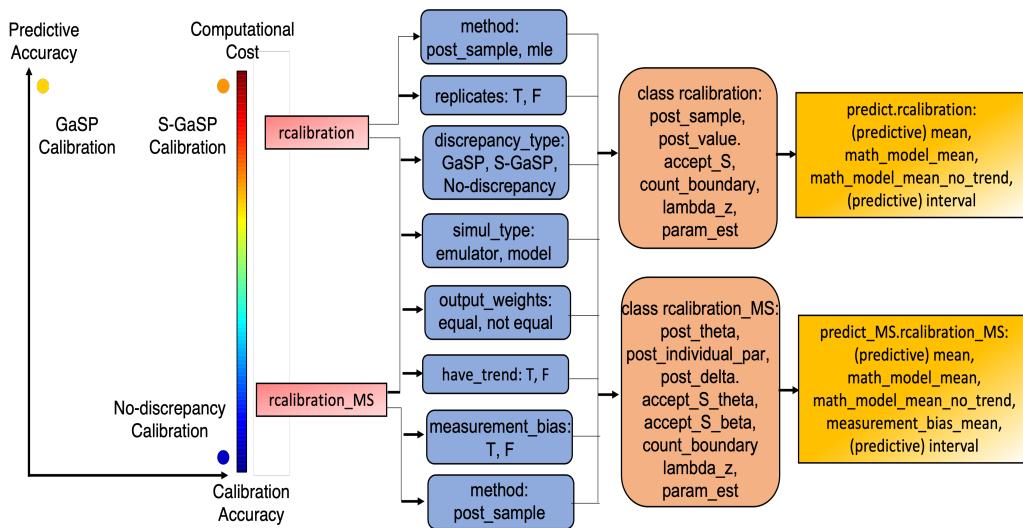


Figure 1: Schematic overview of the `RobustCalibration` package. The left panel compares the expected predictive accuracy and calibration accuracy by different calibration methods. The calibration and calibration_MS in the right panel are two main functions for parameter estimation for observations from single and multiple sources, respectively. The arguments of these functions are given in blue boxes. The object classes `calibration` and `calibration_MS` (orange boxes) can be supplied in `predict.rcalibration` and `predict_MS.rcalibration_MS` functions for making predictions.

unlike GaSP calibration and S-GaSP calibration. Because model discrepancy is modeled, predictive accuracy by GaSP and S-GaSP calibration is typically higher than no-discrepancy calibration. The `RobustCalibration` package can handle model calibration and prediction at a wide range of settings. We first introduce the main functions of the package.

2.1 Main functions

The `RobustCalibration` package can be used for estimating unobservable parameters from computer models and predicting reality. The conventional model calibration and prediction from a single source of field observations are achieved by `rcalibration` and `predict.rcalibration`, as shown in upper blue and yellow boxes in Figure 1, respectively. The function `rcalibration` allows users to call either a Markov chain Monte Carlo (MCMC) algorithm for posterior sampling or a numerical optimization algorithm for computing the maximum likelihood estimator (MLE) of the parameters. Additional arguments can be specified to handle observations from repeated experiments, select a different trend or discrepancy function, and build a surrogate model for approximating expensive computer models, shown in Figure 1. The `rcalibration` function returns an object of the `rcalibration` S4 class with posterior samples or MLE of the parameters, shown in the upper orange box in Figure 1. Then `rcalibration` S4 class is used as the input in the `predict.rcalibration` function to perform predictions on a set of test inputs. The `predict.rcalibration` function returns a `predictobj.rcalibration` S4 class, which contains the predictive mean of the calibrated computer model, estimated trend, and discrepancy function. The estimated interval at a given quantile can also be returned by specifying the vector of the `interval_est` argument in the `predict.rcalibration` function.

In some applications, we have different sources or types of observations. For instance, in calibrating geophysical models, time series of continuous GPS observations and satellite radar interferogram can be jointly used to estimate unobservable parameters (Anderson et al., 2019). In the `RobustCalibration` package, we implement parameter estimation for observations from multiple sources by the `rcalibration_MS` function, which returns a `rcalibration_MS` S4 class that contains posterior samples, shown in Figure 1. Since each source of observations can induce a separate set of model parameters, making the dimension of the parameter space large, the MLE is unstable, and as such is not implemented in `rcalibration_MS`. The `rcalibration_MS` class can be used as an input into the `predict_MS.rcalibration_MS` function for predicting the reality.

2.2 The `rcalibration` function

To use the `rcalibration` function, users need to specify 4 inputs: 1) design, an $n \times p_x$ matrix of observable inputs, 2) observations, field observations that can have three types specified below,

3) `theta_range`, a $p_\theta \times 2$ matrix of the lower and upper bounds of calibration parameters, and 4) either `math_model`, a function of mathematical model, or `input_simul` and `output_simul`, simulation runs by for building the emulator. The `RobustCalibration` package can handle three different kinds of field observations as specified by the argument `observations`. First, one can input an n-vector $(y^F(x_1), \dots, y^F(x_n))^T$, when one field datum is available at each observable input. Second, when k repeated measurements are available for each observable input, one can input an $n \times k$ matrix of field observations, where each row contains k replicates of one observable input. Third, one can input a list, where each element contains k_i replications at the i th observable input, for $i = 1, \dots, n$.

An important feature of the `RobustCalibration` package is the flexible specification of the computer model. Users can specify a mathematical model by supplying a generic R function through the argument `math_model` and selecting the default choice of the simulator `simul_type=1`. The second choice is to use an emulator to predict the computer model by choosing `simul_type=0`. Users need to input a set of simulation runs by `input_simul` and `output_simul` to call an emulator by the `RobustGaSP` package for this choice. Users can either give a $D \times (p_x + p_\theta)$ matrix of input and a D -vector of output to `input_simul` and `output_simul` arguments, for calling `rgasp` function from `RobustGaSP` for emulation, or give a $D \times p_\theta$ matrix of input and a $D \times p_x$ matrix of output to `input_simul` and `output_simul`, to call for `ppgasp` function from `RobustGaSP` for emulation. To account for stochastic error or numerical approximation error in the simulator, we also allow users to specify a nugget parameter estimated by the simulator data by the argument `simul_nug=T`.

The `rcalibration` function contains a few optional arguments. First, users can specify GaSP or S-GaSP discrepancy models by the argument `discrepancy_type="GaSP"` or `discrepancy_type="S-GaSP"`, respectively. The default choice is to assume a S-GaSP discrepancy model. The model without a discrepancy can be specified by the argument `discrepancy_type="no-discrepancy"`. Second, a trend can be specified by the argument `X`. By default, the trend (or mean) of the calibration model is zero. Third, the parameter estimation approach can be specified by the `method` argument. The default choice is `method="post_sample"`, where the posterior samples will be drawn, and stored in the slot `post_sample` of the `rcalibration` class. The MLE can be specified by the argument `method="mle"`, and the estimated parameters are stored in the slot `param_est` of the `rcalibration` class. Users can specify the number of MCMC samples and burn-in samples by the argument `S` and `S_0`, respectively. A vector containing the standard deviation of the proposal distribution for the calibration parameters, the logarithm of the inverse range parameters, and the logarithm of the nugget parameter can be specified by the `sd_proposal` argument. Furthermore, users can "thin" the MCMC samples and only record a subset by the argument `thinning`. For instance, `thinning=5` means only 1/5 of the posterior samples will be recorded. Besides, the inverse variances of noises in the field observations can be specified by the `output_weights`. Finally, arguments `initial_values` and `num_initial_starts` can be specified in numerical optimization to find MLE, and when posterior sampling is used, initial values of the calibration parameters can be specified through the argument `initial_values`.

The object `rcalibration` created by the `rcalibration` function have a few key properties. First of all, if we have `method='post_sample'`, the after burn-in posterior samples of parameters will be stored in the slot `post_sample`, where each row contains posterior samples in one iteration. The first p_θ columns contain posterior samples of the calibration parameters θ . In the no-discrepancy calibration, the $p_\theta + 1$ column of `post_sample` contains posterior draws of the noise variance σ_0^2 and the $p_\theta + 2$ to $p_\theta + q + 1$ columns contain the trend parameters θ_m , for a non-zero mean basis. In the GaSP or S-GaSP calibration, the $p_\theta + 1$ to $p_\theta + p_x + 1$ columns record posterior samples of the log inverse range and log nugget parameters. The $p_\theta + p_x + 2$ to $p_\theta + p_x + q$ columns record the noise variance parameter and trend parameters if they are specified. The parameter λ_z in S-GaSP are recorded in slot `lambda_z`. The indices of the accepted proposed samples are recorded in slots `accept_S` as one of the diagnostic statistics.

2.3 The `predict.rcalibration` function

After calling the `rcalibration` function, an object of the `rcalibration` S4 class will be created, and it can be used as an input into the `predict.rcalibration` function for predicting the reality on a specified matrix of test input using the argument `testing_input`. Besides, if the mean basis is specified by the argument `X` in `rcalibration` function, the user also needs to specify the mean basis of the reality at the test inputs by the argument `X_testing` in function `predict.rcalibration`. If the emulator was not constructed in `rcalibration`, users also need to specify the mathematical model by the argument `math_model` in the `predict.rcalibration` function. Finally, a predictive interval can be specified by the argument `interval_est`. For instance, the 95% predictive interval can be obtained by the `interval_est=c(0.025, 0.975)` argument. The interval of the field data will be computed if `interval_data=T`. Otherwise, the predictive interval of the reality will be computed. If the variance parameter of the noise in field data was specified by the `output_weights`, users should also specify the variance of test data by `testing_output_weights`, which affects the predictive interval of test data.

After calling `predict.rcalibration` function, an object `predictobj.rcalibration` will be created and it contains three different predictors for reality. The slot `math_model_mean_no_trend` gives the predictive mean based on the calibrated computer model (f^M). The slot `math_model_mean` gives the predictive mean based on the calibrated computer model and the estimated trend ($f^M + \mu$), if the basis functions of the trend at the observable input and the test input are specified through `x` and `x_testing` in `rcalibration` and `rcalibration.predict`, respectively. The slot `mean` gives the predictive mean based on the calibrated computer model, estimated trend, and discrepancy ($f^M + \mu + \delta$). If `interval_est` is specified, a matrix of the intervals will be created for quantifying the uncertainty of predictions.

2.4 The `rcalibration_MS` function and the `predict_MS.rcalibration_MS` function

Model calibration and predictions using multiple sources or different types of data are implemented by `rcalibration_MS` and `rcalibration_MS.predict_MS` functions, respectively. One needs to specify 4 inputs: `design`, `observations`, `theta_range` and a form of mathematical model. Suppose we have a data set produced by k sources of observations. The design is a list of k elements, where each element is a matrix of observable input for each source. The argument `observations` takes a list of field observations, where each element is a vector of field observations for each source. In principle, one can have different number of observations from each source and the type of the observations may not be the same. The argument `theta_range` is a $p_\theta \times 2$ matrix of range of parameters, where the i th row contains the minimum and maximum values of the i th coordinate of the calibration parameter vector θ . Furthermore, if closed form expressions of mathematical models are available for each source of data, users can input a list of functions into the `math_model` for each source of data. We also allow users to emulate the expensive simulation. In this scenario, one needs to let `simul_type` be a vector of 1s, indicating emulators will be called for each computer model.

Additional arguments can be specified in `rcalibration_MS`. For instance, the argument `index_theta` takes a list of indices for the associated calibration parameter for each computer model. Suppose we have three calibration parameters $\theta = (\theta_1, \theta_2, \theta_3)$, and the computer model for first source of data is related to the first two calibration parameters and the computer model for second source of data is related to second two calibration parameters. Then `index_theta` is a list where `index_theta[[1]]=c(1, 2)`, and `index_theta[[2]]=c(2, 3)`. More arguments will be introduced along with the examples. After calling `rcalibration_MS`, an S4 class `rcalibration_MS` will be built, and used as an input to `predict_MS.rcalibration_MS` for making predictions.

3 Methods and examples

3.1 No-discrepancy calibration

Let us start with the simplest method: no-discrepancy calibration. First, suppose we have a vector of real-valued field observations $\mathbf{y}^F = (y^F(\mathbf{x}_1), \dots, y^F(\mathbf{x}_n))^T$ at n observable inputs, and the corresponding computer model output is denoted by $\mathbf{f}_\theta^M = (f^M(\mathbf{x}_1, \theta), \dots, f^M(\mathbf{x}_n, \theta))^T$ for any calibration parameters θ . The vector of measurement noise is assumed to follow $\epsilon \sim \mathcal{MN}(0, \sigma_0^2 \Lambda)$, where the covariance of the noise is diagonal with the i th term being $\sigma_0^2 \Lambda_{ii}$ and \mathcal{MN} denote the multivariate normal distribution. In the default setting, σ_0^2 is estimated from the data and $\Lambda = \mathbf{I}_n$, where \mathbf{I}_n is an $n \times n$ identity matrix. Users can manually specify the inverse of the diagonal terms of Λ by the `output_weights` argument in `rcalibration` and `rcalibration_MS` functions, as the variances of measurement errors can be different. We assume the diagonal matrix Λ is given in this section.

For a no-discrepancy calibration model in (1), the MLE of the variance of the noise is $\hat{\sigma}_0^2 = S_0^2/n$ with $S_0^2 = (\mathbf{y}^F - \mathbf{f}_\theta^M)^T \Lambda^{-1} (\mathbf{y}^F - \mathbf{f}_\theta^M)$. The likelihood function is given in the Appendix. As Λ is a diagonal matrix with diagonal terms denoted as $\Lambda_{ii} := 1/w_i$, the profile likelihood follows

$$\mathcal{L}(\theta | \hat{\sigma}_0^2) \propto \left\{ \sum_{i=1}^n w_i (y^F(\mathbf{x}_i) - f^M(\mathbf{x}_i, \theta))^2 \right\}^{-n/2},$$

where $w_i = 1$ is used in the default scenario. Maximizing the profile likelihood of a no-discrepancy model is equivalent to minimizing the weighted squared error $\sum_{i=1}^n w_i (y^F(\mathbf{x}_i) - f^M(\mathbf{x}_i, \theta))^2$. We numerically find θ by the low-storage quasi-Newton optimization method (Liu and Nocedal, 1989) implemented in `lbfgs` function in the `nloptr` package (Ypma (2014)) for optimization.

In the `RobustCalibration` package, we allow the users to specify the trend or mean function modeled as $\mu(\mathbf{x}_i) = \mathbf{h}(\mathbf{x})\theta_m = \sum_{t=1}^q h_t(\mathbf{x})\theta_{m,t}$ for any \mathbf{x} , where $\theta_m = (\theta_{m,1}, \dots, \theta_{m,q})^T$ is a vector of trend

parameters. Maximizing the profile likelihood of a no-discrepancy model with a trend is equivalent to minimizing the squared error loss:

$$\hat{\theta}_m^{LS} = \operatorname{argmin}_{\theta} \sum_{i=1}^n w_i \left(y^F(x_i) - f^M(x_i, \theta) - h(x_i) \hat{\theta}_m^{LS} \right)^2,$$

where the solution follows $\hat{\theta}_m^{LS} = (\mathbf{H}^T \boldsymbol{\Lambda}^{-1} \mathbf{H})^{-1} \mathbf{H}^T \boldsymbol{\Lambda}^{-1} (\mathbf{y}^F - \mathbf{f}_\theta^M)$ with $\mathbf{H} = (\mathbf{h}^T(\mathbf{x}_1), \dots, \mathbf{h}^T(\mathbf{x}_n))^T$ being an $n \times q$ matrix of the mean basis.

After calling the `rcalibration` function with `method='mle'`, the point estimator $\hat{\theta}_{LS}$ is recorded in the slot `param_est` in the `rcalibration` class. The MLE is a computationally cheap way to obtain estimates of calibration parameter vector θ . After obtaining the MLE, we can plug the MLE into the computer model for predicting reality: $f^M(x_i, \hat{\theta}) + h(x_i) \hat{\theta}_m^{LS}$, using the `predict` function.

We also implement the MCMC algorithm for Bayesian inference. The posterior distribution follows

$$p(\theta_m, \sigma_0^2, \theta | \mathbf{y}^F) \propto p(\mathbf{y}^F | \theta, \theta_m, \sigma_0^2) \pi(\theta) \pi(\theta_m, \sigma_0^2). \quad (3)$$

We assume an objective prior for the mean and variance parameters $\pi(\theta_m, \sigma_0^2) \propto 1/\sigma_0^2$. The posterior distribution $p(\theta_m, \sigma_0^2 | \theta, \mathbf{y}^F)$ can be sampled by the Gibbs algorithm, since the prior is conjugate. We assume that the default choice of the prior of the calibration parameters $\pi(\theta)$ is uniform over the parameter space and sample $(\theta | \mathbf{y}^F, \theta_m, \sigma_0^2)$ using the Metropolis algorithm. The posterior distributions and predictions from the posterior samples will be discussed in the Appendix.

Here we show one example from [Bayarri et al. \(2007b\)](#), where data are observed from unknown reality: $y^F(x) = 3.5 \exp(-1.7x) + 1.5 + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 0.3^2)$. We have 30 observations at 10 inputs in the domain $x_i \in [0, 3]$, for $i = 1, 2, \dots, 10$, each containing 3 replications. The computer model is $f^M(x, \theta) = 5 \exp(-\theta x)$ with $\theta \in [0, 50]$. The goal is to estimate the calibration parameters and predict the unknown reality ($3.5 \exp(-1.7x) + 1.5$) at $x \in [0, 5]$. The observable inputs and observations from [Bayarri et al. \(2007b\)](#) can be generated by the code below.

```
R> input=c(.110, .432, .754, 1.077, 1.399, 1.721, 2.043, 2.366, 2.688, 3.010)
R> n=length(input)
R> k=3
R> output=t(matrix(c(4.730,4.720,4.234,3.177,2.966,3.653,1.970,2.267,2.084,2.079,
+ 2.409,2.371, 1.908,1.665,1.685, 1.773,1.603,1.922,1.370,1.661,
+ 1.757, 1.868,1.505,1.638,1.390, 1.275,1.679,1.461,1.157,1.530),k,n))
R> Bayarri_07<-function(x,theta){
+ 5*exp(-x*theta)
+ }
R> theta_range=matrix(c(0,50),1,2)
```

Here the mathematical model has a closed-form expression so we code it as a function called `Bayarri_07`. The parameter range of calibration parameter θ is given by a vector called `theta_range`.

The no-discrepancy calibration is implemented by the code below.

```
R> set.seed(1)
R> X=matrix(1,n,1)
R> m_no_discrepancy=rcalibration(input, output,math_model = Bayarri_07,
+ theta_range = theta_range, X =X,
+ have_trend = T,discrepancy_type = 'no-discrepancy')
R> m_no_discrepancy
type of discrepancy function: no-discrepancy
number of after burn-in posterior samples: 8000
0.1385 of proposed calibration parameters are accepted
median and 95% posterior credible interval of calibration parameter 1 are
2.951997 2.264356 3.930725
```

Here we use a constant mean basis using the mean (or trend) argument `X`, such that the mathematical model is $f^M(x, \theta) + \mu$. We draw 10,000 posterior samples with the first 2,000 samples used as burn-in samples. One can adjust the number of posterior and burn-in samples by argument `S` and `S_0` in the calibration function, respectively. We found that the median of the posterior samples of θ is 2.95, and the 95% posterior interval is around (2.26, 3.93). Around 14% of the samples were accepted by the Metropolis algorithm. Here the first p_θ terms in `sd_proposal` is the standard deviation of the proposal distribution of the calibration parameter, where the default choice is 0.05, and the next $p_x + 1$ parameters are the standard deviation of the inverse logarithm of the range parameter and nugget parameters in GaSP and S-GaSP calibration, discussed in the next subsection. The code below reduces `sd_proposal` to 0.025 of the range of the parameter space to make a larger proportion of the posterior

samples accepted by the algorithm:

```
R> m_no_discrepancy_small_sd=rcalibration(input, output,math_model = Bayarri_07,
+                                         theta_range = theta_range,
+                                         sd_proposal = c(rep(0.025,dim(theta_range)[1]),
+                                         rep(0.25,dim(as.matrix(input) )[2]),0.25),
+                                         X =X, have_trend = T,discrepancy_type = 'no-discrepancy')
R> m_no_discrepancy_small_sd
type of discrepancy function: no-discrepancy
number of after burn-in posterior samples: 8000
0.2613 of proposed calibration parameters are accepted
median and 95% posterior credible interval of calibration parameter 1 are
2.935007 2.19352 3.933492
```

Around 26% of the posterior samples are accepted. In this example, the median and 95% posterior credible intervals are similar to the one having a proposal distribution with smaller standard deviation.

The following code gives the predictions and the 95% predictive interval of the reality for 200 test inputs equally spaced at $x \in [0, 5]$. Since we use a constant trend in model calibration, we also specify a constant trend for making predictions, through the argument `X_testing` in the `predict` function:

```
R> testing_input=seq(0,5,5/199)
R> X_testing=matrix(1,length(testing_input),1)
R> m_no_discrepancy_pred=predict(m_no_discrepancy,testing_input,math_model=Bayarri_07,
+                                   interval_est=c(0.025, 0.975),X_testing=X_testing)
```

Posterior samples of the calibration parameter θ and the mean parameter θ_m in the no-discrepancy calibration are plotted as the red rectangles in the right panel in Figure 2. The predictive mean and 95% predictive interval are graphed as the red curve and grey shaded area in the left panel, respectively. Compared to models with an estimated discrepancy function, the posterior samples from the no-discrepancy calibration are more concentrated, and the average length of 95% predictive interval is shorter. However, a large proportion of reality at $x \in [0, 1.5]$ is not covered by the 95% posterior credible interval, due to large discrepancy between the mathematical model and reality in this domain. Next, we introduce two discrepancy models to solve this problem.

3.2 Gaussian stochastic process models of discrepancy functions

The discrepancy function can be modeled as a GaSP, meaning that for any $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, the marginal distribution of the discrepancy function follows a multivariate normal distribution:

$$(\delta(\mathbf{x}_1), \dots, \delta(\mathbf{x}_n))^T \mid \boldsymbol{\mu}, \sigma^2, \mathbf{R} \sim \mathcal{MN}(\boldsymbol{\mu}, \sigma^2 \mathbf{R}),$$

where $\boldsymbol{\mu} = (\mu(\mathbf{x}_1), \dots, \mu(\mathbf{x}_n))^T$ is a vector of the mean (or trend), σ^2 is a variance parameter, and \mathbf{R} is the correlation matrix between outputs.

Similar to the no-discrepancy calibration, we allow user to model the trend as $\mu(\mathbf{x}_i) = \mathbf{h}(\mathbf{x}_i)\boldsymbol{\theta}_m = \sum_{t=1}^q h_t(\mathbf{x}_i)\theta_{m,t}$, where $\mathbf{h}(\mathbf{x}_i)$ is a row vector of mean basis and $\boldsymbol{\theta}_m$ is a vector of trend parameters. Since the trend is often modeled in the computer model, the default value of the trend parameters is $\mathbf{0}$.

The (i, j) th element of the correlation matrix is modeled by a kernel function $R_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$. We implement the separable kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \prod_{l=1}^{p_x} K_l(\mathbf{x}_i, \mathbf{x}_j)$, where K_l models the correlation between the l th coordinate of the observable input, for $l = 1, \dots, p_x$. Some frequently used kernel functions are listed in Table 1. The range parameters γ are estimated by default.

We denote the nugget parameter by the noise variance to signal variance ratio $\eta = \sigma_0^2/\sigma^2$. Marginalizing out the random discrepancy function $\delta(\cdot)$, one has $\mathbf{y}^F \sim \mathcal{MN}(\mathbf{f}_\theta^M + \mathbf{H}\boldsymbol{\theta}_m, \sigma_0^2 \tilde{\mathbf{R}})$, where $\tilde{\mathbf{R}} = \mathbf{R}/\eta + \boldsymbol{\Lambda}$. Here by default $\boldsymbol{\Lambda} = \mathbf{I}_n$, and we allow users to specify the inverse diagonal terms of $\boldsymbol{\Lambda}$ by the argument `weights`: $w_i = 1/\Lambda_{ii}$, for $i = 1, \dots, n$.

The MLE of the trend and noise variance parameters in the GaSP calibration follows $\hat{\boldsymbol{\theta}}_m = (\mathbf{H}^T \tilde{\mathbf{R}}^{-1} \mathbf{H})^{-1} \mathbf{H}^T \tilde{\mathbf{R}}^{-1} (\mathbf{y} - \mathbf{f}_\theta^M)$ and $\hat{\sigma}_0^2 = S_K^2/n$, where $S_K^2 = (\mathbf{y} - \mathbf{f}_\theta^M - \mathbf{H}^T \hat{\boldsymbol{\theta}}_m)^T \tilde{\mathbf{R}}^{-1} (\mathbf{y} - \mathbf{f}_\theta^M - \mathbf{H}^T \hat{\boldsymbol{\theta}}_m)$. Plugging in the MLE of trend and noise variance parameters, one can numerically maximize the profile likelihood to obtain the calibration, kernel and nugget parameters in the GaSP calibration:

$$(\hat{\boldsymbol{\theta}}, \hat{\gamma}, \hat{\eta}) = \underset{(\boldsymbol{\theta}, \gamma, \eta)}{\operatorname{argmax}} \mathcal{L}_K(\boldsymbol{\theta}, \gamma, \eta \mid \hat{\boldsymbol{\theta}}_m, \hat{\sigma}_0^2) \quad (4)$$

where the closed form expression of the profile likelihood $\mathcal{L}_K(\boldsymbol{\theta}, \gamma, \eta \mid \hat{\boldsymbol{\theta}}_m, \hat{\sigma}_0^2)$ with kernel $K(\cdot, \cdot)$ is given in the Appendix.

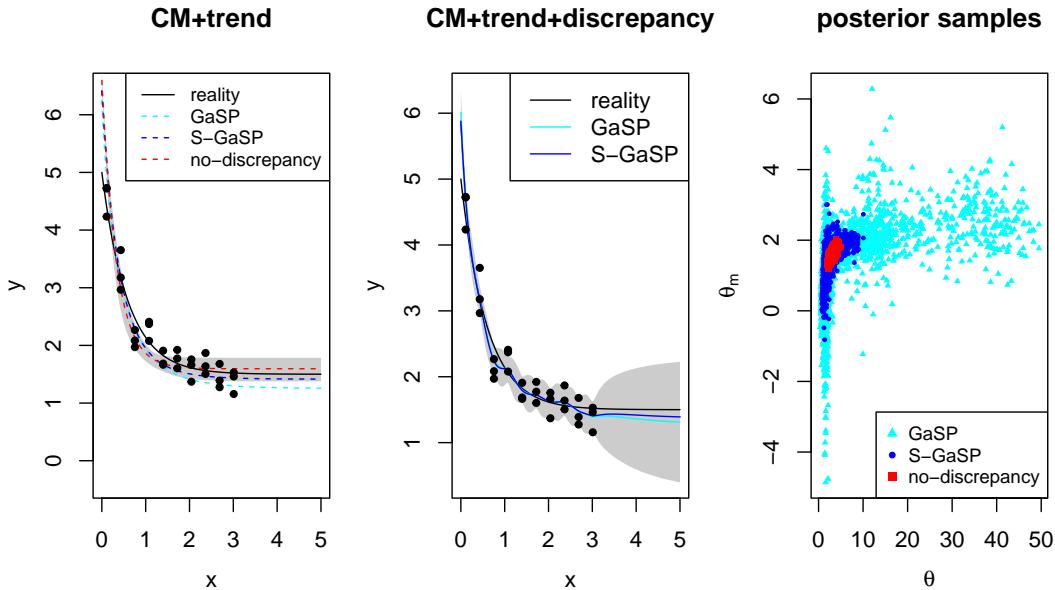


Figure 2: Predictions and posterior samples by the `RobustCalibration` package for the example introduced in Bayarri et al. (2007b). In the left and middle panels, the unknown reality is graphed as black curves, and field observations that contain with 3 replicates at each observable input are plotted as black dots. In the left panel, predictions of reality based on calibrated computer model with trend by GaSP, S-GaSP and no-discrepancy calibration are graphed as light blue, blue and red curves, respectively. ‘CM’ denotes the computer model and the shaded area is the 95% posterior predictive interval by the no-discrepancy calibration. In the middle panel, predictions based on the calibrated computer model, discrepancy, and the trend by GaSP and S-GaSP are graphed as light blue and blue curves, respectively. The shaded area is the 95% predictive interval by the GaSP calibration. After burn-in posterior samples of calibration parameter θ and mean parameter θ_m are plotted in the right panel.

After obtaining the estimation of the parameters, the predictive distribution of reality on any \mathbf{x}^* can be obtained by plugging in the estimator

$$y^R(\mathbf{x}^*) \mid \mathbf{y}^F, \hat{\boldsymbol{\theta}}_m, \hat{\sigma}_0^2, \hat{\boldsymbol{\theta}}, \hat{\gamma}, \hat{\eta} \sim \mathcal{N}(\hat{\mu}(\mathbf{x}^*), \hat{\sigma}_0^2(K^*/\hat{\eta} + \Lambda^*)), \quad (5)$$

where Λ^* is the weight at \mathbf{x}^* set to be 1 by default, and

$$\begin{aligned} \hat{\mu}(\mathbf{x}^*) &= f^M(\mathbf{x}^*, \hat{\boldsymbol{\theta}}) + \mathbf{h}(\mathbf{x}^*)\hat{\boldsymbol{\theta}}_m + \mathbf{r}^T(\mathbf{x}^*)\tilde{\mathbf{R}}^{-1}(\mathbf{y}^F - \mathbf{f}(\mathbf{x}_{1:n}, \hat{\boldsymbol{\theta}}) - \mathbf{H}\hat{\boldsymbol{\theta}}_m), \\ K^* &= K(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{r}^T(\mathbf{x}^*)\tilde{\mathbf{R}}^{-1}\mathbf{r}(\mathbf{x}^*), \end{aligned}$$

with $\mathbf{r}(\mathbf{x}^*) = (K(\mathbf{x}^*, \mathbf{x}_1), \dots, K(\mathbf{x}^*, \mathbf{x}_n))^T$. Though the predictive mean $\hat{\mu}(\mathbf{x}^*)$ may be the most accurate for predictions, the calibrated computer model $f^M(\mathbf{x}^*, \hat{\boldsymbol{\theta}})$ is sometimes of interest for predicting the reality, due to its interpretability. Thus, we recorded three slots for predicting the reality: 1) the calibrated computer model $f^M(\mathbf{x}^*, \hat{\boldsymbol{\theta}})$ by `math_model_mean_no_trend`, 2) The calibrated computer model with trend $f^M(\mathbf{x}^*, \hat{\boldsymbol{\theta}}) + \mathbf{h}(\mathbf{x}^*)\hat{\boldsymbol{\theta}}_m$ by `math_model_mean`, and 3) the predictive mean $\hat{\mu}(\mathbf{x}^*)$ by `mean`. Furthermore, users can also output the predictive interval based on (5) by specifying `interval_est` in the `predict.rcalibration` function. For example, setting `interval_est=c(0.025, 0.975)` gives the 95% predictive interval of the reality. Furthermore, if `interval_data=T`, the predictive interval of the data will be computed.

Numerically computing the MLE can be unstable as the profile likelihood $\mathcal{L}_K(\boldsymbol{\theta}, \gamma, \eta \mid \hat{\boldsymbol{\theta}}_m, \hat{\sigma}_0^2)$ is typically nonlinear and nonconvex with respect to the parameters $(\boldsymbol{\theta}, \gamma, \eta)$. To obtain the uncertainty assessment of the estimates and to avoid instability in the numerical search in computing the MLE, we implement the MCMC algorithm for Bayesian inference. The posterior distribution is

$$p(\boldsymbol{\theta}_m, \sigma_0^2, \boldsymbol{\theta}, \gamma, \eta \mid \mathbf{y}^F) \propto p_K(\mathbf{y}^F \mid \boldsymbol{\theta}, \gamma, \eta, \boldsymbol{\theta}_m, \sigma_0^2) \pi(\boldsymbol{\theta}) \pi(\boldsymbol{\theta}_m, \sigma_0^2, \gamma, \eta), \quad (6)$$

where $p_K(\cdot)$ is a multivariate normal density with covariance function of the discrepancy being $\sigma^2 K(\cdot, \cdot)$,

GaSP calibration	Product kernel ($K(\mathbf{d}) = \prod_{l=1}^{p_x} K_l(d_l)$)
Matérn $\alpha = 5/2$	$K_l(d_l) = \left(1 + \frac{\sqrt{5}d_l}{\gamma_l} + \frac{5d_l^2}{3\gamma_l^2}\right) \exp\left(-\frac{\sqrt{5}d_l}{\gamma_l}\right)$
Matérn $\alpha = 3/2$	$K_l(d_l) = \left(1 + \frac{\sqrt{3}d_l}{\gamma_l}\right) \exp\left(-\frac{\sqrt{3}d_l}{\gamma_l}\right)$
Power exponential	$K_l(d_l) = \exp\left\{-\left(\frac{d_l}{\gamma_l}\right)^{\alpha_l}\right\}, 0 < \alpha_l \leq 2$
S-GaSP calibration	Scaled kernel ($K_{Z_d}(\mathbf{x}_a, \mathbf{x}_b) = K(\mathbf{x}_a, \mathbf{x}_b) - \mathbf{r}^T(\mathbf{x}_a)\mathbf{R}_z^{-1}\mathbf{r}(\mathbf{x}_b)$)

Table 1: Kernel functions implemented in **RobustCalibration**. For any $\mathbf{x}_a, \mathbf{x}_b \in \mathcal{X}$, denote $\mathbf{d} = \mathbf{x}_a - \mathbf{x}_d = (d_1, \dots, d_{p_x})^T$. For any kernel K , the discretized scaled kernel $K_{Z_d}(\mathbf{x}_a, \mathbf{x}_b)$ with discretization points on observed points $\mathbf{x}_1, \dots, \mathbf{x}_n$ is implemented in the S-GaSP calibration, where $\mathbf{R}_z := \mathbf{R} + n\mathbf{I}_n/\lambda_z$, and $\mathbf{r}(\mathbf{x}) = (K(\mathbf{x}, \mathbf{x}_1), \dots, K(\mathbf{x}, \mathbf{x}_n))^T$ for any \mathbf{x} .

and $\pi(\theta)$ is the prior of the calibration parameters, assumed to be uniform over the parameter space. The prior of the mean, variance, kernel and nugget parameters takes the form below

$$\pi(\theta_m, \sigma_0^2, \gamma, \eta) \propto \frac{\pi(\gamma, \eta)}{\sigma_0^2},$$

where the usual reference prior of the trend and variance parameters $\pi(\theta_m, \sigma_0^2) \propto 1/\sigma_0^2$ is assumed. We use jointly robust prior for the kernel and nugget parameters $\pi(\gamma, \eta)$, as it has a similar tail density decay rate as the reference prior, but it is easier and more robust to compute (Gu, 2019). Closed form expressions of the posterior distributions and MCMC algorithm are provided in the Appendix.

In **RobustCalibration**, after burn-in posterior samples $(\theta_m^{(i)}, (\sigma_0^2)^{(i)}, \theta^{(i)}, \gamma^{(i)}, \eta^{(i)})$ are recorded for $i = S_0 + 1, \dots, S$. Users can record a subset of posterior samples to reduce storage by thinning the Markov chain. For instance, `thinning=5` in the `rcalibration` function records 1/5 of posterior samples.

We also output three types of predictions. For instance, suppose we obtain S posterior samples with first S_0 samples used as burn-in samples, the predictive mean of the reality can be computed by

$$\hat{\mu}(\mathbf{x}^*) = \sum_{i=S_0+1}^S f^M(\mathbf{x}^*, \theta^{(i)}) + \mathbf{h}(\mathbf{x}^*)\theta_m^{(i)} + (\mathbf{r}^{(i)}(\mathbf{x}^*))^T (\tilde{\mathbf{R}}^{(i)})^{-1} (\mathbf{y}^F - \mathbf{f}^M(\mathbf{x}_{1:n}, \theta^{(i)}) - \mathbf{H}\theta_m^{(i)}),$$

where $\mathbf{r}^{(i)}(\mathbf{x}^*)$ and $\tilde{\mathbf{R}}^{(i)}$ are obtained by plugging in the i th posterior samples. The predictive interval can also be computed by specifying the argument `interval_est` in `predict.rcalibration` function.

Code below implements the GaSP calibration of parameter estimation and predictions for the example in (Bayarri et al., 2007b) that was discussed in the previous subsection.

```
R> m_gaspc=rcalibration(input, output,math_model = Bayarri_07,theta_range = theta_range,
+                           X=X, have_trend = T,discrepancy_type = 'GaSP')
R> m_gaspc_pred=predict(m_gaspc,testing_input,math_model=Bayarri_07,
+                           interval_est=c(0.025, 0.975),X_testing=X_testing)
```

Predictions and posterior samples from the GaSP calibration are plotted in Figure 2. The associated predictive error is shown in Table 2. Comparing the first two rows in Table 2, the predictive RMSE in the no-discrepancy calibration is around 0.25, and it decreases to 0.15, after adding the discrepancy function modeled by GaSP. Around 97.5% of the held-out truth is covered by the 95% predictive interval of the reality in the GaSP calibration. In comparison, the 95% predictive interval of the reality in the no-discrepancy calibration only covers around 80% of the held-out reality.

As shown in the right panel in Figure 2, the posterior samples from GaSP spread over a large range, reflecting the large uncertainty in parameter estimation. This is because the discrepancy modeled by GaSP with the frequently used kernel listed in Table 1 is very flexible. As a result, the calibrated computer model by GaSP can be less accurate in predicting the reality. To address this problem, we introduce a new approach that induces a scaled kernel to constrain the discrepancy function.

3.3 Scaled Gaussian stochastic process models of discrepancy functions

Scaling the kernel of GaSP for model calibration was introduced in Gu and Wang (2018), where the random L_2 distance between the discrepancy model was scaled to have more prior probability mass at small values, as small distance indicates the computer model fits the reality well. Note that we

	RMSE (CM+trend)	RMSE (with discrepancy)	$P_{CI}(95\%)$	$L_{CI}(95\%)$
GaSP	0.253	0.151	0.975	0.880
S-GaSP	0.228	0.131	0.955	1.15
No-discrepancy	0.250	/	0.795	0.409

Table 2: Predictive accuracy and uncertainty assessment. The RMSE of the out-of-sample prediction based on the calibrated computer model (CM) and trend are shown in the second column. The predictive RMSE by the summation of the calibrated computer model, trend and discrepancy is given in the third column. The proportion of the held-out reality covered in the 95% predictive interval, and average lengths of predictive intervals are given in the last two columns, respectively.

leave σ^2 as a free parameter to be estimated from data, and thus S-GaSP is still a flexible model of discrepancy.

In [RobustCalibration](#), we implemented the discretized S-GaSP to scale the random mean squared error between the reality and computer model:

$$\delta_{Z_d}(\mathbf{x}) = \left\{ \delta(\mathbf{x}) \mid \frac{1}{n} \sum_{i=1}^n \delta(\mathbf{x}_i)^2 = Z_d \right\}, \quad (7)$$

with the subscript ‘d’ denoting discretization, and the density of Z_d is defined as

$$p_{Z_d}(z) = \frac{g_{Z_d}(z) p_\delta(Z_d = z)}{\int_0^\infty g_{Z_d}(t) p_\delta(Z_d = t) dt}, \quad (8)$$

where $p_\delta(Z_d = z)$ is the density of Z_d induced by the GaSP with kernel $K(\cdot, \cdot)$, and $g_Z(\cdot)$ is a nondecreasing function that places more probability mass on smaller values of the L_2 loss. For a frequently used kernel function in Table 1, the probability measure of the GaSP places a large probability at large L_2 loss $p_\delta(Z_d = z)$ when the correlation is large, dragging the calibrated computer model away from the reality. In the S-GaSP calibration, the measure for Z_d was scaled to have more probability mass near zero by a scaling function $g_{Z_d}(z)$. The default scaling function is assumed to be an exponential distribution,

$$g_{Z_d}(z) = \frac{\lambda_z}{2\sigma^2} \exp\left(-\frac{\lambda_z z}{2\sigma^2}\right), \quad (9)$$

where λ_z controls how large the scaling factor is. Large λ_z concentrates more prior probability mass at the origin. As shown in [Gu et al. \(2022\)](#), $\lambda_z \propto \sqrt{n}$ guarantees the convergence of the calibration parameters to the ones that minimize the L_2 distance between the reality and computer model. We let the default choice be $\lambda_z = (\lambda ||\tilde{\gamma}||)^{-1/2}$, where $\lambda = (\sigma_0^2/\sigma^2 n)$ is the regularization parameter in the kernel ridge regression, and $\tilde{\gamma} = (\gamma_1/L_1, \dots, \gamma_{p_x}/L_{p_x})^T$, with L_i being the length of domain of the i th coordinate of the calibration parameter. We allow users to specify λ_z via the argument `lambda_z` in the `rcalibration` function.

The default choice of the scaling function in (9) has computational advantages. After marginalizing out Z_d , it follows from Lemma 2.4 in [Gu and Wang \(2018\)](#) that $\delta_{Z_d}(\cdot)$ has the covariance function

$$\sigma^2 K_{Z_d}(\mathbf{x}_a, \mathbf{x}_b) = \sigma^2 \left\{ K(\mathbf{x}_a, \mathbf{x}_b) - \mathbf{r}^T(\mathbf{x}_a) \left(\mathbf{R} + \frac{n\mathbf{I}_n}{\lambda_z} \right)^{-1} \mathbf{r}(\mathbf{x}_b) \right\}, \quad (10)$$

for any $\mathbf{x}_a, \mathbf{x}_b$, where $\mathbf{r}(\mathbf{x}) = (K(\mathbf{x}, \mathbf{x}_1), \dots, K(\mathbf{x}, \mathbf{x}_n))^T$ for any \mathbf{x} . The covariance matrix of the S-GaSP model of $(\delta_z(\mathbf{x}_1), \dots, \delta_z(\mathbf{x}_n))^T$ follows

$$\sigma^2 \mathbf{R}_z = \sigma^2 \left\{ \mathbf{R} - \mathbf{R} \left(\mathbf{R} + \frac{n\mathbf{I}_n}{\lambda_z} \right)^{-1} \mathbf{R} \right\}. \quad (11)$$

The closed-form expression in Equation (10) avoids sampling $(\delta_z(\mathbf{x}_1), \dots, \delta_z(\mathbf{x}_n))$ to obtain Z_d from the posterior distribution, which greatly improves the computational speed.

For any covariance function $K(\cdot, \cdot)$ in the GaSP calibration, a scaled kernel was induced in S-GaSP calibration, shown in the last row of Table 1. The S-GaSP model of the discrepancy can be specified by the argument `discrepancy_type='S-GaSP'` in the `rcalibration` function. Similar to the model of GaSP-calibration, we also provide MLE and posterior samples for estimating the parameters, with arguments `method='mle'` and `method='post_sample'`, respectively. The MLE of parameters in S-GaSP

are computed by maximizing the profile likelihood

$$(\hat{\theta}, \hat{\gamma}, \hat{\eta}) = \underset{(\theta, \gamma, \eta)}{\operatorname{argmax}} \mathcal{L}_{K_{Z_d}}(\theta, \gamma, \eta, | \hat{\theta}_m, \hat{\sigma}_0^2), \quad (12)$$

where $(\hat{\theta}_m, \hat{\sigma}_0^2)$ denotes the MLE of trend and noise variance parameters in S-GaSP calibration, and $\mathcal{L}_{K_{Z_d}}$ is the profile likelihood with respect to the scaled kernel K_{Z_d} in (10). Furthermore, point predictions and intervals can be obtained by the `predict.rcalibration` function as well.

The code below gives S-GaSP calibration and prediction of the example in Bayarri et al. (2007b).

```
R> m_sgasp=rcalibration(input, output,math_model = Bayarri_07,theta_range = theta_range,
+                         X =X, have_trend = T,discrepancy_type = 'S-GaSP')
R> m_sgasp_pred=predict(m_sgasp,testing_input,math_model=Bayarri_07,
+                         interval_est=c(0.025, 0.975),X_testing=X_testing)
```

Predictions from S-GaSP are plotted as blue curves in Figure 2 and numerical comparisons are given in Table 2. With the discrepancy modeled by S-GaSP, the prediction of reality is more accurate at $x \in [0.5, 1.5]$, compared to no-discrepancy calibration. Note that predictions from the S-GaSP model also extrapolate the reality at $x \in [3, 5]$ accurately, because the calibrated computer model is closer to the truth. Furthermore, the 95% predictive interval by S-GaSP covers around 95% of the held-out samples.

To compare and illustrate different calibration approaches for differential equations, we discuss another example of the Lorenz-96 system used in modeling atmospheric dynamics (Lorenz, 1996). The mathematical model is written as the following differential equations:

$$\dot{x}_j(t) = (x_{j+1}(t) - x_{j-2}(t))x_{j-1}(t) - x_j(t) + \theta, \quad (13)$$

for $j = 1, \dots, k$ states, with $k = 40$ and θ is a scalar value of the force. We further denote that $x_{-1} = x_{k-1}$, $x_0 = x_k$ and $x_{k+1} = x_1$ for any t . The latent variable x_j can model the atmospheric quantities, such as temperature or pressure, measured at k positions along a constant latitude circle. This model is widely used for data assimilation (Maclean and Spiller, 2020; Brajard et al., 2020).

We consider model calibration in two scenarios:

$$\text{Scenario 1: } y_j(t) = x_j(t) + \epsilon, \quad (14)$$

$$\text{Scenario 2: } y_j(t) = x_j(t) + 2t \sin\left(\frac{2\pi j}{k}\right) + \epsilon, \quad (15)$$

for $j = 1, \dots, k$ and $\epsilon \sim \mathcal{N}(0, 1)$ being an independent Gaussian noise. In the first scenario, the mathematical model is the true model and in the second scenario, a discrepancy term is included for simulating the field data. The discrepancy is treated as unknown. We initialize the states by a multivariate normal distribution with the covariance generated from a Wishart distribution with k degrees of freedom and the scale matrix being a diagonal matrix. We use the Runge Kutta method with order 4 to numerically solve the system. Since the computer model is fast, we do not include an emulator. We simulate the reality at 40 time points, with a time step of 0.05 between the time points.

The unobserved reality simulated by the Lorenz-96 system is plotted in upper panel in Figure 3. Only 5% observations (i.e. 2 observations at each time point, plotted as black circles in upper middle panel) are available for model calibration. The posterior parameters of no-discrepancy calibration, GaSP and S-GaSP calibration are plotted in the upper right panel. Since the computer model (Lorenz-96) is the true model in this scenario, the no-discrepancy calibration is expected to perform well. Even though one allows a flexible discrepancy function, modeled as GaSP or S-GaSP, it seems parameters are estimated reasonably well in both approaches. In all methods, posterior samples are close to the true parameter ($\theta = 8$), and the range of the posterior samples is narrow compared to the parameter range $[-20, 20]^2$. Furthermore, the estimate states by the calibrated computer model and the reality are plotted in the lower panels in Figure 3. All methods have small estimation errors, as the parameters are estimated well.

The reality, observations, posterior samples, and predictions from the different calibration models of a discrepancy-included Lorenz-96 system are graphed in Figure 4. Note that since the discrepancy is included, there is no true calibration parameter. The states estimated by the no-discrepancy calibration model (shown in the lower left panel) have a relatively large error. This is not surprising as the Lorenz-96 system is a misspecified model. The predictive means of GaSP and S-GaSP models which includes both the calibrated computer model and discrepancy function is more accurate, as both models capture the discrepancy between the reality and computer model. The estimation error of all models is larger than the ones when there is no discrepancy. This is because the measurement error is large and we only observe 2 states at each time point. Increasing the number of observations or reducing the variance of measurement error can improve predictive accuracy.

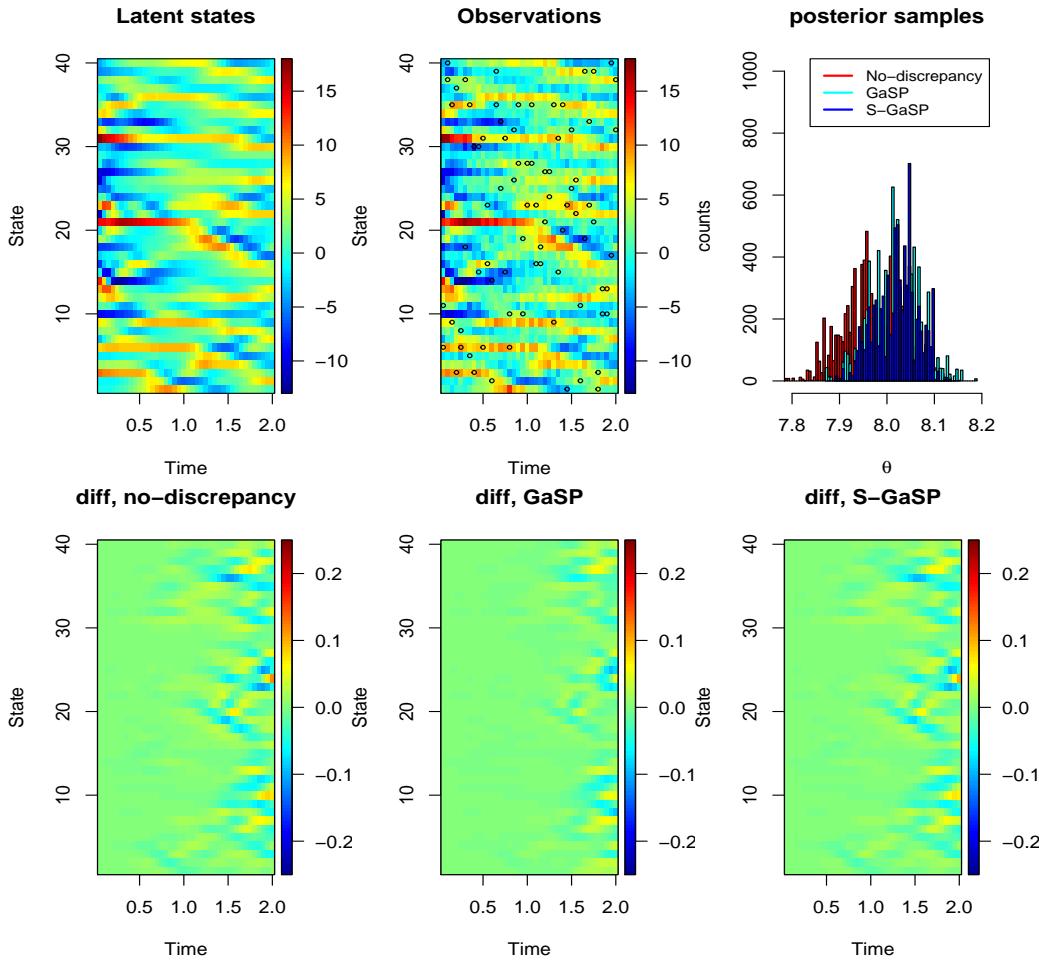


Figure 3: The reality and full observations of Lorenz-96 system in Scenario 1 are plotted in the upper left panel and upper middle panel, respectively, where the black circles are the 5% observations used for model calibration. Posterior samples of calibration parameters of different calibration methods are plotted in the upper right panel. The lower panels give the difference between the reality and calibrated computer model of all latent states.

3.4 Calibration with repeated experiments

Repeated experiments or replicates are commonly used in experiments, as they are helpful for assessing the experimental error. Consider, for example, k_i replicates available for each observable input \mathbf{x}_i , denoted as $\mathbf{y}_i^F = (y_1^F(\mathbf{x}_i), \dots, y_{k_i}^F(\mathbf{x}_i))^T$. Denote $\bar{\mathbf{y}}^F = (\sum_{j=1}^{k_i} y_j^F(\mathbf{x}_i)/k_i, \dots, \sum_{j=1}^{k_n} y_j^F(\mathbf{x}_n)/k_n)^T$ the aggregated data. The probability density of the field observations can be written as:

$$\begin{aligned} p(\mathbf{y}_1^F, \dots, \mathbf{y}_n^F | \boldsymbol{\mu}, \boldsymbol{\delta}, \boldsymbol{\theta}, \sigma_0^2) &= (2\pi\sigma_0^2)^{-\frac{\sum_{i=1}^n k_i}{2}} \exp\left(-\frac{\sum_{i=1}^n \omega_i \sum_{j=1}^{k_i} (y_j^F(\mathbf{x}_i) - f^M(\mathbf{x}_i, \boldsymbol{\theta}) - \mu_i - \delta(\mathbf{x}_i))^2}{2\sigma_0^2}\right) \\ &= (2\pi\sigma_0^2)^{-\frac{\sum_{i=1}^n k_i}{2}} \exp\left(-\frac{S_f^2}{2\sigma_0^2} - \frac{\sum_{i=1}^n k_i \omega_i (\bar{y}_i^F - f^M(\mathbf{x}_i, \boldsymbol{\theta}) - \mu_i - \delta(\mathbf{x}_i))^2}{2\sigma_0^2}\right) \end{aligned}$$

where $S_f^2 = \sum_{i=1}^n \omega_i \sum_{j=1}^{k_i} (y_j^F(\mathbf{x}_i) - \bar{y}_i^F)^2 = \sum_{i=1}^n \omega_i (\mathbf{y}_i^F - \bar{y}_i^F \mathbf{1}_{k_i})^T (\mathbf{y}_i^F - \bar{y}_i^F \mathbf{1}_{k_i})$ with the subscript 'f' denoting the field observations. After integrating out the discrepancy term, assumed to be modeled as a GaSP as an example, the marginal likelihood of the parameters follows

$$\mathcal{L}_K(\boldsymbol{\theta}, \boldsymbol{\theta}_m, \sigma_0^2, \gamma, \eta) \propto (\sigma_0^2)^{-\frac{\sum_{i=1}^n k_i}{2}} |\tilde{\mathbf{R}}|^{-\frac{1}{2}} \exp\left\{-\frac{(\bar{\mathbf{y}}^F - \mathbf{f}_\theta^M - \mu \mathbf{1}_n)^T \tilde{\mathbf{R}}^{-1} (\bar{\mathbf{y}}^F - \mathbf{f}_\theta^M - \mu \mathbf{1}_n)}{2\sigma_0^2} - \frac{S_f^2}{2\sigma_0^2}\right\}, \quad (16)$$

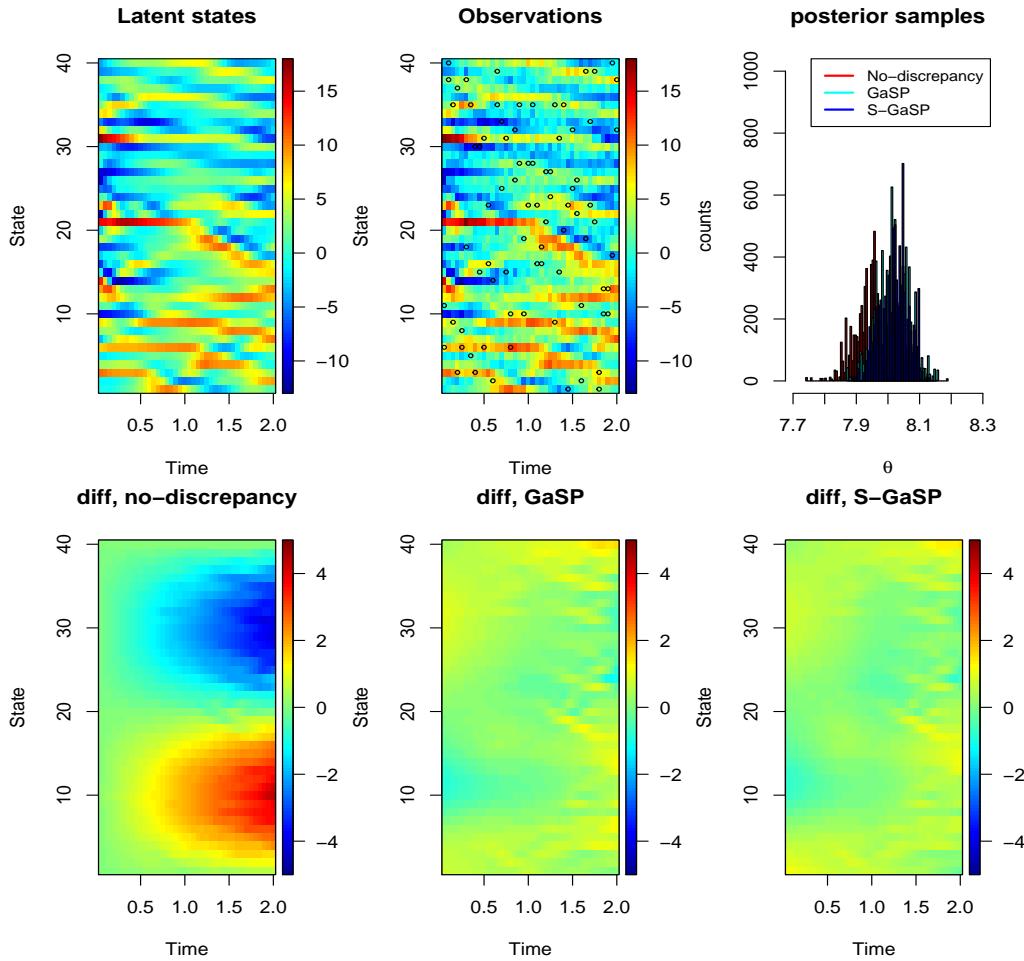


Figure 4: The reality of the discrepancy-included Lorenz-96 system (Scenario 2) is plotted in the upper left panel. The simulated observations are plotted in the upper middle panel, where the 5% of the observations plotted as black circles are used in model calibration. Posterior samples of calibration parameters of different calibration methods are plotted in the upper right panel. The lower panels show the differences between the reality and the predictive mean of all latent states.

where $\tilde{\mathbf{R}} = (\mathbf{R}/\eta + \tilde{\boldsymbol{\Lambda}})$, with $\tilde{\boldsymbol{\Lambda}}$ being an diagonal matrix with diagonal entry $\Lambda_{ii} = 1/(\omega_i k_i)$, for $i = 1, 2, \dots, n$, and $\eta = \sigma_0^2/\sigma^2$. The S-GaSP calibration with replications can also be defined, by simply replacing \mathbf{R} by \mathbf{R}_z where the (i, j) th term is defined by the scaled kernel in Table 1.

The likelihood in (16) has a clear computational advantage. If one directly computes the covariance and its inverse, the computational complexity is $O((\sum_{i=1}^n k_i)^3)$, whereas the computational order by Equation (16) is only $O(n^3) + O(\sum_{i=1}^n k_i)$. Suppose $k_i = k$, for $i = 1, \dots, n$, using Equation (16) is around k^3 times faster than directly computing the likelihood, with no loss of information.

One can specify replications by inputting a $n \times k$ matrix into the argument `observations` in the `rcalibration` function for scenarios with n observable inputs, each containing k repeated measurements. Or one can give a list of observations in the `rcalibration` function with size n , where each element in the list contains k_i values of replicates, for $i = 1, \dots, n$. For observations with replicates, MLE and posterior sampling were implemented in the `rcalibration` function, and predictions were implemented in the `predict.rcalibration` function.

3.5 Statistical emulators

Users can specify a mathematical model via the `math_model` argument in `rcalibration` function if the closed-form expression of the mathematical model is available. However, closed-form solutions of physical systems are typically unavailable, and numerical solvers are required. Simulating the outputs from computer models can be slow. For instance, the TITAN2D computer model, which simulates pyroclastic flows for geological hazard quantification, takes roughly 8 minutes per set

of inputs (Simakov et al., 2019). As hundreds of thousands of model runs are sometimes required for model calibration (Anderson et al., 2019), directly computing the computer model by numerical methods is often prohibitive. In this scenario, we construct a statistical emulator as a surrogate model, to approximate the computer simulation, based on a small number of computer model runs.

To start, assume that we have obtained the output of the computer model at D input design points, $(\theta_1, \dots, \theta_D)$, often selected from an “space-filling” design, such as the Latin hypercube design (Santner et al., 2003). There are two types of computer model outputs: 1) scalar-valued outputs: $f^M(\theta) \in \mathbb{R}$, and 2) vector-valued outputs at k coordinates $f^M(x_{1:k}, \theta) \in \mathbb{R}^k$, where k can be as large as 10^6 . Various packages are available for fitting scalar-valued GP emulators, such as **DiceKriging** (Roustant et al. (2012)), **GPfit** (MacDonald et al. (2015)), and **RobustGaSP** (Gu et al. (2019)). Some of these packages were used in Bayesian model calibration packages (Palomo et al., 2015; Carmassi et al., 2018). However, the emulator of computer models with vector-valued outputs, a common scenario in different disciplines (Higdon et al., 2008; Ma et al., 2022; Li et al., 2022; Fang et al., 2022) was rarely implemented in the model calibration packages.

In **Robustcalibration**, we call the `rgasp` function and `ppgasp` function from the **RobustGaSP** package for emulating scalar-valued and vector-valued computer model outputs, respectively. The parallel partial Gaussian process (PP-GaSP) by the `ppgasp` function has two advantages. First, computing the predictive mean by the PP-GaSP only takes $O(kD) + O(D^3)$ operations, which is particularly suitable for computer models with a large number of outputs (k). Second, as the covariance over θ is shared across all grids, the estimation of PP-GaSP is more stable than building a separate emulator on each output coordinate. Furthermore, the marginal posterior mode with robust parameterization typically avoids the degenerated estimation of the covariance matrix (Gu (2019)).

To call the emulator, users can select the argument `simulator=0`, and then specify the simulation runs in arguments `input_simul`, and `output_simul`. One can input a vector into `output_simul` to emulate scalar-valued computer models, or a matrix into `output_simul` to call the PP-GaSP emulator for vector-valued outputs. For both scenarios, we use a modular approach to fit the PP-GaSP emulator (Bayarri et al., 2007b; Liu et al., 2009), where the predictive distribution of the emulator depends on simulator runs, but not on field observations. After fitting an emulator, the predictive mean of the PP-GaSP emulator will be used to approximate the computer model at any unobserved θ^* .

We illustrate the efficiency and accuracy of the PP-GaSP emulator for approximating differential equations from Box and Coutie (1956), where the interaction between two chemical substances y_1 and y_2 are modeled as

$$\begin{aligned}\dot{y}_1(t) &= 10^{\theta_1-3}y_1(t), \\ \dot{y}_2(t) &= 10^{\theta_1-3}y_1(t) - 10^{\theta_2-3}y_2(t).\end{aligned}$$

In each of the 6 time points $t = 10, 20, 40, 80, 160$, and 320 , 2 replicates of the second chemical substance are available in Box and Coutie (1956). Initial conditions are $y_1(t = 0) = 100$ and $y_2(t = 0) = 0$. The computer model contains two parameters $\theta_1 \in [0.5, 1.5]$ and $\theta_2 \in [0.5, 1.5]$.

To start, we first use the default method in the `ode` function from the **deSolve** package (Soetaert et al., 2010) to numerically solve the ODEs at a given initial condition and parameter set:

```
library(deSolve)
R> Box_model <- function(time, state, parameters) {
+   par <- as.list(c(state, parameters))
+   with(par, {
+     dM1=-10^{parameters[1]-3}*M1
+     dM2=10^{parameters[1]-3}*M1-10^{parameters[2]-3}*M2
+     list(c(dM1, dM2))
+   })
+ }
R> Box_model_solved<-function(input, theta){
+   init <- c(M1 = 100 , M2 = 0)
+   out <- ode(y = init, times = c(0,input), func = Box_model, parms = theta)
+   return(out[-1,3])
+ }
```

We specify the observations and range of calibration parameters from Box and Coutie (1956) below.

```
R> n=6
R> output=t(matrix(c(19.2,14,14.4,24,42.3,30.8,42.1,40.5,40.7,46.4,27.1,22.3),2,n))
R> input=c(10,20,40,80,160,320)
R> theta_range=matrix(c(0.5,0.5,1.5,1.5),2,2)
###the testing input for emulator should contain the observed inputs
```

```
R> testing_input=as.matrix(seq(1,350,1))
#if observed inputs are not included, then add it
R> set_diff_obs=setdiff(input,testing_input)
R> testing_input=sort(c(testing_input,set_diff_obs))
```

We compare model calibration and predictions based on the numerical solver by the `deSolve` package, and by the GaSP emulator approach below. We first implement the direct approach, where the numerical solver is called to generate posterior samples.

```
R> m_sgasp_time=system.time({
+ m_sgasp=rcalibration(input,output,math_model=Box_model_solved,
+ sd_proposal=c(0.25,0.25,1,1),
+ theta_range=theta_range)
+ m_sgasp_pred=predict(m_sgasp,testing_input,math_model=Box_model_solved,
+ interval_est=c(0.025,0.975)))}
```

We then generate 50 design points from maximin Latin hypercube design by the `lhs` package, and run a numerical solver at these design points. As the simulator outputs a vector $\mathbf{f}^M(t_{1:k}, \boldsymbol{\theta}) \in \mathbb{R}^k$ for calibration parameter, we call the PP-GaSP emulator to predict the output at all time points. The simulator runs are then specified as `input_simul` and `output_simul` in the `rcalibration` function. We let `simul_type=0` to call the emulator instead of the numerical solver to generate posterior samples. The `loc_index_emulator` below gives a subset of the output coordinates for the the PP-GaSP emulator to be predicted and by default, the function will predict all output coordinates.

```
R> m_sgasp_emulator_time=system.time({
+   ##constructing simulation data
+   D=50
+   p=2
+   lhs_sample=maximinLHS(n=D,k=p)
+   input_simul=matrix(NA,D,p)
+   input_simul[,1]=lhs_sample[,1]+0.5
+   input_simul[,2]=lhs_sample[,2]+0.5
+   k_simul=length(testing_input)
+
+   output_simul=matrix(NA,D,k_simul)
+   for(i_D in 1:D){
+     output_simul[i_D,]=Box_model_solved(c(testing_input), input_simul[i_D,])
+   }
+   ##create loc_index_emulator as a subset of the simulation output
+   loc_index_emulator=rep(NA,n)
+   for(i in 1:n){
+     loc_index_emulator[i]=which(testing_input==input[i])
+   }
+
+   ##emulator
+   m_sgasp_with_emulator=rcalibration(input,output,simul_type=0,
+                                         input_simul=input_simul, output_simul=output_simul,simul_nug=T,
+                                         loc_index_emulator=loc_index_emulator,
+                                         sd_proposal=c(0.25,0.25,1,1),
+                                         theta_range=theta_range)
+   m_sgasp_with_emulator_pred=predict(m_sgasp_with_emulator,testing_input)
+ })
R> m_sgasp_time[3]/m_sgasp_emulator_time[3]
elapsed
6.252316
```

The approach with an emulator only costs around 1/6 of the computational time in this example, even though the numerical solver of this simple ODE is fast. For computer models that take a few or hours to run, the emulator approach can substantially reduce the computational cost.

Figure 5 gives predictions and posterior samples based on a numerical solver and the PP-GaSP emulator. First, predictions from the calibrated computer model and the discrepancy function, and by the calibrated computer model itself are close to each other, implying that the discrepancy modeled by a S-GaSP is close to be zero. Second, predictions and posterior distributions from the numerical solver and the emulator are close to each other, meaning that the emulator approximates the solver well. Calibration with an emulator is preferred as it is faster than numerically solving ODEs each time when generating the posterior samples.

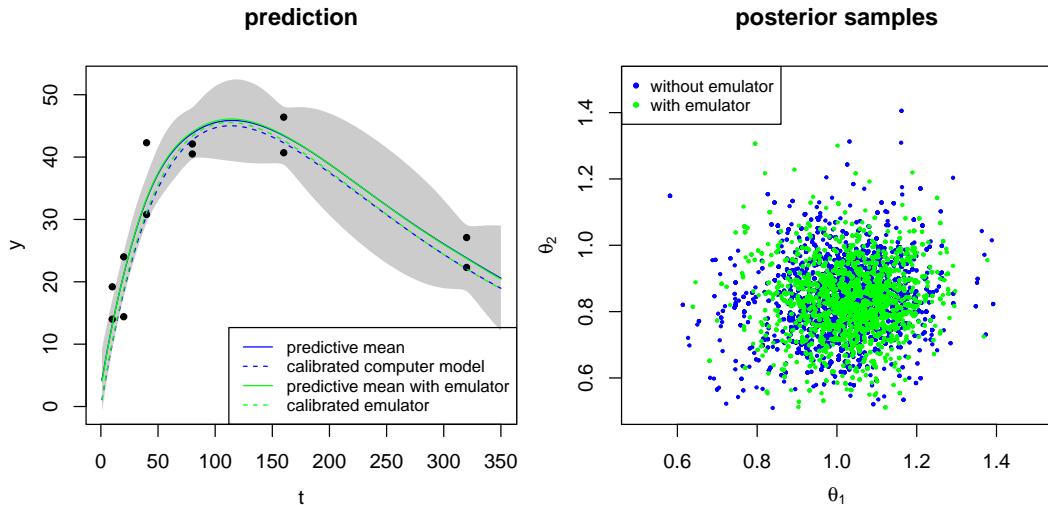


Figure 5: Comparison between the Bayesian model calibration based on the numerical solver and the PP-GaSP emulator of the computer model. In the left panel, the green solid curve is the predictive mean from the calibrated computer model and the discrepancy function, and the blue dashed curves are predictions from the calibrated computer model alone, both of which call the numerical solver for each posterior sample. The same result based on the emulator is plotted as the green curves. The black dots are field observations and the shaded area is the 95% predictive interval of the reality by the approach that uses the PP-GaSP emulator. In the right panel, posterior samples based on the numerical solver and the PP-GaSP emulator are denoted by the blue dots and green dots, respectively.

3.6 Calibration with multiple sources of observations

In reality, one may have observations of different types and they may come from multiple sources. In [Anderson et al. \(2019\)](#), for instance, multiple satellite interferogram and GPS observations measuring the ground deformation during the Kilauea volcano eruption in Summer 2018 are used for calibrating mechanical models that relate the observations to calibration parameters, such as depth and shape of the magma chamber, magma density, pressure change rate, and so on. In these applications, the measurement bias, such as the atmospheric error, can substantially affect the satellite measurements of the ground deformation ([Zebker et al., 1997](#); [Agram and Simons, 2015](#)).

Model calibration using multiple sources of observations was studied in [Gu et al. \(2023\)](#). For each source l , $l = 1, 2, \dots, k$, let the field observations of the l th source at observable input \mathbf{x} be modeled as

$$y_l^F(\mathbf{x}) = f_l^M(\mathbf{x}, \boldsymbol{\theta}) + \delta(\mathbf{x}) + \delta_l(\mathbf{x}) + \mu_l + \epsilon_l, \quad (17)$$

where $f_l^M(\mathbf{x}, \boldsymbol{\theta})$ is the computer model for source l . The discrepancy function is denoted by $\delta(\mathbf{x})$ and the source-specific measurement bias is denoted by $\delta_l(\mathbf{x})$. Measurement bias often appears in the satellite radar interferogram, as atmospheric error could affect the quality of the image. The mean of each data source is denoted by μ_l , and the independent measurement noise is denoted by ϵ_l .

In **RobustCalibration**, we allow users to integrate observations from multiple sources to calibrate computer models by the function `rcalibration_MS` and to make predictions using the function `predict_MS.rcalibration_MS`. The posterior sampling method is implemented for parameter estimation for multiple sources of data. Besides, both GaSP and S-GaSP models can be chosen to model the measurement bias δ_l , for $l = 1, \dots, k$. Similar to calibration model with a single source of data, we allow users to choose a model with no-discrepancy, and with GaSP or S-GaSP model of the discrepancy function δ . Users can choose to have measurement bias or not. We also allow users to integrate different types of measurements for model calibration by using different designs of observable inputs.

To illustrate, we study a synthetic example for calibrating computer models using multiple sources of observations. We assume the l th source of data, $l = 1, \dots, k$, is simulated below

$$y_l^F(x) = \sin(\pi x) + \delta(x) + \delta_l(x) + \epsilon_l(x), \quad (18)$$

where $\delta(\cdot)$ and $\delta_l(\cdot)$ are independently simulated from Gaussian processes with covariance $\sigma^2 K(\cdot, \cdot)$ and $\sigma_l^2 K_l(\cdot, \cdot)$, and the independent noise follows $\epsilon_l(x) \stackrel{i.i.d.}{\sim} N(0, \sigma_0^2)$. We let $\sigma_0 = 0.05$, $\sigma = 0.2$ and $\sigma_l^2 = 0.5 \times l/(l - 1)$, for $l = 1, \dots, k$. The $K(\cdot, \cdot)$ and $K_l(\cdot, \cdot)$ are assumed to follow Matérn kernel with

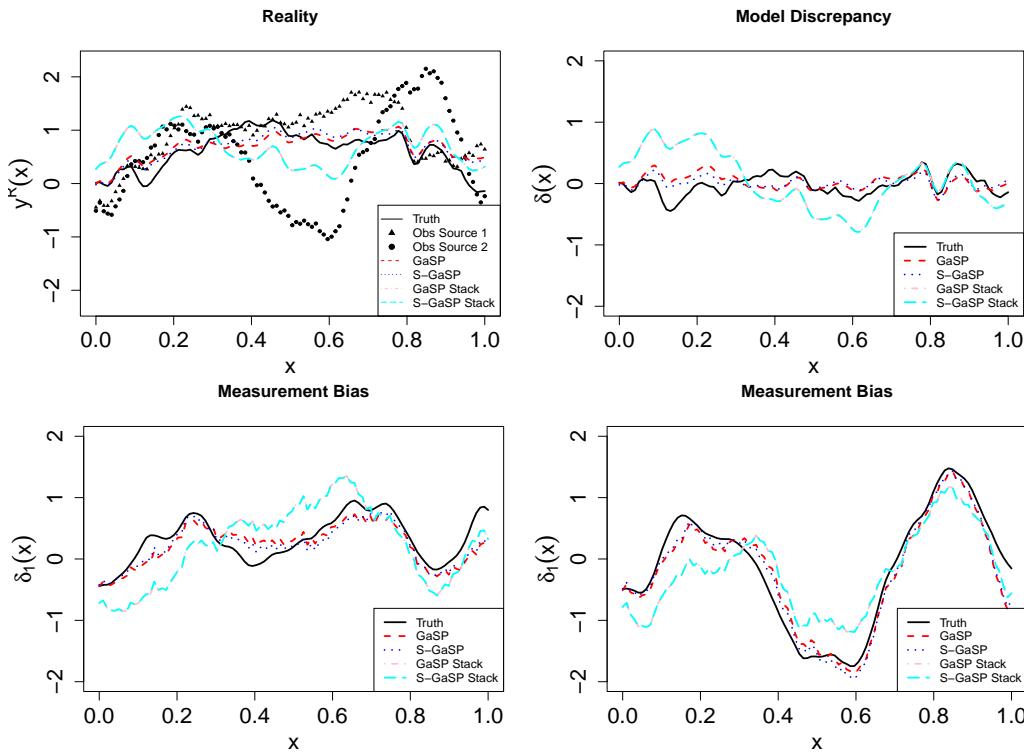


Figure 6: Comparison between modeling individual data and aggregated data for simulation from Equation (18). The upper left and right panels give the reality and model discrepancy, whereas the lower two panels give the measurement bias for the first two sources. The estimation results from GaSP, S-GaSP, GaSP Stack and S-GaSP calibration are plotted as red, blue, pink and cyan curves. The observations from the first two sources are graphed as the triangles and dots in the upper left panel.

roughness parameter being $\alpha = 2.5$, and the range parameter being $\gamma = 1/30$ and $\gamma = 1/10$, respectively. We collect $n = 100$ observations for each source l with x equally spaced from $[0, 1]$. We assume data from $k = 5$ sources are available. Here the mathematical model follows $f_l^M(x, \theta) = \sin(\theta x)$, for $l = 1, \dots, k$, and the reality is $y^R(x) = f_l^M(x, \theta) + \delta(x)$. The goal is to estimate the calibration parameter (θ), the reality ($y^R(x) = f_l^M(x, \theta) + \delta(x)$), model discrepancy ($\delta(x)$), and measurement bias ($\delta_l(x)$).

We compare four methods. The first two methods are GaSP and S-GaSP calibration methods to model individual sources of data, with the discrepancy model specified as a GaSP and S-GaSP respectively. The measurement bias terms are modeled by GaSPs. For instance, the code below implements S-GaSP model of discrepancy and GaSP model of the measurement bias:

```
> model_sgasp=rcalibration_MS(design=input_measurement,observations=output,p_theta=1,
  math_model=math_model, simul_type=rep(1,length(input_measurement)),
  S=5000,S_0=2000,thinning=1,measurement_bias=T,shared_design=input_model,
  have_measurement_bias_recorded=T,
  discrepancy_type=c(rep('GaSP',length(input_measurement)),'S-GaSP'),
  theta_range=matrix(c(-2*pi,2*pi),1,2),sd_proposal_theta=rep(0.02,2));
```

where `math_model`, `input_measurement` and `output` are lists where each contains the model, input and output of the field observations for each source. The `input_model` is a matrix for the input of the discrepancy function.

As modeling each source of the data can be time-consuming for some applications, a common solution is to use the aggregated or stacked data $\bar{y}^R(x) = \sum_{l=1}^k y_l^R(x)/k$ in calibration, as modeling the aggregated data is around k times faster than modeling the individual sources of data. Thus, we also include two methods of modeling the aggregated data for comparison, namely the GaSP Stack and S-GaSP Stack, representing GaSP or S-GaSP discrepancy model, respectively.

In Figure 6, we plot the truth, observations of the first two sources of observations, and the estimates of reality, discrepancy, and measurement bias in two sources from four different methods. Here the GaSP and S-GaSP perform better than the GaSP Stack and S-GaSP Stack calibration methods, as aggregating data leads to loss of information when the data contain measurement bias. The RMSE of reality, discrepancy and measurement bias estimation of different methods are given in the Table 3,

RMSE	reality	discrepancy	measurement bias
GaSP	0.238	0.194	0.247
S-GaSP	0.204	0.159	0.214
GaSP Stack	0.501	0.483	0.504
S-GaSP Stack	0.502	0.484	0.504

Table 3: The RMSE of the reality, discrepancy and measurement bias, where the smaller number indicates a smaller error. For measurement bias, we average the RMSE for each source of data.

which shows the S-GaSP calibration of individual sources of data achieves the highest calibration and predictive accuracy.

Finally, the mean of posterior samples θ from the GaSP, S-GaSP, GaSP Stack and S-GaSP Stack calibration for calibration parameter θ are 2.46, 2.73, 2.15, and 2.17 respectively, whereas the truth $\theta = \pi \approx 3.14$. This is a challenging scenario, as the large measurement bias makes estimation hard. We found model calibration by modeling individual sources of data is more accurate than modeling the aggregated data, whereas modeling the aggregated data is more computationally scalable.

4 Concluding remarks

We have introduced the [RobustCalibration](#) package for Bayesian model calibration and data inversion. This package has implemented a range of estimation methods and models, such as posterior sampling and MLE, for no-discrepancy calibration, GaSP and S-GaSP models of the discrepancy function. We implement statistical emulators for approximating computationally expensive computer models with both scalar-valued output or vector-valued output. The package is applicable to observations from a single source, with or without replications, and to observations from multiple sources or having different types. We illustrate our approaches using mathematical models with closed-form expressions or written as differential equations.

Even though we tried our best to consider different possible scenarios, a comprehensive statistical package for model calibration and data inversion is an ambitious goal. The [RobustCalibration](#) package provides tools for researchers to perform Bayesian model inversion without the need to write emulators or MCMC samplers themselves. We plan to improve the [RobustCalibration](#) package with several specific directions in mind. First, we will make the package more computationally scalable for structured data, such as imaging and time series observations. Furthermore, we plan allow users to specify other prior distributions of the parameters and proposal distributions to sample from in future versions of the package. Third, for problems like the Lorzen-96 system, it may be important to emulate the time evolution operator, if forecasting future states is the goal.

5 Appendix

5.1 Auxiliary facts

- Let $\tilde{\mathbf{R}}_{\gamma,\eta} = \mathbf{R}_{\gamma}/\eta + \Lambda$ be an $n \times n$ positive definite matrix as a function of parameters γ and η . Then for $l = 1, \dots, p_x$:

$$\frac{\partial \log |\tilde{\mathbf{R}}_{\gamma,\eta}|}{\partial \gamma_l} = \text{tr} \left(\frac{\tilde{\mathbf{R}}_{\gamma,\eta}^{-1}}{\eta} \frac{\partial \mathbf{R}_{\gamma}}{\partial \gamma_l} \right) \quad \text{and} \quad \frac{\partial \log |\tilde{\mathbf{R}}_{\gamma,\eta}|}{\partial \eta} = -\text{tr} \left(\frac{\tilde{\mathbf{R}}_{\gamma,\eta}^{-1} \mathbf{R}_{\gamma}}{\eta^2} \right).$$

- Let \mathbf{H} be an $n \times q$ full rank matrix with $q < n$, $\mathbf{Q} = (\tilde{\mathbf{R}}_{\gamma,\eta}^{-1} - \tilde{\mathbf{R}}_{\gamma,\eta}^{-1} \mathbf{H}^T (\mathbf{H}^T \tilde{\mathbf{R}}_{\gamma,\eta}^{-1} \mathbf{H})^{-1} \mathbf{H}^T \tilde{\mathbf{R}}_{\gamma,\eta}^{-1})$ with $\tilde{\mathbf{R}}_{\gamma,\eta} = \mathbf{R}_{\gamma}/\eta + \Lambda$ and \mathbf{y} is an $n \times 1$ vector. Then for $l = 1, \dots, p_x$:

$$\frac{\partial \log(\mathbf{y}^T \mathbf{Q}_{\gamma,\eta} \mathbf{y})}{\partial \gamma_l} = -\frac{\mathbf{y}^T \mathbf{Q}_{\gamma,\eta} \frac{\partial \mathbf{R}_{\gamma}}{\partial \gamma_l} \mathbf{Q}_{\gamma,\eta} \mathbf{y}}{\eta \mathbf{y}^T \mathbf{Q}_{\gamma,\eta} \mathbf{y}} \quad \text{and} \quad \frac{\partial \log(\mathbf{y}^T \mathbf{Q}_{\gamma,\eta} \mathbf{y})}{\partial \eta} = \frac{\mathbf{y}^T \mathbf{Q}_{\gamma,\eta} \mathbf{R}_{\gamma} \mathbf{Q}_{\gamma,\eta} \mathbf{y}}{\eta^2 \mathbf{y}^T \mathbf{Q}_{\gamma,\eta} \mathbf{y}}.$$

- Suppose \mathbf{v}_{θ} is an $n \times 1$ vector as a function of θ and $\tilde{\mathbf{R}}$ is an $n \times n$ positive definite matrix. Then for $i = 1, \dots, p_{\theta}$:

$$\frac{\partial \log(\mathbf{v}_{\theta}^T \tilde{\mathbf{R}}^{-1} \mathbf{v}_{\theta})}{\partial \theta_i} = \frac{2\mathbf{v}_{\theta}^T \tilde{\mathbf{R}}^{-1} \frac{\partial \mathbf{v}_{\theta}}{\partial \theta_i}}{\mathbf{v}_{\theta}^T \tilde{\mathbf{R}}^{-1} \mathbf{v}_{\theta}}.$$

5.2 Likelihood functions and posterior distributions

Posterior distributions of the no-discrepancy calibration. Using the objective prior for the mean and variance parameters, $\pi(\theta_m, \sigma_0^2) \propto 1/\sigma_0^2$, the full conditional posterior distributions of the trend and variance parameters follow:

$$\begin{aligned}\sigma_0^{-2} | \theta_m, \theta &\sim \text{Gamma} \left(\frac{n}{2}, \frac{(\mathbf{y}^F - \mathbf{f}_\theta^M - \mathbf{H}\theta_m)^T \Lambda^{-1} (\mathbf{y}^F - \mathbf{f}_\theta^M - \mathbf{H}\theta_m)}{2} \right), \\ \theta_m | \sigma_0^2, \theta &\sim \mathcal{N} \left(\hat{\theta}_m^{LS}, \sigma_0^{-2} (\mathbf{H}^T \Lambda^{-1} \mathbf{H})^{-1} \right).\end{aligned}$$

The posterior distribution of the calibration parameters follows:

$$p(\theta | \theta_m, \sigma_0^2, \beta, \eta) \propto \exp \left(-\frac{(\mathbf{y}^F - \mathbf{f}_\theta^M - \mathbf{H}\theta_m)^T \Lambda^{-1} (\mathbf{y}^F - \mathbf{f}_\theta^M - \mathbf{H}\theta_m)}{2\sigma_0^2} \right) \pi(\theta),$$

where $\pi(\theta)$ is the prior of calibration parameters, where the default prior distribution is the uniform distribution $\pi(\theta) \propto 1$. We use the Metropolis algorithm to sample the calibration parameters as a block, as we only need to compute the likelihood once in each iteration. The proposal distribution of each coordinate of the calibration parameter vector is chosen as a normal distribution with a pre-specified standard deviation, proportional to the range of the parameters. The default proportion is 0.05, and users can adjust the proportion by changing the first p_θ value in the argument `sd_proposal` in `rcalibration` function.

After obtaining S posterior samples with the first S_0 burn-in samples, the predictive mean of reality based on calibrated computer model and trend by the no-discrepancy calibration can be computed by:

$$\hat{\mu}(\mathbf{x}^*) = \frac{1}{S - S_0} \sum_{i=S_0+1}^S f^M(\mathbf{x}^*, \theta^{(i)}) + \mathbf{h}(\mathbf{x}^*) \theta_m^{(i)}.$$

The posterior credible interval of the reality $y^R(\mathbf{x}^*)$ in the no-discrepancy calibration can be obtained by the empirical quantile of posterior samples $f^M(\mathbf{x}^*, \theta^{(i)}) + \mathbf{h}(\mathbf{x}^*) \theta_m^{(i)}$. For example, the posterior credible interval of $1 - \alpha$ percentile of the data in no-discrepancy calibration can be computed by based on $\frac{1}{S-S_0} \sum_{i=S_0+1}^S \left(f^M(\mathbf{x}^*, \theta^{(i)}) + \mathbf{h}(\mathbf{x}^*) \theta_m^{(i)} + \sqrt{\frac{\sigma_0^{(i)}}{w^*}} Z_\alpha \right)$, where Z_α is an upper α quantile of the standard normal distribution and w^* is the relative test output weight, assumed to be 1 by default.

Likelihood functions and derivatives in GaSP calibration. The profile likelihood function in GaSP calibration follows

$$\mathcal{L}_K(\theta, \gamma, \eta | \hat{\theta}_m, \hat{\sigma}_0^2) \propto |\tilde{\mathbf{R}}|^{-\frac{1}{2}} (S_K^2)^{-\frac{n}{2}}, \quad (19)$$

where $S_K^2 = (\mathbf{y} - \mathbf{f}_\theta^M - \mathbf{H}^T \hat{\theta}_m)^T \tilde{\mathbf{R}}^{-1} (\mathbf{y} - \mathbf{f}_\theta^M - \mathbf{H}^T \hat{\theta}_m)$.

Denote $\mathbf{v}_{\theta_m} = (\mathbf{y} - \mathbf{f}_\theta^M - \mathbf{H}^T \hat{\theta}_m)$. Similar to the no-discrepancy calibration, we call the `lbfsgs` function in the `nloptr` package ([Ypma \(2014\)](#)) for optimization. By facts 1 and 3, directly differentiating the profile likelihood function with respect to kernel and nugget parameters in (19) gives:

$$\begin{aligned}\frac{\partial \log(\mathcal{L}_K(\theta, \gamma, \eta | \hat{\theta}_m, \hat{\sigma}^2))}{\partial \gamma_l} &= -\frac{1}{2} \text{tr} \left(\frac{\tilde{\mathbf{R}}_{\gamma, \eta}^{-1}}{\eta} \frac{\partial \mathbf{R}_\gamma}{\partial \gamma_l} \right) + \frac{n}{2} \times \frac{\mathbf{v}_{\theta_m}^T \mathbf{Q}_{\gamma, \eta} \frac{\partial \mathbf{R}_\gamma}{\partial \gamma_l} \mathbf{Q}_{\gamma, \eta} \mathbf{v}_{\theta_m}}{\eta \mathbf{v}_{\theta_m}^T \mathbf{Q}_{\gamma, \eta} \mathbf{v}_{\theta_m}} \\ \frac{\partial \log(\mathcal{L}_K(\theta, \gamma, \eta | \hat{\theta}_m, \hat{\sigma}^2))}{\partial \eta} &= \frac{1}{2} \text{tr} \left(\frac{\tilde{\mathbf{R}}_{\gamma, \eta}^{-1} \mathbf{R}_\gamma}{\eta^2} \right) + \frac{n}{2} \times \frac{\mathbf{v}_{\theta_m}^T \mathbf{Q}_{\gamma, \eta} \mathbf{R}_\gamma \mathbf{Q}_{\gamma, \eta} \mathbf{v}_{\theta_m}}{\eta^2 \mathbf{v}_{\theta_m}^T \mathbf{Q}_{\gamma, \eta} \mathbf{v}_{\theta_m}} \\ \frac{\partial \log(\mathcal{L}_K(\theta, \gamma, \eta | \hat{\theta}_m, \hat{\sigma}^2))}{\partial \theta_l} &= \frac{n}{2} \times \frac{\mathbf{v}_{\theta_m}^T \mathbf{Q}_{\gamma, \eta} \frac{\partial \mathbf{f}_\theta^M}{\partial \theta_l}}{\mathbf{v}_{\theta_m}^T \mathbf{Q}_{\gamma, \eta} \mathbf{v}_{\theta_m}}\end{aligned}$$

for $l = 1, \dots, p_x$ and $i = 1, \dots, p_\theta$. When the mean function is zero, one can simply replace $\mathbf{Q}_{\gamma, \eta}$ by $\tilde{\mathbf{R}}_{\gamma, \eta}$ in the formula above to obtain the derivatives. Besides, when $\frac{\partial \mathbf{f}_\theta^M}{\partial \theta_l}$ is not available, we use the numerical derivatives to approximate.

Posterior distributions in GaSP calibration. After marginalizing out the discrepancy function, the marginal distribution follows $\mathbf{y}^F \sim \mathcal{MN}(\mathbf{f}_\theta^M + \mathbf{H}\theta_m, \sigma_0^2 \tilde{\mathbf{R}})$, where $\tilde{\mathbf{R}} = \mathbf{R}/\eta + \Lambda$. We assume an objective prior for the mean and noise variance parameters $\pi(\theta_m, \sigma_0^2) \propto 1/\sigma_0^2$. The full conditional

posterior distributions of the trend and variance parameters follow

$$\begin{aligned}\sigma_0^{-2} \mid \theta_m, \theta &\sim \text{Gamma} \left(\frac{n}{2}, \frac{(\mathbf{y}^F - \mathbf{f}_\theta^M - \mathbf{H}\theta_m)^T \tilde{\mathbf{R}}^{-1} (\mathbf{y}^F - \mathbf{f}_\theta^M - \mathbf{H}\theta_m)}{2} \right), \\ \theta_m \mid \sigma_0^2, \theta &\sim \mathcal{MN} \left(\hat{\theta}_m, \sigma_0^{-2} (\mathbf{H}^T \tilde{\mathbf{R}}^{-1} \mathbf{H})^{-1} \right),\end{aligned}$$

where $\hat{\theta}_m = (\mathbf{H}^T \tilde{\mathbf{R}}^{-1} \mathbf{H})^{-1} \mathbf{H}^T \tilde{\mathbf{R}}^{-1} (\mathbf{y} - \mathbf{f}_\theta^M)$. A Gibbs sampler is used to sample σ_0^{-2} and θ_m from the full conditional distribution.

Given the trend and variance parameters, the posterior distribution of the calibration parameters follows

$$p(\theta \mid \theta_m, \sigma^2) \propto \exp \left(-\frac{(\mathbf{y}^F - \mathbf{f}_\theta^M - \mathbf{H}\theta_m)^T \tilde{\mathbf{R}}^{-1} (\mathbf{y}^F - \mathbf{f}_\theta^M - \mathbf{H}\theta_m)}{2\sigma_0^2} \right) \pi(\theta).$$

We implement the Metropolis algorithm where the proposal distribution is a normal distribution with the mean centered around the current value of the posterior sample. Users can adjust the standard deviation of the proposal distribution by argument `sd_proposal` in the `rcalibration` function.

Denote the inverse range parameter $\beta_l = 1/\gamma_l$, $l = 1, \dots, p_x$. The conditional distribution of the inverse range and nugget parameters follow

$$p(\beta, \eta \mid \theta, \theta_m, \sigma_0^2) \propto |\tilde{\mathbf{R}}|^{-\frac{n}{2}} \exp \left(-\frac{(\mathbf{y}^F - \mathbf{f}_\theta^M - \mathbf{H}\theta_m)^T \tilde{\mathbf{R}}^{-1} (\mathbf{y}^F - \mathbf{f}_\theta^M - \mathbf{H}\theta_m)}{2\sigma_0^2} \right) \pi(\gamma, \eta).$$

We define the inverse range parameter $\beta_l = 1/\gamma_l$, and assume that the prior for the inverse range and nugget parameters follows the jointly robust (JR) prior (Gu, 2019)

$$\pi(\beta, \eta) = C \left(\sum_{l=1}^{p_x} C_l \beta_l + \eta \right)^a \exp \left\{ -b \left(\sum_l C_l \beta_l + \eta \right) \right\},$$

where C is a normalizing constant. By default, we use the following prior parameters $a = 1/2 - p_x$, $b = 1$, and $C_l = n^{-1/p_x} |x_l^{max} - x_l^{min}|$, with x_l^{max} and x_l^{min} being the maximum and minimum values of the observable input at the l th coordinate, respectively. The default choices of prior parameters induces a small penalty on very large correlation, preventing the identifiability issues of the calibration parameters due to large correlation (Gu, 2019). Users can adjust a and b in the `rcalibration` function.

We use the Metropolis algorithm to sample the logarithm of the inverse range parameter $\log(\beta_l)$ and nugget parameter $\log(\eta)$. The proposal distribution of each parameter is a normal distribution centered around the current value with the standard deviation chosen to be 0.25 by default. User can change the $p_\theta + 1$ to $p_x + p_\theta + 1$ coordinates of the vector in the argument `sd_proposal` in the `rcalibration` function to specify the standard deviation in the proposal distribution for these parameters.

Likelihood function and posterior distributions S-GaSP calibration. The likelihood function and posterior distribution of S-GaSP follow from those in GaSP calibration, by simply replacing the kernel K to K_{Z_d} in Table 1.

Acknowledgements

This research was supported by the National Science Foundation under Award Number DMS-2053423. We thank Xubo Liu for implementing the numerical method for simulating the Lorenz 96 model.

References

- P. Agram and M. Simons. A noise model for InSAR time series. *Journal of Geophysical Research: Solid Earth*, 120(4):2752–2771, 2015. [p98]
- K. R. Anderson, I. A. Johanson, M. R. Patrick, M. Gu, P. Segall, M. P. Poland, E. K. Montgomery-Brown, and A. Miklius. Magma reservoir failure and the onset of caldera collapse at Kīlauea volcano in 2018. *Science*, 366(6470), 2019. [p84, 85, 96, 98]
- P. D. Arendt, D. W. Apley, and W. Chen. Quantification of model uncertainty: calibration, model discrepancy, and identifiability. *Journal of Mechanical Design*, 134(10):100908, 2012. [p83]

- M. Bayarri, J. Berger, J. Cafeo, G. Garcia-Donato, F. Liu, J. Palomo, R. Parthasarathy, R. Paulo, J. Sacks, and D. Walsh. Computer model validation with functional output. *The Annals of Statistics*, 35(5):1874–1906, 2007a. [p83]
- M. J. Bayarri, J. O. Berger, R. Paulo, J. Sacks, J. A. Cafeo, J. Cavendish, C.-H. Lin, and J. Tu. A framework for validation of computer models. *Technometrics*, 49(2):138–154, 2007b. [p88, 90, 91, 93, 96]
- G. Box and G. Coutie. Application of digital computers in the exploration of functional relationships. *Proceedings of the IEE-Part B: Radio and Electronic Engineering*, 103(1S):100–107, 1956. [p96]
- J. Brajard, A. Carrassi, M. Bocquet, and L. Bertino. Combining data assimilation and machine learning to emulate a dynamical model from sparse and noisy observations: A case study with the Lorenz 96 model. *Journal of Computational Science*, 44:101171, 2020. [p93]
- M. Carmassi, P. Barbillon, M. Chiodetti, M. Keller, and E. Parent. CaliCo: a R package for Bayesian calibration. *arXiv preprint arXiv:1808.01932*, 2018. [p84, 96]
- W. Chang, M. Haran, P. Applegate, and D. Pollard. Calibrating an ice sheet model using high-dimensional binary spatial data. *Journal of the American Statistical Association*, 111(513):57–72, 2016. [p83]
- W. Chang, B. A. Konomi, G. Karagiannis, Y. Guan, and M. Haran. Ice model calibration using semicontinuous spatial data. *The Annals of Applied Statistics*, 16(3):1937–1961, 2022. [p83]
- X. Fang, M. Gu, and J. Wu. Reliable emulation of complex functionals by active learning with error control. *The Journal of Chemical Physics*, 157(21):214109, 2022. [p96]
- M. Gu. Jointly robust prior for Gaussian stochastic process in emulation, calibration and variable selection. *Bayesian Analysis*, 14(1), 2019. [p91, 96, 102]
- M. Gu and L. Wang. Scaled Gaussian stochastic process for computer model calibration and prediction. *SIAM/ASA Journal on Uncertainty Quantification*, 6(4):1555–1583, 2018. [p83, 91, 92]
- M. Gu, J. Palomo, and J. O. Berger. RobustGaSP: Robust Gaussian Stochastic Process Emulation in R. *The R Journal*, 11(1):112–136, 2019. doi: 10.32614/RJ-2019-011. [p84, 96]
- M. Gu, F. Xie, and L. Wang. A theoretical framework of the scaled Gaussian stochastic process in prediction and calibration. *SIAM/ASA Journal on Uncertainty Quantification*, 10(4):1435–1460, 2022. [p92]
- M. Gu, K. Anderson, and E. McPhillips. Calibration of imperfect geophysical models by multiple satellite interferograms with measurement bias. *Technometrics, In Press*, 2023. doi: 10.1080/00401706.2023.2182365. [p98]
- R. K. Hankin. Introducing BACCO, an R bundle for Bayesian analysis of computer code output. *Journal of Statistical Software*, 14:1–21, 2005. [p84]
- D. Higdon, J. Gattiker, B. Williams, and M. Rightley. Computer model calibration using high-dimensional output. *Journal of the American Statistical Association*, 103(482):570–583, 2008. [p83, 96]
- M. C. Kennedy and A. O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464, 2001. [p83, 84]
- H. Li, M. Zhou, J. Sebastian, J. Wu, and M. Gu. Efficient force field and energy emulation through partition of permutationally equivalent atoms. *The Journal of Chemical Physics*, 156(18):184304, 2022. [p96]
- D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989. [p87]
- F. Liu, M. Bayarri, and J. Berger. Modularization in Bayesian analysis, with emphasis on analysis of computer models. *Bayesian Analysis*, 4(1):119–150, 2009. [p96]
- E. N. Lorenz. Predictability: A problem partly solved. In *Proc. Seminar on predictability*, volume 1, 1996. [p93]
- P. Ma, G. Karagiannis, B. A. Konomi, T. G. Asher, G. R. Toro, and A. T. Cox. Multifidelity computer model emulation with high-dimensional output: An application to storm surge. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 71(4):861–883, 2022. [p84, 96]

- B. MacDonald, P. Ranjan, H. Chipman, et al. Gpfit: An R package for fitting a Gaussian process model to deterministic simulator outputs. *Journal of Statistical Software*, 64(i12), 2015. [p84, 96]
- J. Maclean and E. T. Spiller. A surrogate-based approach to nonlinear, non-Gaussian joint state-parameter data assimilation. *arXiv preprint arXiv:2012.04793*, 2020. [p93]
- J. Palomo, R. Paulo, G. García-Donato, et al. Save: an R package for the statistical analysis of computer models. *Journal of Statistical Software*, 64(13):1–23, 2015. [p84, 96]
- R. Paulo, G. García-Donato, and J. Palomo. Calibration of computer models with multivariate output. *Computational Statistics and Data Analysis*, 56(12):3959–3974, 2012. [p83]
- M. Plumlee. Bayesian calibration of inexact computer models. *Journal of the American Statistical Association*, 112(519):1274–1285, 2017. [p84]
- O. Roustant, D. Ginsbourger, and Y. Deville. Dicekriging, Diceoptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization. *Journal of statistical software*, 51:1–55, 2012. [p84, 96]
- J. Sacks, W. J. Welch, T. J. Mitchell, H. P. Wynn, et al. Design and analysis of computer experiments. *Statistical science*, 4(4):409–423, 1989. [p83]
- T. J. Santner, B. J. Williams, and W. I. Notz. *The design and analysis of computer experiments*. Springer Science & Business Media, 2003. [p96]
- N. A. Simakov, R. L. Jones-Ivey, A. Akhavan-Safaei, H. Aghakhani, M. D. Jones, and A. K. Patra. Modernizing Titan2D, a parallel AMR geophysical flow code to support multiple rheologies and extendability. In *International Conference on High Performance Computing*, pages 101–112. Springer, 2019. [p96]
- K. Soetaert, T. Petzoldt, and R. W. Setzer. Solving differential equations in R: package deSolve. *Journal of statistical software*, 33:1–25, 2010. [p96]
- R. Tuo and C. J. Wu. Efficient calibration for imperfect computer models. *The Annals of Statistics*, 43(6):2331–2352, 2015. [p84]
- H. Wickham. ggplot2. *Wiley Interdisciplinary Reviews: Computational Statistics*, 3(2):180–185, 2011. [p84]
- D. Williamson, M. Goldstein, L. Allison, A. Blaker, P. Challenor, L. Jackson, and K. Yamazaki. History matching for exploring and reducing climate model parameter space using observations and a large perturbed physics ensemble. *Climate dynamics*, 41(7–8):1703–1729, 2013. [p84]
- R. K. Wong, C. B. Storlie, and T. Lee. A frequentist approach to computer model calibration. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79:635–648, 2017. [p84]
- J. Ypma. nloptr: R interface to NLOpt, 2014. URL <https://CRAN.R-project.org/package=nloptr>. R package version 1.0.4. [p87, 101]
- H. A. Zebker, P. A. Rosen, and S. Hensley. Atmospheric effects in interferometric synthetic aperture radar surface deformation and topographic maps. *Journal of Geophysical Research: Solid Earth*, 102(B4):7547–7563, apr 1997. ISSN 01480227. doi: 10.1029/96JB03804. URL <http://doi.wiley.com/10.1029/96JB03804>. [p98]

Mengyang Gu

University of California, Santa Barbara

Department of Statistics and Applied Probability

Santa Barbara, California, USA

mengyang@pstat.ucsb.edu

glmmPen: High Dimensional Penalized Generalized Linear Mixed Models

by Hillary M. Heiling, Naim U. Rashid, Quefeng Li, and Joseph G. Ibrahim

Abstract Generalized linear mixed models (GLMMs) are widely used in research for their ability to model correlated outcomes with non-Gaussian conditional distributions. The proper selection of fixed and random effects is a critical part of the modeling process since model misspecification may lead to significant bias. However, the joint selection of fixed and random effects has historically been limited to lower-dimensional GLMMs, largely due to the use of criterion-based model selection strategies. Here we present the R package `glmmPen`, one of the first to select fixed and random effects in higher dimension using a penalized GLMM modeling framework. Model parameters are estimated using a Monte Carlo Expectation Conditional Minimization (MCECM) algorithm, which leverages Stan and `RcppArmadillo` for increased computational efficiency. Our package supports the Binomial, Gaussian, and Poisson families and multiple penalty functions. In this manuscript we discuss the modeling procedure, estimation scheme, and software implementation through application to a pancreatic cancer subtyping study. Simulation results show our method has good performance in selecting both the fixed and random effects in high dimensional GLMMs.

1 Introduction

Generalized linear mixed models (GLMMs) are utilized in many disciplines, including the social sciences (Schmidt-Catran and Fairbrother, 2016), biomedical sciences (Fitzmaurice et al., 2012), public health and epidemiology (Szyszkowicz, 2006; Kleinman et al., 2004; Dean and Nielsen, 2007), natural sciences including ecology and evolution (Bolker et al., 2009), and economics (Langford, 1994). GLMMs are an extension of generalized linear models (GLMs) where the predictors within the model can have “fixed” or “random” effects. Coefficients corresponding to fixed effects predictors can be considered to describe population-level relationships between the predictors and the outcome. Random effects predictors pertain to variables whose relationships with the outcome are presumed to vary randomly across “groups” of observations within the data, leading to group-specific coefficient estimates (Fitzmaurice et al., 2012). In practical applications, these “groups” may pertain to clusters of samples, repeated measures within the same individual, or observations resulting from nested designs. Multiple studies have shown that omitting important random effects can lead to bias in the estimated variance of the fixed effects; conversely, including unnecessary random effects may lead to computational difficulties (Thompson et al., 2017; Gurka et al., 2011; Bondell et al., 2010). As a result, proper specification of fixed and random effects is a critical step in the application of GLMMs.

In many low dimensional settings, researchers may have *a priori* knowledge about which variables are fixed or random. For instance, researchers may reasonably expect treatment effects in multi-site clinical trials to vary by site (Feaster et al., 2011). However, in high dimensional settings, it is often not known *a priori* which variables should be specified as fixed or random in the model. In such settings, the feature space may also be sparse, with many variables unrelated to the outcome. Therefore, variable selection approaches are employed to evaluate and select from a set of candidate models. R packages such as `lme4` (Bates et al., 2015), `mcemGLM` (Archila, 2020), and `MCMCglmm` (Hadfield, 2010) allow users to fit a pre-specified set of models, which may then be compared using model selection criteria such as the profile conditional AIC (Donohue et al., 2011), the BIC-JCQ criterion (Ibrahim et al., 2011), the hybrid Bayesian information criterion, BIC_h (Delattre et al., 2014), or other criteria developed for mixed effects models. However, criterion-based all-subsets selection or direct model comparison strategies are not feasible even in small dimensions, as with p predictors there are 2^p possible combinations of fixed and random effects to be evaluated.

Packages such as `glmnet` (Friedman et al., 2010), `ncvreg` (Breheny and Huang, 2011), and `grpreg` (Breheny and Huang, 2015) avoid this limitation for GLMs via coordinate-descent based penalized likelihood methods for variable selection, and therefore scale much better with respect to p . Unfortunately, none of these methods can account for random effects in their variable selection procedure. Other packages such as `glmmLasso` (Groll, 2017) and `glmmmixedLASSO` (Schellendorfer et al., 2014) alternatively allow the inclusion of random effects in the model while performing variable selection, but only allow for variable selection on the fixed effects. Prior work has shown that simultaneous selection of fixed and random effects is desirable because improper specification of the random effects can significantly affect the selection of the fixed effects, and vice versa (Bondell et al., 2010). In addition, there may not be *a priori* knowledge of which variables have effects that vary randomly across groups. Therefore, the specification of random effects may be difficult in practical applications, particularly as

the dimension of the data grows.

To address these limitations in performing variable selection in high-dimensional GLMMs, we present the **glmmPen** R package. This package allows for the simultaneous selection of fixed and random effects predictors in higher dimensions through the use of penalized generalized linear mixed models (pGLMMs). Similar to **ncvreg** and **glmnet**, this package focuses on variable selection for the purpose of creating prediction models, and does not provide methods for statistical inference. The package leverages Monte Carlo Expectation Conditional Minimization (MCECM) in combination with several techniques to improve the computational efficiency of the algorithm. In the MCECM E-step, **glmmPen** utilizes the **Stan** software implemented in the **rstan** package to efficiently sample from the posterior distribution of the random effects, and a Majorization-Minimization coordinate descent algorithm is utilized to update model parameters in the M-step. The **glmmPen** package utilizes the fast looping capabilities within **Rcpp** and **RcppArmadillo** in order to recalculate large matrices within intermediate computing steps without needing to store them, improving memory use. The **glmmPen** package is also able to improve the speed of the overall variable selection procedure by strategic coefficient initialization (see Section “Initialization and convergence”) and strategic restriction of random effects (see Section “Tuning parameter selection strategy”).

The main estimation functions of the package are **glmmPen** and **glmm**, where the latter can be used to fit traditional generalized linear mixed models without penalization. The user interface and output of the **glmmPen** and **glmm** functions were designed to be very similar to those from the functions **lmer** and **glmer** to facilitate ease of use. Specifically, **glmmPen** outputs a **pglmmObj** object which, like the **merMod** object from **lme4**, can facilitate the application of common S3 method functions used by **lme4** such as **logLik**, **fixef**, **ranef**, and others. In addition, multiple types of penalties and information criteria for selecting optimal penalties are available in the package, and the package supports the Binomial, Gaussian, and Poisson distributional families.

Our manuscript is organized as follows. We begin in Section 2 by reviewing the pGLMMs modeling framework, first described in Rashid et al. (2020). Section 3 describes the MCECM algorithm used by **glmmPen** to fit pGLMM models. Section 4 describes the variable selection procedure of the package and the Bayesian information criterion (BIC) type selection criteria available for use. Section 5 illustrates a practical application of the **glmmPen** R package using data from a recent cancer subtyping study. Section 6 provides some simulation results. Finally, we provide concluding comments in Section 7. The package is available from the Comprehensive R Archive Network (CRAN) at <https://cran.r-project.org/package=glmmPen>. The replication of all code content, tables, and figures presented in this paper can be found in the GitHub repository https://github.com/hheiling/paper_glmmPen_RJournal. Supplementary results mentioned but not reported in this paper can also be found in this GitHub repository.

2 Generalized linear mixed models

We review the notation and model formulation of our approach, first introduced in Rashid et al. (2020). We consider the case where we want to analyze data from K independent groups of any kind. For instance, we could be interested in analyzing data from K different studies, or longitudinal data from K individuals. For each group $k = 1, \dots, K$, there are n_k observations for a total sample size of $N = \sum_{k=1}^K n_k$. For the k^{th} group, let $\mathbf{y}_k = (y_{k1}, \dots, y_{kn_k})^\top$ be the vector of n_k independent responses, let $\mathbf{x}_{ki} = (x_{ki,1}, \dots, x_{ki,p})^\top$ be the p -dimensional vector of predictors, and let $\mathbf{X}_k = (\mathbf{x}_{k1}, \dots, \mathbf{x}_{kn_k})^\top$. Although the **glmmPen** package allows for different n_k for the K groups, we will set $\{n_k\}_{k=1}^K = n$ to simplify the notation within the equations presented in this paper. In GLMMs, we assume that the conditional distribution of \mathbf{y}_k given \mathbf{X}_k belongs to the exponential family and has the following density:

$$f(\mathbf{y}_k | \mathbf{X}_k, \boldsymbol{\alpha}_k, \theta) = \prod_{i=1}^n c(y_{ki}) \exp[\tau^{-1}\{y_{ki}\eta_{ki} - b(\eta_{ki})\}], \quad (1)$$

where $c(y_{ki})$ is a constant that only depends on y_{ki} , τ is the dispersion parameter, $b(\cdot)$ is a known link function, and η_{ki} is the linear predictor. The **glmmPen** algorithm currently allows for the Gaussian, Binomial, and Poisson families with canonical links.

In the GLMM, the linear predictor has the form

$$\eta_{ki} = \mathbf{x}_{ki}^\top \boldsymbol{\beta} + \mathbf{z}_{ki}^\top \boldsymbol{\Gamma} \boldsymbol{\alpha}_k, \quad (2)$$

where $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)^\top$ is a p -dimensional vector for the fixed effects coefficients (including the intercept), $\boldsymbol{\alpha}_k$ is a q -dimensional vector of unobservable random effects (including the random intercept), \mathbf{z}_{ki} is a q -dimensional subvector of \mathbf{x}_{ki} , and $\boldsymbol{\Gamma}$ is a lower triangular matrix. In this notation, \mathbf{z}_{ki} represents the random effects predictors, i.e. the subset of the total predictors (\mathbf{x}_{ki}) that have predictor effects that

randomly vary across levels of the grouping variable.

In Rashid et al. (2020), the random effects vector α_k is assumed to follow $N_q(\mathbf{0}, \mathbf{I})$ so that $\Gamma\alpha_k$ follows $N(\mathbf{0}, \Gamma\Gamma^\top)$. In this way, the random component of the linear predictor has variance $\text{Var}(\Gamma\alpha_k) = \Gamma\Gamma^\top$.

To simplify the procedure of estimating Γ , we consider a vector γ containing all of the nonzero elements of Γ such that γ_t is a $t \times 1$ vector consisting of nonzero elements of the t^{th} row of Γ and $\gamma = (\gamma_1^\top, \dots, \gamma_q^\top)^\top$. We can then reparameterize the linear predictor (Chen and Dunson, 2003; Ibrahim et al., 2011) to

$$\eta_{ki} = \mathbf{x}_{ki}^\top \beta + \mathbf{z}_{ki}^\top \Gamma \alpha_k = \begin{pmatrix} \mathbf{x}_{ki}^\top & (\alpha_k \otimes \mathbf{z}_{ki})^\top \end{pmatrix} J_q \begin{pmatrix} \beta \\ \gamma \end{pmatrix} \quad (3)$$

where J_q is a matrix that transforms γ to $\text{vec}(\Gamma)$ such that $\text{vec}(\Gamma) = J_q \gamma$. J_q is of dimension $q^2 \times q(q+1)/2$ when the random effects covariance matrix $\Gamma\Gamma^\top$ is unstructured; alternatively, J_q is of dimension $q^2 \times q$ when the random effects covariance matrix has an independence structure (i.e., diagonal). The vector of parameters $\theta = (\beta^\top, \gamma^\top, \tau)^\top$ are the main parameters of interest. We denote the true value of θ as $\theta^* = (\beta^{*\top}, \gamma^{*\top}, \tau^*)^\top = \text{argmin}_{\theta} \ell(\theta)$ where $\ell(\theta)$ is the observed marginal log-likelihood across all K groups such that $\ell(\theta) = \sum_{k=1}^K \ell_k(\theta)$, $\ell_k(\theta) = (1/n) \log \int f(\mathbf{y}_k | \mathbf{X}_k, \alpha_k; \theta) \phi(\alpha_k) d\alpha_k$.

Let us consider the high dimensional case where we want to select the true nonzero fixed effects and true nonzero random effects. In other words, we aim to identify the set

$$S = S_1 \cup S_2 = \{j : \beta_j^* \neq 0\} \cup \{t : \|\gamma_t^*\|_2 \neq 0\},$$

where the set S_1 represents the selection of true nonzero fixed effects and the set S_2 represents the selection of true nonzero random effects. When $\gamma_t = \mathbf{0}$, this sets row t of Γ entirely equal to 0, indicating that effect of covariate t is fixed across the K groups.

We aim to solve the following penalized likelihood:

$$\hat{\theta} = \text{argmin}_{\theta} -\ell(\theta) + \lambda_0 \sum_{j=1}^p \rho_0(\beta_j) + \lambda_1 \sum_{t=1}^q \rho_1(\|\gamma_t\|_2), \quad (4)$$

where $\ell(\theta)$ is the observed marginal log-likelihood for all K groups defined earlier, $\rho_0(t)$ and $\rho_1(t)$ are general folded-concave penalty functions, and λ_0 and λ_1 are positive tuning parameters. In the **glmmPen** package, the $\rho_0(t)$ penalty function options include the least absolute shrinkage and selection operator (LASSO) L_1 penalty, the minimax concave penalty (MCP), and the smoothly clipped absolute deviation (SCAD) penalty (Friedman et al., 2010; Breheny and Huang, 2011). For the $\rho_1(t)$ penalty, we treat the elements of γ_t as a group and penalize them in a groupwise manner using the group LASSO, group MCP, or group SCAD penalties presented by Breheny and Huang (2015). These groups of γ_t are then estimated to be either all zero or all nonzero. In this way, we select covariates to have varying effects ($\hat{\gamma}_t \neq \mathbf{0}$) or fixed effects ($\hat{\gamma}_t = \mathbf{0}$) across the K groups.

Similar to other variable selection packages such as package **ncvreg** (Breheny and Huang, 2011), in **glmmPen** we standardize the fixed effects covariates matrix $\mathbf{X} = (\mathbf{X}_1^\top, \dots, \mathbf{X}_K^\top)^\top$ such that $\sum_{k=1}^K \sum_{i=1}^{n_k} x_{ki,j} = 0$ and $N^{-1} \sum_{k=1}^K \sum_{i=1}^{n_k} x_{ki,j}^2 = 1$ for $j = 1, \dots, p$; this process is performed automatically within the algorithm. Although the package **grpreg** (Breheny and Huang, 2015) orthogonalizes grouped effects, we have found through simulations during early package testing that first standardizing the fixed effects and then using subsets of these standardized fixed effects for the random effects (recall: \mathbf{z}_{ki} is a q -dimensional subvector of \mathbf{x}_{ki}) is sufficient. During the selection procedure, the fixed effects intercept and the variance of the random effects intercept remain unpenalized.

3 MCECM algorithm

We solve Equation 4 for a specific (λ_0, λ_1) penalty parameter combination using a Monte Carlo Expectation Conditional Minimization (MCECM) algorithm (Garcia et al., 2010). The MCECM algorithm described in this section uses many of the steps and assumptions described in Rashid et al. (2020), but here we provide further practical details about the E-step, M-step, initialization, and convergence. Additionally, the implementation outlined in this paper has several improvements to the implementation used in Rashid et al. (2020). In **glmmPen**, the E-step allows for several possible sampling schemes, including the fast and efficient No-U-Turn Hamiltonian Monte Carlo sampling procedure (NUTS HMC) from the **Stan** software (Carpenter et al., 2017; Hoffman and Gelman, 2014). The **glmmPen** package was also able to reduce the required memory usage of the MCECM algorithm. In the M-step, we utilized the fast looping capability of packages **Rcpp** and **RcppArmadillo** to allow for fast recalculation of large matrices (see Step 3 of the M-step presented in Algorithm 1) and avoid

their storage, improving model scalability.

During the MCECM algorithm, we aim to evaluate (E-step) and minimize (M-step) the following penalized Q-function in the s^{th} iteration of the algorithm:

$$\begin{aligned} Q_\lambda(\boldsymbol{\theta}|\boldsymbol{\theta}^{(s)}) &= \sum_{k=1}^K E \left\{ -\log(f(\mathbf{y}_k, \mathbf{X}_k, \boldsymbol{\alpha}_k; \boldsymbol{\theta}|\mathbf{D}_o; \boldsymbol{\theta}^{(s)})) \right\} + \lambda_0 \sum_{j=1}^p \rho_0(\beta_j) + \lambda_1 \sum_{t=1}^q \rho_1(||\gamma_t||_2) \\ &= Q_1(\boldsymbol{\theta}|\boldsymbol{\theta}^{(s)}) + Q_2(\boldsymbol{\theta}^{(s)}) + \lambda_0 \sum_{j=1}^p \rho_0(\beta_j) + \lambda_1 \sum_{t=1}^q \rho_1(||\gamma_t||_2), \end{aligned} \quad (5)$$

where $(\mathbf{y}_k, \mathbf{X}_k, \boldsymbol{\alpha}_k)$ gives the complete data for group k , $\mathbf{D}_{k,o} = (\mathbf{y}_k, \mathbf{X}_k)$ gives the observed data for group k , and \mathbf{D}_o represents the entirety of the observed data. In other words, we aim to evaluate and minimize the penalized expectation of the negative joint log-likelihood with respect to the observed data. From [Rashid et al. \(2020\)](#), the expectation can be written as the sum of the following terms:

$$Q_1(\boldsymbol{\theta}|\boldsymbol{\theta}^{(s)}) = - \sum_{k=1}^K \int \log[f(\mathbf{y}_k|\mathbf{X}_k, \boldsymbol{\alpha}_k; \boldsymbol{\theta})] \phi(\boldsymbol{\alpha}_k|\mathbf{D}_{k,o}; \boldsymbol{\theta}^{(s)}) d\boldsymbol{\alpha}_k, \quad (6)$$

$$Q_2(\boldsymbol{\theta}^{(s)}) = - \sum_{k=1}^K \int \log[\phi(\boldsymbol{\alpha}_k)] \phi(\boldsymbol{\alpha}_k|\mathbf{D}_{k,o}; \boldsymbol{\theta}^{(s)}) d\boldsymbol{\alpha}_k \quad (7)$$

The $Q_1(\boldsymbol{\theta}|\boldsymbol{\theta}^{(s)})$ function expresses the conditional model of the observed data given the latent (random) variables and integrates over the latent variables. Using the $Q_1(\boldsymbol{\theta}|\boldsymbol{\theta}^{(s)})$ function, we aim to derive the fixed and random effect coefficient estimates during the M-step of the algorithm. During the E-step, we aim to approximate the integral in the $Q_1(\boldsymbol{\theta}|\boldsymbol{\theta}^{(s)})$ function by incorporating samples from the posterior distribution of the latent variables.

3.1 Monte Carlo E-step

The integrals in the Q-function do not have closed forms when $f(\mathbf{y}_k|\mathbf{X}_k, \boldsymbol{\alpha}_k^{(s,m)}; \boldsymbol{\theta})$ is assumed to be non-Gaussian, and become difficult to approximate as q (the number of random effect predictors) increases. Consequently, we approximate these integrals using a Markov chain Monte Carlo (MCMC) sample of size M from the posterior density $\phi(\boldsymbol{\alpha}_k|\mathbf{D}_{k,o}; \boldsymbol{\theta}^{(s)})$. The **glmmPen** package can draw samples from this posterior using one of several techniques: the No-U-Turn Hamiltonian Monte Carlo sampling procedure (NUTS HMC) implemented by the **Stan** software, which **glmmPen** calls using the **rstan** package ([Carpenter et al. \(2017\)](#); default, and strongly recommended for its speed and efficiency); Metropolis-within-Gibbs with an adaptive random walk sampler ([Roberts and Rosenthal, 2009](#)); and Metropolis-within-Gibbs with an independence sampler ([Givens and Hoeting, 2012](#)). Each sampler type uses a standard normal candidate distribution. Let $\boldsymbol{\alpha}_k^{(s,m)}$ be the m^{th} simulated value, $m = 1, \dots, M$, at the s^{th} iteration of the algorithm for group k . The integral in Equation 6 can then be approximated as

$$Q_1(\boldsymbol{\theta}|\boldsymbol{\theta}^{(s)}) \approx - \frac{1}{M} \sum_{m=1}^M \sum_{k=1}^K \log f(\mathbf{y}_k|\mathbf{X}_k, \boldsymbol{\alpha}_k^{(s,m)}; \boldsymbol{\theta}).$$

Although the optimal number of MCMC samples $M^{(s)}$ in the E-step at EM iteration s is not well defined, the general consensus is that a smaller sample size of the posterior is suitable for the start of the algorithm but larger sample sizes are needed later in the algorithm ([Booth and Hobert, 1999](#)). We set the default number of MCMC samples in the first iteration of the MCECM algorithm $M^{(1)} = 250$ when $q \leq 10$, and $M^{(1)} = 100$ otherwise (we decrease the initial sampling size when the number of random effects predictors is large in order to help speed up the algorithm). Then, in a manner similar to the **mcemGLM** package ([Archila, 2020](#)), the MCMC sample size is increased by a multiplicative factor v at each step of the algorithm such that $M^{(s)} = v \times M^{(s-1)}$ until either the value of $M^{(s)}$ reaches its maximum allowed value or the EM algorithm converges. In **glmmPen**, the default maximum allowed value is dependent on the the number of random effects in the model, q (see the documentation of **optimControl** for more details). For the first 15 iterations of the EM algorithm, the value of v is set to 1.1. For the remaining steps of the algorithm, v is set to 1.2.

3.2 M-step

In the M-step of the algorithm, we aim to minimize

$$Q_{1,\lambda}(\boldsymbol{\theta}|\boldsymbol{\theta}^{(s)}) = Q_1(\boldsymbol{\theta}|\boldsymbol{\theta}^{(s)}) + \lambda_0 \sum_{j=1}^p \rho_0(\beta_j) + \lambda_1 \sum_{t=1}^q \rho_1(||\gamma_t||_2) \quad (8)$$

with respect to $\boldsymbol{\theta} = (\boldsymbol{\beta}^\top, \boldsymbol{\gamma}^\top, \tau)^\top$. The minimization of Equation 8 with respect to $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ is performed using a Majorization-Minimization approach. For the general exponential family, Rashid et al. (2020) suggested minimizing with respect to τ using the standard optimization algorithm Newton-Raphson. In `glmmPen`, the only family implemented with a dispersion parameter is the Gaussian family, and the variance σ^2 can be estimated directly from a derivation of the Q function conditional on the most recent updates of $\boldsymbol{\beta}^{(s)}$ and $\boldsymbol{\gamma}^{(s)}$:

$$\sigma^2 = \frac{1}{M \times N} \sum_{m=1}^M \sum_{k=1}^K \sum_{i=1}^{n_k} (y_{ki} - \eta_{ki}^{(s,m)})^2, \quad (9)$$

where $\eta_{ki}^{(s,m)}$ is the linear predictor η_{ki} evaluated with $\boldsymbol{\beta}^{(s)}$, $\boldsymbol{\gamma}^{(s)}$, and sample $\boldsymbol{\alpha}_k^{(s,m)}$.

Let s represent the iteration of the MCECM algorithm, and h represent the iteration within a particular M-step of the MCECM algorithm. The M-step of the s^{th} iteration of the MCECM algorithm proceeds as in Algorithm 1.

Algorithm 1 M-step of the s -th iteration of the MCECM algorithm

1. Coefficient parameter estimates from the previous M-step, $\boldsymbol{\theta}^{(s-1)}$, are used to initialize the coefficient parameters of the current M-step at M-step iteration $h = 0$, denoted $\boldsymbol{\theta}^{(s,0)}$.
2. Conditional on $\boldsymbol{\gamma}^{(s,h-1)}$ and $\tau^{(s-1)}$, each $\beta_j^{(s,h)}$ for $j = \{1, \dots, p\}$ is given a single update using the Majorization-Minimization algorithm specified by Breheny and Huang (2015).
3. For each group k in $k = \{1, \dots, K\}$, the augmented matrix $\tilde{\mathbf{z}}_{ki} = (\tilde{\boldsymbol{\alpha}}_k^{(s)} \otimes \mathbf{z}_{ki})J_q$ is created for $i = 1, \dots, n_k$ where $\tilde{\boldsymbol{\alpha}}_k^{(s)} = ((\boldsymbol{\alpha}_k^{(s,1)})^\top, \dots, (\boldsymbol{\alpha}_k^{(s,M)})^\top)^\top$. This augmented matrix is used in the random effect portion of the linear predictor specified in Equation 3. The dimension of $\tilde{\mathbf{z}}_{ki}$ is $M \times q(q+1)/2$ for an unstructured covariance matrix and $M \times q$ for an independent covariance matrix. This augmented matrix is used to calculate Equation 2.9 in Breheny and Huang (2015).
4. Conditional on the $\tau^{(s-1)}$ and the recently updated $\boldsymbol{\beta}^{(s,h)}$, each $\gamma_t^{(s,h)}$ for $t = \{1, \dots, q\}$ is updated using the Majorization-Minimization coordinate descent grouped variable selection algorithm specified by Breheny and Huang (2015), except the residuals are not updated after every $\gamma_t^{(s,h)}$ coefficient update.
5. Steps 2 through 4 are repeated until the M-step convergence criteria specified in Equation 10 are reached or until the M-step reaches its maximum number of iterations:

$$\max \left\{ \max_j |\beta_j^{(s,h+1)} - \beta_j^{(s,h)}|, \max_{t,l} |\gamma_{tl}^{(s,h+1)} - \gamma_{tl}^{(s,h)}| \right\} < \delta, \quad (10)$$

where γ_{tl} is an individual element of $\boldsymbol{\gamma}_t$. The default value of δ is 0.0005.

6. Conditioning on the newly updated $\boldsymbol{\beta}^{(s)}$ and $\boldsymbol{b}^{(s)}$, $\tau^{(s)}$ is updated (generically, using the Newton-Raphson algorithm; for Gaussian family, using Equation 9).
-

Algorithm 1 recomputes the augmented matrices $\tilde{\mathbf{z}}_{ki}$ for $k = 1, \dots, K$ and $i = 1, \dots, n_k$ in step 3 of every M-step iteration h for several reasons. These repeat calculations prevent the M-step from having to store the augmented matrix $\tilde{\mathbf{Z}} = (\tilde{\mathbf{Z}}_1^\top, \dots, \tilde{\mathbf{Z}}_K^\top)^\top$ where $\tilde{\mathbf{Z}}_k = (\tilde{\mathbf{z}}_{k1}^\top, \dots, \tilde{\mathbf{z}}_{kn_k}^\top)^\top$. This full augmented matrix is of dimension $(M \times N) \times q(q+1)/2$ or $(M \times N) \times q$ depending on whether the random effect covariance matrix is unstructured or independent, respectively. As the MCMC sample size increases throughout the MCECM algorithm and as q increases, saving this $\tilde{\mathbf{Z}}$ becomes more and more memory

prohibitive even when utilizing large matrix implementation tools such as the package **bigmemory** (Kane et al., 2013). During testing, we found that recomputing the \tilde{z}_{ki} matrices during each M-step iteration utilizing **Rcpp** (Eddelbuettel and François, 2011) and **RcppArmadillo** (Eddelbuettel and Sanderson, 2014) significantly reduced the time and memory required to compute each M-step.

In step 4 of the M-step, the residuals are not updated after every update to the random effects coefficients $\gamma_t^{(s,h)}$ for $t = 1, \dots, q$ in order to speed up computation. Otherwise, this would require re-calculation of the augmented matrix specified in step 3 for each of the q random effects within each M-step iteration. When q is large, this makes the M-step prohibitively time-consuming. Based on early package testing, simplifying step 4 with no residual updates speeds up the computation time in high dimensional settings and was found to have negligible impact on estimation accuracy.

The full MCECM algorithm then proceeds in Algorithm 2.

Algorithm 2 Full MCECM algorithm for single (λ_0, λ_1) penalty combination

1. Fixed and random effects $\beta^{(0)}$ and $\gamma^{(0)}$ are initialized as discussed in Section “Initialization and convergence”.
 2. E-step: In each E-step for EM iteration s , a burn-in sample from the posterior distribution of the random effects is run and discarded. A sample of size $M^{(s)}$ from the posterior is then drawn and retained for the M-step (see Section “Monte Carlo E-step” for details on default burn-in sample size, default $M^{(s)}$, and other E-step details).
 3. M-step: Parameter estimates $\beta^{(s)}$, $\gamma^{(s)}$, and $\tau^{(s)}$ are then updated as described in Algorithm 1.
 4. Steps 2 and 3 are repeated until the average Euclidean distance between the vector containing the current coefficients $\beta^{(s)}$ and $\gamma^{(s)}$ and the vector containing the coefficients from t EM iterations prior (default $t = 2$) is less than ϵ (default $\epsilon = 0.0015$) for at least two consecutive EM iterations or until the maximum number of EM iterations is reached (see Section “Initialization and convergence” for additional details).
 5. Using the estimates of β , γ , and τ at EM convergence, a final sample from the posterior distribution of the random effects is drawn for use in the calculation of the marginal log-likelihood as well as for diagnostics of the MCMC chain. The marginal log-likelihood is used for model selection and is discussed in detail in Section 2.4.
-

3.3 Initialization and convergence

The initial values of the fixed effects $\beta^{(0)}$ and the Cholesky decomposition of the random effects covariance matrix $\gamma^{(0)}$ for MCECM iteration $s = 0$ are chosen in one of two ways. We discuss first the initialization procedure used when the package **glmmPen** is used to fit a single model (`glmm` function) or the first model in the sequence of models fit for variable selection (`glmmPen` function). In this scenario, the fixed effects $\beta^{(0)}$ are initialized by fitting a ‘naive’ model using the coordinate descent techniques of Breheny and Huang (2011) assuming no random effects, and the random effects covariance matrix is initialized as a diagonal matrix with positive variance. This approach is similar to the **mcemGLM** package.

By default, this starting variance is initialized in an automated fashion. First, a GLMM composed of only a fixed and random intercept is fit using the **lme4** package. The random intercept variance from this model is then multiplied by 2, and this value is set as the starting values of the diagonal of the random effects covariance matrix. We use this approach so that the starting variance of the random effects is sufficiently large, which helps improve the stability of the algorithm (Misztal, 2008).

The MCMC chain used in the E-step of the algorithm, which approximately samples from the posterior density $\phi(\alpha_k | D_{k,o}; \theta^{(s)})$ for groups $k = \{1, \dots, K\}$, is initialized in iteration $s = 1$ with draws from the standard normal distribution. For all following iterations $s > 1$, the MCMC chain is initialized with the last draw from the previous EM iteration $s - 1$.

When the algorithm performs variable selection using the `glmmPen` function, the model pertaining to the first tuning parameter combination evaluated is initialized using approach described above. For all subsequent tuning parameter combinations evaluated in the sequence, the fixed effects, random

effects covariance matrix, and random effects MCMC chain are initialized using results from the previous tuning parameter fit. More details about initialization for variable selection is discussed in Section “Tuning parameter selection”.

The EM algorithm runs until the algorithm converges, defined as meeting the condition given in Equation 11 at least 2 consecutive times (default), or until the maximum number of EM iterations is reached:

$$\|(\beta^{(s)\top}, \gamma^{(s)\top})^\top - (\beta^{(s-t)\top}, \gamma^{(s-t)\top})^\top\|_2^2 / c^{s-t} < \epsilon \quad (11)$$

where the superscript $(s-t)$ indicates the EM iteration t iterations prior, $\|\cdot\|_2^2$ represents the L_2 norm, and c^{s-t} equals the total number of non-zero $(\beta^\top, \gamma^\top)^\top$ coefficients in iteration $(s-t)$. In other words, the algorithm computes the average Euclidean distance between the current coefficient vector $(\beta^\top, \gamma^\top)^\top$ and the coefficient vector from t EM iterations prior (default $t = 2$) and compares it with ϵ , which has a default value of 0.0015.

This MCECM algorithm is able to handle much larger dimensions of p fixed effect predictors and q random effect predictors relative to prior methods for simultaneous fixed and random effects variable selection (Bondell et al., 2010; Ibrahim et al., 2011). When the number of random effect predictors is greater than or equal to 10, we recommend approximating the random effect covariance matrix Π^\top as a diagonal matrix. In the mixed model setting, Fan and Li (2012) demonstrated both theoretical and empirical advantages to estimating the random effects covariance matrix in this manner as the number of random effect predictors q grows. Empirically, they found that this approximation had a relatively low impact on the overall bias of the coefficients and resulted in a relatively large reduction of accumulated estimation error since many fewer covariance parameters needed to be estimated. This simplification also has the advantage of enabling the package to have greater computational efficiency when fitting higher-dimensional models. The above-mentioned recommendation to switch from an unstructured to an independent random effect covariance matrix at the 10 random effect predictor mark is an ad hoc recommendation determined by our experience creating and testing this package.

The MCECM algorithm outlined in Algorithm 2 describes how the `glmmPen` package estimates the model parameters for a single set of penalty parameters (λ_0, λ_1) . Section “Tuning parameter selection” discusses how the package chooses optimal set of tuning parameters during the model selection procedure.

4 Tuning parameter selection

This section provides details on how the `glmmPen` function selects the set of optimal tuning parameters from a prespecific grid of values. Section “Software” provides further details on how to use both the `glmmPen` and `glmm` functions, where the latter function allows the user to fit a single model without performing variable selection on the fixed and random effects.

For `glmmPen`, we generally recommend that the user specify the ‘full model’, i.e., specify the set of candidate random effects predictors to be equal to the set of candidate fixed effects predictors, and let the algorithm select the best fixed and random effects using the procedure outlined in this section. However, if the user has some prior knowledge about the form of the random effects, they can restrict the random effects considered to an appropriate subset. As discussed in the previous section, the package requires that the random effects be a subset of the fixed effects.

4.1 Penalty sequence specification

The `glmmPen` package calculates default sequences of penalty values for λ_0 (penalizing the fixed effects β) and λ_1 (penalizing the random effects γ), but allows users to enter their own penalty sequences if desired. We define the penalty parameter sequences for the fixed and random effects as $\lambda_0 = (\lambda_{0,1}, \dots, \lambda_{0,\omega_0})$ and $\lambda_1 = (\lambda_{1,1}, \dots, \lambda_{1,\omega_1})$, respectively, where ω_0 and ω_1 are the length of the fixed and random effect penalty sequences. These sequences are ordered from the minimum penalty ($\lambda_{0,1} = \lambda_{0,min}$ and $\lambda_{1,1} = \lambda_{1,min}$) to the maximum penalty ($\lambda_{0,\omega_0} = \lambda_{0,max}$ and $\lambda_{1,\omega_1} = \lambda_{1,max}$). By default, these sequences are calculated in a similar manner to the approach used by the package `ncvreg` (Breheny and Huang, 2011). The maximum penalty parameter λ_{max} is calculated using the same procedure as `ncvreg`; this value is assumed to penalize all fixed and random effects coefficients to 0. We then set the sequence of penalty parameters $\lambda_0 = \lambda_1$ such that $\lambda_{0,max} = \lambda_{1,max} = \lambda_{max}$ and $\lambda_{0,min} = \lambda_{1,min} = \lambda_{min}$, where the minimum penalty parameter λ_{min} is a small portion of the λ_{max} . More details about these default sequences are given in Section “Software”. In Section “Tuning parameter selection strategy”, we consider a generic case where the λ_0 and λ_1 sequences do not need to be equal.

4.2 Tuning parameter selection strategy

By default, the algorithm runs a computationally efficient two-stage approach to pick the optimal set of tuning parameters. In the first stage, the algorithm fits a sequence of models where the fixed effect penalty is kept constant at the minimum value of $\lambda_0, \lambda_{0,min}$, and the random effects penalty proceeds from the minimum value of $\lambda_1, \lambda_{1,min}$, to the maximum value $\lambda_{1,max}$. The optimal tuning parameter from this first stage is then identified using Bayesian information criterion (BIC) type selection criteria, described in more detail later in this section. This first stage identifies the optimal random effect penalty value, $\lambda_{1,opt}$. In the second stage, the algorithm fits a sequence of models where the random effects penalty is kept fixed at $\lambda_{1,opt}$ and the fixed effects penalty λ_0 proceeds from $\lambda_{0,min}$ to $\lambda_{0,max}$. The overall best model is chosen from the models in the second stage. In both stages, the results from each model are used to initialize the coefficients in the subsequent model in the sequence.

Unlike other packages that perform variable selection, such as `ncvreg` and `grpreg`, we run the λ_0 and λ_1 sequences from their minimum value to their maximum value and not the traditional progression from their maximum value to their minimum value. In this mixed model setting, we have found (through simulations conducted during early package testing) that this approach provides better initialization of subsequent models in the tuning parameter sequence, giving an overall better performance to the algorithm and improving algorithm speed. This progression of penalty sequences also speeds up the overall variable selection procedure by restricting the random effects considered during later penalty combinations within the variable selection procedure. Within stage one, if a previous tuning parameter in the grid penalized out a set of random effects from the model, the following model in the tuning parameter sequence will automatically ignore these random effects. Within stage two, the random effects considered are restricted to the non-zero random effects from the best model in stage one.

In the original MCECM algorithm implementation given in Rashid et al. (2020), the authors searched for the best model by performing a ‘full grid search’ and evaluating all possible combinations of λ_0 and λ_1 . (We sometimes refer to the two-stage approach as the ‘abbreviated grid search’). While the `glmmPen` package can perform this full grid search, we strongly recommend the two-stage abbreviated grid search. Compared with the full grid search, the two-stage grid search significantly reduces the required time to complete the algorithm, particularly when the number of random effects predictors is large. Furthermore, we have found that the two-stage grid search works very well in practice (see Section “Simulations” for performance results).

If users wish to perform a full grid search, the path of solutions is initialized by fitting a model using the minimum penalty for both the fixed and random effects ($\lambda_{0,1} = \lambda_{0,min}, \lambda_{1,1} = \lambda_{1,min}$). The algorithm then proceeds to estimate models over the full grid of λ_0 and λ_1 . For each value of $\lambda_{1,h} \in \lambda_1$ that penalizes the random effects, the fixed effects penalty parameter sequence proceeds from the minimum value $\lambda_{0,min}$ to the maximum value $\lambda_{0,max}$ while keeping $\lambda_{1,h}$ fixed. Each model is initialized using the result from the model fit with the previous tuning parameter combination in the sequence. The algorithm then updates the penalty parameter to the next $\lambda_{1,h+1}$ and repeats the process. The model with the penalty parameter combination $(\lambda_{0,min}, \lambda_{1,h+1})$ is initialized using the model from the previous $(\lambda_{0,min}, \lambda_{1,h})$ penalty parameter combination.

4.3 Optimal tuning parameter selection

Once models have been fit pertaining to all tuning parameter combinations within the first and second stages of the tuning parameter search strategy (or over the full tuning parameter grid search), the `glmmPen` package chooses the best model from one of several BIC-type selection criteria options. For simplification of notation, consider the generic penalty parameter combination $\lambda = (\lambda_0, \lambda_1)$ that penalizes the fixed and random effects, respectively. By default, the package uses the BIC-ICQ criterion (Ibrahim et al., 2011), where the abbreviation ICQ stands for “Information Criterion based on the Q-function”. This BIC-ICQ criteria is expressed below:

$$\text{BICq}(\theta_\lambda) = 2\{Q_1(\theta_\lambda | \alpha_0) + Q_2(\alpha_0)\} + d_\lambda \log(N) \\ \approx \left\{ -\frac{2}{M} \sum_{m=1}^M \sum_{k=1}^K \left[\log f(\mathbf{y}_k | \mathbf{X}_k, \alpha_{0,k}^{(m)}; \theta_\lambda) + \log \phi(\alpha_{0,k}^{(m)}) \right] \right\} + d_\lambda \log(N), \quad (12)$$

where θ_λ are the coefficients of the model fit with the penalty $\lambda = (\lambda_0, \lambda_1)$, α_0 are the posterior samples from a “minimal penalty model”—the model with either no penalty (when the number of random effects predictors is less than 5) or a minimum penalty used on the fixed and random effects—and $\alpha_{0,k}^{(m)}$ is the m^{th} posterior sample for group k from such a minimal penalty model, Q_1 and Q_2 were defined in Section “MCECM algorithm”, d_λ is the number of nonzero coefficients for the model (all nonzero β plus all nonzero γ), and N is the total number of observations in the data (N_{obs}).

The package can also calculate the traditional BIC criterion as specified below:

$$\text{BIC}(\boldsymbol{\theta}_\lambda) = -2\ell(\boldsymbol{\theta}_\lambda) + d_\lambda \log(N),$$

where $\boldsymbol{\theta}_\lambda$ are the coefficients of the penalization model, $\ell(\boldsymbol{\theta}_\lambda)$ is the marginal log-likelihood for the model, d_λ is the number of nonzero coefficients for the model, and N can be either the total number of observations in the data (N_{obs}) or the total number of independent observations (i.e., number of levels within the grouping factor, N_{grps}) in the data. The marginal log-likelihood is as follows:

$$\ell(\boldsymbol{\theta}) = \sum_{k=1}^K \ell_k(\boldsymbol{\theta}) = \sum_{k=1}^K \frac{1}{n_k} \log \int f(\mathbf{y}_k | \mathbf{X}_k, \boldsymbol{\alpha}_k; \boldsymbol{\theta}) \phi(\boldsymbol{\alpha}_k) d\boldsymbol{\alpha}_k. \quad (13)$$

There is a lack of consensus regarding the use of $\log(N_{obs})$ versus $\log(N_{grps})$ in the BIC penalty term for mixed models. For instance, the $\log(N_{obs})$ penalty is used in the R package **nlme** (Pinheiro et al., 2021), and the $\log(N_{grp})$ penalty is used in SAS proc NLMIXED (SAS Institute Inc., 2008; Delattre et al., 2014). In practice, the performance of the different versions of the BIC penalty term may depend on the true underlying model (Lorah and Womack, 2019; Delattre et al., 2014), with Delattre et al. (2014) observing that the $\log(N_{obs})$ penalty performed better when the true model had very few random components, and the $\log(N_{grp})$ penalty performed better when the true model had a large number of random components. Both Delattre et al. (2014) and Lorah and Womack (2019) suggest using some combination of these sample size definitions.

To this point, the package also allows the best model to be selected using a ‘hybrid’ BICh selection criteria developed by Delattre et al. (2014):

$$\text{BICh}(\boldsymbol{\theta}_\lambda) = -2\ell(\boldsymbol{\theta}_\lambda) + d_{\lambda,\beta} \log(N_{obs}) + d_{\lambda,\gamma} \log(N_{grps}), \quad (14)$$

where $d_{\lambda,\beta}$ and $d_{\lambda,\gamma}$ are the number of nonzero fixed and random effect coefficients, respectively.

In simulations not shown here (see content in GitHub repository https://github.com/hheiling/paper_glmmPen_RJournal for details), we found that the BIC-ICQ gave the best performance in choosing the correct set of fixed and random effects. The BIC and BICh methods tended to underestimate the number of true fixed effects compared to BIC-ICQ in the simulations we considered. However, in order to calculate the BIC-ICQ, a minimal penalty model needs to be fit using a small penalty (i.e., λ_{min}) on the fixed and random effects. Posterior samples from this minimal penalty model are then used to calculate the BIC-ICQ value for each model fit in the variable selection procedure. Depending on the size of the full model with all fixed and random effects predictors, this calculation can be time-intensive since fitting the model with a small penalty will keep many fixed and random effects predictors in the model.

Alternatively, the calculation of the BIC and BICh criteria require a calculation of the marginal log-likelihood $\ell(\boldsymbol{\theta})$ for each model. Since the integrals within $\ell(\boldsymbol{\theta})$ are intractable, we estimate the marginal log-likelihood using the corrected arithmetic mean estimator (CAME) described by Pajor (2017). We have found this CAME estimator to be relatively quick and easy to calculate, as well as consistent with the marginal log-likelihood estimate calculated by the package **lme4** (Bates et al., 2015) for a range of conditions (see content in GitHub repository https://github.com/hheiling/paper_glmmPen_RJournal for details).

To calculate the CAME, we focus on a single group k and define a set $A_k \subseteq \Theta$ as a subset of the parameter space of the random effects for group k , where $P(A_k)$ and $P(A_k | \mathbf{y}_k, \mathbf{X}_k; \boldsymbol{\theta})$ are nonzero probabilities. We first start with the knowledge

$$\begin{aligned} P(A_k | \mathbf{y}_k, \mathbf{X}_k; \boldsymbol{\theta}) &= \int_{A_k} \phi(\boldsymbol{\alpha}_k | \mathbf{y}_k, \mathbf{X}_k; \boldsymbol{\theta}) d\boldsymbol{\alpha}_k \\ &= \int_{\Theta} \frac{1}{f(\mathbf{y}_k | \mathbf{X}_k; \boldsymbol{\theta})} f(\mathbf{y}_k | \mathbf{X}_k, \boldsymbol{\alpha}_k; \boldsymbol{\theta}) \phi(\boldsymbol{\alpha}_k) I(\boldsymbol{\alpha}_k \in A_k) d\boldsymbol{\alpha}_k, \end{aligned} \quad (15)$$

where $I(\cdot)$ is an indicator function, $f(\mathbf{y}_k | \mathbf{X}_k; \boldsymbol{\theta}) = \int f(\mathbf{y}_k | \mathbf{X}_k, \boldsymbol{\alpha}_k; \boldsymbol{\theta}) \phi(\boldsymbol{\alpha}_k) d\boldsymbol{\alpha}_k$ is the marginal likelihood for group k , and all other terms are described in Section “Generalized linear mixed models”. The above relationship allows us to obtain the result:

$$\begin{aligned} f(\mathbf{y}_k | \mathbf{X}_k; \boldsymbol{\theta}) &= \frac{1}{P(A_k | \mathbf{y}_k, \mathbf{X}_k; \boldsymbol{\theta})} \int_{\Theta} f(\mathbf{y}_k | \mathbf{X}_k, \boldsymbol{\alpha}_k; \boldsymbol{\theta}) \phi(\boldsymbol{\alpha}_k) I(\boldsymbol{\alpha}_k \in A_k) d\boldsymbol{\alpha}_k \\ &= \frac{1}{P(A_k | \mathbf{y}_k, \mathbf{X}_k; \boldsymbol{\theta})} \int_{\Theta} \frac{f(\mathbf{y}_k | \mathbf{X}_k, \boldsymbol{\alpha}_k; \boldsymbol{\theta}) \phi(\boldsymbol{\alpha}_k) I(\boldsymbol{\alpha}_k \in A_k) s(\boldsymbol{\alpha}_k)}{s(\boldsymbol{\alpha}_k)} d\boldsymbol{\alpha}_k, \end{aligned} \quad (16)$$

where $s(\cdot)$ is an importance sampling function.

Suppose at the end of the MCECM algorithm we obtain M samples from the posterior distribution

of the random effects for group k , $\tilde{\alpha}_k = ((\alpha_k^{(1)})^\top, \dots, (\alpha_k^{(M)})^\top)^\top$. Let us set $A_k = \tilde{\alpha}_k$; this reduces $P(A_k|y_k, X_k; \theta)$ to 1. Let us also set the importance sampling function $s(\cdot)$ to be a multivariate normal distribution with a mean vector equal to the mean of the posterior samples $\frac{1}{M} \sum_{m=1}^M \alpha_k^{(m)}$ and a covariance matrix equal to the covariance matrix of a thinned subset of the posterior samples (to obtain a pseudo-independent set of samples). If we draw M^* samples $\alpha_k^{*m} = ((\alpha_k^{*(1)})^\top, \dots, (\alpha_k^{*(M^*)})^\top)^\top$ from this importance sampling function, then Equation 16 indicates that we can estimate the marginal likelihood for group k as

$$f(y_k|X_k; \theta) \approx \frac{1}{M^*} \sum_{m=1}^{M^*} \frac{f(y_k|X_k, \alpha_k^{*m}; \theta) \phi(\alpha_k^{*m}) I(\alpha_k^{*m} \in A_k)}{s(\alpha_k^{*m})}. \quad (17)$$

We repeat the estimation in Equation 17 for all K groups in order to calculate the full desired marginal log-likelihood $\ell(\theta)$. This final marginal log-likelihood is then used in the previously mentioned BIC and BICh calculations for each fitted model across the λ_0 and λ_1 grid search. We refer to this marginal log-likelihood as the Pajor log-likelihood in later sections of the paper.

5 Software

The main function of the `glmmPen` package is `glmmPen`, which is used to perform fixed and random effects variable selection after the specification of a full model with all candidate fixed and random effects. The `glmmPen` package is also capable of fitting a GLMM with pre-specified fixed and random effects (under no penalization) using the function `glmm`. Here we will use the basal dataset (Rashid et al., 2020) to illustrate the use of the `glmmPen` function in practical applications.

5.1 Data example

The basal dataset is composed of four studies that contain gene expression data and tumor subtype information from patients spanning three cancer types (Moffitt et al., 2015; Weinstein et al., 2013). Two of these datasets contain gene expression data for subjects with Pancreatic Ductal Adenocarcinoma (PDAC), one dataset contains data for subjects with Breast Cancer, and the last contains data for subjects with Bladder Cancer. While each cancer type has separate sets of defined subtypes, all share a common subtype defined as “basal-like”, which was shown to be similar in character across cancer types and have an impact on survival (Moffitt et al., 2015). The goal of the original study was to select features that are relevant in predicting the basal-like subtype. To increase the sample size, it was proposed that samples were merged from each study into one large dataset.

Multiple approaches have been proposed to integrate gene expression data from multiple studies to improve the accuracy of downstream prediction models (Riester et al., 2014; Ma et al., 2018; Patil and Parmigiani, 2018). The pGLMM methodology from Rashid et al. (2020) was originally motivated by the need to select genes that are predictive of cancer outcomes (e.g. cancer subtype), where the effects of genes may vary randomly across studies. It was shown that accounting for this heterogeneity improved the performance of gene selection after data merging.

Using `glmmPen` package, we fit a pGLMM that can accommodate a large number of features in the model and account the heterogeneity in gene effects across studies. It is unclear *a priori* which features truly have a non-zero association with the outcome and which features truly have variation in their effects across studies. Therefore, we will use the `glmmPen` function to simultaneously select fixed and random effects from a set of candidate features. Rashid et al. (2020) integrated gene expression data from each study using a binary rank transformation technique (Top Scoring Pairs or TSPs), which we use as our covariates in this example. To illustrate the concept of TSPs, let $g_{ki,A}$ and $g_{ki,B}$ be the raw expression of genes A and B in subject i of group k . For each gene pair $(g_{ki,A}, g_{ki,B})$, the TSP is the indicator $I(g_{ki,A} > g_{ki,B})$ which specifies which gene of the two has higher expression in the subject. We denote a TSP predictor as “GeneA_GeneB”. A total of 50 binary TSP covariates are provided in the basal dataset available in the package. For illustration purposes, we randomly select 10 TSP covariates. Our goal is to identify TSPs that are associated with patient tumor subtype while accounting for study-level heterogeneity in gene effects. In each study subtype is defined a binary variable with two levels: basal-like or non-basal-like. Therefore, for this example, our example dataset consists of our matrix of covariates X , our subtype vector y (a factor with two levels), and our study membership vector (a factor with four levels).

Summary information about the data is included below.

```
> library("glmmPen")
> data("basal")
> y = basal$y
> set.seed(1618)
> idx = sample(1:50, size = 10, replace = FALSE)
> idx = idx[order(idx)]
> X = basal$X[,idx]
> colnames(X)

[1] "GPR160_CD109"   "SPDEF_MFI2"    "CHST6_CAPN9"   "SLC40A1_CDH3"
[5] "PLEK2_HSD17B2"  "GPX2_ER01L"   "CYP3A5_B3GNT5" "LY6D_ATP2C2"
[9] "MYO1A_FGFBP1"   "CTSE_COL17A1"

> group = basal$group
> levels(group)

[1] "UNC_PDAC"      "TCGA_PDAC"     "TCGA_Bladder"  "UNC_Breast"
```

We will fit a penalized random effects logistic regression model using the `glmmPen` function to model patient subtype, as it is unclear which of the 10 TSPs should be included in the model, and which may also randomly vary across studies in their effects. We perform variable selection using the following code:

```
> set.seed(1618)
> fitB = glmmPen(formula = y ~ X + (X | group),
+                  family = "binomial", covar = "independent",
+                  tuning_options = selectControl(BIC_option = "BICq",
+                                                 pre_screen = TRUE,
+                                                 search = "abbrev"),
+                  penalty = "MCP", BICq_posterior = "Basal_Posterior_Draws")
```

Here we utilize the pre-screening and abbreviated grid search options, as well as select the optimal tuning parameter using the BIC-ICQ model selection criteria (denoted “BICq”). Further details about the pre-screening procedure is described in the Section “`selectControl` arguments” and the consequences of this pre-screening procedure are illustrated through simulations and discussed in Section “Pre-screening performance”. If we were instead interested in fitting a GLMM utilizing all 10 TSPs as fixed effects and assuming a random effect for each (without penalization), we could run the following code:

```
> set.seed(1618)
> fit_glmm = glmm(formula = y ~ X + (X | group),
+                  family = "binomial", covar = "independent",
+                  optim_options = optimControl())
```

The set of random effects specified does not necessarily have to be equal to the set of fixed effects as in the above example. Because of the number of random effects that we are considering in the model, we approximate the random effects covariance matrix as an independent, or diagonal, matrix, which we specify by using the argument option `covar = "independent"`. Our reasoning for such an approximation, as well as a discussion of the pros and cons of such an approximation, are given in Section “Initialization and convergence.”

In the following subsections, we will discuss in detail the `glmmPen` (and `glmm`) arguments and relevant output. We will also examine the output from the variable selection procedure given by the `glmmPen` example.

5.2 Full model specification

The syntax for specifying the full model formula (the model with all relevant fixed and random effects predictors) using the `formula` argument closely follows the formula syntax of the `lme4` package (Bates et al., 2015). The formula follows the form `response ~ fix_expr + (rand_expr | factor)` where the `fix_expr` specifies the variables to use as the fixed effects, the `rand_expr` specifies the variables to use as the random effects, and the `factor` specifies the grouping factor of the observations. When a data frame is given for the `data` argument, the fixed and random effects can be specified using the column names of the data frame. For higher-dimensional data, users may find it easier to directly specify the

matrix containing the covariates of interest and the response vector, such as the $y \sim X + (X \mid \text{group})$ formula given in the earlier `glmmPen` fit example. No specification of the `data` argument is needed in this case. Similar to `ncvreg`, an intercept is always assumed and required, and therefore an intercept column need not be specified in `X` or explicitly in the model formula; `glmmPen` will output an error if the input predictor matrix `X` contains an intercept column.

Regarding the specification of random effects in `formula`, the `glmmPen` package currently does not allow for multiple grouping factors. In addition, the random effects must be a subset of the fixed effects, and a random intercept is always assumed and required in the model. Lastly, the structure of the random effects covariance matrix is determined by the `covar` argument, which may take on the value of ‘unstructured’ or ‘independent’ (diagonal). By default, the `covar` parameter is set to `NULL`. This automatically selects the ‘independent’ option if the number of random effect predictors is 10 or more and selects ‘unstructured’ otherwise. For a large number of random effect predictors, it is strongly recommend that the covariance structure to ‘independent’ in order to improve computational efficiency.

The `glmmPen` algorithm allows the Binomial, Gaussian, and Poisson families with canonical links.

5.3 Penalization and optimal tuning parameter selection

In `glmm`, the default is to fit the single model with user-specified fixed and random effects with no penalization. Although it is generally not recommended, users have the option to specify a single penalty parameter combination using

`tuning_options = lambdaControl(lambda0, lambda1)`. In `glmmPen`, the arguments `penalty`, `gamma_penalty`, `alpha`, `fixef_noPen`, and `tuning_options` all play a part in the variable selection process. The following subsections discuss these argument options in detail and how the arguments impact variable selection.

Penalty, gamma penalty, alpha parameters

To perform variable selection, `glmmPen` allows the fixed effect coefficients to be penalized using the minimax concave penalty (MCP, the default), the smoothly clipped absolute deviation (SCAD) penalty, or the least absolute shrinkage and selection operator (LASSO) penalty (Breheny and Huang, 2011; Friedman et al., 2010) via the `penalty` argument, which takes as input the character strings “MCP”, “SCAD”, or “lasso”. The random effects are then penalized using the grouped version of the selected penalty type (Breheny and Huang, 2015), e.g., if the MCP penalty is used to penalize the fixed effects, then the grouped MCP penalty is used to penalize the random effects covariance matrix coefficients.

In addition to the previously discussed penalty parameters (λ_0, λ_1), the MCP and SCAD penalties also use a scaling factor (Breheny and Huang, 2011, 2015). The argument `gamma_penalty` specifies this scaling factor, with the default of 3 and 4 for the MCP and SCAD penalties, respectively. Additionally, the argument `alpha` allows for the elastic net estimator, controlling the relative contribution of the MCP/SCAD/LASSO penalty and the ridge, or L_2 , penalty. Setting `alpha` to 1 (the default) is equivalent to the regular penalty with no L_2 contribution.

selectControl arguments

The grid search over the fixed effects and random effects penalty parameters λ_0 and λ_1 is controlled by the arguments in `selectControl()`. The user can specify particular sequences for λ_0 (fixed effects penalty parameters) and λ_1 (random effects penalty parameters) using the arguments `lambda0_seq` and `lambda1_seq`, respectively; by default, a sequence of penalty parameters (of length `nlambda`, default 10) are automatically calculated within `glmmPen`. These default sequences are calculated using the method discussed in Section “Tuning parameter selection”. The minimum penalty λ_{min} is a small fraction of the λ_{max} value; the argument `lambda.min` controls what fraction is used. By default, `lambda.min = 0.01` so that $\lambda_{min} = 0.01(\lambda_{max})$.

The structure of the optimal tuning parameter search is specified by the argument `search`. If `search = "abbrev"` (default), the algorithm performs the abbreviated two-stage tuning parameter search specified in Section “Tuning parameter selection”. If `search = "full_grid"`, the algorithm looks over the full grid search of $\text{length}(\text{lambda0_seq}) \times \text{length}(\text{lambda1_seq})$ models before picking the best model.

After all of the tuning parameters have been evaluated, the optimal combination of tuning parameters can be selected using a BIC-type selection criteria, which can be specified using the `selectControl()` argument `BIC_option`. Using the `BIC_option` argument, the user can select one of four BIC-type selection criteria, given in Table 1, to select the best model.

Selection criteria	Description
BICq	(Default) BIC-ICQ selection criteria (Ibrahim et al., 2011); requires fitting the minimal penalty model
BICh	Alternative BICh selection criteria specified by Delattre, Lavielle, and Poursat (2014)
BIC	Traditional BIC whose penalty term sets N to the number of total observations in the data
BICNgrp	Traditional BIC whose penalty term sets N to the number of independent observations (i.e., number of levels of the grouping factor)

Table 1: BIC-type model selection criteria options for argument BIC_option.

Refer to the discussion in Section “Tuning parameter selection” for further details about these BIC-type options as well as their respective pros and cons.

The argument pre_screen allows users to screen out some random effects at the start of the algorithm. When pre_screen is set to TRUE (the default) and the number of random effects predictors is 5 or more, a minimal penalty model is fit using a small penalty for the fixed and random effects and relatively lax convergence criteria. If at the end of the pre-screening procedure the variance of a random effect is penalized to 0 or is estimated to be less than 10^{-2} , that predictor is restricted to have a zero-valued random effect variance for all models fit by the algorithm. The pre-screening procedure is not implemented if the number of random effects is less than five. This threshold of five random effect predictors is an ad hoc choice by the authors; the purpose of the pre-screening procedure is to allow the user to speed up the variable selection procedure when the full model contains a large number of random effects.

The argument lambda.min.presc adjusts the value of the random effect penalty parameter λ_1 used in the pre-screening step and the minimal penalty model fit for the BIC-ICQ calculation, where the minimum penalty used for the random effects is $\text{lambda}.\text{min}.\text{presc} \times \lambda_{\max}$. See package documentation for further details about this argument and other minor arguments not discussed here.

Additional selection arguments in glmmPen

The default variable selection procedure assumes that we have no prior knowledge of which fixed effects should not be penalized during the model fitting procedure. In order to indicate that a covariate should not be subject to penalization (and therefore always remain in the model), one can use the fixef_noPen argument. See the `glmmPen` function documentation for further details.

After running an initial grid search over the default fixed and random effects penalty parameters, users may desire to re-run the variable selection procedure using alternative settings, such as different penalty sequences (e.g. a finer grid search) or different convergence criteria. In this scenario, re-computing the the minimal penalty model for the BIC-ICQ criterion calculation can be time-consuming. In order to save the minimal penalty model posterior samples needed for the BIC-ICQ calculation and re-use these samples to compute the BIC-ICQ within a subsequent tuning parameter selection grid search, the user can save the posterior samples as a file-backed `big.matrix` using the argument `BICq_posterior = "file_location/file_name"`. This saves the backing file and the descriptor file as '`file_location/file_name.bin`' and '`file_location/file_name.desc`', respectively. If the file name is not specified, then the posterior samples are automatically saved to the working directory with the file name "`BICq_Posterior_Draws`". These saved posterior samples can then be re-loaded in R as a `big.matrix` using the `attach.big.matrix` function from the package `bigrnmemory` (Kane et al., 2013). In secondary calculations using `glmmPen`, the recalculation of this minimal penalty model fit can be avoided and these posterior samples can be used by calling `BICq_posterior = "file_location/file_name"`.

5.4 Examination of output

The `glmm` and `glmmPen` functions all output a reference class object of class `pglmmObj`. A full list of the methods available for `pglmmObj` objects are provided in Table 2. These methods and their output were designed to be very similar to the methods and output available for `merMod` objects used in the `lme4` package. Further information about the output provided in a `pglmmObj` object and additional methods documentation is available in the `glmmPen` package documentation (see `?pglmmObj`).

When the `pglmmObj` object is created using the `glmm` function, the output from the methods listed in Table 2 pertains to the single model fit specified by the `glmm` arguments. When the `pglmmObj` object

Generic	Brief description of return value
BIC	Numeric vector returning the BIC, BICh, BICNgrp, and, if specified for model selection, BIC-ICQ selection criteria evaluations for either the fitted glmm model or the optimal fitted glmmPen model (i.e. the ‘best’ model according to the model selection criteria)
coef	Matrix reporting the sum of the fixed effects coefficients and the posterior modes of the random effects for each variable at each level of the grouping factor
fitted	Numeric vector of fitted values (the values of the linear predictor) based on either the fixed effects only (recommended for most applications) or both the fixed effects and the posterior modes of the random effects for each level of the grouping factor (potentially useful for diagnostics)
fixef	Numeric vector of the fixed effects coefficient estimates $\hat{\beta}$
formula	The mixed-model formula of the fitted model
logLik	Estimated log-likelihood for the best model of the glmmPen procedure or the final model from glmm evaluated using the Pajor (2017) marginal likelihood calculation discussed in Section “Tuning parameter selection”
model.frame	A data.frame object containing the output and predictors used to fit the model
model.matrix	The fixed-effects model matrix
ngrps	Number of levels in the grouping factor
nobs	Number of total observations
plot	Diagnostic plots for mixed-model fits
predict	Predicted values based on either the fixed effects only (recommended) or the combined fixed effects and posterior modes of the random effects for each variable and each level of the grouping factor
print	Basic printout of mixed-model objects
ranef	Matrix of posterior modes of the random effects for each variable and each level of the grouping factor
residuals	Numeric vector of residual values: deviance (default), Pearson, response, or working residuals
sigma	Random effect covariance matrix ($\Gamma\Gamma^\top$)
summary	Summary of the mixed model results

Table 2: List of currently available methods for objects of class pg1mmObj.

is created using the glmmPen function, the output from the methods pertains to the best model chosen during the model selection procedure. Additional information about each model fit can be found in the results_all field of the pg1mmObj object. Using the basal output object fitB from the glmmPen function, we illustrate the use of several of these methods in the remainder of this section.

Model summary

The summary method output the function call information such as the sampler used in the E-step (in this case, Stan), the family, the model formula, the estimates of the fixed effects, the variance and standard deviation estimates of the random effects, and a summary of the deviance residuals. (Note: Due to the style of our formula specification using a matrix instead of column names of a data.frame, all variable names begin with the name of the matrix, X.)

```
> summary(fitB)

Penalized generalized linear mixed model fit by Monte Carlo Expectation
Conditional Minimization (MCECM) algorithm (Stan)  ['pg1mmObj']
Family: binomial ( logit )
Formula: y ~ X + (X | group)

Fixed Effects:
  (Intercept) XGPR160_CD109   XSPDEF_MFI2    XCHST6_CAPN9   XSLC40A1_CDH3
-1.1530        -0.7099       -0.7355       0.5082        -0.5831
  XPLEK2_HSD17B2  XGPX2_ER01L  XCYP3A5_B3GNT5  XLY6D_ATP2C2  XMY01A_FGFBP1
  0.4337        -0.5895       0.0000        0.4620        -0.7411
```

```
XCTSE_COL17A1
 0.0000

Random Effects:
Group Name      Variance Std.Dev.
group (Intercept) 0.8193  0.9052
group XGPR160_CD109 0.2036  0.4512
group XSPDEF_MFI2  0.684   0.827
group XCHST6_CAPN9 0       0
group XSLC40A1_CDH3 0       0
group XPLEK2_HSD17B2 0.0804  0.2835
group XGPX2_ER01L  0.0842  0.2901
group XCYP3A5_B3GNT5 0       0
group XLY6D_ATP2C2 0       0
group XMY01A_FGFBP1 0.1551  0.3938
group XCTSE_COL17A1 0.7596  0.8715
Number Observations: 938, groups: group, 4
```

```
Deviance residuals:
  Min    1Q Median    3Q   Max
-2.9338 -0.4026 -0.1512  0.3457  2.9630
```

We see that the best model included 9 TSPs with non-zero fixed effects and 6 TSPs with non-zero random effects (i.e., 6 TSPs with varying predictor effects across the studies). The print method supplies very similar information to the summary method minus the summary of the residuals.

The individual components of the print and summary outputs can be obtained using several accessory functions described in Table 2. Similar to the package `lme4`, the fixed effects can be summarized using `fixef` and the group-specific random effects can be summarized using `ranef`. The random effect covariance matrix is summarized using `sigma`. In the case of the Gaussian family, `sigma` also provides the residual standard error.

```
> fixef(fitB)
> ranef(fitB)
> sigma(fitB)
```

The residuals for the final model can be called using the `residuals` method. The different type options for the residuals include “deviance”, “pearson”, “response”, and “working”, which correspond to the deviance, Pearson, response, and working residuals, respectively.

```
> residuals(fitB, type = "deviance")
```

Predictions and fitted values

Using the `predict` method, we can make predictions using only the population level information (i.e., the fixed effects only) or the group-specific level information (i.e., the fixed and random effects results). The `glmmPen` package restricts predictions on new data to only use the fixed effects since it is generally unlikely that the grouping levels within other datasets will exactly match the grouping levels within the data used to create the prediction model. The `predict` method has the following arguments:

- `object`: an object of class `pglmmObj` output from `glmm` or `glmmPen`.
- `newdata`: a data frame of new data that contains all of the fixed effects covariates from the model fit. The variables provided in `newdata` must match the fixed effects used in the model fit.
- `type`: a character string specifying whether to output the linear predictor (“link”, default) or the expected mean response (“response”).
- `fixed.only`: boolean value specifying if the prediction is made with only the fixed effects (TRUE, default) or both the fixed and random effects (FALSE). Predictions are restricted to `fixed.only = TRUE` for new data predictions.

The `fitted` method also includes the `fixed.only` argument, allowing the fitted values of the linear predictor to be estimated with or without the random effects estimates.

```
> predict(object = fitB, newdata = NULL, type = "link", fixed.only = TRUE)
> fitted(object = fitB, fixed.only = TRUE)
```

Diagnostics

The `glmmPen` package provides methods to perform diagnostics on the final model fit object. The `plot` method plots the residuals against the fitted values. The `plot` function defaults to plotting the Pearson residuals for the Gaussian family, and deviance residuals otherwise.

```
> plot(object = fitB)
```

The `plot_mcmc` function performs graphical MCMC diagnostics on the random effect posterior samples. This command has six arguments with the first argument specifying the `pglmmObj` output object. The second argument `plots` is used to specify which diagnostics plots to produce. The `plots` argument is capable of creating sample path plots ("sample.path", default), autocorrelation plots ("autocorr"), cumulative sum plots ("cumsum"), and histograms ("histogram") of the posterior samples. The plots are output as faceted `ggplot2` (Wickham, 2016) plots with the graphics arranged by groups in the columns and variables in the rows. As objects of class `ggplot`, they are capable of being edited as any other `ggplot` object. The `plots` argument can specify a vector of multiple plot types or the choice of "all", which automatically produces all four types of diagnostic plots. The function outputs a list object containing the plots specified. The third and fourth arguments `grps` and `vars` allow the user to restrict which groups and/or variables are summarized in the diagnostic plots. The default values of "all" for these arguments give the results for all groups and variables. To request specific groups and variables, provide vectors of character strings specifying the variable or group names. The argument `numeric_grp_order` tells the function to order the group levels numerically (default FALSE), and `bin_width` allows the user to manipulate the bin widths of the histograms (default NULL results in `geom_histogram` defaults, only relevant if the "histogram" plot is requested).

The example code below specifies the names of three of TSP predictors with non-zero random effects across the studies and then uses the `plot_mcmc` function to produce the sample path plots and autocorrelation plots for the corresponding posterior samples. Some plot aesthetics are adjusted using the `ggplot2` package (Wickham, 2016). These sample path and autocorrelation plots can be seen in Figure 1.

```
> TSP = c("XGPR160_CD109", "XSPDEF_MFI2", "XPLEK2_HSD17B2")
> plot_diag = plot_mcmc(object = fitB, plots = c("sample.path", "autocorr"),
+                         grps = "all", vars = TSP)
> library("ggplot2")
> plot_diag$sample_path + theme(axis.text.x = element_text(angle = 270))
> plot_diag$autocorr
```

5.5 Optimization

Additional optimization control options can be passed to the `glmm` and `glmmPen` functions using the `optim_options` argument and the `optimControl()` control structure. Some default settings in `optimControl` depend on the family of the data or the number of random effects. Descriptions of several of the main `optimControl()` arguments and their defaults are listed below. Disclaimer: Some optimization argument default values may be refined in future versions of the package if additional package testing suggests that changes could improve package performance (e.g., adjustments for certain data conditions or outcome families); please check the current `glmmPen` documentation for the most up-to-date default information.

`sampler`: a character string specifying the sampling type used in the E-step of the MCECM algorithm. The default sampler is "stan", which requests the No-U-Turn Hamiltonian Monte Carlo sampling performed by the `rstan` package (Stan Development Team, 2020; Carpenter et al., 2017). We strongly recommend using this sampling method due to its speed and efficiency. Other options include "random_walk", which requests the Metropolis-within-Gibbs adaptive random walk sampler (Roberts and Rosenthal, 2009), or "independence", which requests the Metropolis-within-Gibbs independence sampler (Givens and Hoeting, 2012).

`var_start`: either a character string "recommend" (default) or a positive numeric value. This argument specifies the initial starting variance of the random effects covariance matrix. If `var_start` is set to "recommend", the function fits a fixed and random intercept only model using the `lme4` package and sets the starting variance to the random intercept variance multiplied by 2. The random effects covariance matrix is initialized as a diagonal matrix with the value of `var_start` as the diagonal elements.

`var_restrictions`: either a character string "none" (default) or the character string "fixef". This argument can be used to restrict which random effects are considered at the start of the algorithm. If this argument is set to "none", then all random effect predictors are initialized to have a non-zero

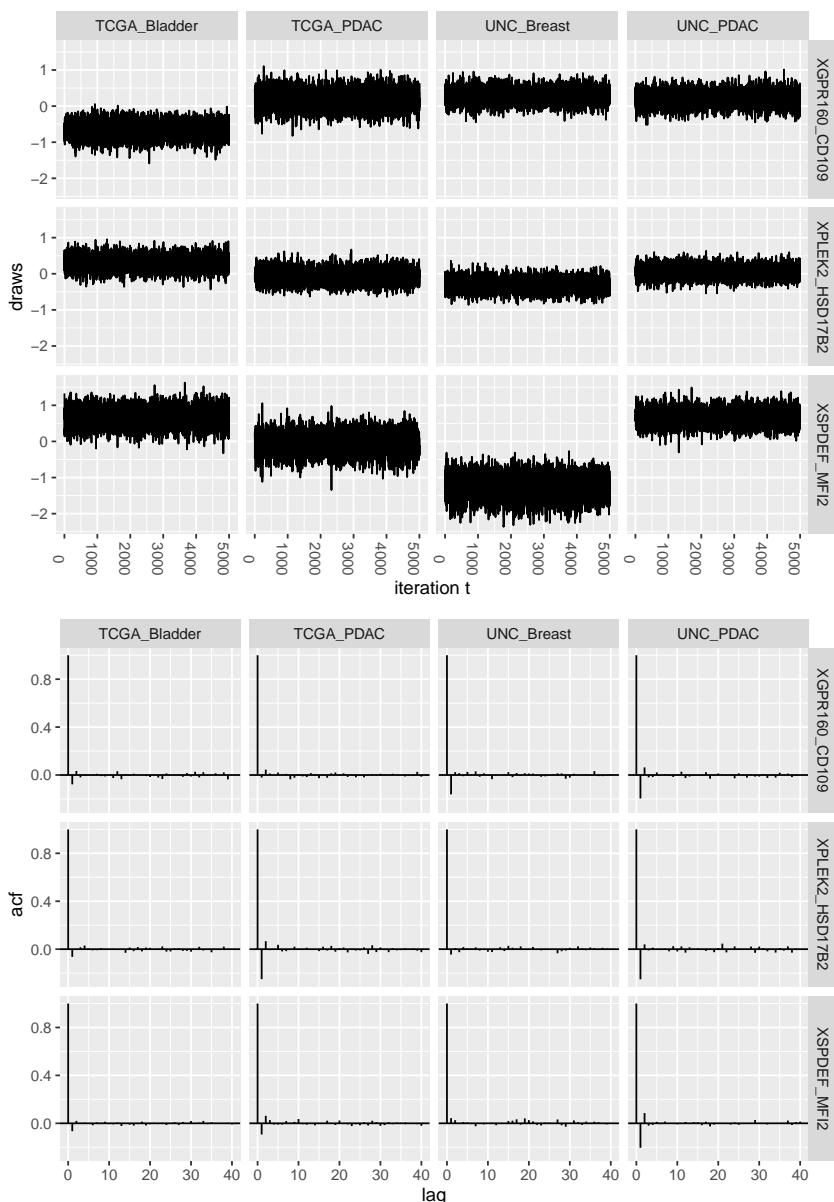


Figure 1: MCMC diagnostic plots for the basal best model results, created using the `plot_mcmc` function. Top: Sample path plots for the random slopes of three TSP covariates. Bottom: Autocorrelation plots for the random slopes of three TSP covariates.

variance in the random effect covariance matrix. If this argument is set to “fixef”, then only the random effect predictors that are initialized to have non-zero fixed effects estimates during the fixed effects initialization procedure are given non-zero variances when initializing the random effect covariance matrix. In effect, this restricts predictors that are initialized with zero-valued fixed effects coefficients to not have random effects. See `glmmPen` simulation results utilizing this feature within the GitHub repository https://github.com/hheiling/paper_glmmPen_RJournal. By using this restriction, the user assumes that predictors penalized out of the naive model do not have random effects. While this could be a strong assumption, using this restriction can be helpful in speeding up the algorithm by removing excessive random effects at the start of the variable selection procedure.

`conv_EM`: a positive numeric value specifying the convergence threshold for the EM algorithm. The argument `conv_EM` specifies the value of ϵ in Equation 11. The default value is 0.0015.

`t`: a positive integer that specifies the value of t in the EM algorithm convergence criteria specified in Equation 11. The convergence criteria is based on the average Euclidean distance between the most recent coefficient estimate and the coefficient estimate from t EM iterations back. Default value is set to 2.

`mcc`: a positive integer indicating the number of times the EM convergence criteria must be met before the algorithm is seen as having converged (`mcc` short for ‘meet condition counter’). Default value is set to 2, and `mcc` is restricted to be no less than 2.

`maxitEM`: The maximum number of EM iterations allowed by the algorithm. When the default value of `NULL` is input, `maxitEM` is set to a value that depends on the family type of the data. For the Binomial and Poisson families, the default is set to 50. For the Gaussian family, the default is set to 100 (we have observed that the Gaussian family data generally takes longer to converge).

Additional optimization parameters include M-step convergence parameters (`conv_CD`, `maxit_CD`), parameters specifying the number of posterior samples to acquire for the E-step throughout the algorithm (`nMC_burnin`, `nMC_start`, and `nMC_max`), the number of posterior samples needed to calculate the log-likelihood (`M`), and the number of posterior samples to save for diagnostics (`nMC_report`). Additional details about these parameters and their defaults can be found in the [glmmPen](#) documentation of the function `optimControl`.

6 Simulations

In this section, we present results from simulations in order to examine the performance of our package. We use the [glmmPen](#) package to perform variable selection and examine the resulting fixed effects estimates as well as the true and false positives for the fixed and random effects. All simulations are performed using the default optimization settings discussed in Section “Optimization”. While the performance of the original implementation of the pGLMM algorithm was demonstrated in [Rashid et al. \(2020\)](#), here we confirm the performance of the computational improvements made since then as well as newer features such as the pre-screening procedure.

6.1 Simulation set-up

We simulated binary responses from a logistic mixed-effects regression model with p predictors. Of p total predictors, we assume that 2 predictors have truly non-zero fixed and random effects, and the other $p - 2$ predictors have zero-valued fixed and random effects. Our aim in the simulations was to select the true predictors.

In these simulations, we consider the following situations: predictor dimensions of $p = \{10, 50\}$, sample size $N = 500$, number of groups $K = \{5, 10\}$, and standard deviation of the random effects $\sigma = \{1, \sqrt{2}\}$. As discussed in Section “Generalized linear mixed models”, we approximate the covariance matrix of the random effects as a diagonal matrix for these higher dimensions. We further consider the scenarios of moderate predictor effects, where the true fixed effects are $\beta = (0, 1, 1)^\top$.

For group k , we generated the binary response y_{ki} , $i = 1, \dots, n_k$ such that $y_{ki} \sim Bernoulli(p_{ki})$ where $p_{ki} = P(y_{ki} = 1 | x_{ki}, z_{ki}, \alpha_k, \theta) = \exp(x_{ki}^\top \beta + z_{ki}^\top \alpha_k) / \{1 + \exp(x_{ki}^\top \beta + z_{ki}^\top \alpha_k)\}$, and $\alpha_k \sim N_3(0, \sigma^2 I_3)$. The fixed effect coefficients were set to $\beta = (0, 1, 1)^\top$ (moderate predictor effects). We also simulated imbalance in sample sizes between the groups. Of the N samples, $N/3$ samples were given to study $k = 1$ and the remaining $2N/3$ samples were evenly distributed among the remaining studies. Each condition was evaluated using 100 total simulated datasets.

For individual i in group k , the vector of predictors for the fixed effects was $x_{ki} = (1, x_{ki,1}, \dots, x_{ki,p})^\top$, and we set the random effects $z_{ki} = x_{ki}$, where $x_{ki,j} \sim N(0, 1)$ for $j = 1, \dots, p$.

Setting the input random effects equal to the fixed effects represents the worst-case scenario where we have no idea what predictors may or may not have random effects. This may be an extreme assumption; in many real-world scenarios, users will have reason to set the input random effects to a strict subset of the fixed effects.

In all of these simulations, we use the default settings discussed earlier, which includes using the default λ_0 and λ_1 penalty sequences, BIC-ICQ for the selection criteria, pre-screening, and the MCP penalty. For all simulations, we performed the abbreviated two-stage grid search as described in Section “Tuning parameter selection”. The results for these simulations are presented in Table 3. These results include the average coefficients, the average true positive and false positive percentages for both fixed and random effects, and the median time for the simulations to complete. The true positive percentages reflect the average percent of the true predictors included in the best models chosen by the BIC-ICQ model selection criteria, which should ideally be near 100%. Likewise, the false positive percentages reflect the average percent of the false predictors included in the best models, which should ideally be near 0%. All simulations were completed on the UNC Longleaf computing cluster (CPU Intel processors between 2.3Ghz and 2.5GHz).

By examining the simulation results, we can observe that the performance of the variable selection procedure in [glmmPen](#) is impacted by the underlying structure of the data. As the magnitude of the

N	p	K	σ	$\hat{\beta}_1$	$\hat{\beta}_2$	TP % Fixed	FP % Fixed	TP % Random	FP % Random	T^{median} (hours)
500	10	5	1	1.02	1.12	89.0	2.1	90.5	3.5	0.20
			$\sqrt{2}$	1.12	1.18	83.0	1.4	96.0	3.6	0.26
	10	10	1	0.99	1.04	99.0	3.0	95.0	4.8	0.24
			$\sqrt{2}$	1.02	1.11	91.0	1.8	99.5	7.0	0.32
500	50	5	1	1.18	1.14	84.5	1.2	83.5	2.2	8.07
			$\sqrt{2}$	1.42	1.43	75.5	2.5	89.0	2.5	12.20
	10	10	1	1.12	1.11	95.0	1.8	93.0	3.9	10.67
			$\sqrt{2}$	1.33	1.31	84.5	2.4	95.5	6.2	15.75

Table 3: Variable selection simulation results with moderate predictor effects (slopes equal to 1). Results include the estimated coefficients for true non-zero fixed effects, true positive (TP) percentages for fixed and random effects, false positive (FP) percentages for fixed and random effects, and the median time in hours for the algorithm to complete.

random effect variance increases, the true positive percentage of the fixed effects decreases and the true positive percentage of the random effects increases. Additionally, as the number of groups K increases, the true positive percentage of both the fixed and random effects increases. We see that as the dimension of the total number of predictors increases ($p = 10$ to $p = 50$), the true positive percentages of both the fixed and random effects decreases. In regards to the run time, Table 3 shows that increases in the number of groups and increases in the variance of the random effects generally increases the time for the algorithm to complete.

In simulations not presented in this paper, we saw that increases to the magnitude of the fixed effects (e.g., increasing the true slope to 2, see content in GitHub repository https://github.com/hheiling/paper_glmmPen_RJournal for details) increased the true positive fixed effects and generally decreased the true positive random effects.

6.2 Pre-screening performance

The time it takes the package to complete the tuning parameter selection procedure depends strongly on the number of random effects considered by the algorithm. Therefore, the pre-screening procedure, which reduces the number of random effects considered within the variable selection algorithm, speeds up the algorithm. Table 4 summarizes the performance of the pre-screening algorithm for the variable selection simulations described above. This table reports the average percent of true positive and false positive random effect predictors that remain under consideration within the variable selection procedure after the pre-screening step has completed. The pre-screening settings were the default settings described in Section “Software”, which include specifying `lambda.min.presc = 0.01` for $p = 10$ and `lambda.min.presc = 0.05` for $p = 50$ such that the minimum penalty on the random effects is $\text{lambda}.\text{min}.\text{presc} \times \lambda_{max}$. We note that there are currently no methods that are capable of scaling to the values of q random effect predictors evaluated in our simulation for estimating and performing variable selection in GLMMs.

N	p	K	σ	TP %	FP %
500	10	5	1	98.0	25.8
			$\sqrt{2}$	100.0	26.1
	10	10	1	100.0	33.0
			$\sqrt{2}$	100.0	32.2
500	50	5	1	96.0	24.6
			$\sqrt{2}$	96.5	25.7
	10	10	1	97.5	25.9
			$\sqrt{2}$	98.5	27.3

Table 4: Pre-screening results for variable selection simulations with moderate predictor effects (slopes equal to 1). Results include the true positive percentages and false positive percentages of the random effect predictors remaining after pre-screening.

Using this higher penalty in the $p = 50$ simulations helps reduce the false positive percentage of the random effects after pre-screening and consequently helps speed up the time of the algorithm to complete. However, we can see by comparing the $p = 50$ and $p = 10$ simulations that this approach can also slightly decrease the true positive percentage. In general, increasing `lambda.min.presc` will help decrease the number of false positive non-zero random effects in the pre-screening step, but it may also decrease the number of true positive non-zero random effects. Decreasing `lambda.min.presc` will generally have the opposite effect. We also see that the true positive percentage for the selection of the random effects after pre-screening is generally higher when the magnitude of the true random effect variance is higher.

7 Conclusion

This paper introduces the R package `glmmPen` for fitting penalized generalized linear mixed models, including Binomial, Gaussian, and Poisson models. The `glmmPen` package's main advantage over other packages that estimate GLMMs is that it can perform variable selection on the fixed and random effects simultaneously. The algorithm utilizes a Monte Carlo Expectation Conditional Minimization (MCECM) algorithm. Several established MCMC sampling techniques are available for the E-step, and a Majorization-Minimization coordinate descent algorithm is used in the M-step. The package utilizes the established methods of `Stan` and `RcppArmadillo` to increase the computational efficiency of the E-step and M-step, respectively. As a result, the `glmmPen` package can fit models with higher dimensions compared to other packages that fit GLMMs, supporting models with 50 or more fixed and random effects.

The `glmmPen` package employs several additional techniques to improve the speed of the algorithm. Such techniques include initialization of subsequent models with the coefficients from the previous model fit and pre-screening to remove unnecessary random effects.

The `glmmPen` package has several attributes that make it user-friendly. For one, the package was designed to have an interface that is similar to the `lme4` package, with which many users may be familiar. Additionally, the `glmmPen` package has several automated procedures that make it more convenient to use, including automated data-dependent initialization of the random effect covariance matrix and automated recommendations for the penalization parameters.

A unique aspect of the package is the calculation of the marginal log-likelihood. The corrected arithmetic mean estimator (CAME) calculation described by Pajor (Pajor, 2017) is relatively simple and fast to calculate, and we have found that it performs well when compared with the log-likelihood estimate used in the `lme4` package (see content in the GitHub repository https://github.com/hheiling/paper_glmmPen_RJournal). This marginal log-likelihood calculation allows the algorithm to perform tuning parameter selection using traditional BIC selection criterion as well as other BIC-derived selection criteria. This gives users the option to forgo calculating the BIC-ICQ selection criterion, which requires the minimal penalty model fit where a minimum penalty is applied to both the fixed and random effects.

In its current implementation at the time of this paper's publication, the `glmmPen` R package can apply to Binomial, Gaussian, and Poisson families with canonical links. In the future, we plan to extend the application of this package to survival data, to non-canonical links for the existing families, and to additional families such as the negative binomial family.

References

- F. A. Archila. `mcemGLM: Maximum Likelihood Estimation for Generalized Linear Mixed Models`, 2020. URL <https://CRAN.R-project.org/package=mcemGLM>. R package version 1.1.1. [p105, 108]
- D. Bates, M. Mächler, B. Bolker, and S. Walker. Fitting linear mixed-effects models using `lme4`. *Journal of Statistical Software*, 67(1):1–48, 2015. URL <https://doi.org/10.18637/jss.v067.i01>. [p105, 113, 115]
- B. M. Bolker, M. E. Brooks, C. J. Clark, S. W. Geange, J. R. Poulsen, M. H. H. Stevens, and J.-S. S. White. Generalized linear mixed models: a practical guide for ecology and evolution. *Trends in ecology & evolution*, 24(3):127–135, 2009. URL <https://doi.org/10.1016/j.tree.2008.10.008>. [p105]
- H. D. Bondell, A. Krishna, and S. K. Ghosh. Joint variable selection for fixed and random effects in linear mixed-effects models. *Biometrics*, 66(4):1069–1077, 2010. URL <https://doi.org/10.1111/j.1541-0420.2010.01391.x>. [p105, 111]

- J. G. Booth and J. P. Hobert. Maximizing generalized linear mixed model likelihoods with an automated monte carlo em algorithm. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(1):265–285, 1999. URL <https://doi.org/10.1111/1467-9868.00176>. [p108]
- P. Breheny and J. Huang. Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection. *Annals of Applied Statistics*, 5(1):232–253, 2011. URL <https://doi.org/10.1214/10-AOAS388>. [p105, 107, 111, 116]
- P. Breheny and J. Huang. Group descent algorithms for nonconvex penalized linear and logistic regression models with grouped predictors. *Statistics and Computing*, 25(2):173–187, 2015. URL <https://doi.org/10.1007/s11222-013-9424-2>. [p105, 107, 109, 116]
- 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>. [p107, 108, 120]
- Z. Chen and D. B. Dunson. Random effects selection in linear mixed models. *Biometrics*, 59(4):762–769, 2003. URL <https://doi.org/10.1111/j.0006-341X.2003.00089.x>. [p107]
- C. Dean and J. D. Nielsen. Generalized linear mixed models: a review and some extensions. *Lifetime data analysis*, 13:497–512, 2007. URL <https://doi.org/10.1007/s10985-007-9065-x>. [p105]
- M. Delattre, M. Lavielle, M.-A. Poursat, et al. A note on bic in mixed-effects models. *Electronic Journal of Statistics*, 8(1):456–475, 2014. URL <https://doi.org/10.1214/14-EJS890>. [p105, 113]
- M. Donohue, R. Overholser, R. Xu, and F. Vaida. Conditional akaike information under generalized linear and proportional hazards mixed models. *Biometrika*, 98(3):685–700, 2011. URL <https://doi.org/10.1093/biomet/asr023>. [p105]
- D. Eddelbuettel and R. François. **Rcpp**: Seamless r and c++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL <http://www.jstatsoft.org/v40/i08/>. [p110]
- D. Eddelbuettel and C. Sanderson. **RcppArmadillo**: Accelerating r with high-performance c++ linear algebra. *Computational Statistics and Data Analysis*, 71:1054–1063, March 2014. URL <https://doi.org/10.1016/j.csda.2013.02.005>. [p110]
- Y. Fan and R. Li. Variable selection in linear mixed effects models. *Annals of Statistics*, 40(4):2043, 2012. URL <https://doi.org/10.1214/12-AOS1028>. [p111]
- D. J. Feaster, S. Mikulich-Gilbertson, and A. M. Brincks. Modeling site effects in the design and analysis of multi-site trials. *The American journal of drug and alcohol abuse*, 37(5):383–391, 2011. URL <https://doi.org/10.3109/00952990.2011.600386>. [p105]
- G. M. Fitzmaurice, N. M. Laird, and J. H. Ware. *Applied Longitudinal Analysis*, volume 998. John Wiley & Sons, 2nd edition, 2012. URL <https://doi.org/10.1002/9781119513469>. [p105]
- J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010. URL <https://www.jstatsoft.org/v33/i01/>. [p105, 107, 116]
- R. I. Garcia, J. G. Ibrahim, and H. Zhu. Variable selection for regression models with missing data. *Statistica Sinica*, 20(1):149, 2010. URL <https://pubmed.ncbi.nlm.nih.gov/20336190/>. [p107]
- G. H. Givens and J. A. Hoeting. *Computational Statistics*, volume 703, chapter 7. John Wiley & Sons, 2nd edition, 2012. URL https://doi.org/10.1111/j.1467-985X.2006.00430_5.x. [p108, 120]
- A. Groll. *glmmLasso: Variable Selection for Generalized Linear Mixed Models by L1-Penalized Estimation*, 2017. URL <https://CRAN.R-project.org/package=glmmLasso>. R package version 1.5.1. [p105]
- M. J. Gurka, L. J. Edwards, and K. E. Muller. Avoiding bias in mixed model inference for fixed effects. *Statistics in Medicine*, 30(22):2696–2707, 2011. URL <https://doi.org/10.1002/sim.4293>. [p105]
- J. D. Hadfield. Mcmc methods for multi-response generalized linear mixed models: The **MCMCglmm** r package. *Journal of Statistical Software*, 33(2):1–22, 2010. URL <https://www.jstatsoft.org/v33/i02/>. [p105]
- M. D. Hoffman and A. Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014. URL <https://dl.acm.org/doi/abs/10.5555/2627435.2638586>. [p107]

- J. G. Ibrahim, H. Zhu, R. I. Garcia, and R. Guo. Fixed and random effects selection in mixed effects models. *Biometrics*, 67(2):495–503, 2011. URL <https://doi.org/10.1111/j.1541-0420.2010.01463.x>. [p105, 107, 111, 112]
- M. J. Kane, J. Emerson, and S. Weston. Scalable strategies for computing with massive data. *Journal of Statistical Software*, 55(14):1–19, 2013. URL <http://www.jstatsoft.org/v55/i14/>. [p110, 117]
- K. Kleinman, R. Lazarus, and R. Platt. A generalized linear mixed models approach for detecting incident clusters of disease in small areas, with an application to biological terrorism. *American Journal of Epidemiology*, 159(3):217–224, 2004. URL <https://doi.org/10.1093/aje/kwh029>. [p105]
- I. H. Langford. Using a generalized linear mixed model to analyze dichotomous choice contingent valuation data. *Land Economics*, pages 507–514, 1994. URL <https://doi.org/10.2307/3146644>. [p105]
- J. Lorah and A. Womack. Value of sample size for computation of the bayesian information criterion (bic) in multilevel modeling. *Behavior Research Methods*, 51(1):440–450, 2019. URL <https://doi.org/10.3758/s13428-018-1188-3>. [p113]
- S. Ma, S. Ogino, P. Parsana, R. Nishihara, Z. Qian, J. Shen, K. Mima, Y. Masugi, Y. Cao, J. A. Nowak, et al. Continuity of transcriptomes among colorectal cancer subtypes based on meta-analysis. *Genome Biology*, 19(1):142, 2018. URL <https://doi.org/10.1186/s13059-018-1511-4>. [p114]
- I. Misztal. Reliable computing in estimation of variance components. *Journal of Animal Breeding and Genetics*, 125(6):363–370, 2008. URL <https://doi.org/10.1111/j.1439-0388.2008.00774.x>. [p110]
- R. A. Moffitt, R. Marayati, E. L. Flate, K. E. Volmar, S. G. H. Loeza, K. A. Hoadley, N. U. Rashid, L. A. Williams, S. C. Eaton, A. H. Chung, et al. Virtual microdissection identifies distinct tumor- and stroma-specific subtypes of pancreatic ductal adenocarcinoma. *Nature Genetics*, 47(10):1168, 2015. URL <https://doi.org/10.1038/ng.3398>. [p114]
- A. Pajor. Estimating the marginal likelihood using the arithmetic mean identity. *Bayesian Analysis*, 12(1):261–287, 2017. URL <https://doi.org/10.1214/16-BA1001>. [p113, 124]
- P. Patil and G. Parmigiani. Training replicable predictors in multiple studies. *Proceedings of the National Academy of Sciences*, 115(11):2578–2583, 2018. URL <https://doi.org/10.1073/pnas.1708283115>. [p114]
- J. Pinheiro, D. Bates, S. DebRoy, D. Sarkar, and R Core Team. **nlme: Linear and Nonlinear Mixed Effects Models**, 2021. URL <https://CRAN.R-project.org/package=nlme>. R package version 3.1-152. [p113]
- N. U. Rashid, Q. Li, J. J. Yeh, and J. G. Ibrahim. Modeling between-study heterogeneity for improved replicability in gene signature selection and clinical prediction. *Journal of the American Statistical Association*, 115(531):1125–1138, 2020. URL <https://doi.org/10.1080/01621459.2019.1671197>. [p106, 107, 108, 109, 112, 114, 122]
- M. Riester, W. Wei, L. Waldron, A. C. Culhane, L. Trippa, E. Oliva, S.-h. Kim, F. Michor, C. Huttenhower, G. Parmigiani, et al. Risk prediction for late-stage ovarian cancer by meta-analysis of 1525 patient samples. *JNCI: Journal of the National Cancer Institute*, 106(5), 2014. URL <https://doi.org/10.1093/jnci/dju048>. [p114]
- G. O. Roberts and J. S. Rosenthal. Examples of adaptive mcmc. *Journal of Computational and Graphical Statistics*, 18(2):349–367, 2009. URL <https://doi.org/10.1002/wics.1307>. [p108, 120]
- SAS Institute Inc. *SAS/STAT Software, Version 9.2*. Cary, NC, 2008. URL <http://www.sas.com/>. [p113]
- J. Schelldorfer, L. Meier, and P. Bühlmann. **Glmlasso**: An algorithm for high-dimensional generalized linear mixed models using l1-penalization. *Journal of Computational and Graphical Statistics*, 23(2):460–477, 2014. URL <https://doi.org/10.1080/10618600.2013.773239>. [p105]
- A. W. Schmidt-Catran and M. Fairbrother. The random effects in multilevel models: Getting them wrong and getting them right. *European Sociological Review*, 32(1):23–38, 2016. URL <https://doi.org/10.1093/esr/jcv090>. [p105]
- Stan Development Team. **RStan**: The r interface to stan, 2020. URL <http://mc-stan.org/>. R package version 2.21.2. [p120]
- M. Szyszkowicz. Use of generalized linear mixed models to examine the association between air pollution and health outcomes. *International Journal of Occupational Medicine and Environmental Health*, 19(4):224–227, 2006. URL <https://doi.org/10.2478/v10001-006-0032-7>. [p105]

- J. A. Thompson, K. L. Fielding, C. Davey, A. M. Aiken, J. R. Hargreaves, and R. J. Hayes. Bias and inference from misspecified mixed-effect models in stepped wedge trial analysis. *Statistics in Medicine*, 36(23):3670–3682, 2017. URL <https://doi.org/10.1002/sim.7348>. [p105]
- J. N. Weinstein, E. A. Collisson, G. B. Mills, K. R. Shaw, B. A. Ozenberger, K. Ellrott, I. Shmulevich, C. Sander, and J. M. Stuart. The cancer genome atlas pan-cancer analysis project. *Nature genetics*, 45(10):1113–1120, 2013. URL <https://doi.org/10.1038/ng.2764>. [p114]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. URL <https://ggplot2.tidyverse.org>. [p120]

Hillary M. Heiling
University of North Carolina Chapel Hill

hmheiling@gmail.com

Naim U. Rashid
University of North Carolina Chapel Hill

nur2@email.unc.edu

Quefeng Li
University of North Carolina Chapel Hill

quefeng@email.unc.edu

Joseph G. Ibrahim
University of North Carolina Chapel Hill

ibrahim@bios.unc.edu

Unified ROC Curve Estimator for Diagnosis and Prognosis Studies: The sMSROC Package

by Susana Díaz-Coto, Pablo Martínez-Camblor, and Norberto Corral-Blanco

Abstract The binary classification problem is a hot topic in Statistics. Its close relationship with the diagnosis and the prognosis of diseases makes it crucial in biomedical research. In this context, it is important to identify biomarkers that may help to classify individuals into different classes, for example, diseased vs. not diseased. The Receiver Operating-Characteristic (ROC) curve is a graphical tool commonly used to assess the accuracy of such classification. Given the diverse nature of diagnosis and prognosis problems, the ROC curve estimation has been tackled from separate perspectives in each setting. The Two-stages Mixed-Subjects (sMS) ROC curve estimator fits both scenarios. Besides, it can handle data with missing or incomplete outcome values. This paper introduces the R package sMSROC which implements the sMS ROC estimator, and includes tools that may support researchers in their decision making. Its practical application is illustrated on three real-world datasets.

1 Introduction

The binary classification problem is crucial in biomedical environments. Researchers and physicians face the task of classifying patients (as e.g. diseased vs. disease-free, at risk vs. risk-free, etc.) daily. Frequently, biological measures (biomarkers) are used as objective medical signs that may indicate, for example, the presence/progress of an event of interest or the response to specific treatments. Diagnostic biomarkers are normally employed to detect the presence of a disease. For example, the *circulating cardiac troponin I* aids in noninvasive detection of myocardial injury in cardiovascular diseases (Ni and Wehrens, 2018). The *blood glucose* and the *hemoglobin A1c* are recognized diagnostic biomarkers of type 2 diabetes mellitus (Long et al., 2020) and the *sweat chloride* is often used to confirm the cystic fibrosis (Farrell et al., 2008).

The Receiver Operating-Characteristic (ROC) curve is a popular graphical tool for assessing the ability of biomarkers to discriminate between positive and negative individuals (with and without the event of interest, respectively) (Zhou et al., 2002; Pepe, 2003). For each possible cut-off point, it plots the pairs formed by the complement of *specificity* against the *sensitivity*. The sensitivity, Se , and the specificity, Sp , are the proportions of positive and negative individuals, respectively, who have been correctly classified. The closer the ROC curve to the upper left corner, the more accurate the biomarker.

Mathematically, for each $u \in [0, 1]$, the ROC curve has the expression:

$$\mathcal{R}(u) = Se \left(Sp^{-1}(1 - u) \right), \quad (1)$$

where $Sp^{-1}(\cdot) = \inf\{x : Sp(x) \geq \cdot\}$. Conventionally, it is assumed that individuals with larger biomarker values are more likely positive. Therefore, the classification sets defining an individual as positive are those in the way $[c, \infty)$, with $c = Sp^{-1}(1 - u) \in \mathbb{R}$. Besides, the related area under the ROC curve, (AUC) ($= \int_0^1 \mathcal{R}(u) du$), is commonly used as a summary index of the global classification accuracy (Hanley and McNeil, 1982).

The estimation of ROC curves has been addressed from different perspectives (see, for instance, Gonçalves et al. (2014) and references therein). Estimation procedures generally assume that data come from case-control designs, where the actual status of all individuals as positive or negative is known in advance. Further, they do not handle the issue of potential missing values in the outcome. Notice that such issues arise, for instance, in cohort designs where the outcome is defined ad hoc through a subsidiary variable. The missing values for that variable lead to missing values in the outcome.

Most of the statistical software (SPSS, SAS, STATA, etc.) offer routines for computing different ROC curve estimators and related elements. There are also packages in R (www.r-project.org) dealing with the ROC curve topic. We highlight the pROC package (Robin et al., 2011), which provides functions for visualizing, smoothing and comparing ROC curves; the package nsROC (Pérez-Fernández, 2017), where some of the non-standard tools for the ROC curve analyses described in Pérez-Fernández et al. (2018) are implemented; the package ROC (Sing et al., 2005), which supplies user-friendly tools for creating graphics for visualizing classification performance (the ROC curve is a particular case); the plotROC package (Sachs, 2017), offering interactive ROC curve plots suitable for use on the web, and

finally the **ROCNReg** package (Rodríguez-Álvarez and Inácio, 2021), which, among other extensive functionality, implements Bayesian methods for the estimation of ROC curves.

Prognostic biomarkers are used to identify individuals who are likely to experience a future clinical event (death, the onset/recurrence of a disease or the development of a new medical condition). For example, the breast cancer genes 1 and 2 mutations are often employed to assess the likelihood of a second breast cancer (Basu et al., 2015). Similarly, the C-reactive protein level is a prognostic biomarker used to identify individuals with unstable angina at risk of developing other adverse events (Ferreirós et al., 1999), and the Gleason score helps to assess the likelihood of prostate cancer progression (Epstein et al., 2016). These biomarkers are measured at baseline and individuals are then followed over time to observe whether or not the event of interest occurs. A *time-to-event* variable is involved in this process. Different definitions of positive and negative outcomes have been proposed, which has given rise to extensions of the sensitivity and specificity measures and, of course, to the corresponding time-dependent ROC curves (Etzioni et al., 1999).

The Cumulative/Dynamic (C/D) ROC curve (Heagerty et al., 2000) is perhaps the most natural extension of the ROC curve for time-dependent outcomes. Once set to a specific point of time t , the time-to-event variable is reduced to a dichotomous variable at that time. Then, the sensitivity and the specificity can be extended to the so-called *cumulative sensitivity* and *dynamic specificity* whose expressions are

$$\begin{aligned} Se_t^C(c) &= \mathcal{P}(X > c \mid T \leq t), \\ Sp_t^D(c) &= \mathcal{P}(X \leq c \mid T > t), \end{aligned}$$

where $c \in \mathbb{R}$ is the cut-off point, and X and T are the random variables modeling the biomarker and the time-to-event variables, respectively. The C/D ROC curve is the plot of the pairs formed by the complement to the dynamic specificity and the cumulative sensitivity, for all possible cut-off points. Alternatively, it is given by

$$\mathcal{R}_t^{C/D}(u) = Se_t^C([Sp_t^D]^{-1}(1-u)), \quad u \in [0, 1].$$

The area under the C/D ROC curve is used as well as a summarize index of the prognostic accuracy of a biomarker and has the expression

$$AUC_t^{C/D} = \int_0^1 \mathcal{R}_t^{C/D}(u) du.$$

The main challenge when estimating the C/D ROC curve is the potential lack of complete information for some individuals (caused by censoring). It arises because of loss of follow-up, either due to dropouts or because the study ended before the event of interest had the chance to occur in the individual (right censoring). It may also come up when individuals are not constantly monitored and the only available information is that the event of interest occurred between two observed timepoints (interval censorship). The simplest C/D ROC curve estimator removes from the sample the censored observations and approximates the cumulative sensitivity and the dynamic specificity through their empirical estimators (naive method). Other procedures integrate, in some way, the information from the censored observations. Kamarudin et al. (2017) provides an illustrative revision of the available R packages implementing some of these methods, all of them addressing the right censorship problem. For example, the **survivalROC** package (Heagerty and Saha-Chaudhuri, 2022) computes the C/D ROC curve through the two procedures proposed by Heagerty et al. (2000). The **survAUC** package (Potapov et al., 2023) collects several routines for computing the $AUC_t^{C/D}$, at different times, estimated by the Inverse Probability Censoring Weighting (IPCW) method (Uno et al., 2007; Hung and Chiang, 2010) and by the Chambles and Diao (2006) approach. The **timeROC** package (Blanche et al., 2013b) implements the Conditional Inverse Probability Censoring Weighting (CIPCW) procedure (Blanche et al., 2013a). In addition to confidence intervals for the $AUC_t^{C/D}$, the package performs tests for comparing two areas under the curve corresponding to different prognostic biomarkers. We add to this list the already mentioned **nsROC** package, which allows to compute the estimation procedures proposed in Martínez-Camblor et al. (2016) and Li et al. (2018). The latter method is also available in the **tdROC** package (Li and Wu, 2016). Finally, the **smoothROCtime** (Díaz-Coto et al., 2020b) and **cenROC** (Beyene and El Ghouch, 2023) packages implement the smooth C/D ROC curve estimators suggested by Martínez-Camblor and Pardo-Fernández (2018) and Beyene and El Grouch (2020), respectively. We only found two packages implementing the C/D ROC curve estimation under interval censorship. The **intcensROC** package (Lin et al., 2021) computes the estimator for the C/D ROC curve and $AUC_t^{C/D}$ proposed in Wu et al. (2020), while the **cenROC** implements the method proposed in Beyene and El Ghouch (2022). Not a package but an R function is provided in Díaz-Coto et al. (2020a), computing the C/D ROC curve estimator proposed in that paper.

We present here the package **sMSROC**, which implements the so-called Two-stage Mixed-Subjects

(sMS) estimator. The sMS estimator uses, in a first stage, a probabilistic model for linking the biomarker with the outcome, and then, in a second stage and for each potential threshold, it computes both the sensitivity and the specificity values, which can be used to draw the ROC curve. This approach can be used to answer both diagnostic and prognostic questions, and, by imposing additional constraints on the missing-value mechanism, is able to handle missing data in the outcome variable. The probabilistic model is used for allocating subjects into the positive and the negative groups with certain probabilities. In this sense, subjects are simultaneously classified as positive and negative, that is, they are mixed. Interested readers are referred to [Díaz-Coto et al. \(2021\)](#) for a more in depth explanation of the theoretical properties of the sMS estimator. The presented **sMSROC** package offers a set of exploratory tools which help to choose the most suitable probabilistic model (logistic regression, proportional hazard Cox regression, etc.). These (more standard) estimation proposals are already implemented in the package, which also allows to manually enter any other estimates of the probabilities of being positive or negative, which may be estimated by other methods. Among other functionalities, the **sMSROC** package computes the AUC with confidence intervals, and provides plots for the ROC curve estimates, the predictive models, and the evolution of the AUCs across the follow-up time, providing different options for customizing the final graphics.

The remainder of the paper is organized as follows. In the Section 2, we present the sMS estimator and review its main properties. We provide a general insight of the structure of the **sMSROC** package in the Section 3. In Section 4 the main functions are described in detail. Two real-world datasets are used to illustrate the use of these functions in the diagnosis and prognosis scenarios. In the Section 5, we present a third real-world example, and show how the package can be used to assess the prognostic ability of a biomarker when data are interval censored. We want to present a disclaimer that the examples that the analyses provided here are used to demonstrate the use of the package only and they should not be used to inform any clinical decisions. Finally, we end in the Section 6 with a discussion of the potential uses of the **sMSROC** package.

2 The two-stage mixed subjects receiver operating-characteristic curve estimator

We first introduce the notation that will be used along this paper. Let X be a continuous random variable, with Cumulative Distribution Function (CDF) $H(\cdot)$, which models the behavior of the biomarker values. Let D be the binary random variable representing the event of interest, taking, without lost of generality, values 0 and 1, identifying negative and positive individuals, respectively. For prognosis scenarios, let T be the involved time-to-event random variable and let our aim be to predict the occurrence of the event of interest before a fixed point of time t . The binary variable depicting this event is given by D_t , which again takes the values 0, when $T > t$ (negative individuals) and 1, when $T \leq t$ (positive individuals). For sake of simplicity, we will remove the subscript t and will use the same notation in both scenarios.

The expression of the sensitivity can be written as

$$\begin{aligned} Se(c) &= \mathcal{P}(X > c | D) \\ &= \frac{\mathcal{P}(X > c, D)}{\mathcal{P}(D)} \\ &= \frac{\mathbb{E}_X[\mathcal{P}(X > c, D | X = x)]}{\mathbb{E}_X[\mathcal{P}(D | X = x)]} \\ &= \frac{\mathbb{E}_X[\mathbb{I}_{(c,\infty)}(x) \cdot \mathcal{P}(D | x)]}{\mathbb{E}_X[\mathcal{P}(D | x)]} \\ &= \frac{\int [\mathbb{I}_{(c,\infty)}(x) \cdot \mathcal{P}(D | x)] dH(x)}{\int (\mathcal{P}(D | x)) dH(x)}, \end{aligned} \tag{2}$$

where $c \in \mathbb{R}$; $\mathbb{I}_A(x)$ depicts the indicator function; D stands for the positive outcome ($D = 1$), and $\mathcal{P}(D | x) = \mathcal{P}(D | X = x)$.

Similarly, the specificity has the expression:

$$\begin{aligned}
 Sp(c) &= \mathcal{P}(X \leq c \mid \bar{D}) \\
 &= \frac{\mathcal{P}(X \leq c, \bar{D})}{\mathcal{P}(\bar{D})} \\
 &= \frac{\mathbb{E}_X[\mathcal{P}(X \leq c, \bar{D} \mid X = x)]}{\mathbb{E}_X[\mathcal{P}(\bar{D} \mid X = x)]} \\
 &= \frac{\mathbb{E}_X[\mathbb{I}_{(-\infty, c]}(x) \cdot (1 - \mathcal{P}(D \mid x))]}{\mathbb{E}_X[(1 - \mathcal{P}(D \mid x))] \\
 &= \frac{\int [\mathbb{I}_{(-\infty, c]}(x) \cdot (1 - \mathcal{P}(D \mid x))] dH(x)}{\int (1 - \mathcal{P}(D \mid x)) dH(x)}, \tag{3}
 \end{aligned}$$

where \bar{D} depicts the negative outcome ($D = 0$).

Let $(\mathcal{X}_N, \mathcal{D}_N) = \{(x_1, d_1), \dots, (x_N, d_N)\}$ be an independent random sample where, for the i -th individual ($1 \leq i \leq N$), x_i is the biomarker value and d_i reports some information regarding the outcome of interest. Such information, in the diagnosis scenario, may provide the actual status of the individual ($d_i = \delta_i$, where $\delta_i = 0$ for a negative individual, or $\delta_i = 1$, for a positive one) or may be missing. When dealing with a time-dependent outcome, this information can also include the event/censoring time, according to the censorship pattern. That is, in the case of right censorship, $d_i = \{\delta_i, z_i\}$, being $z_i = \min\{t_i, c_i\}$, where t_i and c_i stand for the event and censoring times, respectively. Under interval censorship, $d_i = \{l_i, r_i\}$, where l_i and r_i are the lower and upper bounds of the observed interval containing the event time ($l_i \leq t_i \leq r_i$). The sensitivity and specificity given in (2) and (3) can be estimated through:

$$\hat{Se}(c) = \frac{\sum_{i=1}^N I_{(c, \infty)}(x_i) \cdot \hat{\mathcal{P}}_N(D \mid x_i)}{\sum_{i=1}^N \hat{\mathcal{P}}_N(D \mid x_i)}, \tag{4}$$

$$\hat{Sp}(c) = \frac{\sum_{i=1}^N I_{(-\infty, c]}(x_i) \cdot (1 - \hat{\mathcal{P}}_N(D \mid x_i))}{\sum_{i=1}^N (1 - \hat{\mathcal{P}}_N(D \mid x_i))}, \quad c \in \mathbb{R}, \tag{5}$$

where $\hat{\mathcal{P}}_N(D \mid x)$ is chosen to be an adequate estimator of $\mathcal{P}(D \mid x)$.

Plugging-in the expressions (4) and (5) in the definition of the ROC curve given in (1), we obtain the **Two-stage Mixed Subject (sMS) ROC curve estimator**, to which we will refer as sMS estimator:

$$\hat{\mathcal{R}}(u) = \hat{Se}\left([1 - \hat{Sp}]^{-1}(u)\right), \quad u \in [0, 1],$$

where $\hat{Sp}^{-1}(\cdot) = \inf\{x : \hat{Sp}(x) \geq \cdot\}$.

We briefly review some features of the sMS estimator already introduced in [Díaz-Coto et al. \(2021\)](#):

- The relationship between the biomarker and the outcome is modeled by $\mathcal{P}(D \mid x)$ (the predictive model). In the first stage, the sMS estimator approximates the predictive model through the most suitable probabilistic model (e.g. proportional hazards, logistic regression). In the second stage, the rest of the unknown parameters is estimated by the corresponding empirical estimators. The first stage is specially important because the performance of the sMS estimator is highly dependent on the fit of the predictive model to the actual relationship between the biomarker and the outcome.
- The sMS estimator does not need to consider the individuals as fully positive or fully negative. Each individual can be modeled as mixed: partially positive and partially negative (hence the name “mixed subjects”). The weight allocated to each possibility is determined by the predictive model considering the biomarker value in the specific individual.
- The sMS estimator can handle missing values in the outcome as well as censored observations (latter frequently associated with prognosis studies). The individuals with missing outcome are supposed to be missing at random (MAR) however; that is, their characteristics in the sample should be similar to those with complete information. Under this assumption, their potential outcome is determined by the predictive model for the particular biomarker value.
- The sMS estimator generalizes some of the ROC and C/D ROC curve existing estimators. In the simplest diagnosis scenario, where the real status of all individuals is known, we can estimate the predictive model $\mathcal{P}(D \mid x)$ through the average of the status of those having a biomarker value of x . The resulting estimator would be the well-known empirical ROC curve estimator

([Hsieh and Turnbull, 1996](#)). In the prognosis scenario, considering the adequate estimators for the predictive model, it is clear the connection with the C/D ROC curve estimators under right censorship proposed in [Martínez-Camblor et al. \(2016\)](#) and [Li et al. \(2018\)](#), and with the estimator proposed in [Díaz-Coto et al. \(2020a\)](#) under interval censorship. For particular parametrizations, the sMS estimator can as well be the C/D ROC curve estimator proposed by [Chambles and Diao \(2006\)](#) and by [Song and Zhou \(2008\)](#).

- Under certain conditions both the sMS estimator and its corresponding estimator for the AUC are asymptotically normal distributed. We provide two approximations for the variance of the AUC estimator: the Theoretical Variance Estimation (TVE), based on a theoretical expression, and the Empirical Variance Estimator (EVE), which avoids dealing with the expression of the variance of the predictive model. Explicit expressions for both the TVE and the EVE approximations are provided in the Appendix of this manuscript. Reported confidence intervals are $\mathcal{A} \pm \lambda_\alpha \cdot \hat{\sigma}$, with \mathcal{A} and $\hat{\sigma}^2$ the AUC and variance approximations, respectively, and λ_α the adequate quantile based on the normal distribution.

It is worth clarifying that, although related, the sMS estimator is not a single imputation procedure. In this sense, we are not considering here the presence of missing data in the biomarker. The goal of the sMS estimator is not to impute the unknown values of the outcome, but to use the estimated probabilities to approximate the ROC curve. These probabilities could be used even for those subjects for whom we already know the actual status. For instance, in a standard study in which we collect the status of each single participant, the empirical model would be an extreme situation in which the probability of being positive is determined by the actual observed status of the subject (probability 1 or 0). As we have already noted, in this case, the resulting sMS estimator would be the empirical ROC curve estimator. However, we could model these probabilities by the standard binary logistic regression to obtain a smoothed ROC curve estimate. For sure, the quality of this estimation would depend on the goodness of fit of the regression model. For time-to-event outcomes, we can use the actually observed follow-up times for computing the probabilities. Notice that, if the target of interest is to predict events prior to the point $t = 5$, participants who still alive at point 4.99 are more likely to be alive at 5 than those censored at 0.01. One of the main advantage of the sMS estimator is its flexibility, which allows to adapt the procedure to several types of data, including different censoring models, and provides a variety of techniques under the same umbrella.

3 An overview of the package

The main goal of the **sMSROC** package (available at <https://CRAN.R-project.org/package=sMSROC>) is to compute the sMS estimator and related elements, which support the assessment of the diagnostic/prognostic ability of continuous biomarkers. Since R programming is mostly based on objects ([López-Ratón et al., 2014](#)), the **sMSROC** package consists in a set of functions performing specific tasks.

Table 1 provides a summary of these functions, grouped by their common features. The functions have been classified as primary and secondary. Among the former, we consider those directly run by the end-user to perform the exploratory data analysis, compute the sMS estimator and other metrics (such as the AUC and its confidence interval), and to summarize the computed results. We refer to the rest of the functions as secondary, as these are mainly called by other functions and not meant to be used by the end-user, primarily. We will describe the primary functions in more detail in the next sections.

The **sMSROC** package uses some functionalities already implemented in other packages. In a non-exhaustive list we highlight: the functions `Surv` and `ic_sp`, from the **survival** ([Terry M. Therneau and Patricia M. Grambsch, 2000](#)) and **icenReg** ([Anderson-Bergman, 2017](#)) packages, which provide estimates of the survival function under right and interval censorship, respectively; the `rcs` function, from the **rsm** package ([Harrell Jr, 2023](#)), that computes the cubic splines approximation; the `%dopar%` function from the package **foreach** ([Microsoft and Weston, 2022](#)), used to perform parallel computing; the `flextable` function, from the package with the same name ([Gohel and Skintzos, 2023](#)), which provides formatted outputs for the tables and the `ggplot` and `plotROC` functions, from the packages **ggplot2** ([Wickham, 2016](#)) and **plotROC** ([Sachs, 2017](#)), used to obtain well-formatted and interactive final plots, respectively.

Table 1: Functions included in the sMSROC package grouped by the similarity of the tasks performed.

Main Functionality	Functions	General Description
Exploratory data analysis	<code>explore_table</code> <code>explore_plot</code>	Perform a descriptive analysis of the biomarker values on the different samples
sMS ROC curve estimator	<code>sMSROC</code>	Core function that is actually a wrapper of those functions computing each element related to the sMS estimator
Check-ups	<code>check_type_outcome</code> <code>check_conf_int</code> <code>check_grid</code> <code>check_marker_binout</code> <code>check_marker_timerc</code> <code>check_marker_timeic</code> <code>check_tim</code> <code>check_meth</code> <code>check_probs</code> <code>check_ncpus</code> <code>check_nboost</code> <code>check_ci_cl</code>	Verify the integrity and the consistency of the parameters of the functions entered by the end-users
sMS ROC	<code>sMS_binout</code> <code>sMS_timerc</code> <code>sMS_timeic</code>	Compute the sMS ROC curve estimates in each particular scenario
Predictive models	<code>pred_model_binout</code> <code>pred_model_timerc</code> <code>pred_model_timeic</code> <code>pred_model_emp</code>	Compute the estimates for the predictive models according to specific probabilistic models (first stage of the sMS estimator)
ROC curve	<code>compute_ROC</code>	Computes the ROC curve and the AUC through the estimators of sensitivity and specificity (second stage of the sMS estimator)
Confidence intervals for the AUC	<code>auc_ci_boot</code> <code>auc_ci_emp</code> <code>auc_ci_var</code>	Compute confidence intervals for the AUC according to the selected method
Plots	<code>sMSROC_plot</code> <code>evol_AUC</code> <code>prob_pred</code>	Plot the sMS ROC estimate, the evolution of the AUCs, and the predicted probabilities
Print	<code>conf_int_print</code>	Prints certain components of the sMS estimate

4 Primary functions

4.1 Exploratory data analysis

The exploratory analysis of the data is carried out by the `explore_table` and `explore_plot` functions. They allow to have an insight of the distribution of the biomarker on positive and negative individuals and on those whose belonging group is unknown. This may help to the selection of the most suitable predictive model for each particular problem. Both functions share the input parameters collecting the sample information that was formally introduced previously: the biomarker values and the information regarding the outcome of interest, which varies depending on the scenario. They also have specific parameters according to the performed task. The functions provide numerical and graphical outputs.

The function `explore_table` computes the most common descriptive statistics for the pooled sample and the samples of the different types of individuals. The input parameters are:

- **marker** a vector of the biomarker values.
- **status** a numeric vector with the status of the individuals. The highest value represents the event of interest. The lowest value represents the absence of the event of interest. All other values are ignored.
- **observed.time** a vector with the observed times for each subject (prognosis scenario under right censorship). Notice that these values may be the event times or the censoring times.
- **left** a vector with the lower bounds of the observed intervals. It is mandatory, when computing the sMS estimator for assessing prognostic biomarkers under interval censorship. It will be ignored in other situations.
- **right** a vector with the upper bounds of the observed intervals. Like the previous parameter, it is mandatory in the prognosis scenario under interval censorship and ignored in other situations. Non available, NA and ∞ ('Inf') are admissible values to indicate that the event of interest did not occur prior to the last observation time.
- **time** point of time at which the time-dependent sMS estimator will be computed. The default value is 1. This parameter is mandatory in the prognosis scenario.
- **d** number of decimal positions to which all results will be rounded. The default value is 2.
- ... rest of the parameters supplied to the `flextable` function. These can be used to customize the output table as desired.

In diagnosis scenarios, it is clear when individuals are either positive or negative. When dealing with time-dependent outcomes, this status depends on a fixed point time t at which they are evaluated. Particularly, in the interval censorship case, if the last revision time in which the event had already happened took place before the set time t , the individual is positive at t . If there exists a revision time beyond t and the event has not been observed yet, the individual is negative at t . When the event occurs between two consecutive revision times containing the set time t , nothing is known about the status of the individuals at t , because it is not actually observed when the event happened. We refer to these individuals as *undefined* or *mixed*.

The consistency of the incoming parameters is verified by secondary functions. Next, the type of scenario handled (diagnosis/prognosis, under right or interval censorship) is determined. The functions `explore_table`, `explore_plot` and `sMSROC` share both steps.

The output of the function is a list with two components:

- **summary** a matrix whose columns are the name of the groups, their size and the descriptive statistics: minimum, maximum, mean, standard deviation and first, second and third quartiles. The rows show the results for positive, negative, missing/censored/undefined individuals and the pooled sample.
- **table** an object of class `flextable` that collects the descriptive information from the previous matrix in a table, which can be customized according to the preferences of the users by the entered parameters.

The function `explore_plot` plots the kernel density estimations for the biomarker within both the positive and the negative individuals. The input parameters are those related to the sample information described for the `explore_table` function.

The output is a list with three components:

- **plot** an object of class `ggplot` with the density functions of the biomarker on the positive and the negative individuals. The user can add layers to customize the final plot according to the rules of the `ggplot2` package.

- **neg** a vector with the marker values of the negative individuals.
- **pos** a vector with the marker values of the positive individuals.

Example 1 [Exploratory data analysis]: the diabetes dataset

We first consider the study of the ability of *stabilized glucose* to diagnose diabetes (defined through a subsidiary measure: a value of glycosylated hemoglobin greater than 7.0) in an African-American population of central Virginia (USA). We consider the dataset freely available at <https://hbiostat.org/data/>. The subset data **diabet**, used here, is delivered as part of the **sMSROC** package. More information about this study can be found in [Willems et al. \(1997\)](#).

A total of 60 individuals out of the 403 included were diabetic (positive), and 330 were classified as non-diabetic (negative). Besides, there were 13 individuals without glycosylated hemoglobin value, so we cannot determine their actual status (undefined). The next piece of code provides the distribution of the *stabilized glucose* on these groups, shown in HTML format in Table 2:

```
> library(sMSROC)
> data(diabet)
> expl <- explore_table(marker=diabet$stab.glu, status=diabet$diab)
> expl$table
```

Table 2: Object of class **flextable**, one of the elements of the output list of the `explore_table` function. The usual descriptive statistics of the *stabilized glucose* biomarker on the total sample and the samples of Positive, Negative and Undefined observations are shown.

Sample	Size	Minimun	Maximun	Mean	Sd	Variance	Q1	Median	Q3
Positive	60	60	385	194.17	77.44	5,996.68	120	186.0	241.25
Negative	330	48	371	91.55	26.87	721.77	79	86.5	97.00
Miss/Cens/Und	13	68	105	86.69	10.26	105.23	80	84.0	88.00
Total	403	48	385	106.67	53.08	2,817.13	81	89.0	106.00

Left panel of Figure 1 shows the kernel density estimations of the *stabilized glucose* on positive and negative individuals generated through the code:

```
> library(ggplot2)
> density <- explore_plot(marker=diabet$stab.glu, status=diabet$diab)
> output <- density$plot + xlab("Stabilized Glucose") +
  scale_x_continuous(breaks = seq(0, 400, 50),
                     labels = seq(0, 400, 50),
                     limits = c(0, 400))
> output
```

4.2 Main function

sMSROC is the main function in the package. It computes the sMS ROC curve estimator and its associated AUC with confidence intervals, estimated using the bootstrap percentile (BP) method, and according the approximations EVE and TVE given in Section 2. The function has the following input parameters:

- **meth** method for approximating the predictive model $\mathcal{P}(D|x)$. There are several options available:
 - E implements the naive method where missing, censored and undefined individuals are removed from the data.
 - L in the diagnosis scenario this option models the probability of being positive as:

$$\mathcal{P}(D|x) = \frac{1}{1 + \exp\{-(\beta_0 + \beta_1 \cdot x)\}}, \quad \beta_0, \beta_1 \in \mathbb{R}.$$

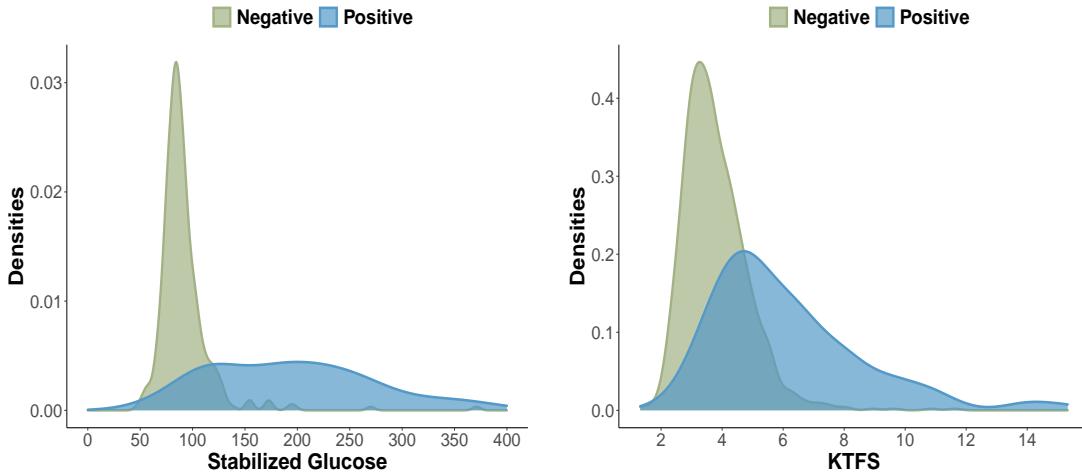


Figure 1: Left: Kernel density estimations of the *stabilized glucose* biomarker on negative and positive individuals. Right: kernel density estimations of the *KTFS* biomarker on negative and positive individuals after 5 years from transplantation.

In the case of prognosis scenarios under right censorship, a proportional hazards Cox regression model is used to approximate $\mathcal{P}(T \leq t|x)$ (and therefore, $\mathcal{P}(D|x)$).

$$\mathcal{P}(T \leq t|x) = 1 - \exp\{-\Gamma_0(t) \cdot \exp\{\beta \cdot x\}\},$$

where $\Gamma_0(\cdot)$ stands for the so-called cumulative baseline hazard function (Cox, 1972). Under interval censorship, $\mathcal{P}(T \leq t|x)$ is approximated through the model proposed in Díaz-Coto et al. (2020a):

$$\mathcal{P}(T \leq t|x) = \frac{S(U|x) - S(t|x)}{S(U|x) - S(V|x)}, \quad (6)$$

where $U = \min\{t, L\}$ and $V = \max\{t, R\}$, being L and R the random variables depicting the edges of the observable interval $(L, R]$. The expression $S(t|x)$ stands for $S(t|X = x)$, that is, the survival function at t , given the marker value. It is estimated by proportional hazards model under interval censorship (Finkelstein, 1986), applying linear interpolation inside the Turnbull intervals (Turnbull, 1976).

In this case, the parameter `all` indicates whether this approximation applied to all individuals or just to the mixed/censored/undefined ones.

- `S` approximates, in diagnosis scenarios, the logit transformation of the predictive model via a cubic spline function. That is:

$$\mathcal{P}(D|x) = \frac{1}{1 + \exp\{-s(x)\}},$$

where $s(\cdot)$ depicts some smooth function, estimated from restricted cubic splines (Harrel, 2015). In the prognosis scenario under right censorship, we consider a proportional hazards model with a more flexible option to approximate the predictive model:

$$\mathcal{P}(T \leq t|x) = 1 - \exp\{-\Gamma_0(t) \cdot \exp\{s(x)\}\},$$

where a penalized splines procedure (Hurvich et al., 1998) estimates the smooth function $s(\cdot)$. Under interval censorship the predictive model is obtained by

$$\mathcal{P}(T \leq t|x) = 1 - S(t|x),$$

where $S(t|x)$ is estimated as before through a proportional hazards model under interval censorship.

- `probs` a vector of manually entered predicted probabilities. This argument is useful if the user wants to estimate the predicted probabilities via a different model than the ones currently offered by the package. In this case, the argument `meth` will be ignored.

- **sd.probs** a vector with the standard deviation of the probabilities entered as **probs**. It is an optional parameter.
- **grid** grid size for computing the ROC curve estimate. The default value is 1000. It is also used to compute the AUC.
- **conf.int** argument with two possible values indicating whether confidence intervals for the AUC will be computed (T) or not (F).
- **ci.cl** confidence level at which the confidence intervals for the AUC will be calculated. The default value is 95%.
- **ci.meth** method for computing the AUC confidence intervals. There are three options available:
 - V method that uses the TVE variance approximation.
 - E method which uses the EVE approximation.
 - B confidence intervals based on BP.
- **ci.nboost** number of bootstrap samples to be run when the option B is chosen as **ci.meth** parameter. The default value is 500.
- **parallel** argument with two possible values which indicates whether parallel computing will be carried out (T) or not (F). There are two processes that currently support parallel computing: the B and V options to obtain confidence intervals for the AUC.
- **ncpus** number of CPUS that will be used when parallel computing is chosen.
- **all** parameter indicating whether all probabilities given by the predictive model should be considered (T) or only those corresponding to individuals whose condition as positive or negative is unknown (F).

The sMSROC function returns an object of class **sMSROC**. It is a list of the following elements:

- **thres** a vector of thresholds at which the sensitivity and the specificity have been computed.
- **SE** a vector with the sensitivities at the considered thresholds **thres**.
- **SP** a vector with the specificities calculated at the thresholds **thres**.
- **probs** a vector with the predictive model estimates for each threshold. Recall that they represent the probability for the observations of being positive, given their marker values.
- **u** sequence of points at which the sMS estimator will be computed. Its length is determined by the **grid** selection parameter.
- **ROC** sMS ROC curve estimate computed at each value of the sequence **u**.
- **time** the point of time at which the ROC curve has been computed, in the case of time-dependent outcomes.
- **auc** area under the sMS ROC curve estimate. It is computed as the sum of the area of the rectangles whose base lies on two consecutive values of the sequence **u** and whose height is the sMS estimate value that corresponds to the lower edge of the base.
- **auc.ci.cl** confidence level at which the confidence interval for the AUC has been computed.
- **auc.ci.l** lower bound of the confidence interval for the AUC.
- **auc.ci.u** upper bound of the confidence interval for the AUC.
- **ci.meth** method used for estimating the confidence intervals for the AUC.
- **data** a list whose elements are, in addition to the type of outcome handled, the set of parameters that had been used to compute the sMS estimator. The values for the **grid**, **meth**, **parallel** and **ncpus** elements are the default or those entered by the users. The **marker**, **status**, **observed**, **time**, **left** and **right** vectors do not contain the positions that correspond to the missing marker values. The **outcome** element is a vector taking the value 1 for the positive individuals; 0, for negative and -1 for the missing/censored/undefined ones.
- **message** a table with the warning messages generated along the computation process.

The computation of the sMS ROC curve estimator is wrapped in three secondary functions according to each scenario. They have the same structure: a part in which the predictive model is estimated (first stage of the sMS estimator), and a common part where the remainder unknown elements are approximated through their empirical estimators (second stage of the sMS estimator).

The functions computing the predictive models have a common output: a list with the ordered marker values and their corresponding estimated probabilities. These two components and the **grid** are the input parameters of the [compute_ROC](#) function, which implements the computation of the

second stage of the sMS estimator. This function provides: the sMS estimates for the sensitivity and the specificity according to the expressions given in (4) and (5); the ROC curve approximation obtained from these estimates (at the granularity level chosen through the **grid** parameter) and the underlying AUC. It is possible as well to directly enter the probabilities corresponding to the predictive model as parameter **probs**. In this case, none of the functions that compute the predictive model are called.

When the **conf.int** parameter is set to T, confidence intervals for the AUC are provided at the confidence level indicated as **ci.cl** parameter. They are computed according to the method entered as **ci.meth** parameter.

- When the chosen method is B the function **auc_ci_boot** is called and the confidence intervals based on BP are calculated. Depending on the type of outcome handled, the corresponding functions for computing the AUC under the sMS estimator are called **ci.nboost** times. Since the bootstrap processes can be time-consuming, the function can be run in parallel via the argument **parallel**. In this case, the number of desired CPUs to be used should be indicated through the parameter **ncpus**. The package **foreach** was used to implement the parallel computation.
- Placing the option V as **ci.method**, the function **auc_ci_nvar** is called and the confidence intervals are computed according to the asymptotic normality of the AUC estimator, based on the TVE approximation. The expression for the variance in this method depends on the variance of the predictive model estimates. That variance is calculated in an independent auxiliary function by bootstrapping (it is possible as well to perform parallel computation to carry out this task). When the probabilities of the predictive model have been directly entered by the users, there is also the option of indicating, manually, the corresponding standard deviation for these probabilities.
- When the selected method is E, the function **auc_ci_emp** uses the asymptotic normality of the AUC estimator, in this case, the variance is calculated through the EVE approximation.

The three functions have the same output: a list with two components, the lower and upper edges of the computed confidence intervals for the AUC.

Example 1 [ROC curve models]: the diabetes dataset

Coming back to the diabetes dataset, we compute the ROC curves to assess the ability of *Stabilized glucose* to identify patients with diabetes. First, we only include this biomarker in the model, and since we observed that its distribution within the positive and the negative populations differs in location, spread, and shape, we used a smooth logistic regression for the first-stage estimation.

```
> roc_diabetes <- sMSROC(marker=diabet$stab.glu, status=diabet$diab, meth="S")
> roc_diabetes
```

The AUC is 0.926.

Predictive model computed through a smooth logistic regression model, based on 60 positive, 330 negative, and 13 undefined (mixed) subjects.

The object **roc_diabetes** also contains the following components

```
> summary(roc_diabetes)
```

	Length	Class	Mode
thres	403	-none-	numeric
SE	403	-none-	numeric
SP	403	-none-	numeric
probs	403	-none-	numeric
u	1001	-none-	numeric
ROC	1001	-none-	numeric
auc	1	-none-	numeric
ci.meth	3	-none-	character
data	6	-none-	list
message	4	-none-	character

which allow to perform a number of customized figures and analyses, including the selection of thresholds under different criteria.

On the other hand, if we want to use an alternative model in the first-stage, for instance, by including additional information which could help us to have a more accurate prediction of the real

status of the undefined subjects, we just have to perform this first-stage out of the `sMSROC` function, and save the vector of the predicted probabilities. This is illustrated in the following chunk of code, where we use logistic regression with first-order effects for both the biomarker and age.

```
> alt_mod <- glm(diab ~ diabet$stab.glu + diabet$age, family = 'binomial')
> prob_model <- predict(alt_mod, type = 'response',
                           newdata = data.frame(diabet$diab, diabet$stab.glu, diabet$age))
```

Then, we include these probabilities in the function

```
> roc_diabetes_prob <- sMSROC(marker=diabet$stab.glu, status=diabet$diab,
                                 probs=prob_model)
> roc_diabetes_prob
```

The AUC is 0.886.

Predictive model externally computed. Based on 0 positive, 0 negative, and 403 undefined (mixed) subjects.

Notice that, in this example, all the subjects are considered as mixed, since none of them are considered as fully positive not fully negative. If we would want to apply this model only on those subjects for which the actual status is unknown, we should introduce probabilities of 1 of 0 for the actually positive, and actually negative, respectively.

Example 2 [ROC curve models]: the kidney transplant failure score (KTFS) dataset

In this second example, our aim is to evaluate the prognostic ability of the *Kidney Transplant Failure Score* (KTFS) to predict the graft failure after five years from kidney transplantation. The KTFS is a composite score build on the base of accepted risk factors of graft loss (Foucher et al., 2010). We will use a subset of the DIVAT cohort (<https://www.divat.fr>) delivered at the `RISCA` package, and now also included in our package. This dataset, `ktfs`, contains the follow-up time from transplantation in years to either of the graft failure or the censoring time (in many cases due to death), a graft failure indicator, and the KTFS score for 2,169 kidney transplant recipients. The distribution of the KTFS score on both patients with graft failure within 5 years (108), and those with a functional graft after 5 years (954) was depicted in Figure 1 (right). Notice that 1107 patients were undefined (follow-up below 5 years and graft working).

```
> data(ktfs)
> roc_KTFS <- sMSROC(marker=ktfs$score, status=ktfs$failure,
>                      observed.time=ktfs$time, time=5, meth="L",
>                      conf.int="T", ci.meth ="E")
```

The AUC is 0.763.

Predictive model computed through a Cox PH regression model, based on 108 positive, 954 negative, and 1107 undefined (mixed) subjects.

4.3 Summarize and plot functions

The `sMSROC` package includes functions which provide numerical and graphical summaries of the data contained in the object returned by the `sMSROC` function. We describe them below.

The `sMSROC_plot` provides informative plots of the sMS ROC curve estimate. The function has the following input parameters:

- **sMS** an object of class `sMSROC`.
- **m.value** marker value. When specified, the point which corresponds to that marker value is added over the plot of the ROC curve.

The function generates two different types of graphics. On one hand, it computes a basic plot approximating the ROC curve by the pairs given by the sequences **1 - SP** and **SE**, from the `sMSROC` object. We have added to this plot the layers `geom_roc()` and `roc_style()` from the `plotROC` package, to obtain a final object that could take advantage of the whole functionality of this package. On the other hand, we produce a customized graphic of the ROC curve whose class is `ggplot` by plotting the sequence **1 - SP** against **SE**. In the case that a number of **m.value** is indicated, the final plot displays over the ROC curve line the point that corresponds to the entered value.

The output of the function is a list with two components:

- **basic.plot** an object that can be used and customized using the tools from the **plotROC** package.
- **roc.plot** an object of class **ggplot**. Although it is already customized (e.g. title, colors or axis labels) the users can make their own changes by adding the corresponding layers through the tools available in the **ggplot2** package.

The function **evol_AUC** provides a graphic with the areas under the time-dependent ROC curves computed by the sMS estimator over a sequence of times. Its input parameters include the sample information for time-dependent outcomes (**marker**, **status**, **observed.time**, **left** and **right**); the **time**, that in this case is a vector containing the points of time at which the AUC will be computed; the method of computation (**meth**) and the **grid**. The features of these parameters are the same described for the corresponding functions in the package. The function **evol_AUC** calls **sMSROC** at each of the times indicated in the vector **time**, and the AUC is computed according to the parameters indicated. The output is a list with three elements:

- **evol.auc** an object of class **ggplot**. It is a graphic line plotting the AUCs at the considered times.
- **time** a vector with the values of the **time** entered as parameter.
- **auc** a vector with the values of the AUCs computed at the times indicated at the **time** parameter.

The function **pred_probs** plots the predicted probabilities estimated from the predictive model for each of the marker values. It may provide a 95% pointwise confidence intervals. The input parameters of the function are:

- **sMS** an object of class **sMSROC**.
- **var** argument with two possible values indicating whether the pointwise confidence intervals should be computed (T) or not (F).
- **nboost** number of bootstrap samples for computing the pointwise confidence interval.
- **parallel** argument indicating whether parallel computing will be carried out (T) or not (F).
- **ncpus** number of CPUS to be used in the case of choosing parallel computing.

The function **pred_probs** generates a graphic for the probability estimation of the predictive model versus the marker values. As usual, this is a **ggplot** object which can be customized by the user. In the case that the **var** option is set to T, the function computes and plots 95% pointwise confidence intervals on the same graphic. The variance of the probability estimates is computed via bootstrap. The output of the function is a list with four components:

- **plot** an object of class **ggplot**.
- **thres** a vector of marker values (x-axis coordinates).
- **probs** a vector containing the predicted probabilities (y-axis coordinates).
- **sd.probs** a vector containing the estimation of the deviation of the predicted probabilities.

Example 2 [ROC curve plots]: the kidney transplant failure score (KTFS) dataset

The next piece of code returns the plot of the ROC curve computed on the data from the *KTFS* example (top-left). We only show the basic plot, however, it can be customized with elements from the **plotROC** package. The code also generates the probabilities derived from the predictive model used in the first stage (i.e. proportional hazard Cox regression), and included in the same panel (top-right), and the evolution of the AUC over ten years from kidney transplantation (bottom).

```
> # ROC curve
> plot_KTFS <- sMSROC_plot(sMS = roc_KTFS, m.value = 3.9)
> plot_KTFS$rocplot
>
> # Evolution of the AUCs
> aucs <- evol_auc(marker = ktfs$score, status = ktfs$failure,
>                   observed.time = ktfs$time,
>                   time = seq(2, 10), meth = "L")
> plot_aucs <- aucs$evol.auc +
>   scale_x_continuous(limit = c(2, 10),
>                     breaks = seq(2, 10, 1)) +
>   scale_y_continuous(limit = c(0.4, 1),
>                     breaks = seq(0.4, 1, 0.1))
> df1 <- data.frame(x = c(2,10), y = c(0.5, 0.5))
```

```

> plot_aucs <- plot_aucs +
> geom_line(data = df1, aes(x, y), linewidth = 0.9, colour = "gray", linetype = "twodash")
> plot_aucs
>
> # Predictive model
> probs <- probs_pred(roc_KTFS, var = "T")
> plot_probs_pred <- probs$plot + xlab("KTFS")
> plot_probs_pred

```

All these plots are arranged in the Figure 2.

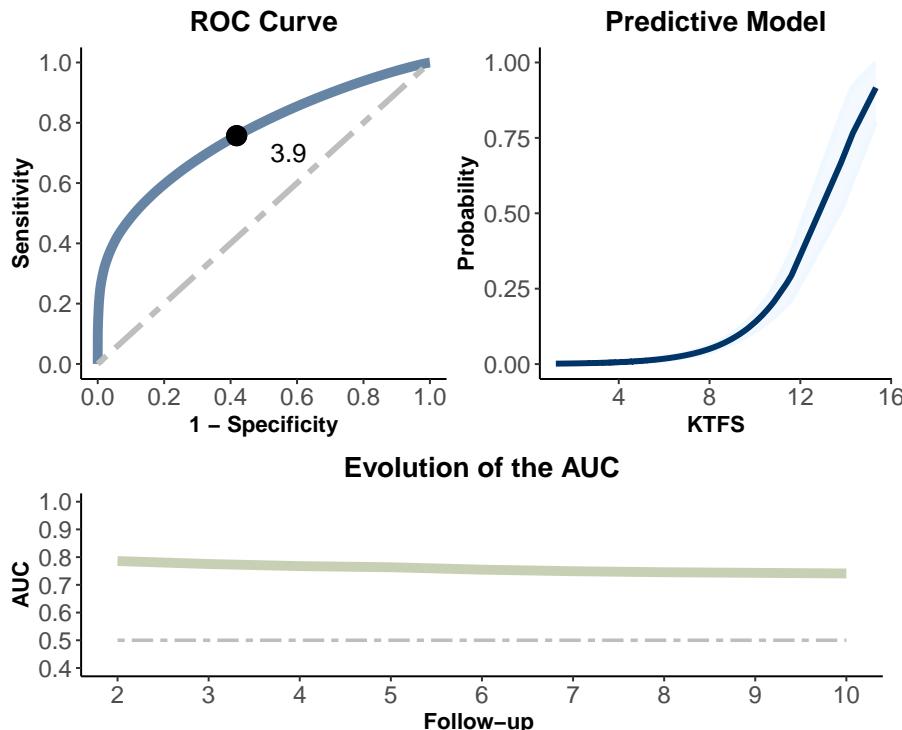


Figure 2: Upper left: graphic of ROC curve estimate obtained from the `sMSROC_plot` function, customized to show the point of the curve corresponding to a given *KTFS* value. Upper right: estimated probabilities from the predictive model with 95% pointwise confidence intervals computed for the biomarker. Bottom: evolution of the AUCs over 10 years from kidney transplantation for the *KTFS* score.

Finally, the function `conf_int_print` prints the values, method of computation, and the confidence level of the confidence intervals calculated for the AUC. Since the `sMSROC` object may contain information stored in large list of components, we only print a single summary, such as the lower and the upper bounds of the confidence intervals, the level at which they were obtained, and the method used for their computation. The input of this function is an object of class `sMSROC` and the output is a string including the described information. As example, below is shown the code for printing the AUC and 95% confidence intervals for the *KTFS* at 5 years:

```

> conf.int.print(roc_KTFS)
"AUC: 0.76; 0.95%\% C.I.[0.6, 0.93]"

```

5 Example 3: the fibrosis dataset

We finally consider the fibrosis dataset. This synthetic data set ships alongside singR and emulates a retrospective study carried out at three different medical centers in Spain. The goal was to determine the capacity of a score punctuation (based on the age, different polymorphisms, and other variables)

to predict the worsening of the fibrosis stage in patients with chronic Hepatitis C (HC). A total of 722 individuals infected by the HC virus, and underwent revision since a particular date were enrolled. The date of the diagnosis of HC and a number of variables determining the risk score such as the age, gender, alcohol consumption, and different polymorphism variants were also collected. Reader interested in having more information about the original study are referred to [Vidal-Castineira et al. \(2020\)](#).

Particularly, we are interested in knowing the prediction ability of the constructed risk score at 5, 10, and 15 years from the HC onset. However, for each patient, we only know whether or not the fibrosis worsened within the interval $(0, R_i]$ or (R_i, ∞) ($1 \leq i \leq 722$), where R_i is the time between the diagnosis and the revision dates. Therefore, we have an interval censorship scenario in which, for instance, at 5 years, the i th patient is positive if they were positive in the revision and $R_i \leq 5$, negative if they was negative in the revision and $R_i \geq 5$, and undefined otherwise. The next piece of code deals with the ROC curve construction at 5 years. Since higher values of the score are associated with smaller probabilities of having the event, we have transformed the values adequately.

```
> data(fibrosis)
> explore_table(marker = -fibrosis$Score, left = fibrosis$Start,
>                 right = fibrosis$Stop, time = 5)$summary

      Sample Size Minimun Maximum Mean   Sd Variance Q1 Median Q3
1 Positive    21      -8      -2 -5.95 1.63    2.65  -7     -6  -5
2 Negative   112     -22     -3 -11.30 3.21   10.30 -13    -11  -9
3 Miss/Cens/Und 589     -21     -3 -10.02 3.09    9.56 -12    -10  -8
4 Total      722     -22     -2 -10.10 3.19   10.19 -12    -10  -8

> roc_fibrosis_5 <- sMSROC(marker = -fibrosis$Score, left = fibrosis$Start,
>                               right = fibrosis$Stop, meth = "L", time = 5)
> roc_fibrosis_5
```

The AUC is 0.647.

Predictive model computed through a D. Finkelstein PH regression model, accounting to the length of the observed intervals, based on 21 positive, 112 negative and 589 undefined (mixed) subjects.

```
> sMSROC_plot(roc_fibrosis_5)$rocplot
```

Figure 3 shows the ROC curves at 5, 10, and 15 years using the sMS ROC curve estimates (top-left), and the estimator proposed by [Beyene and El Ghouch \(2022\)](#) (top-right) and recently implemented in the package [cenROC](#). Besides, since the object `roc_fibrosis_5` also contains the values of both the sensitivity and the specificity for each potential threshold, the next simple piece of code allows to compute the weighted Youden index ([Martínez-Camblor, 2011](#)).

$$J_\lambda = \max_{x \in \mathbb{R}} \{\lambda \cdot Se(x) + (1 - \lambda) \cdot Sp(x)\} \quad \lambda \in [0, 1],$$

and its associated threshold. Figure 3 (middle) depicts J_λ at 5 years, and highlights some of the thresholds. Notice that $J_{1/2}$ is equivalent to the Youden index. We also include the AUC evolution along the follow-up computed through the `sMSROC` (blue line) and `cenROC` (red line) packages. AUCs at 5, 10, and 15 years were 0.647, 0.680 and 0.687 for the sMS ROC curve (Figure 3 bottom), and 0.640, 0.653 and 0.680 for the cenROC-based estimations.

```
> lambda <- seq(0, 1, length = 101)
> Yw <- seq(0, 1, length = 101)
> Tw <- seq(0, 1, length = 101)

> for (j in 1:101) {
>   Yw[j] <- max(lambda[j]*roc_fibrosis_5$SE + (1-lambda[j])*roc_fibrosis_5$SP)
>   Tw[j] <- roc_fibrosis_5$thres[which.max(lambda[j]*roc_fibrosis_5$SE +
>                                         (1-lambda[j])*roc_fibrosis_5$SP)]}
```

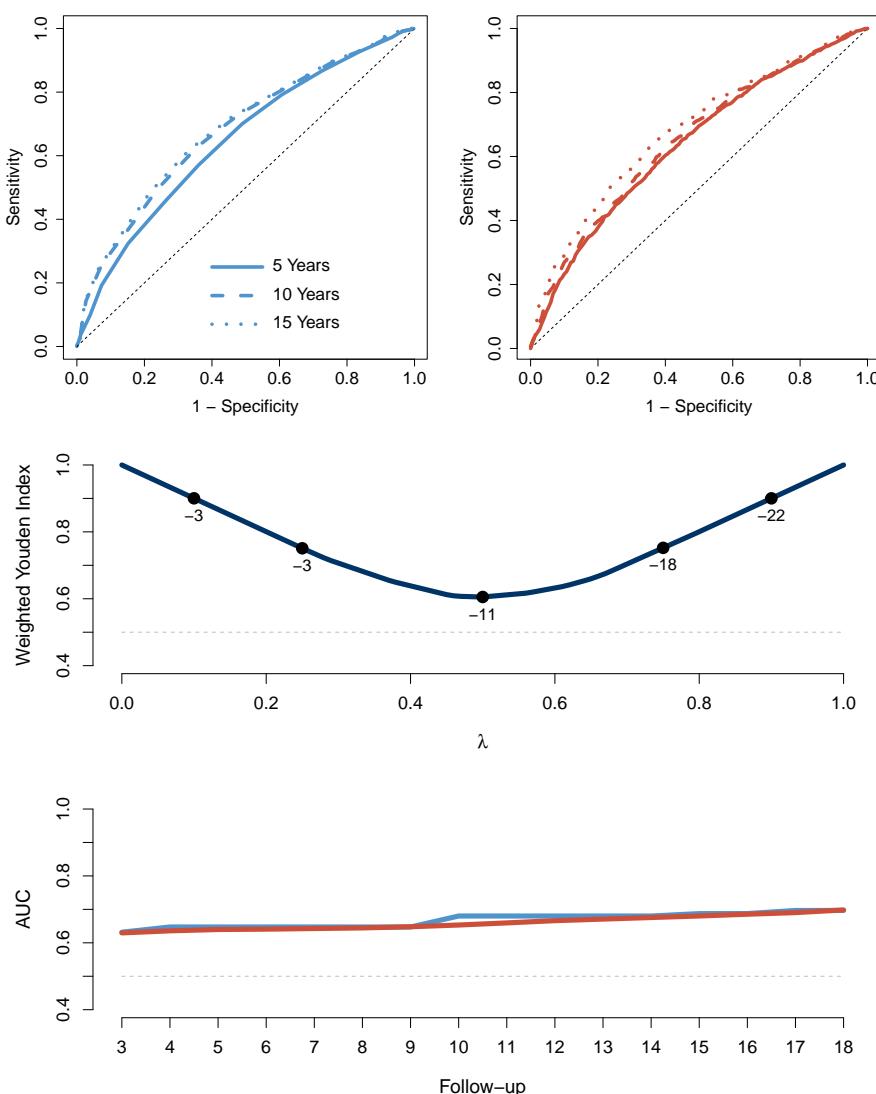


Figure 3: Top: ROC curves estimates at 5, 10 and 15 years from the **sMSROC** (left), and from the **cenROC** (right) packages. Middle: Weighted Youden Index at 5 years from the sMS ROC curve estimation. Bottom: AUC evolution from 3 to 18 years for the two considered estimators.

6 Conclusions

We presented the new R package **sMSROC** which implements the two-stage mixed-subjects ROC curve estimator. This procedure allows the user to assess the classification performance of both diagnostic and prognostic biomarkers. The package offers a set of exploratory functions which allow researchers to have an insight of the distribution of the biomarkers on positive and negative individuals, and on those whose status is unknown. A single main function (**sMSROC**) wraps secondary functions developed to compute the sMS estimator, and the AUC with a confidence interval. This method allows to link the diagnosis and prognosis scenarios via a predictive model which models the relationship between the biomarker and the event under study. The most common probabilistic models (e.g. logistic regression, Cox proportional hazards regression) are implemented out-of-the-box and the user can also enter their own predicted probabilities which can be computed using any other appropriate model. A separate function computes the weighted empirical estimator of the biomarker to get the corresponding estimates for the sensitivity and specificity (second stage). We also implemented three different ways of computing the variance of the AUC and these are available in the package. The package also contains several summarize functions which provide useful numerical and graphical outputs. These include the ROC curve plots, a plot of the evolution of the AUC over time, or the plots of the predictive models.

Appendix

Assume a sample with N subjects. Let $\{\hat{P}(D|x_1), \dots, \hat{P}(D|x_N)\}$ be the individual estimated probabilities of being within the positive group, and let $\{\hat{\sigma}^2(x_1), \dots, \hat{\sigma}^2(x_N)\}$ be their respective variance. Then, if $\hat{\pi}_N = N^{-1} \sum_{i=1}^N \hat{P}(D|x_i)$, the TVE approximation for the variance of AUC is

$$\text{TVE} = \frac{1}{[\hat{\pi}_N \cdot (1 - \hat{\pi}_N)]^2} \cdot [\hat{\sigma}_1^2 + \hat{\sigma}_2^2],$$

where

$$\begin{aligned}\hat{\sigma}_1^2 &= N^{-1} \sum_{j=1}^N \left\{ [\hat{W}_{\text{se}}(x_j) + \hat{W}_{\text{sp}}(x_j)] \cdot \mathcal{P}(D|x_j) - \hat{W}_{\text{sp}}(x_j) \right\}^2, \quad \text{and} \\ \hat{\sigma}_2^2 &= \left\{ N^{-1} \cdot \sum_{j=1}^N [\hat{W}_{\text{sp}}(x_j) - \hat{W}_{\text{se}}(x_j)] \hat{\sigma}(x_j) \right\}^2,\end{aligned}$$

and with

$$\begin{aligned}\hat{W}_{\text{se}}(x) &= N^{-1} \sum_{j=1}^N [I_{(x_j, \infty)}(x) - \hat{Se}(x_j)] \cdot [1 - \hat{P}(D|x_j)], \quad \text{and} \\ \hat{W}_{\text{sp}}(x) &= N^{-1} \sum_{j=1}^N [I_{(x_j, \infty)}(x) - \hat{Sp}(x_j)] \cdot \hat{P}(D|x_j),\end{aligned}$$

$I_A(s)$ is the indicator function (takes the value 1 if $s \in A$, and 0 otherwise) and $\hat{Se}(\cdot)$ and $\hat{Sp}(\cdot)$ are the estimates for the sensitivity and the specificity, respectively. The most challenging part of approximating the variance is usually the computation of $\hat{\sigma}^2(\cdot)$. When $\hat{P}(D|x)$ is based on logistic or Cox-type regression models, closed-form equations for estimating the variance are available. However, these equations are based on the Delta-method and the obtained results are sometimes not good estimates. The EVE approximation considers that the proposed AUC estimator variance is similar to the one based on the empirical estimator of the observed subjects, and therefore it could be approximated through

$$\text{EVE} = \frac{N}{N_O} \cdot \left\{ \frac{1}{1 - \hat{\pi}_N} \cdot \langle \hat{Se}, \hat{Sp} \rangle + \frac{1}{\hat{\pi}_N} \cdot \langle \hat{Sp}, \hat{Se} \rangle \right\},$$

where given two real functions f and g , $\langle f, g \rangle = \int f^2 dg - (\int f dg)^2$, and N_O is the number of subjects with complete information (those used for estimating the predictive model).

References

- C. Anderson-Bergman. icenReg: Regression models for interval censored data in R. *Journal of Statistical Software*, 81(12):1–23, 2017. URL <https://doi.org/10.18637/jss.v081.i12>. [p132]
- N. N. Basu, S. Ingham, J. Hodson, F. Laloo, M. Bulman, A. Howell, and D. G. Evans. Risk of contralateral breast cancer in BRCA1 and BRCA2 mutation carriers: a 30-year semi-prospective analysis. *Familial Cancer*, 14(4):531–538, 2015. URL <https://doi.org/10.1007/s10689-015-9825-9>. [p129]
- K. M. Beyene and A. El Ghouch. Time-dependent ROC curve estimation for interval-censored data. *Biometrical Journal*, 64(6):1056–1074, 2022. URL <https://doi.org/10.1002/bimj.202000382>. [p129, 142]
- K. M. Beyene and A. El Ghouch. *cenROC: estimating time-dependent ROC curve and AUC for censored data*, 2023. URL <https://CRAN.R-project.org/package=cenROC>. R package version 2.0.0. [p129]
- K. M. Beyene and A. El Grouch. Smoothed time-dependent receiver operating characteristic curve for right censored survival data. *Statistics in Medicine*, 39(24):3373–3396, 2020. URL <https://doi.org/10.1002/sim.8671>. [p129]
- P. Blanche, J. F. Dartigues, and H. Jacqmin-Gadda. Review and comparison of ROC curve estimators for a time-dependent outcome with marker-dependent censoring. *Biometrical Journal*, 55(5):687–704, 2013a. URL <https://doi.org/10.1002/bimj.201200045>. [p129]

- 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, 2013b. URL <https://doi.org/10.1002/sim.5958>. [p129]
- L. Chambles and G. Diao. Estimation of time-dependent area under ROC curve for long-term risk prediction. *Statistics in Medicine*, 20(25):3474–3486, 2006. URL <https://doi.org/10.1002/sim.2299>. [p129, 132]
- D. R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(2):187–220, 1972. URL <https://doi.org/10.1111/j.2517-6161.1972.tb00899.x>. [p136]
- S. Díaz-Coto, P. Martínez-Camblor, and N. O. Corral-Blanco. Cumulative/dynamic ROC curve estimation under interval censorship. *Journal of Statistical Computation and Simulation*, 90(9):1570–1590, 2020a. URL <https://doi.org/10.1080/00949655.2020.1736071>. [p129, 132, 136]
- S. Díaz-Coto, P. Martínez-Camblor, and S. Pérez-Fernández. smoothROCtime: an R package for time-dependent ROC curve estimation. *Computational Statistics*, 2020b. URL <https://doi.org/10.1007/s00180-020-00955-7>. [p129]
- S. Díaz-Coto, N. Corral-Blanco, and P. Martínez-Camblor. Two-stage receiver operating-characteristic curve estimator for cohort studies. *The International Journal of Biostatistics*, 17:117–137, 2021. URL <https://doi.org/10.1515/ijb-2019-0097>. [p130, 131]
- J. I. Epstein, L. Egevad, M. B. Amin, B. Delahunt, J. R. Srigley, P. A. Humphrey, and the Grading-Committee. The 2014 international society of urological pathology (ISUP) consensus conference on gleason grading of prostatic carcinoma. *The American Journal of Surgical Pathology*, 40(2):244–252, 2016. URL <https://doi.org/10.1097/PAS.0000000000000530>. [p129]
- R. Etzioni, M. Pepe, G. Longton, C. Hu, and G. Goodman. Incorporating the time dimension in receiver operating characteristic curves: A case study of prostate cancer. *Medical Decision Making*, 19(3):242–251, 1999. URL <https://doi.org/10.1177/0272989X9901900303>. [p129]
- P. M. Farrell, B. J. Rosenstein, T. B. White, F. J. Accurso, C. Castellani, G. R. Cutting, P. R. Durie, V. A. LeGrys, J. Massie, R. B. Parad, M. J. Rock, P. W. Campbell 3rd, and Cystic-Fibrosis-Foundation. Guidelines for diagnosis of cystic fibrosis in newborns through older adults: Cystic fibrosis foundation consensus report. *The Journal of Pediatrics*, 153(2):S4–S14, 2008. URL <https://doi.org/10.1016/j.jpeds.2008.05.005>. [p128]
- E. Ferreirós, C. Boissonnet, R. Pizarro, P. Merletti, G. Corrado, A. Cagide, and O. Bazzino. Independent prognostic value of elevated C-reactive protein in unstable angina. *Circulation*, 100(19):1958–1963, 1999. URL <https://doi.org/10.1161/01.CIR.100.19.1958>. [p129]
- D. M. Finkelstein. A proportional hazards model for interval-censored failure time data. *Biometrics*, 42(4):845–854, 1986. URL <https://doi.org/10.2307/2530698>. [p136]
- Y. Foucher, P. Daguin, A. Akl, M. Kessler, M. Ladrière, C. Legendre, H. Kreis, L. Rostaing, N. Kamar, G. Mourad, V. Garrigue, F. Bayle, B. H. de Ligny, M. Büchler, C. Meier, J. P. Daurès, J. P. Soullou, and M. Giral. A clinical scoring system highly predictive of long-term kidney graft survival. *Kidney International*, 78(12):1288–1294, 2010. URL <https://doi.org/10.1038/ki.2010.232>. [p139]
- D. Gohel and P. Skintzos. *flextable: functions for tabular reporting*, 2023. URL <https://CRAN.R-project.org/package=flextable>. R package version 0.9.3. [p132]
- L. Gonçalves, A. Subtil, M. Rosário Oliveira, and P. De Zea Bermudez. ROC curve estimation: An overview. *Statistical Journal*, 12(1):1–20, 2014. URL <https://doi.org/10.57805/revstat.v12i1.141>. [p128]
- J. Hanley and B. McNeil. The meaning and use of the area under the receiver operating characteristic (ROC) curve. *Radiology*, 20(143):29–36, 1982. URL <https://doi.org/10.1148/radiology.143.1.7063747>. [p128]
- F. E. Harrel. *Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression and Survival Analysis*. Springer Series in Statistics. Springer International Publishing, 2015. [p136]
- F. E. Harrell Jr. *rms: Regression modeling strategies*, 2023. URL <https://CRAN.R-project.org/package=rms>. R package version 6.7-1. [p132]

- P. J. Heagerty and P. Saha-Chaudhuri. *survivalROC: time-dependent ROC curve estimation from censored survival data*, 2022. URL <https://CRAN.R-project.org/package=survivalROC>. R package version 1.0.3.1. [p129]
- P. J. Heagerty, T. Lumley, and M. S. Pepe. Time-Dependent ROC Curves for Censored Survival Data and a Diagnostic Marker. *Biometrics*, 56(2):337–344, 2000. URL <https://doi.org/10.1111/j.0006-341x.2000.00337.x>. [p129]
- F. Hsieh and B. W. Turnbull. Nonparametric and semiparametric estimation of the receiver operating characteristic curve. *The Annals of Statistics*, 24(1):25–40, 1996. URL <https://doi.org/10.1214/aos/1033066197>. [p132]
- H. Hung and C. Chiang. Optimal composite markers for time-dependent receiver operating characteristic curves with censored survival data. *Scandinavian Journal of Statistics*, 20(37):664–679, 2010. URL <https://doi.org/10.1111/j.1467-9469.2009.00683.x>. [p129]
- C. M. Hurvich, J. S. Simonoff, and C.-L. Tsai. Smoothing parameter selection in nonparametric regression using an improved Akaike information criterion. *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*, 60(2):271–293, 1998. URL <https://doi.org/10.1111/1467-9868.00125>. [p136]
- A. N. Kamarudin, T. Cox, and R. Kolamunnage-Dona. Time-dependent ROC curve analysis in medical research: current methods and applications. *BMC Medical Research Methodology*, 17(53), 2017. URL <https://doi.org/10.1186/s12874-017-0332-6>. [p129]
- L. Li and C. Wu. *tdROC: non-parametric estimation of time-dependent ROC curve for right censored survival data*, 2016. URL <https://CRAN.R-project.org/package=tdROC>. [p129]
- L. Li, T. Greene, and B. Hu. A simple method to estimate the time-dependent receiver operating characteristic curve and the area under the curve with right censored data. *Statistical Methods in Medical Research*, 27(8):2264–2278, 2018. URL <https://doi.org/10.1177/0962280216680239>. [p129, 132]
- J. Lin, Y. Wu, X. Wang, and K. Owzar. *intcensROC: AUC estimation of interval censored survival data*, 2021. URL <https://CRAN.R-project.org/package=intcensROC>. R package version 0.1.3. [p129]
- J. Long, Z. Yang, L. Wang, Y. Han, C. Peng, C. Yan, and D. Yan. Metabolite biomarkers of Type 2 diabetes mellitus and pre-diabetes: a systematic review and meta-analysis. *BMC Endocrine Disorders*, 20(1):SP174, 2020. URL <https://doi.org/10.1186/s12902-020-00653-x>. [p128]
- M. López-Ratón, M. X. Rodríguez-Álvarez, C. Cadarso-Suárez, and F. Gude-Sampedro. OptimalCutpoints: An R package for selecting optimal cutpoints in diagnostic tests. *Journal of Statistical Software*, 61(8):1–36, 2014. URL <https://doi.org/10.18637/jss.v061.i08>. [p132]
- P. Martínez-Camblor. Nonparametric cutoff point estimation for diagnostic decisions with weighted errors. *Revista Colombiana de Estadística*, 34(1):133–146, 2011. URL <https://doi.org/10.15446/rce>. [p142]
- P. Martínez-Camblor and J. C. Pardo-Fernández. Smooth time-dependent receiver operating characteristic curve estimators. *Statistical Methods in Medical Research*, 27(3):651–674, 2018. URL <https://doi.org/10.1177/0962280217740786>. [p129]
- P. Martínez-Camblor, G. F. Bayón, and S. Pérez-Fernández. Cumulative/dynamic ROC curve estimation. *Journal of Statistical Computation and Simulation*, 86(17):3582–3594, 2016. URL <https://doi.org/10.1080/00949655.2016.1175442>. [p129, 132]
- Microsoft and S. Weston. *foreach: Provides foreach looping construct*, 2022. URL <https://CRAN.R-project.org/package=foreach>. R package version 1.5.2. [p132]
- L. Ni and X. H. Wehrens. Cardiac troponin I - more than a biomarker for myocardial ischemia? *Annals of Translational Medicine*, Suppl 1(6):S17, 2018. URL <https://doi.org/10.21037/atm.2018.09.07>. [p128]
- M. S. Pepe. *The Statistical Evaluation of Medical Tests for Classification and Prediction*. Oxford Statistical Sciences Series, 2003. [p128]
- S. Pérez-Fernández. *nsROC: non-standard ROC curve analysis*, 2017. URL <https://CRAN.R-project.org/package=nsROC>. [p128]

- S. Pérez-Fernández, P. Martínez-Camblor, P. Filzmoser, and N. Corral. nsROC: An R package for non-standard ROC curve analysis. *The R Journal*, 10(2):55–77, 2018. URL <https://doi.org/10.32614/RJ-2018-043>. [p128]
- S. Potapov, W. Adler, and M. Schmid. survAUC: estimators of prediction accuracy for time-to-event data, 2023. URL <https://CRAN.R-project.org/package=survAUC>. R package version 1.2-0. [p129]
- X. Robin, N. Turck, A. Hainard, N. Tiberti, F. Lisacek, J. Sánchez, and M. M. uller. pROC: an open-source package for R and S+ to analyze and compare ROC curves. *BMC Bioinformatics*, 12(3):77, 2011. URL <https://doi.org/10.1186/1471-2105-12-77>. [p128]
- M. X. Rodríguez-Álvarez and V. Inácio. ROCnReg: An R package for receiver operating characteristic curve inference with and without covariates. *The R Journal*, 13:525, 2021. URL <https://doi.org/10.32614/RJ-2021-066>. [p129]
- M. C. Sachs. plotROC: a tool for plotting ROC curves. *Journal of Statistical Software*, 79(2):1–19, 2017. URL <https://doi.org/10.18637/jss.v079.c02>. [p128, 132]
- T. Sing, O. Sander, N. Beerenwinkel, and T. Lengauer. ROCR: visualizing classifier performance in R. *Bioinformatics*, 21(20):7881, 2005. URL <https://doi.org/10.1093/bioinformatics/bti623>. [p128]
- X. Song and X. H. Zhou. A semiparametric approach for the covariate-specific ROC curve with survival outcome. *Statistica Sinica*, 18:947–965, 2008. URL <http://www.jstor.org/stable/24308524>. [p132]
- Terry M. Therneau and Patricia M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer, New York, 2000. ISBN 0-387-98784-3. [p132]
- B. Turnbull. The empirical distribution function with arbitrarily grouped, censored and truncated data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 38(3):290–295, 1976. URL <http://www.jstor.org/stable/2984980>. [p136]
- 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 Statistics Association*, 478(102):527–537, 2007. URL <https://doi.org/10.1198/016214507000000149>. [p129]
- J. R. Vidal-Castiñeira, A. López-Vázquez, P. Díaz-Bulnes, S. Díaz-Coto, L. Márquez-Kisinousky, J. Martínez-Borra, C. A. Navascués, P. Sanz-Cameno, A. A. Juan de la Vega, M. Rodríguez, and C. López-Larrea. Genetic contribution of endoplasmic reticulum aminopeptidase 1 polymorphisms to liver fibrosis progression in patients with HCV infection. *Journal of Molecular Medicine*, 98:1245–1254, 2020. URL <https://doi.org/10.1007/s00109-020-01948-1>. [p142]
- H. Wickham. *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p132]
- J. P. Willems, J. T. Saunders, D. E. Hunt, and J. B. Schorling. Prevalence of coronary heart disease risk factors among rural blacks: A community-based study. *Southern Medical Journal*, 90(8):814–820, 1997. URL <https://doi.org/10.1097/00007611-199708000-00008>. [p135]
- Y. Wu, X. Wang, J. Lin, J. Beilin, and K. Owzar. Predictive accuracy of markers or risk scores for interval censored survival data. *Statistics in Medicine*, 39(18):2437–2446, 2020. URL <https://doi.org/10.1002/sim.8547>. [p129]
- X.-H. Zhou, N. A. Obuchowski, and D. K. McClish. *Statistical Methods in Diagnostic Medicine*. Wiley Blackwell, New York, 2002. [p128]

Susana Díaz-Coto
Department of Orthopaedics, Dartmouth Health, Lebanon, NH, USA
Geisel School of Medicine at Dartmouth, Hanover, NH, USA

Pablo Martínez-Camblor
Faculty of Health Sciences, Universidad Autonoma de Chile, Chile
and
Department of Anesthesiology, Dartmouth Health, Lebanon, NH, USA
Geisel School of Medicine at Dartmouth, Hanover, NH, USA

Norberto Corral-Blanco

*Department of Statistics, Operational Research and Mathematics Didactics, University of Oviedo, Oviedo
(Asturias), Spain*

Sparse Model Matrices for Multidimensional Hierarchical Aggregation

by Øyvind Langsrud

Abstract Multidimensional hierarchical sum aggregations can be formulated as matrix multiplications involving dummy matrices which can be referred to as model matrices. In contrast to standard model matrices, all categories of all variables must be included. For this purpose, the R package **SSBtools** includes functionality to create model matrices in two alternative ways, by model formulas or by so-called hierarchies. The latter means a coding of hierarchical relationships, and this can be done in several ways. Tree-shaped hierarchies are not required. The internal standard in the package is a parent-child coding. Functionality to find hierarchies automatically from the data is also included. The model matrix functionality is applied in several R packages for statistical disclosure control. This enables general implementation of methods and a flexible user interface. This paper describes the model matrix and hierarchy functions in **SSBtools**, as well as the methods and functions behind it.

1 Introduction

In general, a vector of sum aggregates (z) can be calculated from a data vector (y) through a dummy matrix (x) by

$$z = x^T y \quad (1)$$

The matrix x can be referred to as a model matrix. In package **SSBtools** (Langsrud and Lupp 2023b) there are several tools for creating such model matrices and for computing aggregates via such matrices. This article focuses on these tools, while there are additional tools included in the package that are not addressed herein. Note that the package gathers functions that are used by other packages for specific purposes.

For some applications it is important to have access to the model matrix. In other applications, the interface or the computational efficiency is what is needed. For efficiency, the aggregates may sometimes be computed by

$$z = x_1^T y x_2 \quad (2)$$

where y is a matrix of input data that is appropriately reorganized into multiple columns and where x_1 and x_2 are two dummy matrices.

An important part of the model matrix framework within **SSBtools** is the handling of hierarchical relationships. That is, the categorical variables in the input data are hierarchically related. Or, alternatively, some codings of hierarchical relationships, such as parent-child, are supplied as separate input. We refer to these codings as hierarchies and these are not limited to being tree-shaped. Hierarchies are used both in the process of constructing the model matrix and also to organize the categorical variables in the aggregated output data.

Model matrices are usually associated with regression models which can be expressed by model formulas and fitted by the `lm` function or by other more advanced functions. In this article, we will also use model formulas to specify model matrices. For this purpose, only the right-hand side of a model formula is needed. A formula consisting of the variables a and b including the interaction ($a:b$) can be written as $\sim a + b + a:b$, or equivalently as $\sim a * b$. By using model formulas, we can take advantage of the asterisk sign and other possibilities to write comprehensive models in compact ways (see `?formula` in an interactive R console). The intercept term, which is included by default, can be removed by subtracting a one ($\sim a * b - 1$). In the model matrix, the intercept term is a column of ones. In our context, we would rather refer to this model term as the overall total.

Most elements of the model matrices considered in this paper will be zero. When using regular dense matrices, as created by the base-R function `matrix`, an unnecessary amount of memory is used to store all the zeros. Instead, we use sparse matrices defined in the **Matrix** package (Bates, Maechler, and Jagan 2023). Then the matrices are stored in a compressed format, while ordinary matrix operations can still be performed. In many cases, the matrices will be so large that applications which do not utilize sparse matrices will be impossible in practice. Please note that the functions discussed in this article do not create any new classes. The primary outputs of these functions include sparse matrices

according to [Matrix](#), data frames, or a combination of both in the form of lists.

The rest of this paper is structured as follows. First, we describe the function, `ModelMatrix`, for generating model matrices. The following section takes a closer look at some underlying computations. Then, a section is devoted to a more advanced function for two-way computation. Within this function, equation (2) is applied. The next section looks at aggregation beyond summation, such as median calculation. Then, there is a section where various alternatives for creating model matrices and aggregates are compared in terms of both time and memory usage. In the penultimate section, we consider applications in other packages. Finally, a short conclusion is given.

The theory in this paper is described using examples and R code.

2 The function `ModelMatrix`

The base-R function `model.matrix` is a workhorse function for constructing model or design matrices used in regression modeling. To create sparse matrices, the corresponding function `sparse.model.matrix` in the [Matrix](#) package (Bates, Maechler, and Jagan 2023) can be used. When all variables involved are categorical, the model matrices are dummy matrices of zeroes and ones. The function `ModelMatrix` in package [SSBtools](#) is an alternative function with a slightly different purpose. The aim is to represent the transformation from input to output when producing various types of aggregated data. As described below, the model matrix can be constructed with both a formula interface and a hierarchy interface.

2.1 Model matrix from formula

In the examples below, `d` is a data frame.

```
> d
```

	age	geo	eu	value
1	young	Spain	EU	66.9
2	young	Iceland	nonEU	1.8
3	young	Portugal	EU	11.6
4	old	Spain	EU	120.3
5	old	Iceland	nonEU	1.5
6	old	Portugal	EU	20.2

A model matrix can be specified with a formula.

```
> ModelMatrix(d, formula = ~age + geo)
```

	Total-Total	old-Total	young-Total	Total-Iceland	Total-Portugal	Total-Spain
[1,]	1	.	1	.	.	1
[2,]	1	.	1	1	.	.
[3,]	1	.	1	.	1	.
[4,]	1	1	.	.	.	1
[5,]	1	1	.	1	.	.
[6,]	1	1	.	.	1	.

In contrast to standard model matrices, all categories are included. All variables are considered categorical. The column names correspond to rows in a data frame, called `crossTable`, which can be requested as output. Such a model matrix can be used to compute aggregates according to (1), for example by:

```
> t(ModelMatrix(d, formula = ~age + geo)) %*% d$value
```

	[,1]
Total-Total	222.3
old-Total	142.0
young-Total	80.3
Total-Iceland	3.3
Total-Portugal	31.8
Total-Spain	187.2

Exactly the same aggregates can be computed more directly using a related **SSBtools** function: `FormulaSums(d, formula = value~age + geo)`. With this function, the aggregates are computed somewhat faster when the data is large. Internally, each model term triggers a call to the function `aggregate`. Correspondingly, `ModelMatrix` calls the `Matrix` function `fac2sparse` repeatedly. Note that the formula interface within `aggregate` is different from our approach. The formula `~age + geo` interpreted by `aggregate` corresponds to `~age:geo - 1` interpreted by `ModelMatrix` or `FormulaSums`. So the formula interface in `aggregate` is less flexible. For example, `aggregate(value ~age + geo, d, sum)` and `aggregate(value ~age * geo, d, sum)` give the same results.

To illustrate hierarchical variables, we proceed with a more complicated formula:

```
> ModelMatrix(d, formula = ~age*eu + geo, crossTable = TRUE)

$modelMatrix
[[ suppressing 12 column names 'Total-Total', 'old-Total', 'young-Total' ... ]]

[1,] 1 . 1 1 . . . 1 . . 1 .
[2,] 1 . 1 . 1 1 . . . . 1
[3,] 1 . 1 1 . . 1 . . . 1 .
[4,] 1 1 . 1 . . . 1 1 . .
[5,] 1 1 . . 1 1 . . . 1 .
[6,] 1 1 . 1 . . 1 . 1 . .

$crossTable
      age      geo
1  Total    Total
2  old      Total
3  young    Total
4  Total      EU
5  Total    nonEU
6  Total   Iceland
7  Total   Portugal
8  Total     Spain
9  old       EU
10 old     nonEU
11 young    EU
12 young   nonEU
```

The column names suppressed from the `$modelMatrix` output can be seen in `$crossTable`. Here there are two variables in `crossTable` even though there are three in the formula. This is because it has automatically been discovered that `geo` and `eu` are hierarchically related. Combining hierarchical variables in this way is often useful and this is common within official statistics. Automatic detection and handling of hierarchical relationships can be turned off with the `avoidHierarchical` parameter. The calculations made in the background are related to the automatic generation of hierarchies discussed below.

Note that `ModelMatrix` does not define any new classes. The dummy matrix will have a sparse matrix class according to `Matrix`. Though, an attribute called `startCol` has been added to make locating individual model terms easy and a related function, `FormulaSelection`, is available in **SSBtools**.

2.2 Model matrix from hierarchies

From the data we can generate hierarchies coded as trees:

```
> dimLists <- FindDimLists(d[c("age", "geo", "eu")])
> dimLists

$age
  levels codes
1      @ Total
2    @@   old
3    @@ young
```

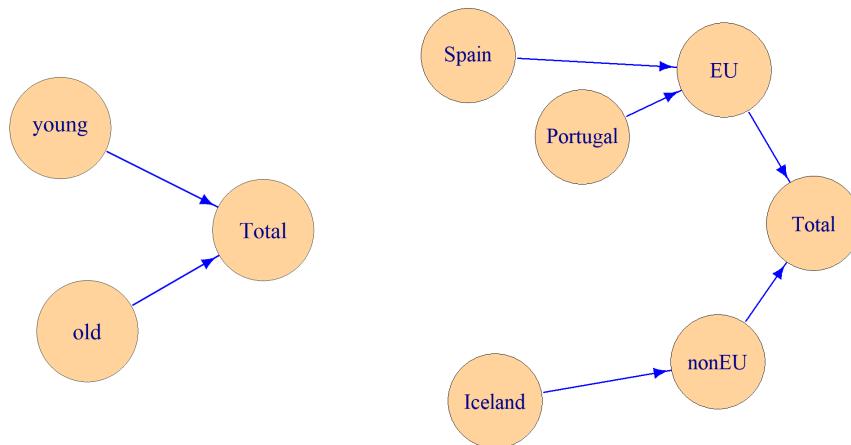


Figure 1: The hierarchies in the examples.

```

$geo
  levels      codes
1      @     Total
2      @@       EU
3      @@@ Portugal
4      @@@   Spain
5      @@  nonEU
6      @@@ Iceland
  
```

The two hierarchies are illustrated in Figure 1. This way of coding hierarchies follows the standard used in the R package [sdcTable](#) (Meindl 2023), which is a tool for statistical disclosure control (SDC) (Hundepool et al. 2012). A model matrix can be constructed from the data and the hierarchies.

```

> ModelMatrix(d[-6, -3], hierarchies = dimLists, inputInOutput = c(TRUE, FALSE),
+               crossTable = TRUE)

$modelMatrix
[[ suppressing 9 column names 'Total:Total', 'Total:EU', 'Total:nonEU' ... ]]

[1,] 1 1 . . . . 1 1 .
[2,] 1 . 1 . . . 1 . 1
[3,] 1 1 . . . . 1 1 .
[4,] 1 1 . 1 1 . . .
[5,] 1 . 1 1 . 1 . .

$crossTable
  age   geo
1 Total Total
2 Total     EU
3 Total  nonEU
4  old Total
5  old     EU
6  old  nonEU
7 young Total
8 young     EU
9 young  nonEU
  
```

By default, with hierarchy input, all possible combinations are made. Here, parameter `inputInOutput` is used to prevent columns being constructed from input codes for geo (country names). The `eu` column

(3rd) was removed from the input data to illustrate that this column is not being used. Instead, the information about EU and nonEU is now given in the hierarchy. The last row (6th) was also removed to illustrate that complete data is not required. Either way, the rows in the output correspond to the rows in the input.

An alternative coding of the above hierarchies follow the standard used in the SDC program called τ -Argus (Wolf et al. 2014). Conversion to and from this standard is possible:

```
> hrc <- DimList2Hrc(dimLists)
> hrc

$age
[1] "old"    "young"

$geo
[1] "EU"      "@Portugal"  "@Spain"   "nonEU"    "@Iceland"
```

However, the standard we use within the framework of ModelMatrix is more general and tree-shaped hierarchies are not required. By the function AutoHierarchies, hierarchies coded in several ways are converted to our internal standard. As shown below, it is also possible to specify total codes.

```
> hi <- AutoHierarchies(hrc, total = c("All", "Europe"))
> hi

$age
  mapsFrom mapsTo sign level
1     old     All    1    1
2   young     All    1    1

$geo
  mapsFrom mapsTo sign level
1     EU Europe    1    2
2 Portugal     EU    1    1
3   Spain     EU    1    1
4   nonEU Europe    1    2
5 Iceland   nonEU    1    1
```

The variable names `mapsFrom` and `mapsTo` were chosen because the starting point was to reprogram an application implemented in the Validation and Transformation Language (VTL) described in Airo et al. (2015). The `level` variable indicates the order in which the codes must be calculated. This variable can be computed automatically and is not needed in input. `Sign`, as coded in the `sign` column, can also be negative (-1), which means subtraction instead of addition. The hierarchies can be very general (but not circular) and it is possible that the final model matrix consists of values beyond zeros and ones. A dummy matrix can be ensured by `unionComplement = TRUE`. Then, in accordance with VTL, `sign` is interpreted as union or complement instead of addition or subtraction.

Another possible coding that can be input to `AutoHierarchies` is formulas. This should not be confused with model formulas in the section above. Conversion to formulas from the internal standard is possible.

```
> Hierarchies2Formulas(hi)

$age
[1] "All = old + young"

$geo
[1] "Europe = EU + nonEU"  "EU = Portugal + Spain" "nonEU = Iceland"
```

By using the parameter `select`, the columns in the model matrix, i.e. `crossTable`, can be specified in advance. This can be much more efficient than making the selection afterwards.

```
> ModelMatrix(d, hierarchies = hi, select = data.frame(
+               age = c("young", "young", "All", "All"),
+               geo = c("EU", "nonEU", "nonEU", "Europe")))
```

```
young:EU young:nonEU All:nonEU All:Europe
[1,]      1       .       .       1
[2,]      .       1       1       1
[3,]      1       .       .       1
[4,]      .       .       .       1
[5,]      .       .       1       1
[6,]      .       .       .       1
```

With few rows in input, many columns in the model matrix will only consist of zeros. As illustrated below, such columns can be omitted by the parameter, `removeEmpty`.

```
> ModelMatrix(d[c(1, 4), ], hierarchies = hi, removeEmpty = TRUE)
```

```
All:Europe All:EU All:Spain old:Europe old:EU old:Spain young:Europe
[1,]      1       1       1       .       .       .       1
[2,]      1       1       1       1       1       1       .
young:EU young:Spain
[1,]      1       1
[2,]      .       .
```

Due to `removeEmpty`, nine columns were omitted ("All", "old" and "young" crossed with "nonEU", "Iceland" and "Portugal"). In large real data sets, the effect of `removeEmpty` can be enormous.

Two special possibilities are that the hierarchies can be encoded as a string or as an empty string. A string is considered a total code. An empty string means that the variable is considered a pure categorical variable without a hierarchy.

```
> ModelMatrix(d[c(1,2,4), ], hierarchies = list(age = "", geo = "Europe"))
```

```
old:Europe old:Iceland old:Spain young:Europe young:Iceland young:Spain
[1,]      .       .       .       1       .       1
[2,]      .       .       .       1       1       .
[3,]      1       .       1       .       .       .
```

In this case only two countries are present in the input data. If `removeEmpty` had been TRUE, the `old:Iceland` column would have been omitted.

Note that both `ModelMatrix` parameters, `hierarchies` and `formula`, can be used simultaneously. How the hierarchies are to be crossed is then defined by the formula.

3 Underlying computations

3.1 Hierarchies automatically from the data

To find hierarchies automatically from the data, a kind of correlation matrix is first computed by the function `FactorLevCorr`. The function was named with factor variables and their levels in mind, but any variable type is possible. To illustrate a little more complexity, we add `isSpain` as a logical variable.

```
> d$isSpain <- d$geo == "Spain"
> fCorr <- FactorLevCorr(d[c("age", "geo", "eu", "isSpain")])
> fCorr
```

	age	geo	eu	isSpain
age	2	0	0.0	0.0
geo	0	3	-1.0	-1.0
eu	0	1	2.0	0.5
isSpain	0	1	-0.5	2.0

The diagonal elements in this matrix are used to store the number of unique values of each variable (n_i). To calculate our correlations, we first find the number of unique combinations (m_{ij}) of each pair of variables. For this purpose, the function, `unique`, is repeatedly called with two-column data frames as input. The absolute values of off-diagonal elements are 0 when $m_{ij} = n_i * n_j$, 1 when $m_{ij} = \max(n_i, n_j)$

and otherwise computed as $(n_i * n_j - m_{ij}) / (n_i * n_j - \max(n_i, n_j))$. So 0 means that all possible level combinations exist in the data and 1 means that the two variables are hierarchically related. In our example the correlation between eu and isSpain isn't zero since the combination (nonEU, TRUE) is missing in the data. The signs of off-diagonal elements are set to be positive when $n_i < n_j$ and negative when $n_i > n_j$. In cases where $n_i = n_j$, elements will be positive above the diagonal and negative below. Values other than 0, -1 and 1 could be useful for detecting errors in the data. For example, values close to one may be suspicious.

In our application, to generate hierarchies, we only need to look for ones. A one means that two hierarchically related variables are found. More generally, one can find trees by connecting relations when the variables are ordered according to the diagonal elements. The function, `HierarchicalGroups`, does this repeatedly using a recursive algorithm so that all possible trees are found. Below we run this function with `eachName = TRUE` so that names are included instead of indices.

```
> HierarchicalGroups(fCorr = fCorr, eachName = TRUE)
```

```
$age
[1] "age"

$geo
[1] "geo"   "eu"

$geo
[1] "geo"   "isSpain"
```

Here two groups were named geo. This means also that `FindDimLists` will result in two hierarchies named geo. Each hierarchy is achieved by looking at sorted unique rows. For example, this hierarchy,

```
> FindDimLists(d[c("age", "geo", "eu", "isSpain")])[3]
```

	levels	codes
1	@	Total
2	@@	FALSE
3	@@@	Iceland
4	@@@	Portugal
5	@@	TRUE
6	@@@	Spain

was constructed from this data frame:

```
> SSBtools::SortRows(unique(d[c("isSpain", "geo")]))
```

	isSpain	geo
2	FALSE	Iceland
3	FALSE	Portugal
1	TRUE	Spain

The top level (@) is always the total code. The remaining levels correspond to the columns in the data as defined by the hierarchical groups. Here this means "isSpain" (@@) and "geo" (@@@).

Note that the corresponding function `FindHierarchies` produces only a single geo hierarchy.

```
> FindHierarchies(d[c("age", "geo", "eu", "isSpain")])$geo
```

	mapsFrom	mapsTo	sign	level
1	Iceland	nonEU	1	1
2	Iceland	Total	1	1
3	Iceland	FALSE	1	1
4	Portugal	EU	1	1
5	Portugal	Total	1	1
6	Portugal	FALSE	1	1
7	Spain	EU	1	1
8	Spain	Total	1	1
9	Spain	TRUE	1	1

FindHierarchies wraps FindDimLists and AutoHierarchies into a single function. By default, when multiple hierarchies exist for the same variable, AutoHierarchies combines them by generating a flattened hierarchy with a single level. The output codes are the parents (`mapsTo`) and all childs (`mapsFrom`) are input codes. Parent-child relationships corresponding to multilevel tree structures are not directly available, but the information needed to create model matrices is available. In fact, the hierarchies are combined via dummy matrices described below.

3.2 From hierarchies to dummy matrices

To produce model matrices, the hierarchies are converted to dummy matrices and the parameter `inputInOutput` is used. Here we convert the hierarchies, `hi`, in the previous section:

```
> hiM <- DummyHierarchies(hi, inputInOutput = c(TRUE, FALSE))
> hiM

$age
      old young
old     1   .
young   .   1
All     1   1

$geo
      Iceland Portugal Spain
EU       .       1       1
nonEU    1       .       .
Europe   1       1       1
```

The first two rows of `hiM$age` are a perturbed identity matrix since `inputInOutput = TRUE` for this variable. The first two rows of `hiM$geo` can be found directly from the level 1 rows of the hierarchy, `hi$geo`. The nonzero elements are taken from the `sign` variable and the corresponding rows and columns can be read from the variables `mapsFrom` and `mapsTo`, respectively. A similar matrix can be constructed from level 2. In our example this matrix is:

```
> geo2 <- Matrix(1, 1, 2, dimnames = list("Europe", c("EU", "nonEU")))
> geo2

      EU nonEU
Europe 1     1
```

The last row of `hiM$geo` can now be found by the matrix multiplication:

```
> geo2 %*% hiM$geo[1:2, ]
      Iceland Portugal Spain
Europe     1       1       1
```

In general, the dummy matrix is constructed successively by adding one level at a time. The new rows are found as a matrix constructed from the new level multiplied by the preliminary dummy matrix. The parameter `unionComplement` mentioned above is used within this process.

The next step is to extend these matrices to match the data and this can be achieved by the function `DataDummyHierarchies`:

```
> hiD <- DataDummyHierarchies(d, hiM, colNamesFromData = TRUE)
> hiD

$age
      young young young old old old
old     .     .     .     1     1     1
young   1     1     1     .     .     .
All     1     1     1     1     1     1

$geo
```

	Spain	Iceland	Portugal	Spain	Iceland	Portugal
EU	1	.	1	1	.	1
nonEU	.	1	.	.	1	.
Europe	1	1	1	1	1	1

For readability `colNamesFromData` was here set to `TRUE`. This also means that the relationship to `hiM` is direct. Now a transposed model matrix can be achieved by the function `KhatriRao` (column-wise Kronecker product) within the `Matrix` package.

```
> Matrix::KhatriRao(hiD$geo, hiD$age, make.dimnames = TRUE)
```

	Spain	Iceland	Portugal	Spain	Iceland	Portugal
old:EU	.	.	.	1	.	1
young:EU	1	.	1	.	.	.
All:EU	1	.	1	1	.	1
old:nonEU	1	.
young:nonEU	.	1
All:nonEU	.	1	.	.	1	.
old:Europe	.	.	.	1	1	1
young:Europe	1	1	1	.	.	.
All:Europe	1	1	1	1	1	1

Now the column names are misleading since only names from the first input matrix are used. However, the row names can be useful. With more than two variables, `KhatriRao` is run several times by adding one variable at a time.

We now recall how a model matrix with pre-selected columns was created in the previous section using the `select` parameter to `ModelMatrix`. When we look at dummy matrices here, this means pre-selected rows. To produce such a matrix with pre-selected rows, one possibility is to first produce separate matrices for each variable.

```
> hiD$age[c("young", "young", "All", "All"), ]
```

	young	young	young	old	old	old
young	1	1	1	.	.	.
young	1	1	1	.	.	.
All	1	1	1	1	1	1
All	1	1	1	1	1	1

```
> hiD$geo[c("EU", "nonEU", "nonEU", "Europe"), ]
```

	Spain	Iceland	Portugal	Spain	Iceland	Portugal
EU	1	.	1	1	.	1
nonEU	.	1	.	.	1	.
nonEU	.	1	.	.	1	.
Europe	1	1	1	1	1	1

The transposed model matrix is then obtained by multiplying these matrices together. This is a fast way to obtain the model matrix. A possible problem is, however, that the matrices to be multiplied are not as sparse as the final matrix. Therefore `KhatriRao` combined with reducing the matrices as much as possible first, can still be better. The choice between the two methods is controlled by a parameter, called `selectionByMultiplicationLimit`. The functionality of the parameter `removeEmpty` mentioned above, is implemented by `KhatriRao` combined with reducing the matrices.

4 Two-way computation

4.1 The function `HierarchyCompute`

We recall that sum aggregates of a data vector (y) can be calculated by equation (1) where the model matrix (x) can be produced by the function `ModelMatrix`. As mentioned above, with formula input an alternative function is `FormulaSums`. Then the aggregates are calculated without x being created. In the cases of hierarchies, a corresponding function is `HierarchyCompute`. Although this function

makes aggregates via one or two dummy matrices, a model matrix that matches the input data is not necessarily made. A special case of `HierarchyCompute` is to make such a model matrix and this matrix can also be requested as output. Actually, `ModelMatrix` with hierarchy input make use of `HierarchyCompute`. The `HierarchyCompute` function does not take many hierarchy types as input, so `AutoHierarchies` may have to be run first.

In general, aggregates are computed by `HierarchyCompute` according to equation (2). But the original version of the function, without the `colVar` parameter, is limited to equation (1). However, `y` may be a multi-column matrix containing a reorganized version of the input data vector. Then the columns represent a single categorical variable with no hierarchy. To tell `HierarchyCompute` to treat a categorical variable in this way, this categorical variable is encoded as the string "colFactor" in the hierarchy input. Other categorical variables are encoded as "rowFactor". The function `AutoHierarchies` re-encodes an empty string to "rowFactor".

To illustrate this function, we add an extra variable, `year`, to the data. Although, it is only a single year.

```
> HierarchyCompute(data = cbind(d, year = 2022),
+   hierarchies = list(age = "colFactor", geo = hi$geo, year = "rowFactor"),
+   valueVar = "value")

      age    geo year value
1 old      EU 2022 140.5
2 old nonEU 2022   1.5
3 old Europe 2022 142.0
4 young    EU 2022  78.5
5 young nonEU 2022   1.8
6 young Europe 2022  80.3
```

Except for column and row order, the output will be the same if "rowFactor" and "colFactor" are swapped or if both age and year are set to "rowFactor". If `output = "matrixComponents"` is included in the function call, the internal differences can be seen. Then, in our example, output become:

```
> hc <- HierarchyCompute(data = cbind(d, year = 2022),
+   hierarchies = list(age = "colFactor", geo = hi$geo, year = "rowFactor"),
+   valueVar = "value", output = "matrixComponents")
> hc

$dataDummyHierarchy

EU:2022      . 1 1
nonEU:2022   1 . .
Europe:2022  1 1 1

$valueMatrix
      old young
[1,] 1.5  1.8
[2,] 20.2 11.6
[3,] 120.3 66.9

$fromCrossCode
      geo year
1 Iceland 2022
2 Portugal 2022
3 Spain 2022

$toCrossCode
      geo year
1 EU 2022
2 nonEU 2022
3 Europe 2022
```

The matrix, `valueMatrix` is a reorganized version of the `value` variable in the input data. If a variable is selected as "colFactor", there is one column for each level of that variable. Even without such a variable, there may be fewer rows in the `valueMatrix` than in the input data. This is because

rows where the values are zero can be removed and because duplicated code combinations in the input can be aggregated. This is controlled by parameters "reduceData" and "handleDuplicated". The data frame `fromCrossCode` contains variables that characterize the rows of `valueMatrix`. The matrix `dataDummyHierarchy` is a transposed model matrix that matches `valueMatrix`. This means that the aggregate output values can be calculated as `dataDummyHierarchy %*% valueMatrix`:

```
> hc$dataDummyHierarchy %*% hc$valueMatrix

      old young
EU:2022    140.5  78.5
nonEU:2022   1.5   1.8
Europe:2022 142.0  80.3
```

The data frame `toCrossCode` contains variables that characterize the rows of this matrix. Along with column names, this is the information needed to reorganize the aggregate result into a regular output data frame.

4.2 HierarchyCompute with colVar

When the parameter `colVar` is applied, two (transposed) model matrices are made, one for rows and one for columns. Parameter `colVar` splits the hierarchy variables in two groups and this variable overrides the difference between "rowFactor" and "colFactor". Actually, there will be two runs of `HierarchyCompute`. With `output = "matrixComponents"`, output from the two runs are returned as a list with elements `hcRow` and `hcCol`. Below we illustrate this by including the age hierarchy with `inputInOutput = TRUE` (`FALSE` is default).

```
> hc2 <- HierarchyCompute(data = cbind(d, year = 2022),
+                           hierarchies = list(age = hi$age, geo = hi$geo, year = "rowFactor"),
+                           colVar = "age", inputInOutput = c(TRUE, FALSE),
+                           valueVar = "value", output = "matrixComponents")
```

For rows, we obtain the same matrices as earlier. The aggregates in the previous subsection can now be computed as:

```
> hc2$hcRow$dataDummyHierarchy %*% hc2$hcRow$valueMatrix

      1     2
EU:2022    140.5 78.5
nonEU:2022   1.5  1.8
Europe:2022 142.0 80.3
```

The model matrix for columns can be seen as:

```
> t(hc2$hcCol$dataDummyHierarchy)

      old young All
[1,]   1     .   1
[2,]   .     1   1
```

Finally, the aggregate output values can be calculated as:

```
> hc2$hcRow$dataDummyHierarchy %*% hc2$hcRow$valueMatrix %*% t(hc2$hcCol$dataDummyHierarchy)

      old young All
EU:2022    140.5 78.5 219.0
nonEU:2022   1.5  1.8  3.3
Europe:2022 142.0 80.3 222.3
```

Taking into account that the dummy matrices are transposed model matrices, equation (2) is applied. With ordinary output, these values are organized into a data frame and we do not see what is going on internally.

```
> HierarchyCompute(data = cbind(d, year = 2022),
+   hierarchies = list(age = hi$age, geo = hi$geo, year = "rowFactor"),
+   colVar = "age", inputInOutput = c(TRUE, FALSE),
+   valueVar = "value")

  age     geo year value
1 old      EU 2022 140.5
2 old    nonEU 2022   1.5
3 old Europe 2022 142.0
4 young    EU 2022  78.5
5 young  nonEU 2022   1.8
6 young Europe 2022  80.3
7 All      EU 2022 219.0
8 All    nonEU 2022   3.3
9 All Europe 2022 222.3
```

We obtain the same results without specifying `colVar`. One reason to use `colVar` is to do the computations more efficiently. Another reason may be that the internal matrices may be of interest. Note that variable combinations for output can be specified by parameters `rowSelect`, `colSelect` and `select`. Then `colVar` also matters.

5 Aggregation beyond summation

The most recent functions in the `SSBtools` package are about general aggregation beyond summation. With respect to future application packages, this is written in a different coding style (lower snake case). The function `model_aggregate` combines the model specification capabilities of `ModelMatrix` with aggregation with general functions. Below is an example:

```
> model_aggregate(d, formula = ~age + eu,
+   fun_vars = c(max = "value", median = "value", length = "geo"))

  age     eu value_max value_median geo_length
1 Total Total  120.3       15.90       6
2 old Total  120.3       20.20       3
3 young Total   66.9       11.60       3
4 Total   EU  120.3       43.55       4
5 Total nonEU    1.8       1.65       2
```

The underlying computations make use of a model matrix. By default (`pre_aggregate = TRUE`), the number of rows in the input data frame is reduced before `ModelMatrix` is called. In practice, this pre-aggregation step is done by running `aggregate` with `FUN = function(x){x}` and `simplify = FALSE`. The individual observations to be aggregated are then included as lists. In this example, the reduced input data is:

```
  age     eu      value           geo
1 old      EU 120.3, 20.2 Spain, Portugal
2 young    EU  66.9, 11.6 Spain, Portugal
3 old    nonEU     1.5        Iceland
4 young  nonEU     1.8        Iceland
```

The corresponding model matrix is:

	Total-Total	old-Total	young-Total	Total-EU	Total-nonEU
[1,]	1	1	.	1	.
[2,]	1	.	1	1	.
[3,]	1	1	.	.	1
[4,]	1	.	1	.	1

The function `model_aggregate` has many possibilities for specifying the aggregation. Functions of several data variables are possible. Output variable names can be passed as input, even for functions with multiple outputs. This is possible since `model_aggregate` calls `dummy_aggregate` (model matrix is input) which calls `aggregate_multiple_fun` which is an advanced wrapper to `aggregate`. The implementation uses indexes to rows, which made it possible to call `aggregate` only once. Since matrix multiplication is a fast way to do sum aggregation, this is also included as a possibility in `model_aggregate` (parameter `sum_vars`).

6 Comparisons with comments

To facilitate comparisons, we utilize hierarchical example data with 10^i rows obtained by crossing i dimensions. For each dimension there is a child variable with 10 categories and a parent variable with 3 categories (2, 3 and 5 categories aggregated). The child and parent variables are named alphabetically by small and capital letters, respectively. Below is an illustration of a `HierarchyCompute` call when $i = 4$, showing eight rows in both the input and output.

```
> d <- SSBtoolsData("power10to4")
> d[c(1:4, 3715:3716, 9999:10000), ]

      a     A     b     B     c     C     d     D
1    a1 A100   b1 B100   c1 C100   d1 D100
2    a2 A100   b1 B100   c1 C100   d1 D100
3    a3 A200   b1 B100   c1 C100   d1 D100
4    a4 A200   b1 B100   c1 C100   d1 D100
3715 a5 A200   b2 B100   c8 C300   d4 D200
3716 a6 A300   b2 B100   c8 C300   d4 D200
9999 a9 A300   b10 B300  c10 C300  d10 D300
10000 a10 A300  b10 B300  c10 C300  d10 D300

> hi <- FindHierarchies(d)
> d$y <- 1:nrow(d)
> output <- HierarchyCompute(d, hi, "y", inputInOutput = TRUE)
> output[c(1, 5473:5475, 37852:37854, 38416), ]

      a     b     c     d     y
1    a1   b1   c1   d1    1
5473 A300 B300 Total d10 2382000
5474 Total B300 Total d10 4762750
5475   a1 Total Total d10 949600
37852   a9 b10 C200 Total 146970
37853 A100 b10 C200 Total 293490
37854 A200 b10 C200 Total 440460
38416 Total Total Total 50005000
```

Here an integer y is added to the input data. Either way, the variable becomes numeric in the output. In the comparisons below, the input class is also numeric. We use `inputInOutput = TRUE`, which is the default in `ModelMatrix`. With 10 child categories, 3 parent categories, and the inclusion of a total code, this results in 14^i rows in the output.

Table 1 displays the CPU times for different function calls. The measurements were made by `system.time` (first element). The median values from five runs are shown. Table 2 displays the memory usage as peak RAM, measured using the `peakRAM` package (Quinn 2017). The median values from five runs are shown. These runs were conducted separately from the CPU time measurements. Table 2 also presents object sizes obtained using the `object.size` function. The source code is shown in Table 3. These comparisons were made with R version 4.2.2 running on Linux. The server had about 500 GiB of RAM, and it appeared that less than 10% of it were occupied by other users.

Since some functions are based on hierarchy input that may need to be generated from data, `FindHierarchies` are also included in the tables. Some functions produce aggregate results (`HierarchyCompute`, `FormulaSums` and `model_aggregate`), while others produce a model matrix. However, the time required for calculating sum aggregates through matrix multiplication is almost negligible. When $i = 6$, `crossprod(x, d$y)` takes four seconds. Here x refers to the sparse model matrix.

Generation of the model matrix from hierarchies by `ModelMatrix` is done via `HierarchyCompute`. The transposed model matrix is outputted before the ordinary results are produced. The main reason why `ModelMatrix` still takes longer, when $i \geq 4$, is a parameter setting that ensures that the column order is more similar to a usual model matrix (`reorder = TRUE`). Some time is also spent on matrix transposition. Even if `ModelMatrix` is run with `crossTable = FALSE`, which is the default, this information is still internally generated in order to create column names.

If the desired output is sum aggregates, there is no doubt that `HierarchyCompute` with `colVar` is the most efficient option. In this case, equation (2) is utilized, and instead of creating one large model matrix, two smaller matrices are generated. For example, when $i = 5$, dimensions a and b are used for columns, while c , d , and e are used for rows.

Table 1: User CPU time in seconds based on hierarchical input data with 10^i rows, resulting in 14^i output aggregates. Thus, the size of the model matrix is $10^i \times 14^i$. The source code is given in Table 3.

function call	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$
FindHierarchies	0.010	0.032	0.392	8	131
HierarchyCompute	0.017	0.027	0.188	5	147
HierarchyCompute with colVar	0.018	0.028	0.086	1	22
ModelMatrix from hierarchies	0.014	0.026	0.229	7	218
ModelMatrix from formula	0.016	0.083	1.294	60	2922
sparse.model.matrix	0.026	0.173	11.703	1597	
model.matrix	0.004	0.024	3.238		
FormulaSums	0.021	0.110	1.417	43	1041
median by model_aggregate	0.030	0.208	3.353	74	1403

Table 2: Peak RAM in MiB for the calculations described in Tables 1 and 3. For $i \geq 4$, the size of the output object is provided within parentheses, denoted in MiB as well.

function call	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$
FindHierarchies	0.9	1.9	12 (0.010)	57 (0.012)	610 (0.015)
HierarchyCompute	3.0	3.8	69 (1.5)	1074 (25)	30792 (402)
HierarchyCompute with colVar	1.2	2.1	20 (1.5)	201 (25)	1273 (402)
ModelMatrix from hierarchies	1.0	4.3	71 (12.2)	1079 (324)	32246 (9003)
ModelMatrix from formula	0.7	16.8	71 (12.2)	919 (324)	27809 (9003)
sparse.model.matrix	2.4	40.0	536 (13.1)	61654 (333)	
model.matrix	0.6	42.1	5864 (2935)		
FormulaSums	1.0	18.2	431 (3.1)	6884 (48)	1252 (689)
median by model_aggregate	0.9	15.2	230 (1.5)	2257 (25)	63435 (402)

Table 3: The source code for the calculations underlying Tables 1 and 2. With i defined, the code can be run directly from top to bottom. Be careful not to run `FindHierarchies(d)` with `y` included in `d`.

function call, etc.	source code
Input data	<code>d <- SSBtoolsData(paste0("power10to", i))</code>
FindHierarchies	<code>hi <- FindHierarchies(d)</code>
Adding numeric variable	<code>d\$y <- as.numeric(1:nrow(d))</code>
HierarchyCompute	<code>HierarchyCompute(d, hi, "y", inputInOutput = TRUE)</code>
HierarchyCompute with colVar	<code>HierarchyCompute(d, hi, "y", inputInOutput = TRUE, colVar = letters[1:(floor(i/2))])</code>
ModelMatrix from hierarchies	<code>ModelMatrix(d, hierarchies = hi)</code>
Formula	<code>f <- as.formula(strtrim("y ~ (a+A)*(b+B)*(c+C)*(d+D)*(e+E)*(f+F)", 3 + 6*i))</code>
ModelMatrix from formula	<code>ModelMatrix(d, formula = f)</code>
sparse.model.matrix	<code>ModelMatrix(d, formula = f, viaOrdinary = TRUE)</code>
model.matrix	<code>ModelMatrix(d, formula = f, viaOrdinary = TRUE, sparse = FALSE)</code>
FormulaSums	<code>FormulaSums(d, formula = f)</code>
median by model_aggregate	<code>model_aggregate(d, hierarchies = hi, fun_vars = c(median = "y"))</code>

Even though the functions handle hierarchical relationships, the formula used in the input must be written appropriately in order to achieve the desired output. To generate all combinations, the input formula to `ModelMatrix` at $i = 4$ would be $\sim(a+A)*(b+B)*(c+C)*(d+D)$. In Table 3, a response term is included in the formula, but it is only necessary for the `FormulaSums` function. The output from `FormulaSums` is a one-column matrix containing aggregates. However, the object size is larger compared to the data frame output of `HierarchyCompute`. This is because row names are a less efficient method of storing the information. It is interesting to note that `FormulaSums` uses more memory at $i = 5$ than $i = 6$. This is probably because the aggregate function uses a different method for large data.

The initial implementation of `ModelMatrix` utilized the functions `sparse.model.matrix` and `model.matrix`. This option still remains available by setting `viaOrdinary = TRUE`. Since these functions omit a factor level, the input variables are encoded as factors and an empty factor level is added. In this way, the desired matrix is achieved, here a $10^i \times 14^i$ matrix. With small data sets ($i \leq 3$), `model.matrix` is the fastest method, but with $i \geq 5$, there was not enough available memory. With $i = 5$ and $i = 6$, the size of the output object would be approximately 400 GiB and 55 TiB, respectively. The results show, perhaps surprisingly, that `sparse.model.matrix` consumes significant time and memory. It was not feasible to run it at $i = 6$. Profiling indicates that this is primarily due to interaction calculations. In contrast, in `ModelMatrix`, which only handles categorical variables, this process is simplified by combining multiple variables from the input before calling `fac2sparse`. The hierarchy interface to `ModelMatrix`, however, is significantly faster than the formula interface. This is because a few calls to the `KhatriRao` function are much faster than building the matrix with `cbind` based on numerous `fac2sparse` calls. The object size of the output matrix is slightly larger when using `sparse.model.matrix` compared to the regular `ModelMatrix`. This is primarily because the output from `sparse.model.matrix` includes row names as well.

Here, `model_aggregate` is used to calculate median values instead of the sums calculated by `HierarchyCompute`. Output is similar, but the rows are sorted differently. Internally, `ModelMatrix` is called, and the formula interface is also possible. Instead of matrix multiplication, the `median` function is now called 14^i times. This is more resource-intensive, but still feasible when $i = 6$.

7 Applications in other R packages

The first of the functions demonstrated above to be used in a specific purpose package was `FindDimLists`. Package `easySdcTable` (Langsrud 2022) provides a simplifying interface to the statistical disclosure control (SDC) package `sdcTable` (Meindl 2023). This is about protecting frequency tables by the suppression method. Automatic hierarchies by `FindDimLists` is an important part of the simplification.

The function `HierarchyCompute` was originally made for Statistics Norway's modernized calculations of `municipal accounts`, which involve complicated hierarchies. The original intention was to do the computations by an implementation of VTL (Airo et al. 2015), but this approach proved to be too inefficient and R was chosen instead. The calculations are implemented in the package `Kostra` (Lillegård et al. 2023) that is made for internal use in Statistics Norway. The two complicated hierarchies involved are illustrated on Figure 2. This figure as well as Figure 1 is made using package `igraph` (Csardi and Nepusz 2006). Plotting could be achieved by converting the hierarchies produced by `AutoHierarchies` into `igraph` objects using the `graph_from_data_frame` function.

The hierarchy interface part of `ModelMatrix` was created as a spin-off from `HierarchyCompute`. The first application was in the SDC package `SmallCountRounding` (Langsrud and Heldal 2022), which protects frequency tables data by the perturbation method described by Langsrud and Heldal (2018). The earliest versions of this package were limited to formula input and the package then contained the first version of `ModelMatrix`. Two other SDC packages also use `ModelMatrix`. The packages `SSBcellKey` (Lupp and Langsrud 2023) and `GaussSuppression` (Langsrud and Lupp 2023a) can be used to protect both frequency and magnitude tables by, respectively, perturbation (Thompson, Broadfoot, and Elazar 2013) and suppression (Hundepool et al. 2012).

All the SDC packages mentioned above are about protection of tabular data. When `ModelMatrix` is used, the model matrix itself is important to the algorithm. The model matrix is therefore necessary for more than aggregating the data by a matrix multiplication. In the package `SSBcellKey`, however, use of the model matrix is limited to running the `SSBtools` function `DummyApply` which is a precursor to the `dummy_aggregate` function mentioned above.

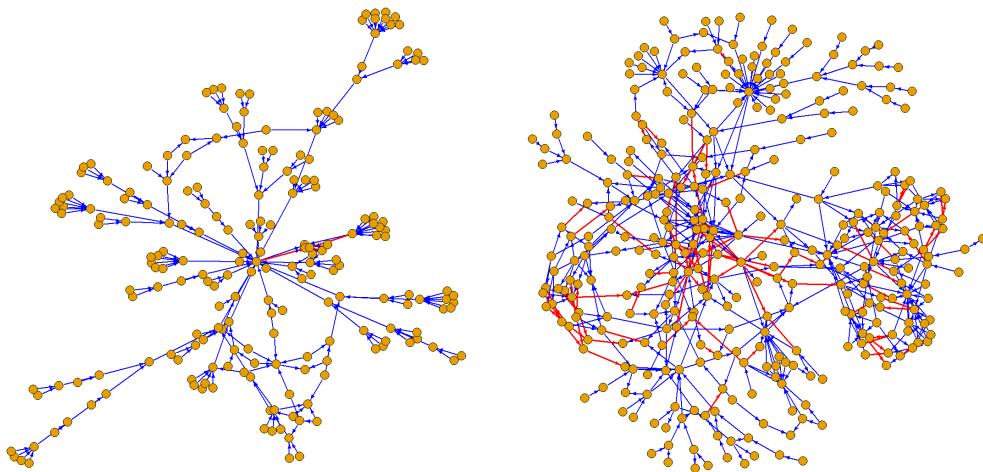


Figure 2: The function (left) and art (right) hierarchy involved in Statistics Norway's municipal accounts calculations. Red arrow means negative sign.

8 Conclusion

The **SSBtools** package provides functionality to handle hierarchical relationships and to generate corresponding model matrices. This is useful in contexts where multidimensional aggregation and related computations are to be done. This is applied by several R packages for statistical disclosure control, a field that is important for official statistics.

Acknowledgements

I would like to thank my colleagues Daniel Lupp and Johan Fosen at Statistics Norway, as well as an anonymous reviewer, for their valuable comments that led to improvements.

References

- Airo, Sami, Foteini Andrikopoulou, David Barraclough, Luigi Bellomarini, Marc Bouffard, Maurizio Capaccioli, Vincenzo Vecchio, et al. 2015. *Validation & Transformation Language, Part 2 Library of Operators, Version 1.0.* https://sdmx.org/wp-content/uploads/VTL1_2015_part2_operators_final.pdf.
- Bates, Douglas, Martin Maechler, and Mikael Jagan. 2023. *Matrix: Sparse and Dense Matrix Classes and Methods.* <https://CRAN.R-project.org/package=Matrix>.
- Csardi, Gabor, and Tamas Nepusz. 2006. "The Igraph Software Package for Complex Network Research." *InterJournal Complex Systems*: 1695. <https://igraph.org>.
- Hundepool, Anco, Josep Domingo-Ferrer, Luisa Franconi, Sarah Giessing, Eric Schulte Nordholt, Keith Spicer, and Peter-Paul de Wolf. 2012. *Statistical Disclosure Control*. John Wiley & Sons, Ltd. <https://doi.org/10.1002/9781118348239.ch1>.
- Langsrød, Øyvind. 2022. *easySdcTable: Easy Interface to the Statistical Disclosure Control Package 'sdcTable' Extended with Own Implementation of 'GaussSuppression'.* <https://CRAN.R-project.org/package=easySdcTable>.
- Langsrød, Øyvind, and Johan Heldal. 2018. "An Algorithm for Small Count Rounding of Tabular Data." Privacy in statistical databases, Valencia, Spain.
- . 2022. *SmallCountRounding: Small Count Rounding of Tabular Data.* <https://CRAN.R-project.org/package=SmallCountRounding>.
- Langsrød, Øyvind, and Daniel Lupp. 2023a. *GaussSuppression: Tabular Data Suppression Using Gaussian Elimination.* <https://CRAN.R-project.org/package=GaussSuppression>.
- . 2023b. *SSBtools: Statistics Norway's Miscellaneous Tools.* <https://CRAN.R-project.org/package=SSBtools>.
- Lillegård, Magnar, Anna-Karin Mevik, Johan Heldal, and Øyvind Langsrød. 2023. *Kostra: Functions for Kostra.* <https://github.com/statisticsnorway/Kostra>.

- Lupp, Daniel, and Øyvind Langsrud. 2023. *SSBcellKey: Cell-Key Method for Tabular Data*. <https://github.com/statisticsnorway/SSBcellKey>.
- Meindl, Bernhard. 2023. *sdcTable: Methods for Statistical Disclosure Control in Tabular Data*. <https://CRAN.R-project.org/package=sdcTable>.
- Quinn, Thomas. 2017. *peakRAM: Monitor the Total and Peak RAM Used by an Expression or Function*. <https://CRAN.R-project.org/package=peakRAM>.
- Thompson, G., S. Broadfoot, and D. Elazar. 2013. "Methodology for the automatic confidentialisation of statistical outputs from remote servers at the Australian Bureau of Statistics." Joint UNECE/Eurostat Work Session on Statistical Data.
- Wolf, Peter-Paul de, Anco Hundepool, Sarah Giessing, Juan-José Salazar, and Jordi Castro. 2014. "tau-ARGUS user's manual, version 4.1." Statistics Netherlands. <https://github.com/sdcTools/tauargus>.

Øyvind Langsrud
Statistics Norway
P.O. Box 8131 Dep.,
0033 Oslo, Norway
<https://github.com/olangsrud>
ORCID: 0000-0002-1380-4396
Oyvind.Langsrud@ssb.no

openalexR: An R-Tool for Collecting Bibliometric Data from OpenAlex

by Massimo Aria, Trang Le, Corrado Cuccurullo, Alessandra Belfiore, and June Choe

Abstract Bibliographic databases are indispensable sources of information on published literature. OpenAlex is an open-source collection of academic metadata that enable comprehensive bibliographic analyses (Priem, Piwowar, and Orr 2022). In this paper, we provide details on the implementation of openalexR, an R package to interface with the OpenAlex API. We present a general overview of its main functions and several detailed examples of its use. Following best API package practices, openalexR offers an intuitive interface for collecting information on different entities, including works, authors, institutions, sources, and concepts. openalexR exposes to the user different API parameters including filtering, searching, sorting, and grouping. This new open-source package is well-documented and available on CRAN.

1 Introduction

Bibliographic data sources are organized digital collections of reference metadata and citation links related to published scientific literature. One primary purpose is to assess the scholarly performance, although it is important to exercise caution when employing bibliometric indicators for performance evaluation (Van Noorden, 2013; Hicks et al., 2015; Priem et al., 2011). Another objective of bibliometric analyzes is science mapping, a process that involves synthesizing extensive data, prioritizing impactful research, and extracting key knowledge structures (Chen, 2017). Given the rapid proliferation of scientific publications, science mapping evidence is particularly valuable for reconstructing the theoretical framework of empirical studies or literature reviews.

Recently, new sources of multidisciplinary bibliographic data have emerged, including Microsoft Academic, launched in 2016 (Sinha et al., 2015; Wang et al., 2019, 2020), Semantic Scholar launched in 2015 (Ammar et al., 2018), and CrossRef, an open bibliographic data source launched in 2017 (Hendricks et al., 2020; Van Eck et al., 2018). Dimensions is a scientometric data source that provides also information on grants, datasets, clinical trials, patents, and policy documents (Herzog et al., 2020; Hook et al., 2018). These databases complement the many existing open and commercial sources. The two most widely used commercial multidisciplinary databases are Web of Science and Scopus while the specialized ones include PubMed, EconBiz, and arXiv, which are the main open bibliographic sources for medicine, economics, and physical sciences and engineering, respectively.

The value of these open and commercial bibliographic data sources hinges on several characteristics: (1) document coverage, (2) completeness and accuracy of citation links, (3) update speed, (4) automation of data access through web interfaces, APIs, and data dumps, and (5) the terms of use for a data source (Wanyama et al., 2022; Kulkanjanapiban and Silwattananusarn, 2022; Singh et al., 2021; Martín-Martín et al., 2021; Visser et al., 2021; Waltman and Larivière, 2020; Winter, 2017). OpenAlex is recognized for providing the most extensive coverage of scientific literature, encompassing a notably larger number of documents compared to other data sources (see <https://openalex.org/about>). Its document coverage outpaces all major databases, including Microsoft Academic (Visser et al., 2021; Wang et al., 2020) and CrossRef, which are its primary sources. While Google Scholar reportedly boasts an estimated 389 million database, it is crucial to note that Google Scholar does not adhere to the conventional database model due to its lack of comprehensive metadata, and it does not permit users to download query search results (Dallas et al., 2018).

Another remarkable strength of OpenAlex is its substantial collection of open-access works, totaling 48 million. This extensive repository grants users open and free access to a wealth of scholarly resources, aligning with OpenAlex's dedication to open science principles. This commitment promotes both accessibility and transparency in the sharing of knowledge. Furthermore, OpenAlex stands out not only in terms of quantity but also in data quality. With a substantial number of citations amounting to 1.9 billion, OpenAlex emphasizes its relevance for researchers engaged in citation-based studies. This robust citation data enhances its appeal as a valuable resource for scholars conducting research reliant on citation analysis.

The purpose of this article is to introduce the OpenAlexR R package, which facilitates the retrieval of metadata from OpenAlex, performs specific functions, and formats the data for utilization in bibliometrix, particularly for science mapping and research assessment purposes.

1.1 OpenAlex

Named after the ancient Library of Alexandria, [OpenAlex](#) is a free and fully open catalogue of scholarly metadata with open data, open APIs, open-source code ([Priem et al., 2022](#)). Behind this tour de force is OurResearch, a nonprofit organisation dedicated to open principles of academic works with other impactful projects such as CiteAs ([Du et al., 2021](#)) and Unpaywall ([Chawla, 2017](#)). OpenAlex was launched in January 2022, timely replacing the retired Microsoft Academic system. OpenAlex is already extensively used in many scholarly articles ([Belfiore et al., 2022](#)). The data is currently accessible through a complete database snapshot and a [REST API](#) that is updated daily.

The OpenAlex database consists of eight academic entity types: works, authors, institutions, sources, concepts, publishers, funders, geo (Fig. 1). It is important to know the OpenAlex entities because it is possible to make query for each entity. In fact, each entity is assigned an OpenAlex ID (OAID) which represents the primary key to access the data. However, OpenAlex also recognizes different [external canonical IDs](#) for different entities. We briefly summarise the eight entities below. For more detail, visit the [documentation page](#) and the more recent [Postgres schema diagram](#) by OpenAlex.

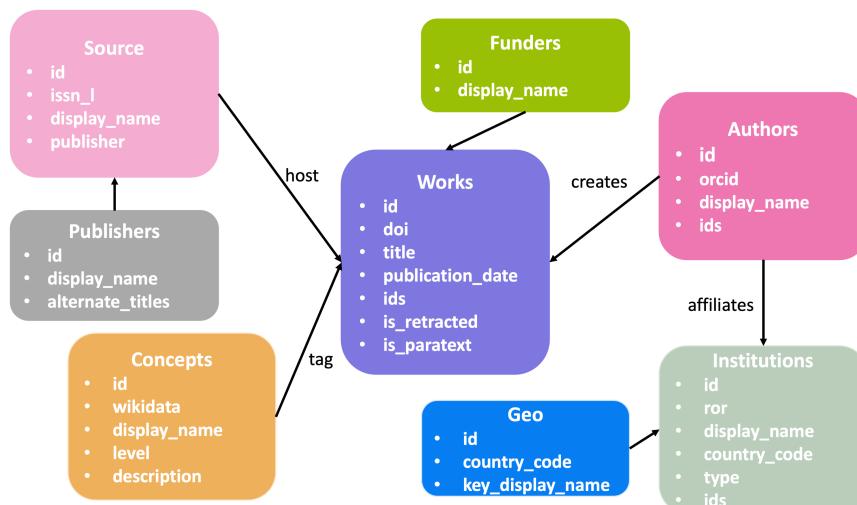


Figure 1: Eight OpenAlex entities and their first few attributes.

- **Works:** academic documents such as journal articles, proceedings, books, and datasets. The Works entity is central in that it ties together the other four entities (Fig. 1). OpenAlex indexes over 200 million works. One can identify a work by its OAID or DOI, a unique alphanumeric string assigned to a digital document, often a research article.
- **Authors:** individuals who create works. Authors create Works, study Concepts, and are affiliated with Institutions. OpenAlex indexes more than 200 million authors. One can identify an author by their OAID or ORCID, a persistent and unique identifier assigned to researchers.
- **Institutions:** universities and organisations affiliated with authors. Institutions are linked to Works via Authors. OpenAlex indexes more than 100,000 institutions. One can identify an institution by its OAID or ROR ID, a persistent identifier for research organizations.
- **sources:** repositories that house works such as journals, conferences, preprint repositories, or institutional repositories. OpenAlex uses a fingerprinting algorithm to match multiple locations a work may be hosted in and flag the version of the record's host as primary. OpenAlex indexes more than 100,000 sources. One can identify a source by its OAID or ISSN-L, *i.e.*, a single ISSN that groups the publication's all possible ISSNs (standardized numeric identifiers assigned to serial publications).
- **Concepts:** topics of works, deduced from their titles and abstracts. To identify a concept, you can use the OAID or its Wikidata ID, a unique identifier assigned to that entity in Wikidata because all OpenAlex concepts are also Wikidata concepts. The concepts follow a hierarchy; there are 19 concepts at the root level (0-level) and 5 layers of descendants. OpenAlex indexes ~ 65,000 concepts.
- **Publishers:** companies and organizations that distribute academic documents. Each publisher publishes multiple journals, so publisher data is aggregate data. OpenAlex indexes about 10,000 publishers.

- **Funders:** public and private companies that fund research. OpenAlex indexes about 30,000 funders. Funder data comes from Crossref, and is enhanced with data from Wikidata and ROR.
- **Geo:** works produced in the country based on the nationality of the institution with which the author is affiliated. In particular, there are some ways to filter and group academic documents by continents and the Global South. OpenAlex uses United Nations data to divide the globe into continents and regions, making it easier to filter the data.

2 Implementation of openalexR

Interacting with the OpenAlex API, **openalexR** provides easy querying and downloading of scholarly metadata as well as converting the output into a classic bibliographic dataframe (Fig. 2), which allows the user to streamline their data collection and downstream analyses (Aria, 2022).

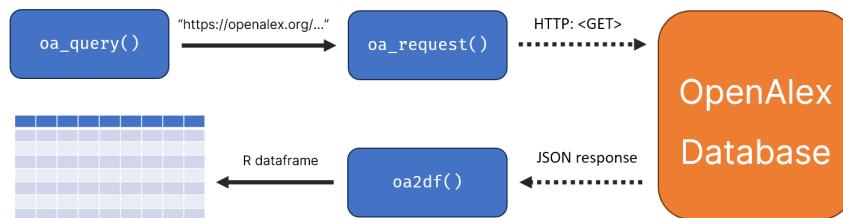


Figure 2: openalexR workflow

With minimal dependencies, **openalexR** lowers the barrier to using the [REST API](#) by simplifying input entry, handling rate limits, and automatically parsing API responses. The package also offers other useful functionality such as snowball search and N-grams of works. **openalexR** is currently listed on the [OpenAlex website](#) as the supported R library for API access. The main function of **openalexR**, `oa_fetch`, is a convenience wrapper for three smaller functions to

1. generate a valid query from a set of arguments provided by the user (`oa_query`),
2. download a collection of entities matching the query (`oa_request`),
3. and convert the list output to a classical bibliographic data frame (`oa2df`).

Specifically, in constructing valid queries following the OpenAlex API syntax, `oa_query` utilizes the `modify_url` function of the **httr** package (Wickham, 2022) to pass the `filter`, `sort`, `search`, and `group_by` parameters specified by the user. Next, `oa_request` sends a request to OpenAlex, downloads the JSON output matching the created query, and returns the result in a nested list. Finally, `oa2df` converts this output list into a classic bibliographic data frame (similar to an Excel sheet) that can be used as input in a bibliometric analysis or scientific mapping (Wais, 2016), e.g., using the **bibliometrix** package (Aria and Cuccurullo, 2017).

In addition to `oa_fetch`, the package **openalexR** includes three other functions specific to certain features: `oa_random`, `oa_snowball`, and `oa_ngrams`. The function `oa_random`, similar to `oa_fetch`, returns a record randomly. This function is particularly useful for casual sampling purposes. For instance, when conducting an analysis of gender bias within the academy, one could utilize this function to randomly query the database.

`oa_snowball` enables the user to perform *snowballing*. Snowballing, or snowball search, is a literature search technique where the researcher starts with a set of articles and finds other articles that cite (forward citations) or were cited by (backward citations) the original set (Wohlin, 2014). Meta-analysis researchers often employ this technique to collect relevant primary studies, where sufficient iterations of snowballing can converge on and exhaust the target literature space (Siddaway et al., 2019). The traditional method of conducting a snowball search involves manual effort. However, computer-assisted snowball search can effectively reduce the time and resources required while maintaining a representative coverage of the target literature (McWeeny et al., 2021, 2022). `oa_snowball` takes a vector of OpenAlex IDs as input and returns a list of 2 elements: nodes and edges. `oa_snowball` locates and retrieves information on articles that cite or are cited by the initial set of articles (nodes) and also the relationships between these articles (edges). Following **tidygraph**'s convention (Pedersen, 2022b), the edges dataframe contains 2 columns of OpenAlex IDs, *from* and *to*. The row *from A to B* means *A cites B*.

The `oa_ngrams` function is used to obtain N-grams from a specific set of works. N-grams are defined as sequences of *n* words that occur within a work and are commonly used in language

modeling and text analysis to identify relationships between words. The use of N-grams allows for the analysis of the frequency and distribution of words within a text or set of texts and is widely employed in fields such as natural language processing, machine translation, and sentiment analysis. Some work entities in OpenAlex include N-grams of their full text, which are obtained from the Internet Archive using the spaCy parser to index academic works. The **openalexR** package offers the capability to extract N-grams of works through the `oa_ngrams` function. This function takes a vector of OpenAlex IDs as input and returns a list of N-grams and their corresponding frequencies.

3 Installation of openalexR

The package is available from the Comprehensive R Archive Network (CRAN) via the command `install.packages("openalexR")`. The current version is v1.2.0. Development versions (latest v1.2.0.9000) are available on GitHub and can be installed using `devtools` (Wickham et al., 2022) or `remotes` (Csárdi et al., 2021).

```
install.packages("devtools")
devtools::install_github("ropensci/openalexR")
or
install.packages("remotes")
remotes::install_github("ropensci/openalexR")
```

Other installation details are available on the GitHub page <https://github.com/ropensci/openalexR>.

4 Polite use

The OpenAlex API doesn't require authentication but requires following polite usage. To get into the polite pool, it is necessary to provide a user e-mail address through the `mailto` parameter in R options

```
options(openalexR.mailto = "example@email.com")
```

in all API requests. The polite pool has much faster and more consistent response times.

5 Examples of use

We show many different examples of typical use cases on the package's `README` and `vignettes`. Examples we show in this manuscript can be found at <https://github.com/trangdata/oarj/blob/main/paper-examples.md>.

First, we demonstrate an example that uses a few different `filters`. We want to describe the use of "*bibliometrics*" approaches in the scientific literature. We first describe how to make the query that allows us to answer this search question. After a brief description of the concept "*bibliometrics*", we identify the scientific literature that has used the concept "*bibliometrics*" in OpenAlex. Finally, focusing on the metadata offered by OpenAlex, we analyse the most relevant sources, authors, institutions, and works on bibliometrics.

5.1 The *bibliometrics* concept

We define the search on the entity "concepts" by filtering the "*bibliometrics*" topic associated with the id [C178315738](#). The function `oa_fetch` generates the query from the set of arguments provided to it, downloads the set of concepts that match the query, and converts the output into a classical bibliographic data frame. Concepts can be queried using concept IDs or by concept name searching.

Searching "*bibliometrics*" concept by name:

```
concept <- oa_fetch(
  entity = "concepts",
  display_name.search = "bibliometrics" # search by concept name "bibliometrics"
)

concept$id
# [1] "https://openalex.org/C178315738"
```

```
cat(concept$description, "\nis a level", concept$level, "concept")
# [1] statistical analysis of written publications, such as books or articles
is a level 2 concept
```

Alternatively, once we know the OAID for a concept, we can search by its ID and get a similar result:

```
concept <- oa_fetch(
  entity = "concepts",
  identifier = "C178315738" # OAID for "bibliometrics"
)

cat(concept$description, "\nis a level", concept$level, "concept")
# [1] statistical analysis of written publications, such as books or articles
is a level 2 concept
```

To describe which concepts are related to the term *bibliometrics*, let's analyze the OpenAlex hierarchy. In OpenAlex each work is tagged with multiple concepts, based on the title, abstract and host source title. A score is available for each concept in a work, demonstrating how well that concept represents the work to which it was assigned. However, when a lower-level descendant concept is assigned, all of its antecedent concepts are also assigned. Since *bibliometrics* is a level 2 concept in OpenAlex, we have detailed information on the concepts related to ancestors (level 0 or 1), peers (level 2), and descendants (level 3).

```
related_concepts <- concept$related_concepts[[1]] |>
  dplyr::mutate(relation = case_when(
    level < 2 ~ "ancestor",
    level == 2 ~ "equal level",
    TRUE ~ "descendant"
  )) |>
  dplyr::arrange(level) |>
  dplyr::relocate(relation) |>
  dplyr::select(-wikidata)

# output in Table 1:
related_concepts
```

We find 4 ancestor, 11 equal-level and 9 descendant concepts of *bibliometrics* (Tab. 1). The resulting hierarchy of *bibliometrics* can enable us to analyse, for example, all equal-level concepts.

```
concept_df <- oa_fetch(
  entity = "concepts",
  identifier = c(concept$id, equal_level$id)
)

concept_df |>
  dplyr::select(display_name, counts_by_year) |>
  tidyverse::unnest(counts_by_year) |>
  dplyr::filter(year < 2022) |>
  ggplot(aes(x = year, y = works_count, color = display_name)) +
  facet_wrap(~display_name) +
  geom_line() +
  ...
```

Visualising all *bibliometrics*-related concepts together, we observe an increasing trend in the sub-fields of *bibliometrics*, *peer review*, *scientific literature* and *scientometrics*. Conversely, there has been a reduction in the number of papers in the subfields of *webometrics*, *knowledge organisation*, *information science*, *collection development*, and *altmetrics*. *PageRank* and *Impact factor*, concepts have remained stable in popularity over the last 10 years. Compared to other topics, *citation* has the highest number of papers over time (Fig. 3).

5.2 Bibliometrics dataset

We download all works, included in OpenAlex, that have the words *bibliometrics* or *science mapping* in the title to map bibliometric approaches in the scientific literature. We set the query using the

relation	id	display_name	level	score
ancestor	C124101348	Data mining	1	
ancestor	C161191863	Library science	1	
ancestor	C136764020	World Wide Web	1	
ancestor	C41008148	Computer science	0	
equal level	C525823164	Scientometrics	2	6.6193560
equal level	C2779455604	Impact factor	2	4.1035270
equal level	C2778407487	Altmetrics	2	2.5396087
equal level	C521491914	Webometrics	2	2.3026270
equal level	C2781083858	Scientific literature	2	1.6163236
equal level	C2778805511	Citation	2	1.6110690
equal level	C95831776	Information science	2	1.5750017
equal level	C2779172887	PageRank	2	1.5363927
equal level	C138368954	Peer review	2	1.4112837
equal level	C2779810430	Knowledge organization	2	1.0037539
equal level	C2780416505	Collection development	2	0.8137859
descendant	C105345328	Citation analysis	3	4.9036117
descendant	C2778793908	Citation impact	3	4.0405297
descendant	C2780378607	Informetrics	3	2.1396947
descendant	C2778032371	Citation index	3	1.8888942
descendant	C83867959	Scopus	3	1.6536747
descendant	C2776822937	Bibliographic coupling	3	1.3375385
descendant	C2779693592	Journal ranking	3	1.1321522
descendant	C45462083	Documentation science	3	0.8473609
descendant	C2777765086	Co-citation	3	0.8002241

Table 1: Concepts related to *bibliometrics*: ancestors, equal-levels, and descendants.

following parameters: entity is "works"; title.search is "bibliometrics|science mapping" and, for the first part, count_only is TRUE so we can see how many records will be returned.

```
oa_fetch(
  entity = "works",
  title.search = "bibliometrics|science mapping",
  count_only = TRUE
)
#      count db_response_time_ms page per_page
# [1,] 26,953           118     1       1

biblio_works <- oa_fetch(
  entity = "works",
  title.search = "bibliometrics|science mapping",
  count_only = FALSE
)
```

Our query returns 26,953 works concerning bibliometrics. By default, the `oa_fetch` converts these records in a tibble object (dataframe) with each row containing information about a work. This dataframe has 28 columns containing important information about a work, such as the publication date, DOI, reference works, and so on. If users wish to convert the original nested list into another object, they can change the parameters in the following way `oa_fetch(..., output = "list")`. From this dataset, we could describe the most relevant sources, authors, and institutions.

5.3 Most relevant sources

We first identify the core journals for the discipline by tallying all bibliometrics-related works for each source and selecting the top 5 sources with the most works.

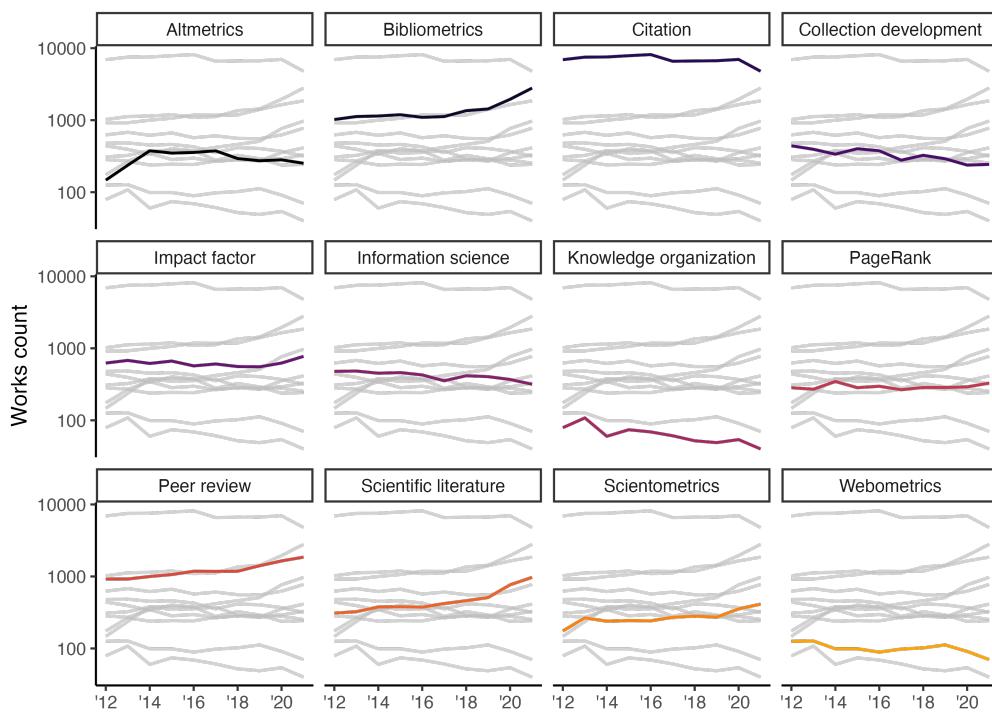


Figure 3: Trends in *bibliometrics*-related topics in the past 10 years.

```

sources <- biblio_works |>
  dplyr::count(so) |>
  tidyrr::drop_na(so) |>
  dplyr::slice_max(n, n = 5) |>
  dplyr::pull(so)

sources
# [1] "Scientometrics"
# [2] "Sustainability"
# [3] "Social Science Research Network"
# [4] "International Journal Environmental Research and Public Health"
# [5] "Environmental Science and Pollution Research"
  
```

Visualising these counts over the years (Fig.4), we observe the field has expanded overall, especially starting around 2015. *Scientometrics* is the oldest journal publishing on bibliometrics and remains the top source for these articles. Other journals started to publish these works in 2015 and have maintained some volume in this field. *Sustainability* only started publishing in this field in 2017 but has rapidly increased its number of publications since. In 2021, it was the second source to have published the most bibliometrics articles, after *Scientometrics*.

5.4 Most relevant authors and institutions

Secondly, we identify the authors and institutions most relevant to the discipline by extracting the list of author and institution-related metadata from the collection, tallying all bibliometrics-related works for each author, and selecting the top 10 authors who write the most articles and 10 institutions with the most publications in the field.

```

biblio_authors_raw <- do.call(rbind.data.frame, biblio_works$author)
biblio_insts <- biblio_authors_raw |>
  dplyr::count(institution_display_name) |>
  dplyr::rename("name" = institution_display_name) |>
  tidyrr::drop_na(name) |>
  dplyr::slice_max(n, n = 10) |>
  dplyr::mutate(type = "Institution")
  
```

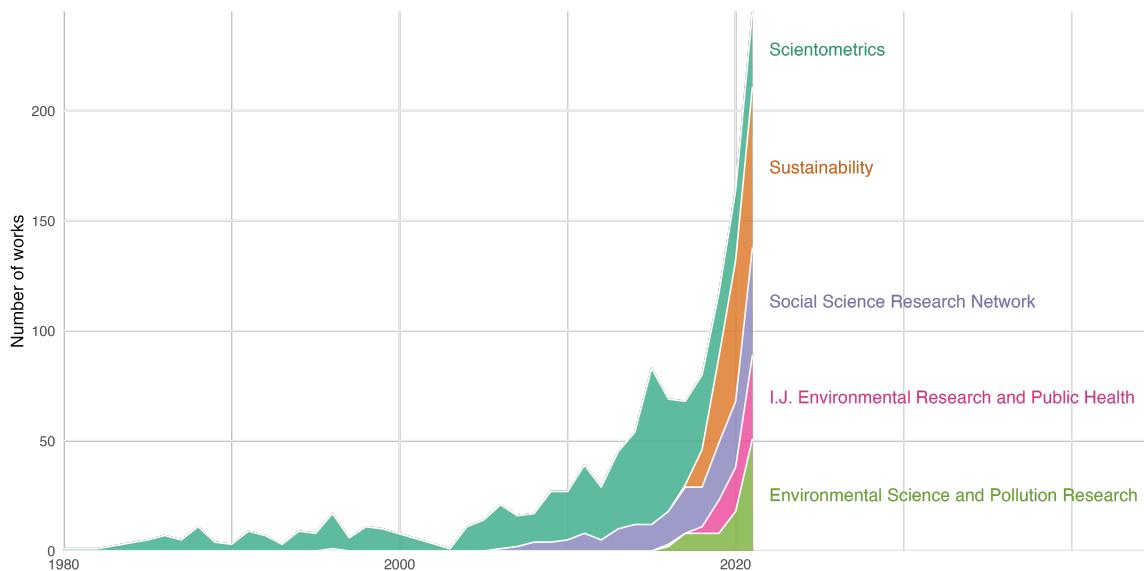


Figure 4: Number of *bibliometrics* articles by journal over the years.

```

biblio_authors <- biblio_authors_raw |>
  dplyr::count(au_display_name) |>
  dplyr::rename("name" = au_display_name) |>
  tidyrr::drop_na(name) |>
  dplyr::slice_max(n, n = 10) |>
  dplyr::mutate(type = "Author")
  
```

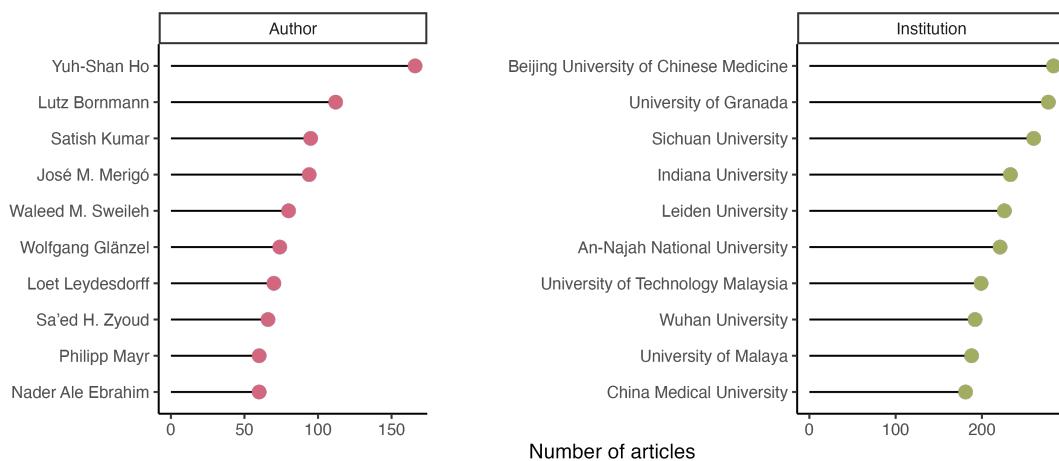


Figure 5: Most relevant authors and institutions.

Among the 84,335 authors in our collection, Yuh-Shan Ho appears to have published the most bibliometrics articles. He has over 150 bibliometric papers in his career. Another relevant author in our collection is Lutz Bornmann with more than 100 bibliometrics papers. The other authors in the top 10 all have between 50 and 100 papers (Fig. 5).

With regard to institutions, we observe that the concept *bibliometrics* is widely developed by different centres of expertise. At the top of the ranking, we see Beijing University of Chinese Medicine (China) and University of Granada (Spain) with over 250 articles. Other relevant institutions in our dataset include Sichuan University (China), Indiana University (US), Leiden University (Netherlands), An-Najah National University (Palestine) with more than 200 bibliometric papers. Other institutions in the top 10 have between 150 and 200 papers (Fig. 5).

5.5 Most relevant works

We identify the most relevant papers for the discipline by tallying all citations of articles related to bibliometrics and selecting the top 10 most cited articles (Tab. 2). We find ‘Software survey: VOSviewer, a computer programme for bibliometric mapping’ at the top with 4805 citations. Other relevant works in our collection are ‘Comparison of PubMed, Scopus, Web of Science, and Google Scholar: strengths and weaknesses’ with 1996 citations and ‘bibliometrix: An R-tool for comprehensive science mapping analysis’ with 1800 citations. The other papers in the top 10 all have between 950 and 1500 citations.

```
seminal_works <- slice_max(biblio_works, cited_by_count, n = 10)

# output in Table 2:
seminal_works |>
  dplyr::select(publication_date, display_name, so, cited_by_count)
```

Year	Article	source	Cited
2010	Software survey: VOSviewer, a computer program for bibliometric mapping	Scientometrics	5364
2017	bibliometrix : An R-tool for comprehensive science mapping analysis	Journal of Informetrics	2127
1976	A general theory of bibliometric and other cumulative advantage processes	Journal of the American Society for Information Science	1505
2015	Bibliometric Methods in Management and Organization	Organizational Research Methods	1488
2015	Bibliometrics: The Leiden Manifesto for research metrics	Nature	1168
2011	Science mapping software tools: Review, analysis, and cooperative study among tools	Journal of the Association for Information Science and Technology	1100
2004	Changes in the intellectual structure of strategic management research: a bibliometric study of the Strategic Management Journal, 1980–2000	Strategic Management Journal	1038
2010	A unified approach to mapping and clustering of bibliometric networks	Journal of Informetrics	922
2015	Green supply chain management: A review and bibliometric analysis	International Journal of Production Economics	920
2006	Forecasting emerging technologies: Use of bibliometrics and patent analysis	Technological Forecasting and Social Change	804

Table 2: Most relevant works

5.6 Snowball search

We perform snowballing with `oa_snowball` to identify the set of articles that cite and are cited by the two seminal works associated with the concept of bibliometrics: Software survey: VOSviewer, a computer programme for bibliometric mapping (W2150220236), bibliometrix : An R-tool for comprehensive science mapping analysis (W2755950973). We insert these OAIDs as identifiers in `oa_snowball`, use the filter on the citations obtaining only those related to 2022, then use `tidygraph` (Pedersen, 2022b) and `ggraph` (Pedersen, 2022a) to display this citation network (Fig. 6). `oa_snowball` returns a list of 2 elements: nodes and edges. The first have information about the work, while edges have the start and end points of the links. This list output from `oa_snowball` can be used directly as input to standard graph functions such as `tidygraph::as_tbl_graph` for further network analyses and visualisations in co-citation analysis, historiograph analysis, etc.

```
sb_docs <- oa_snowball(
  identifier = c("W2150220236", "W2755950973"),
  citing_filter = list(from_publication_date = "2022-01-01")
)

# Reduced output
print(sb_docs)
$nodes
# A tibble: 5,769 × 37
   id      display_name
   <chr>    <chr>
 1 W2150220236 Software survey: VOSviewer, a computer program for bibliometric ...
 2 W2755950973 bibliometrix : An R-tool for comprehensive science mapping analysis
 3 W4306178549 Literature reviews as independent studies: guidelines for academic ...
```

```

4 W4320070415 Is Metaverse in education a blessing or a curse: a combined content ...
# i 5,765 more rows
# i 35 more variables
$edges
# A tibble: 6,444 × 2
  from      to
  <chr>    <chr>
1 W4306178549 W2755950973
2 W4320070415 W2150220236
3 W3203542139 W2150220236
4 W4283392904 W2150220236
# i 6,440 more rows

# Conversion to a `tbl_graph` object for network analysis and visualization
sb_docs_graph <- tidygraph::as_tbl_graph(sb_docs)

```

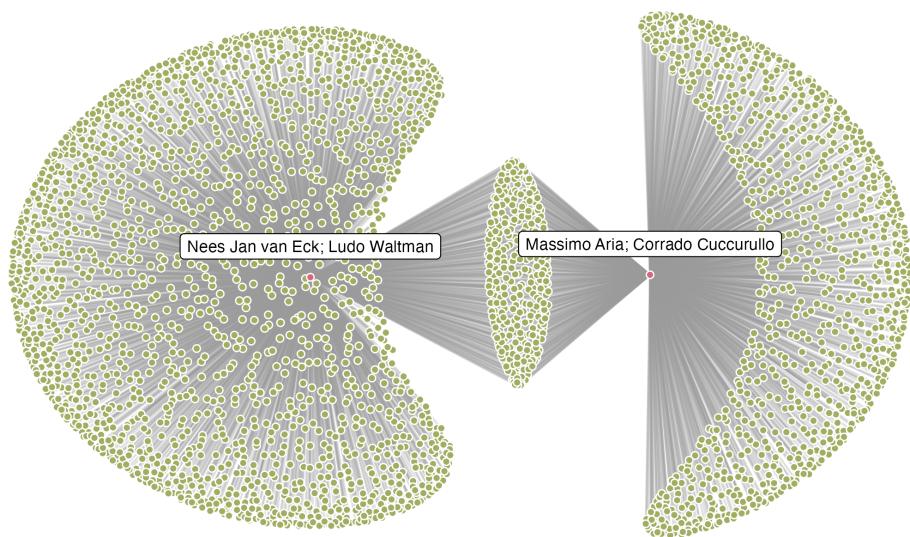


Figure 6: Two seminal works and their citations and references, from oa_snowball output.

oa_snowball finds and returns the metadata on the two seminal works in our dataset, and also information on the articles that cite and are cited by them, in 2022. We have a total of 2,907 citing articles: 2,078 articles cite van Eck et al., 1,161 articles cite Aria and Cuccurullo. In addition, as we can see from the graph, there are 332 articles citing both van Eck et al. and Aria and Cuccurullo, simultaneously.

5.7 N-grams

Finally, we obtain the N-grams of all the bibliometric works that make up our collection. N-grams are groups of words that occur in the full text of a work. To extract n-grams we use the oa_ngrams function from the **openalexR** package. From this list we then extract only the bigrams because we believe they can be more informative.

```

ngrams_data <- oa_ngrams(sample(biblio_works$id, 1000), verbose = TRUE)
top_10 <- do.call(rbind.data.frame, ngrams_data$ngrams) |>
  dplyr::filter(ngram_tokens == 2, nchar(ngram) > 10) |>
  dplyr::arrange(desc(ngram_count)) |>
  dplyr::slice_max(ngram_count, n = 10, with_ties = FALSE)

```

As can be seen from the graph of the 10 most frequent bi-grams, the papers are very much focused on the use of advanced technologies to improve efficiency and sustainability in various fields, such

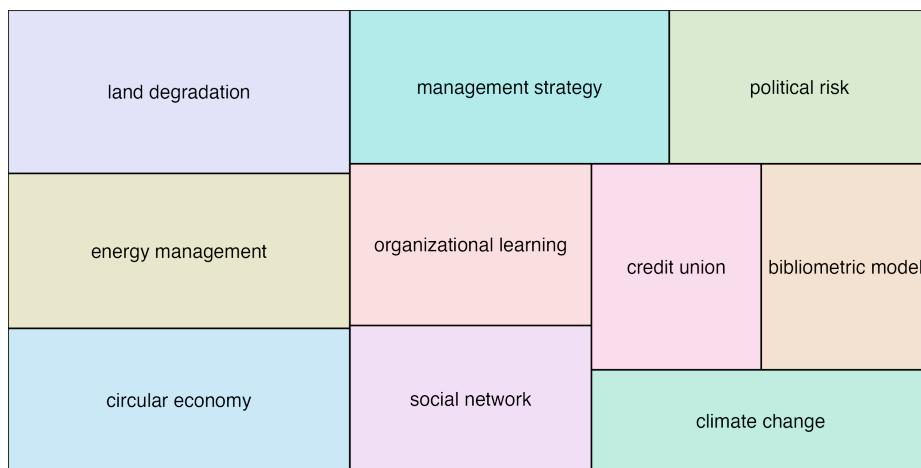


Figure 7: TreeMap of the top 10 bi-grams of *bibliometrics* articles.

as *circular economy*, *climate change*, *management strategy*, and *energy management* (Fig. 7). In addition, there are themes related to *political risk*, *organizational learning*, and cooperation within organizations and *social networks*. In general, these topics suggest the need for greater attention to environmental and social issues in business management and risk prevention. Finally, *bibliometric model* (scientific performance evaluation) and *credit union* (financial regulation) are interrelated to bibliometrics, which requires a global vision and cooperation among different stakeholders to be effectively applied.

6 Summary

openalexR is a new package that facilitates querying, collecting, and downloading bibliographic metadata of OpenAlex entities through the provided REST APIs. It is available on CRAN at <https://cran.r-project.org/package=openalexR>. **openalexR** helps streamline the researcher's workflow in accessing, collecting, and wrangling OpenAlex data. Extensive documentation, comprehensive tests of the package's internal functions, and common use cases are provided, sufficiently covering the current OpenAlex API. The source code and development versions are available at <https://github.com/ropensci/openalexR>. The current version of the package is a stable version, and there are no plans for any breaking changes soon. Of course, **openalexR** will continue to be actively maintained to keep up with CRAN policies and distribute any bug fixes. Bug reports, help requests, or improvement suggestions are welcome in the package software repository. For more information on **openalexR**, vignettes are available at <https://ropensci.github.io/openalexR/articles/>.

7 Acknowledgements

8 Supplementary material

Examples we show in this manuscript can be found at <https://github.com/trangdata/oarj/blob/main/paper-examples.md>.

References

- W. Ammar, D. Groeneveld, C. Bhagavatula, I. Beltagy, M. Crawford, D. Downey, J. Dunkelberger, A. Elgohary, S. Feldman, V. Ha, et al. Construction of the literature graph in semantic scholar. *arXiv preprint arXiv:1805.02262*, 2018. [p166]
- M. Aria. *openalexR: Getting Bibliographic Records from 'OpenAlex' Database Using 'DSL' API*, 2022. <https://github.com/massimoaria/openalexR>, <https://massimoaria.github.io/openalexR/>. [p168]
- M. Aria and C. Cuccurullo. bibliometrix: An r-tool for comprehensive science mapping analysis. *Journal of informetrics*, 11(4):959–975, 2017. [p168]
- A. Belfiore, A. Salatino, and F. Osborne. Characterising research areas in the field of ai. *arXiv preprint arXiv:2205.13471*, 2022. [p167]

- D. S. Chawla. Unpaywall finds free versions of paywalled papers. *Nature*, 2017. [p167]
- C. Chen. Science mapping: A systematic review of the literature. *Journal of Data and Information Science*, 2(2):1–40, 2017. doi: doi:10.1515/jdis-2017-0006. URL <https://doi.org/10.1515/jdis-2017-0006>. [p166]
- G. Csárdi, J. Hester, H. Wickham, W. Chang, M. Morgan, and D. Tenenbaum. *remotes: R Package Installation from Remote Repositories, Including 'GitHub'*, 2021. URL <https://CRAN.R-project.org/package=remotes>. R package version 2.4.2. [p169]
- T. Dallas, A.-L. Gehman, and M. J. Farrell. Variable bibliographic database access could limit reproducibility. *BioScience*, 68(8):552–553, 2018. [p166]
- C. Du, J. Cohoon, J. Priem, H. Piwowar, C. Meyer, and J. Howison. Citeas: Better software through sociotechnical change for better software citation. *Companion Publication of the 2021 Conference on Computer Supported Cooperative Work and Social Computing*, 2021. [p167]
- G. Hendricks, D. Tkaczyk, J. Lin, and P. Feeney. Crossref: The sustainable source of community-owned scholarly metadata. *Quantitative Science Studies*, 1(1):414–427, 2020. [p166]
- C. Herzog, D. Hook, and S. Konkiel. Dimensions: Bringing down barriers between scientometricians and data. *Quantitative Science Studies*, 1(1):387–395, 2020. [p166]
- D. Hicks, P. Wouters, L. Waltman, S. De Rijcke, and I. Rafols. Bibliometrics: the leiden manifesto for research metrics. *Nature*, 520(7548):429–431, 2015. [p166]
- D. W. Hook, S. J. Porter, and C. Herzog. Dimensions: building context for search and evaluation. *Frontiers in Research Metrics and Analytics*, 3:23, 2018. [p166]
- P. Kulkanjanapiban and T. Silwattananusarn. Comparative analysis of dimensions and scopus bibliographic data sources: an approach to university research productivity. *International Journal of Electrical & Computer Engineering* (2088-8708), 12(1), 2022. [p166]
- A. Martín-Martín, M. Thelwall, E. Orduna-Malea, and E. Delgado López-Cózar. Google scholar, microsoft academic, scopus, dimensions, web of science, and opencitations' coc: a multidisciplinary comparison of coverage via citations. *Scientometrics*, 126(1):871–906, 2021. [p166]
- S. McWeeny, J. Choe, and E. S. Norton. *SnowGlobe: An Iterative Search Tool for Systematic Reviews and Meta-Analyses*, 2021. [p168]
- S. McWeeny, S. Choi, J. Choe, A. LaTourrette, M. Y. Roberts, and E. S. Norton. Rapid automatized naming (ran) as a kindergarten predictor of future reading in english: A systematic review and meta-analysis. *Reading Research Quarterly*, 57(4):1187–1211, 2022. doi: <https://doi.org/10.1002/rrq.467>. URL <https://ila.onlinelibrary.wiley.com/doi/abs/10.1002/rrq.467>. [p168]
- T. L. Pedersen. *ggraph: An Implementation of Grammar of Graphics for Graphs and Networks*, 2022a. URL <https://CRAN.R-project.org/package=ggraph>. R package version 2.1.0. [p174]
- T. L. Pedersen. *tidygraph: A Tidy API for Graph Manipulation*, 2022b. URL <https://CRAN.R-project.org/package=tidygraph>. R package version 1.2.2. [p168, 174]
- J. Priem, D. Taraborelli, P. Groth, and C. Neylon. Altmetrics: A manifesto. 2011. [p166]
- J. Priem, H. Piwowar, and R. Orr. Openalex: A fully-open index of scholarly works, authors, venues, institutions, and concepts. *arXiv preprint arXiv:2205.01833*, 2022. URL <http://altmetrics.org/manifesto>. [p167]
- A. P. Siddaway, A. M. Wood, and L. V. Hedges. How to do a systematic review: A best practice guide for conducting and reporting narrative reviews, meta-analyses, and meta-syntheses. *Annual Review of Psychology*, 70(1):747–770, 2019. doi: 10.1146/annurev-psych-010418-102803. [p168]
- V. K. Singh, P. Singh, M. Karmakar, J. Leta, and P. Mayr. The journal coverage of web of science, scopus and dimensions: A comparative analysis. *Scientometrics*, 126(6):5113–5142, 2021. [p166]
- A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-J. Hsu, and K. Wang. An overview of microsoft academic service (mas) and applications. In *Proceedings of the 24th international conference on world wide web*, pages 243–246, 2015. [p166]
- N. J. Van Eck, L. Waltman, V. Larivière, and C. Sugimoto. Crossref as a new source of citation data: A comparison with web of science and scopus. *CWTS Blog*, 17, 2018. [p166]

- R. Van Noorden. Scientists join journal editors to fight impact-factor abuse. *Nature News Blog*, 16, 2013. [p166]
- M. Visser, N. J. van Eck, and L. Waltman. Large-scale comparison of bibliographic data sources: Scopus, web of science, dimensions, crossref, and microsoft academic. *Quantitative Science Studies*, 2 (1):20–41, 2021. [p166]
- K. Wais. Gender prediction methods based on first names with genderizer. *R J.*, 8(1):17, 2016. [p168]
- L. Waltman and V. Larivière. Special issue on bibliographic data sources, 2020. [p166]
- K. Wang, Z. Shen, C. Huang, C.-H. Wu, Y. Dong, and A. Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413, 2020. [p166]
- K. Wang et al. A review of microsoft academic services for science of science studies. *front. big data* 2, 45 (2019), 2019. [p166]
- S. B. Wanyama, R. W. McQuaid, and M. Kittler. Where you search determines what you find: the effects of bibliographic databases on systematic reviews. *International Journal of Social Research Methodology*, 25(3):409–422, 2022. [p166]
- H. Wickham. *httr: Tools for Working with URLs and HTTP*, 2022. URL <https://CRAN.R-project.org/package=httr>. R package version 1.4.4. [p168]
- 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. [p169]
- D. J. Winter. rentrez: An r package for the ncbi eutils api. Technical report, PeerJ Preprints, 2017. [p166]
- C. Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, EASE '14, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450324762. doi: 10.1145/2601248.2601268. URL <https://doi.org/10.1145/2601248.2601268>. [p168]

Massimo Aria

*Università degli Studi di Napoli Federico II
K-Synth srl, Academic Spin-off
Department of Economics and Statistics
Napoli, NA 80126
Italy
(0000-0002-8517-9411)
aria@unina.it*

Trang Le

*Bristol Myers Squibb
Cambridge, MA 02143
USA
(0000-0003-3737-6565)
trang.le@bms.com*

Corrado Cuccurullo

*Università della Campania Luigi Vanvitelli
Capua, CE 81043
Italy
Università degli Studi di Napoli Federico II
K-Synth srl, Academic Spin-off
Department of Economics and Statistics
Napoli, NA 80126
Italy
(0000-0002-7401-8575)
corrado.cuccurullo@unicampania.it*

Alessandra Belfiore

Università degli Studi di Napoli Federico II

*K-Synth srl, Academic Spin-off
Department of Economics and Statistics
Napoli, NA 80126
Italy
(0000-0003-3709-9481)
alessandra.belfiore@unina.it*

*June Choe
University of Pennsylvania
Philadelphia, PA 19104
USA
(0000-0002-0701-921X)
yjchoe@sas.upenn.edu*

Computer Algebra in R Bridges a Gap Between Symbolic Mathematics and Data in the Teaching of Statistics and Data Science

by Mikkel Meyer Andersen and Søren Højsgaard

Abstract The capability of R to do symbolic mathematics is enhanced by the `reticulate` and `caracas` packages. The workhorse behind these packages is the Python computer algebra library SymPy. Via `reticulate`, the SymPy library can be accessed from within R. This, however, requires some knowledge of SymPy, Python and `reticulate`. The `caracas` package, on the other hand, provides access to SymPy (via `reticulate`) but by using R syntax, and this is the main contribution of `caracas`. We show examples of how to use the SymPy library from R via `reticulate` and `caracas`. Using `caracas`, we demonstrate how mathematics and statistics can benefit from bridging computer algebra and data via R. The `caracas` package integrates well with `Rmarkdown` and `Quarto`, and as such supports creation of teaching material and scientific reports. As inspiration for teachers, we include ideas for small student projects.

1 Introduction

The capability of R to do symbolic mathematics is enhanced by the `reticulate` (Ushey, Allaire, and Tang 2020) and `caracas` (Andersen and Højsgaard 2021) packages. The `reticulate` package allows R users to make use of various Python libraries, such as the symbolic mathematics package SymPy, which is the workhorse behind symbolic mathematics in this connection. However, the `reticulate` package does require that the users are somewhat familiar with Python syntax. The `caracas` package, on the other hand, provides an interface to `reticulate` that conforms fully to the existing R syntax. In short form, `caracas` provides the following:

- (1) Mathematical tools like equation solving, summation, limits, symbolic linear algebra in R syntax and formatting of tex output.
- (2) Symbolic mathematics can easily be combined with data which is helpful in e.g. numerical optimization.

In this paper we will illustrate the use of the `caracas` package (version 2.1.0) in connection with teaching mathematics and statistics and how students can benefit from bridging computer algebra and data via R. Focus is on: 1) treating statistical models symbolically, 2) bridging the gap between symbolic mathematics and numerical computations and 3) preparing teaching material in a reproducible framework (provided by, e.g. `rmarkdown` and Quarto; J. Allaire et al. (2021); Xie, Allaire, and Grolemund (2018); Xie, Dervieux, and Riederer (2020); J. J. Allaire et al. (2022)).

The `caracas` package is available from CRAN. Several vignettes illustrating `caracas` are provided with the package and they are also available online together with the help pages, see <https://r-cas.github.io/caracas/>. The development version of `caracas` is available at <https://github.com/r-cas/caracas>.

The paper is organized in the following sections: The section `Introducing caracas` briefly introduces the `caracas` package and its syntax, and relates `caracas` to SymPy via `reticulate`. The section `Statistics examples` presents a sample of statistical models where we believe that a symbolic treatment can enhance purely numerical computations. In the section `Further topics` we demonstrate further aspects of `caracas`, including how `caracas` can be used in connection with preparing texts, e.g. teaching material and working documents. The section `Hands-on activities` contains suggestions about hands-on activities, e.g. for students. The last section `Discussion` contains a discussion of the paper.

1.1 Installation

The `caracas` package is available on CRAN and can be installed as usual with `install.packages('caracas')`. Please ensure that you have SymPy installed, or else install it:

```
if (!caracas::has_sympy()) {
```

```
caracas::install_sympy()
}
```

The caracas package relies on the `reticulate` package to run Python code. Thus, if you wish to configure your Python environment, you need to first load `reticulate`, then configure the Python environment, and at last load `caracas`. The Python environment can be configured as in `reticulate`'s "Python Version Configuration" vignette. Again, configuring the Python environment needs to be done before loading `caracas`. Please find further details in `reticulate`'s documentation.

2 Introducing `caracas`

Here we introduce key concepts and show functionality subsequently needed in the section [Statistics examples](#). We will demonstrate both `caracas` and contrast this with using `reticulate` directly.

2.1 Symbols

A `caracas` symbol is a list with a `pyobj` slot and the class `caracas_symbol`. The `pyobj` is a Python object (often a SymPy object). As such, a `caracas` symbol (in R) provides a handle to a Python object. In the design of `caracas` we have tried to make this distinction something the user should not be concerned with, but it is worthwhile being aware of the distinction. Whenever we refer to a symbol we mean a `caracas` symbol. Two functions that create symbols are `def_sym()` and `as_sym()`; these and other functions that create symbols will be illustrated below.

2.2 Linear algebra

We create a symbolic matrix (a `caracas` symbol) from an R object and a symbolic vector (a `caracas` symbol) directly. A vector is a one-column matrix which is printed as its transpose to save space. Matrix products are computed using the `%*%` operator:

```
R> M0 <- toeplitz(c("a", "b")) # Character matrix
R> M <- as_sym(M0)           # as_sym() converts to a caracas symbol
R> v <- vector_sym(2, "v")   # vector_sym creates symbolic vector
R> y <- M %*% v
R> Minv <- solve(M)
R> w <- Minv %*% y |> simplify()
```

Here we make use of the fact that `caracas` is tightly integrated with R which has a `toeplitz()` function that can be used. Similarly, `caracas` offers `matrix_sym()` and `vector_sym()` for generating general matrix and vector objects. The object `M` is

```
R> M
#> c: [[a, b],
#>       [b, a]]
```

The LaTeX rendering using the `tex()` function of the symbols above are (refer to section [Further topics](#)):

$$M = \begin{bmatrix} a & b \\ b & a \end{bmatrix}; v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}; y = \begin{bmatrix} av_1 + bv_2 \\ av_2 + bv_1 \end{bmatrix}; M^{-1} = \begin{bmatrix} \frac{a}{a^2-b^2} & -\frac{b}{a^2-b^2} \\ -\frac{b}{a^2-b^2} & \frac{a}{a^2-b^2} \end{bmatrix}; w = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}. \quad (1)$$

Symbols can be substituted with other symbols or with numerical values using `subs()`.

```
R> M2 <- subs(M, "b", "a^2")
R> M3 <- subs(M2, "a", 2)
```

$$M2 = \begin{bmatrix} a & a^2 \\ a^2 & a \end{bmatrix}; M3 = \begin{bmatrix} 2 & 4 \\ 4 & 2 \end{bmatrix}. \quad (2)$$

2.3 Linear algebra - using reticulate

The reticulate package already enables SymPy from within R, but does not use standard R syntax for many operations (e.g. matrix multiplication), and certain operations are more complicated than the R counterparts (e.g. replacing elements in a matrix and constructing R expressions). As illustration, the previous linear algebra example can also be done using reticulate:

```
R> library(reticulate)
R> sympy <- import("sympy")
R> M_ <- sympy$Matrix(list(c("a_", "b_"), c("b_", "a_")))
R> v_ <- sympy$Matrix(list("v1_", "v2_"))
R> y_ <- M_* v_
R> w_ <- M_$inv() * y_
R> sympy$simplify(w_)

#> Matrix([
#> [v1_],
#> [v2_]])
```

This shows that it is possible to do the same linear algebra example using only reticulate, but it requires using non-standard R syntax (for example, using `*` for matrix multiplication instead of `%*`).

2.4 Functionality and R syntax provided by caracas

In caracas we use R syntax:

```
R> rbind(v, v)
R> cbind(v, v)
R> c(v, v)
R> v[3] <- "v3" # Insert element
R> M[, 2]
R> M[2]
```

The code correspondence between reticulate and caracas shows that the same can be achieved with reticulate. However, it can be argued that the syntax is more involved, at least for users only familiar with R. Note in particular that Python's "object-oriented" syntax can make code harder to read due to having to call methods with `:`:

```
R> v_$row_join(v_)                                # rbind(v, v)
R> v_-$T$col_join(v_-$T)                         # cbind(v, v)
R> sympy$Matrix(c(v_.$tolist(), v_.$tolist()))    # c(v, v)
R> sympy$Matrix(c(v_.$tolist(), list(list(sympy$symbols("v3_"))))) # v[3] <- "v3"
R> M_$col(1L)                                     # M[, 2]
R> M_$row(1L)$col(0L)                            # M[2]
```

Notice that SymPy uses 0-based indexing (as Python does), whereas caracas uses 1-based indexing (as R does). Furthermore, indexing has to be done using explicit integers so above we write `1L` (an integer) rather than simply `1` (a numeric).

We have already shown that caracas can coerce R matrices to symbols. Additionally, caracas provides various convenience functions:

```
R> M <- matrix_sym(2, 2, entry = "sigma")
R> D <- matrix_sym_diag(2, entry = "d")
R> S <- matrix_sym_symmetric(2, entry = "s")
R> E <- eye_sym(2, 2)
R> J <- ones_sym(2, 2)
R> b <- vector_sym(2, entry = "b")
```

$$M = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix}; D = \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix}; S = \begin{bmatrix} s_{11} & s_{21} \\ s_{21} & s_{22} \end{bmatrix}; E = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; J = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}; b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \quad (3)$$

A caracas symbol can be turned into an R function for subsequent numerical evaluation using `as_func()` or into an R expression using `as_expr()`:

```
R> as_func(M)

#> function (sigma11, sigma12, sigma21, sigma22)
#> {
#>   matrix(c(sigma11, sigma21, sigma12, sigma22), nrow = 2)
#> }
#> <environment: 0x128ad8ea0>

R> as_expr(M)

#> expression(matrix(c(sigma11, sigma21, sigma12, sigma22), nrow = 2))
```

2.5 Algebra and calculus

We can define a polynomial p in the variable x . This is done by defining a caracas symbol x and subsequently a caracas polynomial p in x (notice that p gets automatically coerced into a symbol as well, because p is defined in terms of the symbol x):

```
R> def_sym(x)
R> p <- 1 - x^2 + x^3 + x^4/4 - 3 * x^5 / 5 + x^6 / 6
```

The function `def_sym()` creates the symbol x . Alternatively, `x <- as_sym("x")` can be used, but it has the drawback that you could also write `y <- as_sym("x")`. We investigate p further by finding the first and second derivatives of p , i.e. the gradient and Hessian of p .

```
R> g <- der(p, x)
R> g2 <- factor_(g)
R> h <- der2(p, x)
```

Notice here that some functions have a postfix underscore as a simple way of distinguishing them from R functions with a different meaning. Thus, here the function `factor_()` factorizes the polynomial which shows that the stationary points are $-1, 0, 1$ and 2 :

$$g = x^5 - 3x^4 + x^3 + 3x^2 - 2x; \quad g2 = x(x - 2)(x - 1)^2(x + 1). \quad (4)$$

In a more general setting we can find the stationary points by equating the gradient to zero: The output `sol` is a list of solutions in which each solution is a list of caracas symbols.

```
R> sol <- solve_sys(lhs = g, rhs = 0, vars = x)
R> sol

#> x = -1
#> x = 0
#> x = 1
#> x = 2
```

Notice that `solve_sys` also works with complex solutions:

```
R> solve_sys(lhs = x^2 + 1, rhs = 0, vars = x)

#> x = -1i
#> x = 1i
```

As noted before, a caracas symbol can be coerced to an R expression using `as_expr()`. This can be used to get the roots of g (the stationary points) above as an R object. The sign of the second derivative in the stationary points can be obtained by coercing the second derivative symbol to a function:

```
R> sol_expr <- as_expr(sol) |> unlist() |> unname()
R> sol_expr

#> [1] -1  0  1  2
```

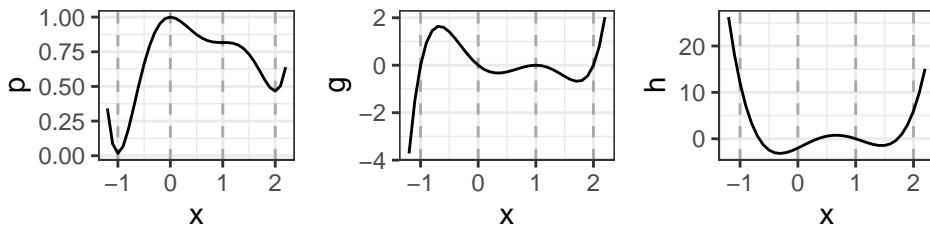


Figure 1: Left: A polynomial. Center: First derivative (the gradient). Right: Second derivative (the Hessian).

```
R> h_fn <- as_func(h)
R> h_fn(sol_expr)
```

```
#> [1] 12 -2  0  6
```

The sign of the second derivative in the stationary points shows that -1 and 2 are local minima, 0 is a local maximum and 1 is an inflection point. The polynomial, the first derivative and the second derivative are shown in Fig. 1. The stationary points, $-1, 0, 1, 2$, are indicated in the plots.

3 Statistics examples

In this section we examine larger statistical examples and demonstrate how caracas can help improve understanding of the models.

3.1 Example: Linear models

While the matrix form of linear models is quite clear and concise, it can also be argued that matrix algebra obscures what is being computed. Numerical examples are useful for some aspects of the computations but not for others. In this respect symbolic computations can be enlightening.

Consider a two-way analysis of variance (ANOVA) with one observation per group, see Table 1.

Table 1: Two-by-two layout of data.

y_{11}	y_{12}
y_{21}	y_{22}

Previously, it was demonstrated that a symbolic vector could be defined with the `vector_sym()` function. Another way to specify a symbolic vector with explicit elements is by using `as_sym()`:

```
R> y <- as_sym(c("y_11", "y_21", "y_12", "y_22"))
R> dat <- expand.grid(r = factor(1:2), s = factor(1:2))
R> X <- model.matrix(~ r + s, data = dat) |> as_sym()
R> b <- vector_sym(ncol(X), "b")
R> mu <- X %*% b
```

For the specific model we have random variables $y = (y_{ij})$. All y_{ij} s are assumed independent and $y_{ij} \sim N(\mu_{ij}, v)$. The corresponding mean vector μ has the form given below:

$$y = \begin{bmatrix} y_{11} \\ y_{21} \\ y_{12} \\ y_{22} \end{bmatrix}; \quad X = \begin{bmatrix} 1 & . & . \\ 1 & 1 & . \\ 1 & . & 1 \\ 1 & 1 & 1 \end{bmatrix}; \quad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}; \quad \mu = Xb = \begin{bmatrix} b_1 \\ b_1 + b_2 \\ b_1 + b_3 \\ b_1 + b_2 + b_3 \end{bmatrix}. \quad (5)$$

Above and elsewhere, dots represent zero. The least squares estimate of b is the vector \hat{b} that minimizes $\|y - Xb\|^2$ which leads to the normal equations $(X^\top X)b = X^\top y$ to be solved. If X has full rank, the unique solution to the normal equations is $\hat{b} = (X^\top X)^{-1}X^\top y$. Hence the estimated mean vector is $\hat{\mu} = X\hat{b} = X(X^\top X)^{-1}X^\top y$. Symbolic computations are not needed for quantities involving only the model matrix X , but when it comes to computations involving y , a symbolic treatment of y is useful:

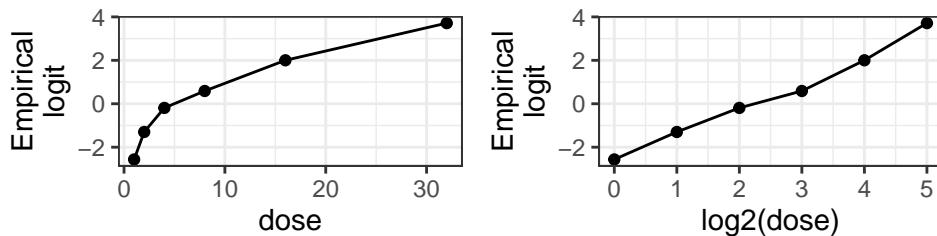


Figure 2: Insecticide mortality of the moth tobacco budworm.

```
R> Xty <- t(X) %*% y
R> b_hat <- solve(t(X) %*% X, Xty)
```

$$X^\top y = \begin{bmatrix} y_{11} + y_{12} + y_{21} + y_{22} \\ y_{21} + y_{22} \\ y_{12} + y_{22} \end{bmatrix}; \quad \hat{b} = \frac{1}{2} \begin{bmatrix} \frac{3y_{11}}{2} + \frac{y_{12}}{2} + \frac{y_{21}}{2} - \frac{y_{22}}{2} \\ -y_{11} - y_{12} + y_{21} + y_{22} \\ -y_{11} + y_{12} - y_{21} + y_{22} \end{bmatrix}. \quad (6)$$

Hence $X^\top y$ (a sufficient reduction of data if the variance is known) consists of the sum of all observations, the sum of observations in the second row and the sum of observations in the second column. For \hat{b} , the second component is, apart from a scaling, the sum of the second row minus the sum of the first row. Likewise, the third component is the sum of the second column minus the sum of the first column. Hence, for example the second component of \hat{b} is the difference in mean between the first and second column in Table 1.

3.2 Example: Logistic regression

In the following we go through details of the logistic regression model, for a classical description see e.g. McCullagh and Nelder (1989) for a classical description.

As an example, consider the budworm data from the `doBy` package (Højsgaard and Halekoh 2023). The data shows the number of killed moth tobacco budworm *Heliothis virescens*. Batches of 20 moths of each sex were exposed for three days to the pyrethroid and the number in each batch that were dead or knocked down was recorded. Below we focus only on male budworms and the mortality is illustrated in Figure 2 (produced with `ggplot2`; Wickham (2016)). The y -axis shows the empirical logits, i.e. $\log((\text{ndead} + 0.5)/(\text{ntotal} - \text{ndead} + 0.5))$. The figure suggests that log odds of dying grows linearly with log dose.

```
R> data(budworm, package = "doBy")
R> bud <- subset(budworm, sex == "male")
R> bud

#>   sex dose ndead ntotal
#> 1 male    1     1    20
#> 2 male    2     4    20
#> 3 male    4     9    20
#> 4 male    8    13    20
#> 5 male   16    18    20
#> 6 male   32    20    20
```

Observables are binomially distributed, $y_i \sim \text{bin}(p_i, n_i)$. The probability p_i is connected to a q -vector of covariates $x_i = (x_{i1}, \dots, x_{iq})$ and a q -vector of regression coefficients $b = (b_1, \dots, b_q)$ as follows: The term $s_i = x_i \cdot b$ is denoted the *linear predictor*. The probability p_i can be linked to s_i in different ways, but the most commonly employed is via the *logit link function* which is $\text{logit}(p_i) = \log(p_i/(1 - p_i))$ so here $\text{logit}(p_i) = s_i$. Based on Figure 2, we consider the specific model with $s_i = b_1 + b_2 \log 2(\text{dose}_i)$. For later use, we define the data matrix below:

```
R> DM <- cbind(model.matrix(~log2(dose), data=bud),
+                 bud[, c("ndead", "ntotal")]) |> as.matrix()
R> DM |> head(3)
```

```
#>   (Intercept) log2(dose) ndead ntotal
#> 1           1        0     1    20
#> 2           1        1     4    20
#> 3           1        2     9    20
```

Each component of the likelihood

The log-likelihood is $\log L = \sum_i y_i \log(p_i) + (n_i - y_i) \log(1 - p_i) = \sum_i \log L_i$, say. Consider the contribution to the total log-likelihood from the i th observation which is $\log L_i = l_i = y_i \log(p_i) + (n_i - y_i) \log(1 - p_i)$. Since we are focusing on one observation only, we shall ignore the subscript i in this section. First notice that with $s = \log(p/(1 - p))$ we can find p as a function of s as:

```
R> def_sym(s, p) # The previous polynomial p is removed by this new declaration
R> sol_ <- solve_sys(lhs = log(p / (1 - p)), rhs = s, vars = p)
R> p_s <- sol_[[1]]$p
```

$$p_s = \frac{e^s}{e^s + 1} \quad (7)$$

Next, find the likelihood as a function of p , as a function of s and as a function of b . The underscore in $\logLb_$ and elsewhere indicates that this expression is defined in terms of other symbols. The log-likelihood can be maximized using e.g. Newton-Raphson (see e.g. Nocedal and Wright (2006)) and in this connection we need the score function, S , and the Hessian, H :

```
R> def_sym(y, n)
R> b <- vector_sym(2, "b")
R> x <- vector_sym(2, "x")
R> logLp_ <- y * log(p) + (n - y) * log(1 - p) # logL as fn of p
R> s_b <- sum(x * b) # s as fn of b
R> p_b <- subs(p_s, s, s_b) # p as fn of b
R> logLb_ <- subs(logLp_, p, p_b) # logL as fn of b
R> Sb_ <- score(logLb_, b) |> simplify()
R> Hb_ <- hessian(logLb_, b) |> simplify()
```

$$p_b = \frac{e^{b_1 x_1 + b_2 x_2}}{e^{b_1 x_1 + b_2 x_2} + 1}; \quad (8)$$

$$\logLb_ = y \log \left(\frac{e^{b_1 x_1 + b_2 x_2}}{e^{b_1 x_1 + b_2 x_2} + 1} \right) + (n - y) \log \left(1 - \frac{e^{b_1 x_1 + b_2 x_2}}{e^{b_1 x_1 + b_2 x_2} + 1} \right); \quad (9)$$

$$Sb_ = \begin{bmatrix} \frac{x_1(-ne^{b_1 x_1 + b_2 x_2} + ye^{b_1 x_1 + b_2 x_2} + y)}{e^{b_1 x_1 + b_2 x_2} + 1} \\ \frac{x_2(-ne^{b_1 x_1 + b_2 x_2} + ye^{b_1 x_1 + b_2 x_2} + y)}{e^{b_1 x_1 + b_2 x_2} + 1} \end{bmatrix}; \quad (10)$$

$$Hb_ = \begin{bmatrix} \frac{nx_1^2 e^{b_1 x_1 + b_2 x_2}}{2e^{b_1 x_1 + b_2 x_2} + e^{2b_1 x_1 + 2b_2 x_2} + 1} & \frac{nx_1 x_2 e^{b_1 x_1 + b_2 x_2}}{2e^{b_1 x_1 + b_2 x_2} + e^{2b_1 x_1 + 2b_2 x_2} + 1} \\ \frac{nx_1 x_2 e^{b_1 x_1 + b_2 x_2}}{2e^{b_1 x_1 + b_2 x_2} + e^{2b_1 x_1 + 2b_2 x_2} + 1} & \frac{nx_2^2 e^{b_1 x_1 + b_2 x_2}}{2e^{b_1 x_1 + b_2 x_2} + e^{2b_1 x_1 + 2b_2 x_2} + 1} \end{bmatrix}. \quad (11)$$

There are some possible approaches before maximizing the total log likelihood. One is to insert data case by case into the symbolic log likelihood:

```
R> nms <- c("x1", "x2", "y", "n")
R> DM_lst <- doBy::split_byrow(DM)
R> logLb_lst <- lapply(DM_lst, function(vls) {
+   subs(logLb_, nms, vls)
+ })
```

For example, the contribution from the third observation to the total log likelihood is:

$$\logLb_lst[[3]] = 9 \log \left(\frac{e^{b_1 + 2b_2}}{e^{b_1 + 2b_2} + 1} \right) + 11 \log \left(1 - \frac{e^{b_1 + 2b_2}}{e^{b_1 + 2b_2} + 1} \right). \quad (12)$$

The full likelihood can be maximized either e.g. using SymPy (not pursued here) or by converting the sum to an R function which can be maximized using one of R's internal optimization procedures:

```
R> logLb_tot <- Reduce(`+`, logLb_lst)
R> logLb_fn <- as_func(logLb_tot, vec_arg = TRUE)
R> opt <- optim(c(b1 = 0, b2 = 0), logLb_fn,
+                 control = list(fnscale = -1), hessian = TRUE)
R> opt$par

#>     b1      b2
#> -2.82  1.26
```

The same model can be fitted e.g. using R's `glm()` function as follows:

```
R> m <- glm(cbind(ndead, ntotal - ndead) ~ log2(dose), family=binomial(), data=bud)
R> m |> coef()

#> (Intercept)  log2(dose)
#>          -2.82           1.26
```

The total likelihood symbolically

We conclude this section by illustrating that the log-likelihood for the entire dataset can be constructed in a few steps (output is omitted to save space):

```
R> N <- 6; q <- 2
R> X <- matrix_sym(N, q, "x")
R> n <- vector_sym(N, "n")
R> y <- vector_sym(N, "y")
R> p <- vector_sym(N, "p")
R> s <- vector_sym(N, "s")
R> b <- vector_sym(q, "b")
```

$$X = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ x_{41} & x_{42} \\ x_{51} & x_{52} \\ x_{61} & x_{62} \end{bmatrix}; \quad n = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \\ n_6 \end{bmatrix}; \quad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix}. \quad (13)$$

The symbolic computations are as follows: We express the linear predictor s as function of the regression coefficients b and express the probability p as function of the linear predictor:

```
R> logLp <- sum(y * log(p) + (n - y) * log(1 - p)) # logL as fn of p
R> p_s <- exp(s) / (exp(s) + 1) # p as fn of s
R> s_b <- X %*% b # s as fn of b
R> p_b <- subs(p_s, s, s_b) # p as fn of b
R> logLb_ <- subs(logLp, p, p_b) # logL as fn of b
```

Next step could be to go from symbolic to numerical computations by inserting numerical values. From here, one may proceed by computing the score function and the Hessian matrix and solve the score equation, using e.g. Newton-Raphson. Alternatively, one might create an R function based on the log-likelihood, and maximize this function using one of R's optimization methods (see the example in the previous section):

```
R> logLb <- subs(logLb_, cbind(X, y, n), DM)
R> logLb_fn <- as_func(logLb, vec_arg = TRUE)
R> opt <- optim(c(b1 = 0, b2 = 0), logLb_fn,
+                 control = list(fnscale = -1), hessian = TRUE)
R> opt$par

#>     b1      b2
#> -2.82  1.26
```

3.3 Example: Constrained maximum likelihood

In this section we illustrate constrained optimization using Lagrange multipliers. This is demonstrated for the independence model for a two-way contingency table. Consider a 2×2 contingency table with cell counts y_{ij} and cell probabilities p_{ij} for $i = 1, 2$ and $j = 1, 2$, where i refers to row and j to column as illustrated in Table 1.

Under multinomial sampling, the log likelihood is

$$l = \log L = \sum_{ij} y_{ij} \log(p_{ij}). \quad (14)$$

Under the assumption of independence between rows and columns, the cell probabilities have the form, (see e.g. Højsgaard, Edwards, and Lauritzen (2012), p. 32)

$$p_{ij} = u \cdot r_i \cdot s_j. \quad (15)$$

To make the parameters (u, r_i, s_j) identifiable, constraints must be imposed. One possibility is to require that $r_1 = s_1 = 1$. The task is then to estimate u, r_2, s_2 by maximizing the log likelihood under the constraint that $\sum_{ij} p_{ij} = 1$. These constraints can be imposed using a Lagrange multiplier where we solve the unconstrained optimization problem $\max_p \text{Lag}(p)$ where

$$\text{Lag}(p) = -l(p) + \lambda g(p) \quad \text{under the constraint that} \quad (16)$$

$$g(p) = \sum_{ij} p_{ij} - 1 = 0, \quad (17)$$

where λ is a Lagrange multiplier. The likelihood equations can be found in closed-form. In SymPy, `lambda` is a reserved symbol so it is denoted by an postfixed underscore below:

```
R> def_sym(u, r2, s2, lambda_)
R> y <- as_sym(c("y_11", "y_21", "y_12", "y_22"))
R> p <- as_sym(c("u", "u*r2", "u*s2", "u*r2*s2"))
R> logL <- sum(y * log(p))
R> Lag <- -logL + lambda_* (sum(p) - 1)
R> vars <- list(u, r2, s2, lambda_)
R> gLag <- der(Lag, vars)
R> sol <- solve_sys(gLag, vars)
R> print(sol, method = "ascii")

#> Solution 1:
#> lambda_ = y_11 + y_12 + y_21 + y_22
#> r2      = (y_21 + y_22)/(y_11 + y_12)
#> s2      = (y_12 + y_22)/(y_11 + y_21)
#> u       = (y_11 + y_12)*(y_11 + y_21)/(y_11 + y_12 + y_21 + y_22)^2

R> sol <- sol[[1]]
```

There is only one critical point. The fitted cell probabilities \hat{p}_{ij} are:

```
R> p11 <- sol$u
R> p21 <- sol$u * sol$r2
R> p12 <- sol$u * sol$s2
R> p22 <- sol$u * sol$r2 * sol$s2
R> p.hat <- matrix_(c(p11, p21, p12, p22), nrow = 2)
```

$$\hat{p} = \frac{1}{(y_{11} + y_{12} + y_{21} + y_{22})^2} \begin{bmatrix} (y_{11} + y_{12})(y_{11} + y_{21}) & (y_{11} + y_{12})(y_{12} + y_{22}) \\ (y_{11} + y_{21})(y_{21} + y_{22}) & (y_{12} + y_{22})(y_{21} + y_{22}) \end{bmatrix} \quad (18)$$

To verify that the maximum likelihood estimate has been found, we compute the Hessian matrix which is negative definite (the Hessian matrix is diagonal so the eigenvalues are the diagonal entries and these are all negative), output omitted:

```
R> H <- hessian(logL, list(u, r2, s2)) |> simplify()
```

3.4 Example: An auto regression model

Symbolic computations

In this section we study the auto regressive model of order 1 (an AR(1) model, see e.g. Shumway and Stoffer (2016), p. 75): Consider random variables x_1, x_2, \dots, x_n following a stationary zero mean AR(1) process:

$$x_i = ax_{i-1} + e_i; \quad i = 2, \dots, n, \quad (19)$$

where $e_i \sim N(0, v)$ and all independent and with constant variance v . The marginal distribution of x_1 is also assumed normal, and for the process to be stationary we must have that the variance $\text{Var}(x_1) = v/(1 - a^2)$. Hence we can write $x_1 = \frac{1}{\sqrt{1-a^2}}e_1$.

For simplicity of exposition, we set $n = 4$ such that $e = (e_1, \dots, e_4)$ and $x = (x_1, \dots, x_4)$. Hence $e \sim N(0, vI)$. Isolating error terms in (19) gives

$$e = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix} = \begin{bmatrix} \sqrt{1-a^2} & . & . & . \\ -a & 1 & . & . \\ . & -a & 1 & . \\ . & . & -a & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = Lx. \quad (20)$$

Since $\text{Var}(e) = vI$ we have $\text{Var}(e) = vI = L\text{Var}(x)L^\top$ so the covariance matrix of x is $V = \text{Var}(x) = vL^{-1}(L^{-1})^\top$ while the concentration matrix (the inverse covariance matrix) is $K = v^{-1}L^\top L$:

```
R> def_sym(a, v)
R> n <- 4
R> L <- diff_mat(n, "-a") # The difference matrix, L, shown above
R> L[1, 1] <- sqrt(1 - a^2)
R> Linv <- solve(L)
R> K <- crossprod_(L) / v
R> V <- tcrossprod_(Linv) * v
```

$$L^{-1} = \begin{bmatrix} \frac{1}{\sqrt{1-a^2}} & . & . & . \\ \frac{a}{\sqrt{1-a^2}} & 1 & . & . \\ \frac{a^2}{\sqrt{1-a^2}} & a & 1 & . \\ \frac{a^3}{\sqrt{1-a^2}} & a^2 & a & 1 \end{bmatrix}; \quad (21)$$

$$K = \frac{1}{v} \begin{bmatrix} 1 & -a & 0 & 0 \\ -a & a^2 + 1 & -a & 0 \\ 0 & -a & a^2 + 1 & -a \\ 0 & 0 & -a & 1 \end{bmatrix}; \quad (22)$$

$$V = \frac{v}{a^2 - 1} \begin{bmatrix} -1 & -a & -a^2 & -a^3 \\ -a & -1 & -a & -a^2 \\ -a^2 & -a & -1 & -a \\ -a^3 & -a^2 & -a & -1 \end{bmatrix}. \quad (23)$$

The zeros in the concentration matrix K implies a conditional independence restriction: If the ij th element of a concentration matrix is zero then x_i and x_j are conditionally independent given all other variables (see e.g. Højsgaard, Edwards, and Lauritzen (2012), p. 84 for details).

Next, we take the step from symbolic computations to numerical evaluations. The joint distribution of x is multivariate normal distribution, $x \sim N(0, K^{-1})$. Let $W = xx^\top$ denote the matrix of (cross) products. The log-likelihood is therefore (ignoring additive constants)

$$\log L = \frac{n}{2}(\log \det(K) - x^\top K x) = \frac{n}{2}(\log \det(K) - \text{tr}(KW)), \quad (24)$$

where we note that $\text{tr}(KW)$ is the sum of the elementwise products of K and W since both matrices are symmetric. Ignoring the constant $\frac{n}{2}$, this can be written symbolically to obtain the expression in this particular case:

```
R> x <- vector_sym(n, "x")
R> logL <- log(det(K)) - sum(K * (x %*% t(x))) |> simplify()
```

$$\log L = \log \left(-\frac{a^2}{v^4} + \frac{1}{v^4} \right) - \frac{-2ax_1x_2 - 2ax_2x_3 - 2ax_3x_4 + x_1^2 + x_2^2(a^2 + 1) + x_3^2(a^2 + 1) + x_4^2}{v}. \quad (25)$$

Numerical evaluation

Next we illustrate how bridge the gap from symbolic computations to numerical computations based on a dataset: For a specific data vector we get:

```
R> xt <- c(0.1, -0.9, 0.4, 0.0)
R> logL_ <- subs(logL, x, xt)
```

$$\log L = \log \left(-\frac{a^2}{v^4} + \frac{1}{v^4} \right) - \frac{0.97a^2 + 0.9a + 0.98}{v}. \quad (26)$$

We can use R for numerical maximization of the likelihood and constraints on the parameter values can be imposed e.g. in the `optim()` function:

```
R> logL_wrap <- as_func(logL_, vec_arg = TRUE)
R> eps <- 0.01
R> par <- optim(c(a=0, v=1), logL_wrap,
+                 lower=c(-(1-eps), eps), upper=c((1-eps), 10),
+                 method="L-BFGS-B", control=list(fnscale=-1))$par
R> par

#>      a          v
#> -0.376   0.195
```

The same model can be fitted e.g. using R's `arima()` function as follows (output omitted):

```
R> arima(xt, order = c(1, 0, 0), include.mean = FALSE, method = "ML")
```

It is less trivial to do the optimization in `caracas` by solving the score equations. There are some possibilities for putting assumptions on variables in `caracas` (see the "Reference" vignette), but it is not possible to restrict the parameter a to only take values in $(-1, 1)$.

3.5 Example: Variance of average of correlated variables

Consider random variables x_1, \dots, x_n where $\text{Var}(x_i) = v$ and $\text{Cov}(x_i, x_j) = vr$ for $i \neq j$, where $0 \leq |r| \leq 1$. For $n = 3$, the covariance matrix of (x_1, \dots, x_n) is therefore

$$V = vR = v \begin{bmatrix} 1 & r & r \\ r & 1 & r \\ r & r & 1 \end{bmatrix}. \quad (27)$$

Let $\bar{x} = \sum_i x_i/n$ denote the average. Suppose interest is in the variance of the average, $w_{nr} = \text{Var}(\bar{x})$, when n goes to infinity. Here the subscripts n and r emphasize the dependence on the sample size n and the correlation r . The variance of a sum $x. = \sum_i x_i$ is $\text{Var}(x.) = \sum_i \text{Var}(x_i) + 2 \sum_{ij:i < j} \text{Cov}(x_i, x_j)$ (i.e., the sum of the elements of the covariance matrix). Then $w_{nr} = \text{Var}(\bar{x}) = \text{Var}(x.)/n^2$. We can do this in `caracas` as follows using the `sum_` function that calculate a symbolic sum:

```
R> def_sym(v, r, n, j, i)
R> s1 <- sum_(r, j, i+1, n) # sum_{j = i+1}^n r
R> s2 <- sum_(s1, i, 1, n-1) |> simplify()
R> var_sum <- v*(n + 2 * s2) |> simplify()
R> w_nr <- var_sum / n^2
```

Above, $s1$ is the sum of elements $i + 1$ to n in row j of the covariance matrix and therefore $s2$ is the sum of the entire upper triangular of the covariance matrix.

$$s1 = r(-i + n); \quad s2 = \frac{nr(n-1)}{2}; \quad w_{nr} = \text{Var}(\bar{x}) = \frac{v(r(n-1)+1)}{n}. \quad (28)$$

The limiting behavior of the variance w_{nr} can be studied in different situations (results shown later):

```
R> l_1 <- lim(w_nr, n, Inf)           # when sample size n goes to infinity
R> l_2 <- lim(w_nr, r, 0, dir = '+') # when correlation r goes to zero
R> l_3 <- lim(w_nr, r, 1, dir = '-') # when correlation r goes to one
```

Moreover, for a given correlation r it is instructive to investigate how many independent variables, say k_{nr} the n correlated variables correspond to (in the sense of giving the same variance of the average), because then k_{nr} can be seen as a measure of the amount of information in data. We call k_{nr} the effective sample size. Moreover, one might study how k_{nr} behaves as function of n when $n \rightarrow \infty$. That is we must (1) solve $v(1 + (n-1)r)/n = v/k_{nr}$ for k_{nr} and (2) find the limit $k_r = \lim_{n \rightarrow \infty} k_{nr}$:

```
R> def_sym(k_n)
R> sol <- solve_sys(w_nr - v / k_n, k_n)
R> k_nr <- sol[[1]]$k_n           # effective sample size
R> k_r <- lim(k_nr, n, Inf)
```

The findings above are:

$$l_1 = \lim_{n \rightarrow \infty} w_{nr} = rv; \quad l_2 = \lim_{r \rightarrow 0} w_{nr} = \frac{v}{n}; \quad l_3 = \lim_{r \rightarrow 1} w_{nr} = v; \quad k_{nr} = \frac{n}{nr - r + 1}; \quad k_r = \frac{1}{r}. \quad (29)$$

It is illustrative to supplement the symbolic computations above with numerical evaluations, which shows that even a moderate correlation reduces the effective sample size substantially. In Fig. 3, this is illustrated for a wider range of correlations and sample sizes.

```
R> dat <- expand.grid(r = c(.1, .2, .5), n = c(10, 50, 1000))
R> k_nr_fn <- as_func(k_nr)
R> dat$k_nr <- k_nr_fn(r = dat$r, n = dat$n)
R> dat$k_r <- 1 / dat$r
R> dat

#>      r      n   k_nr   k_r
#> 1  0.1    10  5.26   10
#> 2  0.2    10  3.57    5
#> 3  0.5    10  1.82    2
#> 4  0.1    50  8.47   10
#> 5  0.2    50  4.63    5
#> 6  0.5    50  1.96    2
#> 7  0.1  1000  9.91   10
#> 8  0.2  1000  4.98    5
#> 9  0.5  1000  2.00    2
```

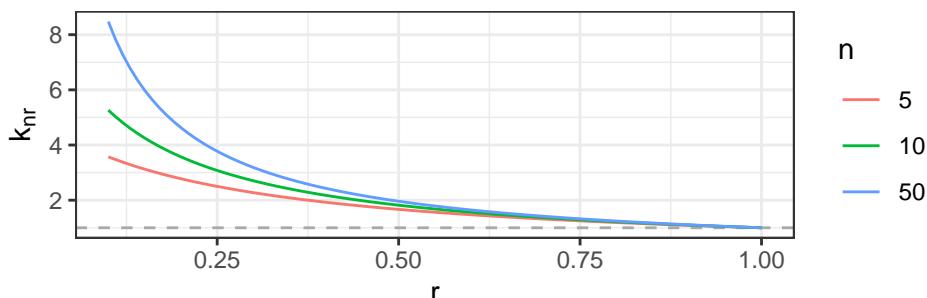


Figure 3: Effective sample size k_{nr} as function of correlation r for different values of n . The dashed line is the limit of k_r as $r \rightarrow 1$, i.e. 1.

4 Further topics

4.1 Integration, limits, and unevaluated expressions

The unit circle is given by $x^2 + y^2 = 1$ so the area of the upper half of the unit circle is $\int_{-1}^1 \sqrt{1 - x^2} dx$ (which is known to be $\pi/2$). This result is produced by caracas while the `integrate` function in R produces the approximate result 1.57.

```
R> x <- as_sym("x")
R> half_circle_ <- sqrt(1 - x^2)
R> ad <- int(half_circle_, "x")           # Anti derivative
R> area <- int(half_circle_, "x", -1, 1) # Definite integral
```

$$\text{ad} = \frac{x\sqrt{1-x^2}}{2} + \frac{\sin(x)}{2}; \quad \text{area} = \frac{\pi}{2}. \quad (30)$$

Finally, we illustrate limits and the creation of unevaluated expressions:

```
R> def_sym(x, n)
R> y <- (1 + x/n)^n
R> l <- lim(y, n, Inf, doit = FALSE)
R> l_2 <- doit(l)
```

$$l = \lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n; \quad l_2 = e^x \quad (31)$$

Several functions have the `doit` argument, e.g. `lim()`, `int()` and `sum_()`. Among other things, unevaluated expressions help making reproducible documents where the changes in code appears automatically in the generated formulas.

4.2 Documents with mathematical content

A LaTeX rendering of a caracas symbol, say `x`, is obtained by typing `$$x = `r tex(x)`$$. This feature is useful when creating documents with a mathematical content and has been used extensively throughout this paper.`

For rendering matrices, the `tex()` function has a `zero_as_dot` argument which is useful:

```
R> A <- diag_(c("a", "b", "c"))
```

$$\text{tex}(A) = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix}; \quad \text{tex}(A, \text{zero_as_dot} = \text{TRUE}) = \begin{bmatrix} a & \cdot & \cdot \\ \cdot & b & \cdot \\ \cdot & \cdot & c \end{bmatrix} \quad (32)$$

When displaying a matrix A , the expression can sometimes be greatly simplified by displaying k and (A/k) for some factor k . A specific example could when displaying M^{-1} . Here one may choose to display $(1/\det(M))$ and $M^{-1}\det(M)$. This can be illustrated as follows:

```
R> M0 <- toeplitz(c("a", "b")) # Character matrix
R> M <- as_sym(M0)           # as_sym() converts to a caracas symbol
R> Minv <- solve(M)
R> Minv2 <- scale_matrix(Minv, det(Minv))
```

$$\text{Minv} = \begin{bmatrix} \frac{a}{a^2-b^2} & -\frac{b}{a^2-b^2} \\ -\frac{b}{a^2-b^2} & \frac{a}{a^2-b^2} \end{bmatrix}; \quad \text{Minv2} = \frac{1}{a^2-b^2} \begin{bmatrix} a & -b \\ -b & a \end{bmatrix} \quad (33)$$

4.3 Extending caracas

It is possible to easily extend caracas with additional functionality from SymPy using `sympy_func()` from caracas which we illustrate below. This example illustrates how to use SymPy's `diff()` function to find univariate derivatives multiple times. The partial derivative of $\sin(xy)$ with respect to x and y is found with `diff` in SymPy:

```
R> library(reticulate)
R> sympy <- import("sympy")
R> sympy$diff("sin(x * y)", "x", "y")

#> -x*y*sin(x*y) + cos(x*y)
```

Alternatively:

```
R> x <- sympy$symbols("x")
R> y <- sympy$symbols("y")
R> sympy$diff(sympy$sin(x*y), x, y)
```

One the other hand, the `der()` function in `caracas` finds the gradient, which is a design choice in `caracas`:

```
R> def_sym(x, y)
R> f <- sin(x * y)
R> der(f, list(x, y))

#> c: [y*cos(x*y), x*cos(x*y)]
```

If we want to obtain the functionality from SymPy we can write a function that invokes `diff` in SymPy using the `sympy_func()` function in `caracas`:

```
R> der_diff <- function(expr, ...) {
+   sympy_func(expr, "diff", ...)
+ }
R> der_diff(sin(x * y), x, y)

#> c: -x*y*sin(x*y) + cos(x*y)
```

This latter function is especially useful if we need to find the higher-order derivative with respect to the same variable:

```
R> sympy$diff("sin(x * y)", "x", 100L)
R> der_diff(sin(x * y), x, 100L)
```

4.4 Switching back and forth between `caracas` and `reticulate`

Another way of invoking SymPy functionality that is not available in `caracas` is the following. As mentioned, a `caracas` symbol is a list with a slot called `pyobj` (accessed by `$pyobj`). Therefore, one can work with `caracas` symbols in `reticulate`, and one can also coerce a Python object into a `caracas` symbol. For example, it is straight forward to create a Toeplitz matrix using `caracas`. The minor sub matrix obtained by removing the first row and column using `reticulate` and the result can be coerced to a `caracas` object with `as_sym()`, e.g. for numerical evaluation (introduced later).

```
R> A <- as_sym(toeplitz(c("a", "b", 0))) # caracas symbol
R> B_ <- A$pyobj$minor_submatrix(0, 1) # reticulate object (notice: 0-based indexing)
R> B <- B_ |> as_sym() # caracas symbol
```

$$A = \begin{bmatrix} a & b & 0 \\ b & a & b \\ 0 & b & a \end{bmatrix}; \quad B = \begin{bmatrix} b & b \\ 0 & a \end{bmatrix}. \quad (34)$$

5 Hands-on activities

1. Related to Section Example: Linear models:

- a) The orthogonal projection matrix onto the span of the model matrix X is $P = X(X^\top X)^{-1}X^\top$. The residuals are $r = (I - P)y$. From this one may verify that these are not all independent.

- b) If one of the factors is ignored, then the two-way analysis of variance model becomes a one-way analysis of variance model, and it is illustrative to redo the computations in this setting.
 - c) Likewise if an interaction between the two factors is included in the model, what are the residuals in this case?
2. Related to Section Example: Logistic regression:
- a) In [Each component of the likelihood](#), Newton-Raphson can be implemented to solve the likelihood equations. Note how sensitive Newton-Raphson is to starting point. This can be solved by another optimisation scheme, e.g. Nelder-Mead (optimising the log likelihood) or BFGS (finding extreme for the score function).
 - b) The example is done as logistic regression with the logit link function. Try other link functions such as cloglog (complementary log-log).
3. Related to Section Example: Constrained maximum likelihood:
- a) Identifiability of the parameters was handled by not including r_1 and s_1 in the specification of p_{ij} . An alternative is to impose the restrictions $r_1 = 1$ and $s_1 = 1$, and this can also be handled via Lagrange multipliers. Another alternative is to regard the model as a log-linear model where $\log p_{ij} = \log u + \log r_i + \log s_j = \bar{u} + \bar{r}_i + \bar{s}_j$. This model is similar in its structure to the two-way ANOVA for Section Example: Linear models. This model can be fitted as a generalized linear model with a Poisson likelihood and log as link function. Hence, one may modify the results in Section Example: Logistic regression to provide an alternative way of fitting the model.
 - b) A simpler task is to consider a multinomial distribution with four categories, counts y_i and cell probabilities p_i , $i = 1, 2, 3, 4$ where $\sum_i p_i = 1$. For this model, find the maximum likelihood estimate for p_i (use the Hessian to verify that the critical point is a maximum).
4. Related to Section Example: An auto regression model:
- a) Compare the estimated parameter values with those obtained from the `arima()` function.
 - b) Modify the model in Equation (19) by setting $x_1 = ax_n + e_1$ ("wrapping around") and see what happens to the pattern of zeros in the concentration matrix.
 - c) Extend the AR(1) model to an AR(2) model ("wrapping around") and investigate this model along the same lines. Specifically, what are the conditional independencies (try at least $n = 6$)?

5. Related to Section Example: Variance of average of correlated variables:

- a) Simulate the situation given in the paper (e.g. using the function `mvrnorm()` in R package MASS) and verify that the results align with the symbolic computations.
- b) It is interesting to study such behaviours for other covariance functions. Replicate the calculations for the covariance matrix of the form

$$V = vR = v \begin{bmatrix} 1 & r & 0 \\ r & 1 & r \\ 0 & r & 1 \end{bmatrix}, \quad (35)$$

i.e., a special case of a Toeplitz matrix. How many independent variables, k , do the n correlated variables correspond to?

6 Discussion

We have presented the `caracas` package and argued that the package extends the functionality of R significantly with respect to symbolic mathematics. In contrast to using `reticulate` and `Sympy` directly, `caracas` provides symbolic mathematics in standard R syntax.

One practical virtue of `caracas` is that the package integrates nicely with `Rmarkdown`, (J. Allaire et al. 2021), (e.g. with the `tex()` functionality)

and thus supports creating of scientific documents and teaching material. As for the usability in practice we await feedback from users.

Another related R package is `Ryacas` based on `Yacas` (Pinkus and Winitzki 2002; Pinkus, Winnitzky, and Mazur 2016). The `Ryacas` package has existed for many years and is still of relevance. `Ryacas` probably has fewer features than `caracas`. On the other hand, `Ryacas` does not require Python (it is compiled). Finally, the `Yacas` language is extendable (see e.g. the vignette "User-defined yacas rules" in the `Ryacas` package).

One possible future development could be an R package which is designed without a view towards the underlying engine (SymPy or Yacas) and which then draws more freely from SymPy and Yacas. In this connection we mention that there are additional resources on CRAN such as `calculus` (Guidotti 2022).

Lastly, with respect to freely available resources in a CAS context, we would like to draw attention to `WolframAlpha`, see e.g. <https://www.wolframalpha.com/>, which provides an online service for answering (mathematical) queries.

7 Acknowledgements

We would like to thank the R Consortium for financial support for creating the `caracas` package, users for pin pointing aspects that can be improved in `caracas` and Ege Rubak (Aalborg University, Denmark), Poul Svante Eriksen (Aalborg University, Denmark), Giovanni Marchetti (University of Florence, Italy) and reviewers for constructive comments.

References

- Allaire, J. J., Charles Teague, Carlos Scheidegger, Yihui Xie, and Christophe Dervieux. 2022. "Quarto." <https://doi.org/10.5281/zenodo.5960048>.
- Allaire, JJ, Yihui Xie, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, Hadley Wickham, Joe Cheng, Winston Chang, and Richard Iannone. 2021. *Rmarkdown: Dynamic Documents for r*. <https://github.com/rstudio/rmarkdown>.
- Andersen, Mikkel Meyer, and Søren Højsgaard. 2021. "caracas: Computer algebra in R." *Journal of Open Source Software* 6 (63): 3438. <https://doi.org/10.21105/joss.03438>.
- Guidotti, Emanuele. 2022. "calculus: High-Dimensional Numerical and Symbolic Calculus in R." *Journal of Statistical Software* 104 (1): 1–37. <https://doi.org/10.18637/jss.v104.i05>.
- Højsgaard, Søren, David Edwards, and Steffen Lauritzen. 2012. *Graphical Models with R*. New York: Springer. <https://doi.org/10.1007/978-1-4614-2299-0>.
- Højsgaard, Søren, and Ulrich Halekoh. 2023. *doBy: Groupwise Statistics, LSmeans, Linear Estimates, Utilities*. <https://github.com/hojsgaard/doBy>.
- McCullagh, P, and John A Nelder. 1989. *Generalized Linear Models*. 2nd ed. Chapman & Hall/CRC Monographs on Statistics and Applied Probability. Philadelphia, PA: Chapman & Hall/CRC. <https://www.routledge.com/Generalized-Linear-Models/McCullagh-Nelder/p/book/9780412317606>.
- Nocedal, Jorge, and Stephen J. Wright. 2006. *Numerical Optimization*. Springer New York. <https://doi.org/10.1007/978-0-387-40065-5>.
- Pinkus, Ayal, and Serge Winitzki. 2002. "YACAS: A Do-It-Yourself Symbolic Algebra Environment." In *Proceedings of the Joint International Conferences on Artificial Intelligence, Automated Reasoning, and Symbolic Computation*, 332–36. AISC '02/Calculemus '02. London, UK, UK: Springer-Verlag. https://doi.org/10.1007/3-540-45470-5_29.
- Pinkus, Ayal, Serge Winitzki, and Grzegorz Mazur. 2016. "Yacas - Yet another computer algebra system." <https://yacas.readthedocs.io/en/latest/>.
- Shumway, Robert H., and David S. Stoffer. 2016. *Time Series Analysis and Its Applications*. Fourth Edition. Springer. <https://doi.org/10.1007/978-3-319-52452-8>.
- Ushey, Kevin, JJ Allaire, and Yuan Tang. 2020. *Reticulate: Interface to 'Python'*. <https://CRAN.R-project.org/package=reticulate>.
- Wickham, Hadley. 2016. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. <https://ggplot2.tidyverse.org>.
- Xie, Yihui, J. J. Allaire, and Garrett Grolemund. 2018. *R Markdown: The Definitive Guide*. Boca Raton, Florida: Chapman; Hall/CRC. <https://bookdown.org/yihui/rmarkdown>.
- Xie, Yihui, Christophe Dervieux, and Emily Riederer. 2020. *R Markdown Cookbook*. Boca Raton, Florida: Chapman; Hall/CRC. <https://bookdown.org/yihui/rmarkdown-cookbook>.

Mikkel Meyer Andersen

Department of Mathematical Sciences, Aalborg University, Denmark

Skjernvej 4A

9220 Aalborg Ø, Denmark

ORCID: 0000-0002-0234-0266

mikl@math.aau.dk

Søren Højsgaard

Department of Mathematical Sciences, Aalborg University, Denmark

*Skjernvej 4A
9220 Aalborg Ø, Denmark
ORCiD: 0000-0002-3269-9552
sorenh@math.aau.dk*

A Comparison of R Tools for Nonlinear Least Squares Modeling

by John C. Nash and Arkajyoti Bhattacharjee

Abstract Our Google Summer of Code project “Improvements to `nls()`” investigated rationalizing R tools for nonlinear regression and nonlinear estimation tools by considering usability, maintainability, and functionality, especially for a Gauss-Newton solver. The rich features of `nls()` are weakened by several deficiencies and inconsistencies such as a lack of stabilization of the Gauss-Newton solver. Further considerations are the usability and maintainability of the code base that provides the functionality `nls()` claims to offer. Various packages, including our `nlsr`, provide alternative capabilities. We consider the differences in goals, approaches, and features of different tools for nonlinear least squares modeling in R. Discussion of these matters is relevant to improving R generally as well as its nonlinear estimation tools.

1 The `nls()` function

`nls()` is a Comprehensive R Archive Network (CRAN: <https://cran.r-project.org>) tool that has remarkable and wide-ranging features for estimating nonlinear statistical models that are expressed as **formulas**. The function dates to the 1980s and the work related to D. M. Bates and Watts (1988) in S (see https://en.wikipedia.org/wiki/S_%28programming_language%29).

In particular, we note that it

- can call calculations in other programming languages
- allows weighted or subset data
- can estimate bound-constrained parameters
- provides partially linear model handling mechanism
- permits parameters to be indexed over a set of related data
- produces measures of variability (i.e., standard error estimates) for the estimated parameters
- has related profiling capabilities for exploring the likelihood surface as parameters are changed
- links to many pre-coded (`selfStart`) models that do not require initial parameter values.

Due to its extensive range of features and prolonged history, the code has become untidy and overly patched, making it challenging to maintain and ripe for improvement in its underlying methods.

2 Scope of our comparison

Besides the base-R `nls()` function, we will pay particular attention to `nlsr` (John C. Nash and Murdoch (2023)), `minpack.lm` (Elzhov et al. (2012)), and `gslnls` (Chau (2023)) which are general nonlinear least-squares solvers in the CRAN repository. While we will provide capsule comments for some other CRAN packages, those in the Bioconductor (Gentleman et al. (2004)) collection are more specialized, and those on repositories such as GitHub (<https://github.com>) and Gitlab (<https://about.gitlab.com>), while interesting, do not have the checking applied to CRAN packages.

Our work aimed at unifying nonlinear modeling functionality in R, ideally in a refactored `nls()` function. The primary messages from this work are:

- For R **users**, we would advise that it is most efficient to carry out nonlinear modeling or least squares by adapting working scripts, preferably those with documentation and using recent tools. If there is a suspicion that there may be ill-conditioning, package `nlsr` or the example we give in the section “Comparison notes for formula-setup solutions” below of how to find singular values of the Jacobian allow these diagnostics to be calculated.
- For R **developers**, we invite and encourage discussion of the design choices, since these have downstream implications for ease of use, adaptation to new features, and efficiency of ongoing maintenance.

3 Some other CRAN packages for nonlinear modeling

`onls` (Spiess, Andrej-Nikolai (2022)) is used for optimising and estimating nonlinear models by minimizing the sum of squares of **orthogonal** residuals rather than vertical residuals. The objec-

tive is therefore different and involves nontrivial extra calculation. A vignette with the package and the blog article <https://www.r-bloggers.com/2015/01/introducing-orthogonal-nonline-least-squares-regression-in-r/> give some description with illustrative graphs. `nls` appears to be limited to problems with one independent and one dependent variable. The Wikipedia article https://en.wikipedia.org/wiki/Total_least_squares presents an overview of some ideas, with references to the literature. The approach needs a wider discussion and tutorial examples to allow its merits to be judged than can be included here.

`crsnls` (Tvrdík (2016)) – This package allows nonlinear estimation by controlled random search via two methods. There is unfortunately no vignette. A modest trial we carried out showed `nlsr::n1xb()` gave the same results in a small fraction of the time required by either of the methods in `crsnls`. The method discussed in Josef Tvrdík and Ivan Křivý and Ladislav Mišík (2007) claims better reliability in finding solutions than a Levenberg-Marquardt code (actually from Matlab), but the tests were conducted on the extreme NIST examples mentioned next.

`NISTnls` (D. M. Bates (2012)) – This package provides R code and data for a set of (numerically ill-conditioned) nonlinear least squares problems from the U.S. National Institute for Standards and Technology. These may not represent real-world situations.

`n1shelper` (Duursma (2017)) – This package, which unfortunately lacks a vignette, provides a few utilities for summarizing, testing, and plotting non-linear regression models estimated with `nls()`, `nlsList()` or `nlme()` that are linked or grouped in some way.

`nlsic` (Sokol (2022)) – This solves nonlinear least squares problems with optional equality and/or inequality constraints. It is clearly not about modeling, and the input and output are quite different from class `nls` methods. However, there do not appear to be other R packages with these capabilities.

`nlsMicrobio` (Baty and Delignette-Muller (2014)) – Data sets and nonlinear regression models dedicated to predictive microbiology, including a vignette, by authors of the `nlstools` package.

`nlstools` (Baty and Delignette-Muller (2013)) – This package provides several tools for aiding the estimation of nonlinear models, particularly using `nls()`. The vignette is actually a journal article, and the authors have considerable experience in the subject.

`nlsmsn` (Prates, Lachos, and Garay (2021)) – Fit univariate non-linear scale mixture of skew-normal (NL-SMSN) regression, with details in Garay, Lachos, and Abanto-Valle (2011). The problem here is to minimize an objective that is modified from the traditional sum of squared residuals.

`nls.multstart` (Padfield and Matheson (2020)) – Non-linear least squares regression using AIC scores with the Levenberg-Marquardt algorithm using multiple starting values for increasing the chance that the minimum found is the global minimum.

`nls2` (Grothendieck (2022)) – Nonlinear least squares by brute force has similar motivations to `nls.multstart`, but uses `nls()` within multiple trials. The author has extensive expertise in R.

`nlstac` (Rodriguez-Arias et al. (2020)) – A set of functions implementing the algorithm described in Torvisco, Rodriguez-Arias, and Sanchez (2018) for fitting separable nonlinear regression curves. The special class of problem for which this package is intended is an important and difficult one. No vignette is provided, unfortunately.

`easynls` – Fit and plot some nonlinear models. Thirteen models are treated, but there is minimal documentation and no vignette. Package `nlraa` is to be preferred.

`nlraa` (Miguez (2021)) – a set of nonlinear `selfStart` models, primarily from agriculture. Most include analytic Jacobian code.

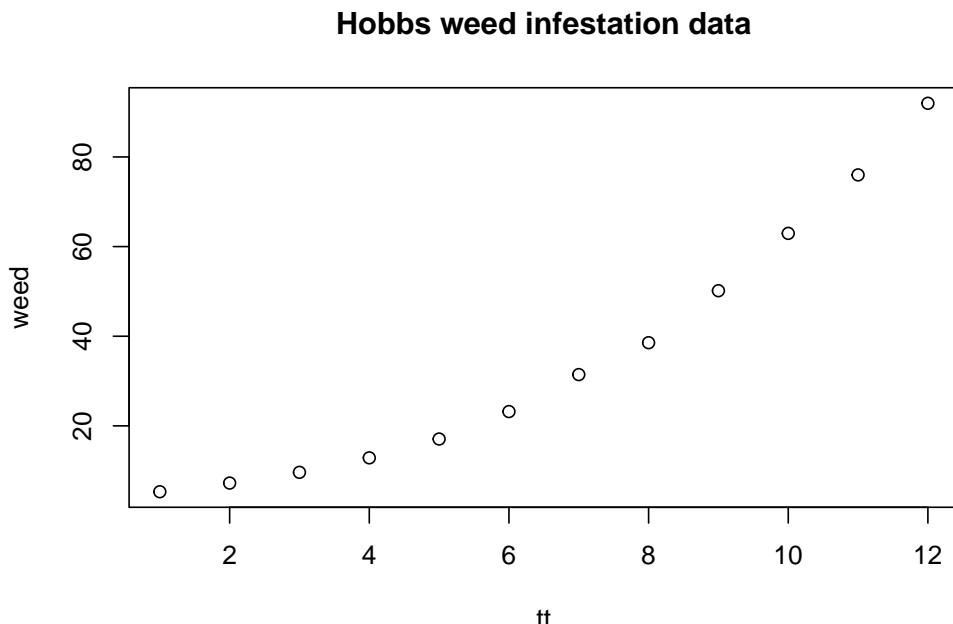
`optimx` (John C. Nash and Varadhan (2011)) – This provides optimizers that can be applied to minimize a nonlinear function which could be a nonlinear sum of squares. Not generally recommended if nonlinear least squares programs can be easily used, but provides a check and alternative solvers.

4 An illustrative example

The Hobbs weed infestation problem (John C. Nash (1979, 120)) is a growth-curve modeling task. Its very succinct statement provides the “short reproducible example” much requested on R mailing lists. The problem is small and seemingly straightforward, yet presents such difficulties that optimization researchers have asked if it is contrived rather than a real problem from a field researcher. The data and graph follow.

```
weed <- c(5.308, 7.24, 9.638, 12.866, 17.069, 23.192, 31.443,
       38.558, 50.156, 62.948, 75.995, 91.972)
tt <- 1:12
weeddf <- data.frame(tt, weed)
```

```
plot(weeddf, main="Hobbs weed infestation data")
```



While model estimation is increasingly automated, nonlinear regression should use background knowledge and graphs to obtain an understanding of the general magnitude of the parameters. Indeed, the “visual fitting” approach (John C. Nash and Velleman (1996)) mentioned later accords with this viewpoint, as does use of the **Logistic3T** variant of the model, as well as discussions in Ross (1990), Seber and Wild (1989), D. M. Bates and Watts (1988), and Gallant (1987). Our emphasis on the software resilience to starting parameters that are, in the sense of statistical modeling, “silly” comes from over half a century of dealing with users whose interests and understanding are very far from those of statistical modelers. Thus, we seek methods and codes that obtain reasonable answers under highly unfavourable conditions. Nevertheless, a proper approach to nonlinear modeling is to apply all available knowledge to the task.

Three suggested models for this data are (with names to allow for easy reference)

Logistic3U:

$$y \approx b_1 / (1 + b_2 * \exp(-b_3 * t))$$

Logistic3S:

$$y \approx 100 * c_1 / (1 + 10 * c_2 * \exp(-0.1 * c_3 * t))$$

Logistic3T:

$$y \approx Asym / (1 + \exp((xmid - t) / scal))$$

where we will use *weed* for *y* and *tt* for *t*. The functions above are equivalent, but the first is generally more awkward to solve numerically due to its poor scaling. The parameters of the three forms are related as follows:

$$\begin{aligned} Asym &= b_1 = 100 * c_1 \\ \exp(xmid / scal) &= b_2 = 10 * c_2 \\ 1 / scal &= b_3 = 0.1 * c_3 \end{aligned}$$

To allow for a simpler discussion, let us say that the parameters form a (named) vector *p* and the model function is called *model(p)* with *r* = *y* − *model(p)*.

We wish to minimize the sum of squared residuals, which is our loss (or objective) function. Starting with some guess for the parameters, we aim to alter these parameters to obtain a smaller loss function. We then iterate until we can make no further progress.

Let us consider there are *n* parameters and *m* residuals. The loss function is

$$S(p) = r'r = \sum_{i=1}^m r_i^2$$

The gradient of *S(p)* is

$$g = 2 * J'r$$

where the Jacobian J is given by elements

$$J_{i,j} = \partial r_i / \partial p_j$$

and the Hessian is defined by elements

$$H_{i,j} = \partial^2 S(p) / \partial p_i \partial p_j$$

If we expand the Hessian for nonlinear least squares problems, we find

$$0.5 * H_{i,j} = \sum_{k=1}^m J_{k,i} J_{k,j} + \sum_{k=1}^m r_k * \partial r_k / \partial p_i \partial p_j$$

Let us use $D_{i,j}$ for the elements of the second term of this expression. What is generally called **Newton's method** for function minimization tries to set the gradient to zero (to find a stationary point of the function $S(p)$). This leads to **Newton's equation**

$$H\delta = -g$$

Given a set of parameters p , we solve this equation for δ , adjust p to $p + \delta$ and iterate, hopefully to converge on a solution. Applying this to a sum of squares problem gives

$$0.5 * H\delta = (J'J + D)\delta = -J'r$$

In this expression, only the elements of D have second partial derivatives. Gauss, attempting to model planetary orbits, had small residuals, and noted that these multiplied the second partial derivatives of r , so he approximated

$$0.5 * H \approx J'J$$

by assuming $D \approx 0$. This results in the Gauss-Newton method where we solve

$$J'J\delta = -J'r$$

though we can avoid some loss of accuracy by not forming the inner product matrix $J'J$ and solving the linear least squares matrix problem

$$J\delta \approx -r$$

by one of several matrix decomposition methods.

In reality, there are many problems where D should not be ignored, but the work to compute it precisely is considerable. Many work-arounds have been proposed, of which the Levenberg-Marquardt stabilization (Levenberg (1944), Marquardt (1963)) is the most commonly used. For convenience, we will use "Marquardt", as we believe he first incorporated the ideas into a practical computer program.

The usual suggestion is that D be replaced by a multiple of the unit matrix or else a multiple of the diagonal part of $J'J$. In low precision, some elements of $J'J$ could underflow to zero (John C. Nash (1977)), and a linear combination of both choices is an effective compromise. Various choices for D , as well as a possible line search along the direction δ rather than a unit step (Hartley (1961)), give rise to several variant algorithms. "Marquardt's method" is a family of methods. Fortunately, most choices work well.

4.1 Problem setup

Let us specify in R the three model formulas and set some starting values for parameters. These starting points are not equivalent and are deliberately crude choices. Workers performing many calculations of a similar nature should try to provide good starting points to reduce computation time and avoid finding a false solution.

```
# model formulas
frmw <- weed ~ b1 / (1 + b2 * exp(-b3 * tt))
frms <- weed ~ 100 * c1/(1 + 10*c2* exp(-0.1 * c3* tt))
frmt <- weed ~ Asym / (1 + exp((xmid - tt) / scal))
#
# Starting parameter sets
stu1 <- c(b1 = 1, b2 = 1, b3 = 1)
sts1 <- c(c1 = 1, c2 = 1, c3 = 1)
stt1 <- c(Asym = 1, xmid = 1, scal = 1)
```

One of the useful features of `nls()` is the possibility of a `selfStart` model, where starting pa-

rameter values are not required. However, if a *selfStart* model is not available, *nls()* sets all the starting parameters to 1. This is tolerable but could be improved by using a set of values that are all slightly different, which, in the case of the example model $y \sim a * \exp(-b * x) + c * \exp(-d * x)$ would avoid a singular Jacobian because b and d were equal in value. Program modifications to give a sequence like 1.0, 1.1, 1.2, 1.3 for the four parameters are fairly obvious.

It is also possible to provide R functions for the residual and Jacobian. This is usually much more work for the user if the formula setup is possible. To illustrate, we show the functions for the unscaled 3 parameter logistic. The particular form of these explicit residual and Jacobian functions comes from their translation from BASIC codes of the 1970s, as adapted in John C. Nash and Walker-Smith (1987). The use of the Laplace form of the residual and the inclusion of data within the functions reflects choices of that era that are at odds with current practice. Some users still want or need to provide problems as explicit functions, particularly for problems that are not regressions. For example, the Rosenbrock banana-valley test problem can be provided this way, where the two “residuals” are different functional forms.

```
# Logistic3U
hobbs.res <- function(x){ # unscaled Hobbs weeds problem -- residual
  if(length(x) != 3) stop("hobbs.res -- parameter vector n!=3")
  y <- c(5.308, 7.24, 9.638, 12.866, 17.069, 23.192, 31.443,
        38.558, 50.156, 62.948, 75.995, 91.972)
  tt <- 1:12
  res <- x[1] / (1 + x[2] * exp(-x[3] * tt)) - y
  # Note: this form of the residual, from Laplace (1788) in the form
  # of "fitted - observed" has been used in our software for half
  # a century, but sometimes concerns users of the more traditional
  # "observed - fitted" form
}

hobbs.jac <- function(x) { # unscaled Hobbs weeds problem -- Jacobian
  jj <- matrix(0.0, 12, 3)
  tt <- 1:12
  yy <- exp(-x[3] * tt)
  zz <- 1.0 / (1 + x[2] * yy)
  jj[tt, 1] <- zz
  jj[tt, 2] <- -x[1] * zz * zz * yy
  jj[tt, 3] <- x[1] * zz * zz * yy * x[2] * tt
  attr(jj, "gradient") <- jj
  jj
}
```

5 Estimation of models specified as formulas

Using a formula specification was a principal advantage made with *nls()* when it became available in S sometime in the 1980s. It uses a Gauss-Newton (i.e., unstabilized) iteration with a step reduction line search. This works very efficiently as long as J is not ill-conditioned. Below we see *nls()* does poorly on the example problem. To save page space, we use 1-line result display functions from package *nlsr*, namely *pnls()* and *pshort()*.

```
#> Error in nls(formula = frmu, start = stu1, data = weeddf) :
#>   singular gradient

#> Error in nls(formula = frms, start = sts1, data = weeddf) :
#>   singular gradient

#> Error in nls(formula = frmt, start = stt1, data = weeddf) :
#>   singular gradient
```

Here we see the infamous “singular gradient” termination message of *nls()*. Users should, of course, be using the *SSlogis* *selfStart* tool, but ignorance of this possibility or a slight variant in the model can easily lead to outcomes similar to those seen here.

5.1 Solution attempts with nlsr

```
#> unlx1:residual sumsquares = 2.5873 on 12 observations
#> after 19 Jacobian and 25 function evaluations
#>   name      coeff       SE     tstat    pval    gradient   JSingval
#>   b1      196.186     11.31    17.35 3.167e-08 -4.859e-09     1011
#>   b2      49.0916     1.688    29.08 3.284e-10 -3.099e-08     0.4605
#>   b3      0.31357    0.006863    45.69 5.768e-12  2.305e-06    0.04714

#> snlx1 -- ss= 2.5873 : c1 = 1.9619 c2 = 4.9092 c3 = 3.1357; 34 res/ 23 jac

#> tnlx1 -- ss= 2.5873 : Asym = 196.19 xmid = 12.417 scal = 3.1891; 36 res/ 27 jac
```

Though we have found solutions, the Jacobian is essentially singular as shown by its singular values. Note that these are displayed by package `nlsr` in a single column in the output to provide a compact layout, but the values do not correspond to the individual parameters in whose row they appear; they are a property of the whole problem.

5.2 Solution attempts with minpack.lm

```
#> unlm1 -- ss= 2.5873 : b1 = 196.19 b2 = 49.092 b3 = 0.31357; 17 itns

#> snlm1 -- ss= 2.5873 : c1 = 1.9619 c2 = 4.9092 c3 = 3.1357; 7 itns

#> Error in nlsModel(formula, mf, start, wts) :
#>   singular gradient matrix at initial parameter estimates
```

5.3 Solution attempts with gslnls

```
#> ugslnls1 -- ss= 2.5873 : b1 = 196.19 b2 = 49.092 b3 = 0.31357; 25 itns

#> sgslnls1 -- ss= 2.5873 : c1 = 1.9619 c2 = 4.9092 c3 = 3.1357; 9 itns

#> tgslnls1 -- ss= 9205.4 : Asym = 35.532 xmid = 20846 scal = -1745.2; 47 itns
```

5.4 Comparison notes for formula-setup solutions

`nlsr:::nlxb()` uses `print()` to output standard errors and singular values of the Jacobian (for diagnostic purposes). By contrast, `minpack.lm:::nlsLM()` and `nls()` use `summary()`, which does not display the sum of squares, while `print()` gives the sum of squares, but not the standard error of the residuals.

The singular values allow us to gauge how “nearly singular” the Jacobian is at the solution, and the ratio of the smallest to largest of the singular values is a simple but effective measure. The ratios are 4.6641e-05 for Logistic3U, 0.021022 for Logistic3S, and 0.001055 for Logistic3T, so Logistic3S is the “least singular”.

The results from `nlsLM` and `gsl_nls` for the transformed model Logistic3T have a very large sum of squares, which may suggest that these programs have failed. Since `nls()`, `nlsLM()`, and `gsl_nls()` do not offer singular values, we need to extract the Jacobian and compute its singular values. The following script shows how to do this, using as Jacobian what is called the gradient element in the returned solution for these solvers.

```
# for nlsLM
if (inherits(tnlm1, "try-error")) {
  print("Cannot compute solution -- likely singular Jacobian")
} else {
  JtnlsLM <- tnlm1$gradient() # actually the Jacobian
  svd(JtnlsLM)$d # Singular values
}

#> [1] "Cannot compute solution -- likely singular Jacobian"
```

```

# for gsl_nls
if (inherits(tgslnls1, "try-error")) {
  cat("Cannot compute solution -- likely singular Jacobian")
} else {
  JtnlsLM <- tgslnls1$gradient()
  svd(JtnlsLM)$d # Singular values
}

#> [1] 3.4641e+00 9.8541e-09 3.8249e-11

We see that there are differences in detail, but the more important result is that two out of three singular values are essentially 0. Our Jacobian is singular, and no method of the Gauss-Newton type should be able to continue. Indeed, from the parameters reported at this saddle point, nlsr::nlxb() cannot proceed.

stspecial <- c(Asym = 35.532, xmid = 43376, scal = -2935.4)
badstart <- try(nlxb(formula = frmt, start = stspecial, data = weeddf))
if (! inherits(badstart, "try-error")) print(badstart)

#> residual sumssquares = 9205.4 on 12 observations
#> after 2 Jacobian and 2 function evaluations
#>   name      coeff       SE     tstat    pval   gradient   JSingval
#> Asym      35.5321     NA      NA      NA -9.694e-09    3.464
#> xmid      43376      NA      NA      NA -1.742e-09  2.61e-10
#> scal      -2935.4     NA      NA      NA -2.4e-08  7.12e-16

```

6 Functional specification of problems

We illustrate how to solve nonlinear least squares problems using a function to define the residual. Note that `gsl_nls()` requires a vector `y` that is the length of the vector returned by the function supplied, e.g., `hobbs.res()`. `gsl_nls` uses a numerical approximation for the Jacobian if the argument `jac` is missing. Note function `nlsr::pnls1m()` for a 1-line display of the results of `minpack.lm::nls.lm()`.

```

#> hobnlfb<-nlfb(start=stu1, resfn=hobbs.res, jacfn=hobbs.jac)

#> hobnlfb -- ss= 2.5873 : b1 = 196.19 b2 = 49.092 b3 = 0.31357; 25 res/ 19 jac

#> hobnlm<-nls.lm(par=stu1, fn=hobbs.res, jac=hobbs.jac)

#> hobnlm -- ss= 2.5873 : b1 = 196.19 b2 = 49.092 b3 = 0.31357; 17 itns

#> hobgsln<-gsl_nls(start=stu1, fn=hobbs.res, y=rep(0,12))

#> hobgsln -- ss= 2.5873 : b1 = 196.19 b2 = 49.092 b3 = 0.31357; 25 itns

#> hobgsl<-gsl_nls(start=stu1, fn=hobbs.res, y=rep(0,12), jac=hobbs.jac)

#> hobgsl -- ss= 2.5873 : b1 = 196.19 b2 = 49.092 b3 = 0.31357; 25 itns

```

7 Design goals, termination tests, and output objects

The output object of `nlxb()` is smaller than the class `nls` object returned by `nls()`, `nlsLM()`, and `gsl_nls()`. Package `nlsr` emphasizes the solution of the nonlinear least squares problem rather than the estimation of a nonlinear model that fits or explains the data. The object of class `nls` allows for a number of specialized modeling and diagnostic extensions. For compatibility, the `nlsr` package has the function `wrapnlsr()`, for which `nlsr()` is an alias. This uses `nlxb()` to find good parameters, then calls `nls()` to return the class `nls` object. Unless particular modeling features are needed, the use of `wrapnlsr()` is unnecessary and wasteful of resources.

The design goals of the different tools may also be revealed in the so-called “convergence tests” for the iterative solvers. In the manual page for `nls()` in R 4.0.0, there was the warning:

> *Do not use nls on artificial “zero-residual” data*

with the suggested addition of small perturbations to the data. This admits `nls()` could not solve well-posed problems unless data is polluted with errors. Zero-residual problems are not always artificial, since problems in function approximation and nonlinear equations can be approached with nonlinear least squares. Fortunately, a small adjustment to the “termination test” for the program, rather than for the “convergence” of the underlying algorithm, fixes the defect. The test is the Relative Offset Convergence Criterion (see D. M. Bates and Watts (1981)). This scales an estimated reduction in the loss function by its current value. If the loss function is very small, we are close to a zero-divide. Adding a small quantity to the divisor avoids trouble. In 2021, one of us (J. Nash) proposed that `nls.control()` have an additional parameter `scaleOffset` with a default value of zero. Setting it to a small number – 1.0 is a reasonable choice – allows small-residual problems (i.e., near-exact fits) to be dealt with easily. We call this the **safeguarded relative offset convergence criterion**, and it has been in `nlsr` since it was introduced. The default value gives the legacy behavior. This improvement has been in the R distributed code since version 4.1.0.

Additional termination tests can be used. `nlsr` has a **small sum of squares test (smallssTest)** that compares the latest evaluated sum of squared (weighted) residuals to `e4` times the initial sum of squares, where `e4 <- (100 * Machine$double.eps)^4` is approximately 2.43e-55.

Termination after what may be considered excessive computation is also important. `nls()` stops after `maxiter` “iterations”. The meaning of “iteration” may require an examination of the code for the different algorithms. `nlsr` terminates execution when the number of residual or Jacobian evaluations exceed set limits. Generally, we prefer larger limits than the default `maxiter = 50` of `nls()` to avoid stopping early, though this may result in some unnecessary computations.

7.1 Returned results of `nls()` and other tools

As mentioned, the output of `nls()`, `minpack.lm::nlsLM()`, or `gslnls::gsl_nls()` is an object of class “`nls`” which has a quite rich structure described in the manual files or revealed by applying the `str()` function to the result of `nls()`. The complexity of this object is a challenge to users. Let us use for example `result <- snlm1` as the returned object from `nlsLM()` for the Logistic3S problem. The data return element is an R symbol. To actually access the data from this element, we need to use the syntax:

```
eval(parse(text = result$data))
```

However, if the call is made with `model = TRUE`, then there is a returned element `model` which contains the data, and we can list its contents using:

```
ls(result$model)
```

and if there is an element called `xdata`, then it can be accessed as `result$model$xdata`.

By contrast, `nlsr::n1xb()` returns a much simpler structure of 11 items in one level. Moreover, `n1xb` explicitly returns the sum of squares, the residual vector, Jacobian, and counts of evaluations.

7.2 When to compute ancillary information

Tools that produce a class `nls` output object create a rich set of functions and structures that are then used in a variety of modeling tasks, including the least squares solution. By contrast, `nlsr` computes quantities as they are requested or needed, with additional features in separate functions. For example, the singular values of the Jacobian are actually computed in the `print` and `summary` methods for the result. These two approaches lead to different consequences for performance and how features are provided. `nlsr` has antecedents in the methods of John C. Nash (1979), where storage for data and programs was at a ridiculous premium in the small computers of the era. Thus, the code in `nlsr` is likely of value for workers to copy and modify for customized tools.

8 Jacobian calculation

Gauss-Newton/Marquardt methods all need a Jacobian matrix at each iteration. By default, `nlsr::n1xb()` will try to evaluate this using analytic expressions using symbolic and automatic differentiation tools. When using a formula specification of the model, `nls()`, `minpack.lm::nlsLM()` and `gslnls::gsl_nls()` use a finite difference approximation to compute the Jacobian, though `gsl_nls()` does have an option to attempt symbolic expressions. Package `nlsr` provides, via appropriate calling syntax, four numeric

approximation options for the Jacobian, with a further control `ndstep` for the size of the step used in the approximation. These options allow programming choices to be examined. Users can largely ignore them.

Using the “gradient” attribute of the output of the Jacobian function to hold the Jacobian matrix lets us embed this in the residual function as well, so that the call to `nlsr::n1fb()` can be made with the same name used for both residual and Jacobian function arguments. This programming trick saves a lot of trouble for the package developer, but it can be a nuisance for users trying to understand the code.

As far as we can understand the logic in `nls()`, the Jacobian computation during parameter estimation is carried out within the called C-language program and its wrapper R code function `numericDeriv()`, part of `./src/library/stats/R/nls.R` in the R distribution source code. This is used to provide Jacobian information in the `nlsModel()` and `nlsModel.plinear()` functions, which are not exported for general use. `gsl_nls()` also appears to use `numericDeriv()`.

`numericDeriv()` uses a simple forward difference approximation of derivatives, though a central difference approximation can be specified in control parameters. We are unclear why `numericDeriv()` in base R calls `C_numeric_deriv`, as we were easily able to create a more compact version entirely in R. See <https://github.com/nashjc/RNonlinearLS/tree/main/DerivsNLS>.

`minpack.lm::nlsLM()` invokes `numericDeriv()` in its local version of `nlsModel()`, but it appears to use an internal approximate Jacobian code from the original Fortran `minpack` code, namely, `lmdif.f`. Such differences in approach can lead to different behavior, usually minor, but sometimes annoying with ill-conditioned problems.

- A pasture regrowth problem (Huet et al. (2004), page 1, based on Ratkowsky (1983)) has a poorly conditioned Jacobian and `nls()` fails with “singular gradient”. Worse, numerical approximation to the Jacobian gives the error “singular gradient matrix at initial parameter estimates” for `minpack.lm::nlsLM` so that the Marquardt stabilization is unable to take effect, while the analytic derivatives of `nlsr::n1xb` give a solution.
- Karl Schilling (private communication) provided an example where a model specified with the formula $y \sim a * (x^b)$ causes `nlsr::n1xb` to fail because the partial derivative w.r.t. b is $a * (x^b * \log(x))$. If there is data for which $x = 0$, the derivative is undefined, but the model can be computed. In such cases, we observed that `nls()` and `minpack.lm::nlsLM` found a solution, though this seems to be a lucky accident.

8.1 Jacobian code in selfStart models

Analytic Jacobian code can be provided to all the solvers discussed. Most `selfStart` models that automatically provide starting parameters also include such code. There is documentation in R of `selfStart` models, but their construction is non-trivial. A number of such models are included with base R in `./src/library/stats/R/zzModels.R`, with package `n1raa` (Miguez (2021)) providing a richer set. There are also some in the now-archived package `NRAIA`. These provide the Jacobian in the “gradient” attribute of the “one-sided” formula that defines each model, and these Jacobians are often the analytic forms.

The `nls()` function, after computing the “right-hand side” or `rhs` of the residual, checks to see if the “gradient” attribute is defined, otherwise using `numericDeriv()` to compute a Jacobian into that attribute. This code is within the `nlsModel()` or `nlsModel.plinear()` functions. The use of analytic Jacobians almost certainly contributes to the good performance of `nls()` on `selfStart` models.

The use of `selfStart` models with `nlsr` is described in the “Introduction to `nlsr`” vignette. However, since `nlsr` generally can use very crude starting values, we have rarely needed them, though it should be pointed out that our work is primarily diagnostic. If we were carrying out a large number of similar estimations, such initial parameters are critical to efficiency.

In considering `selfStart` models, we noted that the base-R function `SSlogis` is intended to solve problem **Logistic3T** above. When this function is used via `getInitial()` to find starting values, it actually calls `nls()` with the ‘`plinear`’ algorithm and finds a (full) solution. It then passes the solution coefficients to the default algorithm unnecessarily. Moreover, the implicit double call is, in our view, prone to creating errors in code maintenance. To provide simpler starting parameters, the function `SSlogisJN` is now part of the package `nlsr`, but is most useful for `nls()`.

Users may also want to profit from the Jacobian code of `selfStart` models but supply explicit starting values other than those suggested by `getInitial()`. This does not appear to be possible with `nls()`. `nlsr::n1xb()` always requires starting parameters, and can either use `getInitial()` to find them from the `selfStart` model or provide explicit values, but the formula used in the call to `n1xb()` still involves the `selfStart` function.

We are also surprised that the analytic expressions for the Jacobian (“gradient”) in the `SSLogis` function and others save quantities in “hidden” variables, i.e., with names preceded by “.”. These are then not displayed by the `ls()` command, making them difficult to access by users who may wish to create their own `selfStart` model via copy and edit. Interactive tools, such as “visual fitting” (John C. Nash and Velleman (1996)) might be worth considering as another way to find starting parameters, but we know of no R capability of this type.

As a side note, the introduction of `scaleOffset` in R 4.1.1 to deal with the convergence test for small residual problems now requires that the `getInitial()` function have dot-arguments (...) in its argument list. This illustrates the entanglement of many features in `nls()` that complicate its maintenance and improvement.

9 Bounds constraints on parameters

For many problems, we know that parameters cannot be bigger or smaller than some externally known limits. Such limits should be built into models, but there are some important details for using the tools in R.

- `nls()` can only impose bounds if the `algorithm = "port"` argument is used in the call. Unfortunately, the documentation warns us:

The algorithm = “port” code appears unfinished, and does not even check that the starting value is within the bounds. Use with caution, especially where bounds are supplied.
- `gsl_nls()` does not offer bounds.
- bounds are part of the default method for package `nlsr`.
- `nlsLM()` includes bounds in the standard call, but we have observed cases where it fails to get the correct answer. From an examination of the code, we believe the authors have not taken into account all possibilities, though all programs have some weakness regarding constrained optimization in that programmers have to work with assumptions on the scale of numbers, and some problems will be outside the scope envisaged.

```
# Start MUST be feasible i.e. on or within bounds
anlshob1b <- try(nls(frms, start = sts1, data = weeddf, lower = c(0,0,0),
                    upper = c(2,6,3), algorithm = 'port'))
if (! inherits(anlshob1b, "try-error")) pnls(anlshob1b) # check the answer (short form)

#> anlshob1b -- ss= 9.4726 : c1 = 2 c2 = 4.4332 c3 = 3; 12  itns

# nlsLM seems NOT to work with bounds in this example
anlsLM1b <- try(nlsLM(frms, start = sts1, data = weeddf, lower = c(0,0,0), upper = c(2,6,3)))
if (! inherits(anlsLM1b, "try-error")) pnls(anlsLM1b)

#> anlsLM1b -- ss= 881.02 : c1 = 2 c2 = 6 c3 = 3; 2  itns

# also no warning if starting out of bounds, but gets a good answer!!
st4 <- c(c1 = 4, c2 = 4, c3 = 4)
anlsLMob <- try(nlsLM(frms, start = st4, data = weeddf, lower = c(0,0,0), upper = c(2,6,3)))
if (! inherits(anlsLMob, "try-error")) pnls(anlsLMob)

#> anlsLMob -- ss= 9.4726 : c1 = 2 c2 = 4.4332 c3 = 3; 4  itns

# Try nlsr::nlxb()
anlx1b <- try(nlxb(frms, start = sts1, data = weeddf, lower = c(0,0,0), upper = c(2,6,3)))
if (! inherits(anlx1b, "try-error")) pshort(anlx1b)

#> anlx1b -- ss= 9.4726 : c1 = 2 c2 = 4.4332 c3 = 3; 12 res/ 12 jac
```

9.1 Philosophical considerations

Bounds on parameters raise some interesting questions about how uncertainty in parameter estimates should be computed or reported. That is, the traditional “standard errors” are generally taken to imply symmetric intervals about the point estimate in which the parameter may be expected to be found

with some probability under certain assumptions. Bounds change those assumptions and hence the interpretation of estimates of uncertainty, whether by traditional approximations from the $J'J$ matrix or from methods such as profile likelihood or bootstrap. At the time of writing, `nlsr::nlxb()` does not compute standard errors nor their derived statistics when bounds are active to avoid providing misleading information.

9.2 Fixed parameters (masks)

Let us try to fix (mask) the first parameter in the first two example problems.

```
# Hobbsmaskx.R -- masks with formula specification of the problem
require(nlsr); require(minpack.lm); traceval <- FALSE
stu <- c(b1 = 200, b2 = 50, b3 = 0.3) # a default starting vector (named!)
sts <- c(c1 = 2, c2 = 5, c3 = 3) # a default scaled starting vector (named!)
# fix first parameter
anxbmsk1 <- try(nlxb(frmu, start = stu, data = weeddf, lower = c(200, 0, 0),
                    upper = c(200, 60, 3), trace=traceval))
if (!inherits(anxbmsk1, "try-error")) print(anxbmsk1)

#> residual sumsquares = 2.6182 on 12 observations
#>      after 4 Jacobian and 4 function evaluations
#>      name        coeff          SE       tstat      pval      gradient    JSingval
#>      b1           200U M        NA        NA        NA          0         NA
#>      b2           49.5108     1.12     44.21  8.421e-13 -2.887e-07   1022
#>      b3           0.311461   0.002278   136.8  1.073e-17  0.0001635   0.4569

anlM1 <- try(nlsLM(frmu, start = stu, data = weeddf, lower = c(200, 0, 0),
                  upper=c(200, 60, 3), trace = traceval))
if (!inherits(anlM1, "try-error")) pnls(anlM1)

#> anlM1 -- ss= 2.6182 : b1 = 200 b2 = 49.511 b3 = 0.31146; 4 itns

anlsmsk1 <- try(nls(frmu, start = stu, data = weeddf, lower = c(200, 0, 0),
                     upper = c(200, 60, 3), algorithm = "port", trace = traceval))
if (!inherits(anlsmsk1, "try-error")) pnls(anlsmsk1)

#> anlsmsk1 -- ss= 2.6182 : b1 = 200 b2 = 49.511 b3 = 0.31146; 5 itns

# Hobbs scaled problem with bounds, formula specification
anlxmsks1 <- try(nlxb(frms, start = sts, data = weeddf, lower = c(2, 0, 0),
                   upper = c(2, 6, 30)))
if (!inherits(anlxmsks1, "try-error")) print(anlxmsks1)

#> residual sumsquares = 2.6182 on 12 observations
#>      after 4 Jacobian and 4 function evaluations
#>      name        coeff          SE       tstat      pval      gradient    JSingval
#>      c1           2U M        NA        NA        NA          0         NA
#>      c2           4.95108    0.112     44.21  8.421e-13 -2.981e-06   104.2
#>      c3           3.11461   0.02278   136.8  1.073e-17  1.583e-05   4.482

anlshmsk1 <- try(nls(frms, start = sts, trace = traceval, data = weeddf,
                      lower = c(2, 0, 0), upper = c(2, 6, 30), algorithm = 'port'))
if (!inherits(anlshmsk1, "try-error")) pnls(anlshmsk1)

#> anlshmsk1 -- ss= 2.6182 : c1 = 2 c2 = 4.9511 c3 = 3.1146; 5 itns

anlsLMmsks1 <- try(nlsLM(frms, start = sts, data = weeddf, lower = c(2,0,0),
                         upper = c(2,6,30)))
if (!inherits(anlsLMmsks1, "try-error")) pnls(anlsLMmsks1)

#> anlsLMmsks1 -- ss= 2.6182 : c1 = 2 c2 = 4.9511 c3 = 3.1146; 4 itns
```

```
# Test with all parameters masked
anlxmskall <- try(nlxb(frms, start=sts, data=weeddf, lower=sts, upper=sts))
if (! inherits(anlxmskall, "try-error")) print(anlxmskall)

#> residual sumofsquares = 158.23 on 12 observations
#> after 0 Jacobian and 1 function evaluations
#>   name       coeff      SE     tstat    pval   gradient   JSingval
#> c1          2U M      NA      NA      NA      NA      NA
#> c2          5U M      NA      NA      NA      NA      NA
#> c3          3U M      NA      NA      NA      NA      NA
```

`nlsr` has an output format that indicates the constraint status of the parameter estimates. For `nlsr`, we have chosen to suppress the calculation of approximate standard errors in the parameters when constraints are active because their meaning under constraints is unclear, though we believe this policy worthy of discussion and further investigation.

10 Stabilization of Gauss-Newton computations

All four major tools illustrated solve some variant of the Gauss-Newton equations. `nls()` uses a modification of an approach suggested by Hartley (1961), while `nlsr`, `gslnls`, and `minpack.lm` use flavors of Marquardt (1963). `gslnls` offers an accelerated Marquardt method and three alternative methods which we have not investigated. Control settings for `nlxb()` or `nlfb()` allow exploration of Hartley and Marquardt algorithm variants. In general, the Levenberg-Marquardt stabilization is important in obtaining solutions in methods of the Gauss-Newton family, as `nls()` terminates too frequently and unnecessarily with singular gradient errors.

10.1 Programming language

An important choice made in developing `nlsr` was to code entirely within the R programming language. `nls()` uses a mix of R, C, and Fortran, as does `minpack.lm`. `gslnls` is an R wrapper to various C-language routines in the GNU Scientific Library (Galassi et al. (2009)). Generally, keeping to a single programming language can allow for easier maintenance and upgrades. It also avoids some work when there are changes or upgrades to libraries for the non-R languages. R is usually considered slower than most computing environments because it keeps track of objects and because it is usually interpreted. In recent years, the performance penalty for using code entirely in R has been much reduced with the just-in-time compiler and other improvements. All-R computation may now offer acceptable performance. In `nlsr`, the use of R may be a smaller performance cost than the aggressive approach to a solution, which can cause more iterations to be used.

11 Data sources for problems

`nls()` can be called without specifying the `data` argument. In this case, it will search in the available environments (i.e., workspaces) for suitable data objects. We do not like this approach, but it is “the R way”. R allows users to leave many objects in the default (`.GlobalEnv`) workspace. Moreover, users have to actively suppress saving this workspace (`.RData`) on exit, otherwise any such file in the path will be loaded on startup. R users in our acquaintance avoid saving the workspace because of lurking data and functions that may cause unwanted results.

11.1 Feature: Subsetting

`nls()` and other class `nls` tools accept an argument `subset`. This acts through the mediation of `model.frame`, which is not obvious in the source code files `/src/library/stats/R/nls.R` and `/src/library/stats/src/nls.C`. Having `subset` at the level of the call to a function like `nls()` saves effort, but it does mean that the programmer of the solver needs to be aware of the origin (and value) of objects such as the data, residuals and Jacobian. By preference, we would implement subsetting by zero-value weights, with observation counts (and degrees of freedom) computed via the numbers of non-zero weights. Alternatively, we would extract a working dataframe from the relevant elements in the original.

11.2 Feature: na.action (missing value treatment)

`na.action` is an argument to the `nls()` function, but it does not appear obviously in the source code, often being handled behind the scenes after referencing the option `na.action`. This feature also changes the data supplied to our nonlinear least squares solver. A useful, but possibly dated, description is given in: <https://stats.idre.ucla.edu/r/faq/how-does-r-handle-missing-values/>. The typical default action, which can be seen by using the command `getOption("na.action")` is `na.omit`. This option removes from computations any observations containing missing values (i.e. any row of a data frame containing an NA). `na.exclude` does much of the same for solver computations, but keeps the rows with NA elements so that predictions are in the correct row position. We recommend that workers test output to verify the behavior is as wanted. See <https://stats.stackexchange.com/questions/492955/should-i-use-na-omit-or-na-exclude-in-a-linear-model-in-r>. As with `subset`, our concern with `na.action` is that users may be unaware of the effects of an option they may not even be aware has been set. Should `na.fail` be the default?

11.3 Feature: model frame

`model` is an argument to the `nls()` and related functions, which is documented as:

model logical. If true, the model frame is returned as part of the object. Default is FALSE.

Indeed, the argument only gets used when `nls()` is about to return its result object, and the element `model` is NULL unless the calling argument `model` is TRUE. Using the same name for both function argument and object element could be confusing. Despite this, the model frame is used within the function code in the form of the object `mf`. We feel that users could benefit from more extensive documentation and examples of its use since it is used to implement features like `subset`.

11.4 Weights on observations

All four main tools we consider here allow a `weights` argument that specifies a vector of fixed weights the same length as the number of residuals. Each residual is multiplied by the square root of the corresponding weight. Where available, the values returned by the `residuals()` function are weighted, and the `fitted()` or `predict()` function are used to compute raw residuals.

While fixed weights may be useful, there are many problems for which we want weights that are determined at least partially from the model parameters, for example, a measure of the standard deviation of observations. Such dynamic weighting situations are discussed in the vignette “Introduction to `nlsr`” of package `nlsr` in section *Weights that are functions of the model parameters*. `minpack.lm` offers a function `wfct()` to facilitate such weighting. Care is advised in applying such ideas.

11.5 Weights in returned functions from `nls()`

The function `resid()` (an alias for `residuals()`) gives weighted residuals like `resultmresid()`. The function `nlsModel()` allows us to compute residuals for particular coefficient sets. We have had to extract `nlsModel()` from the base R code and include it via a code chunk (not echoed here for space) because it is not exported to the working namespace. We could also explicitly `source()` this code.

```
wts <- 0.5 ^ tt # simple weights
frmlogis <- weed ~ Asym / (1 + exp((xmid - tt)/scal))
Asym <- 1; xmid <- 1; scal <- 1
nowt <- try(nls(weed ~ SSlogis(tt, Asym, xmid, scal))) # UNWEIGHTED
if (! inherits(nowt, "try-error")) {
  rnnowt <- nowt$resid() # This has UNWEIGHTED residual and Jacobian. Does NOT take coefficients.
  attr(rnnowt, "gradient") <- NULL
} else rnnowt <- NULL
rnnowt

#> [1] -0.011900  0.032755 -0.092030 -0.208782 -0.392634  0.057594  1.105728
#> [8] -0.715786  0.107647  0.348396 -0.652592  0.287569

usewt <- try(nls(weed ~ SSlogis(tt, Asym, xmid, scal), weights = wts))
if (! inherits(usewt, "try-error")) {
  rusewt <- usewt$resid() # WEIGHTED. Does NOT take coefficients.
```

```

    attr(rusewt, "gradient") <- NULL
} else rusewt <- NULL
rusewt

#> [1] 0.0085640 0.0324442 -0.0176652 -0.0388479 -0.0579575 0.0163623
#> [7] 0.1042380 -0.0411766 0.0052509 0.0084324 -0.0194246 -0.0024053

## source("nlsModel.R") # or use {r nlsmodelsource, echo=FALSE} code chunk
nmod0 <- nlsModel(frmlogis, data = weeddf, start = c(Asym = 1, xmid = 1, scal = 1), wts = wts)
rn0 <- nmod0$resid() # Parameters are supplied in nlsModel() `start` above.
attr(rn0, "gradient") <- NULL; rn0 # weighted residuals at starting coefficients

#> [1] 3.3998 3.2545 3.0961 2.9784 2.8438 2.7748 2.6910 2.3474 2.1724 1.9359
#> [11] 1.6572 1.4214

nmod <- nlsModel(frmlogis, data = weeddf, start = coef(usewt), wts = wts)
rn <- nmod$resid()
attr(rn,"gradient")<-NULL; rn # same as rusewt

#> [1] 0.0085640 0.0324442 -0.0176652 -0.0388479 -0.0579575 0.0163623
#> [7] 0.1042380 -0.0411766 0.0052509 0.0084324 -0.0194246 -0.0024053

```

12 Minor issues with nonlinear least-squares tools

12.1 Interim output from the “port” algorithm

As the `nls()` **man** page states, when the “port” algorithm is used with the `trace` argument `TRUE`, the iterations display the objective function value which is $1/2$ the sum of squares (or deviance). The trace display is likely embedded in the Fortran of the `nlminb` routine that is called to execute the “port” algorithm, but the factor of 2 discrepancy is nonetheless unfortunate for users.

12.2 Failure to return the best result achieved

If `nls()` reaches a point where it cannot continue but has not found a point where the relative offset convergence criterion is met, it may simply exit, especially if a “singular gradient” (singular Jacobian) is found. However, this may occur AFTER the function has made considerable progress in reducing the sum of squared residuals. Here is an abbreviated example:

```

time <- c( 1, 2, 3, 4, 6 , 8, 10, 12, 16)
conc <- c( 0.7, 1.2, 1.4, 1.4, 1.1, 0.8, 0.6, 0.5, 0.3)
NLSdata <- data.frame(time,conc)
NLSstart <- c(lrc1 = -2, lrc2 = 0.25, A1 = 150, A2 = 50) # a starting vector (named!)
NLSformula <- conc ~ A1 * exp(-exp(lrc1) * time) + A2 * exp(-exp(lrc2) * time)
tryit <- try(nls(NLSformula, data = NLSdata, start = NLSstart, trace = TRUE))

#> 61216. (3.56e+03): par = (-2 0.25 150 50)
#> 2.1757 (2.23e+01): par = (-1.9991 0.31711 2.6182 -1.3668)
#> 1.6211 (7.14e+00): par = (-1.9605 -2.6203 2.5753 -0.55599)
#> Error in nls(NLSformula, data = NLSdata, start = NLSstart, trace = TRUE) :
#>   singular gradient

if (! inherits(tryit, "try-error")) print(tryit)

```

Note that the sum of squares has been reduced from 61216 to 1.6211, but unless `trace` is invoked, the user will not get any information about this. Changing this in the `nls()` function could be useful to R users, and would be almost trivial.

13 Estimating models that are partially linear

The variable projection method (Golub and Pereyra (1973), O’Leary and Rust (2013)) is usually much more effective than general approaches in finding good solutions to nonlinear least squares problems when some of the parameters appear linearly. In our logistic examples, the asymptote parameters are an illustration. However, identifying which parameters are linear and communicating this information to estimating functions is not a trivial task. `nls()` has an option `algorithm = "plinear"` that allows some partially linear models to be solved. The other tools, as far as we are aware, do not offer any such capability. The `nlstac` package uses a different algorithm for similar goals.

Within `nls()` itself we must, unfortunately, use different specifications with different `algorithm` options. For example, the explicit model $y \sim a * x + b$ does not work with the linear modeling function `lm()`, which requires this model to be specified as $y \sim x$. Within `nls()`, consider the following FOUR different specifications for the same problem, plus an intuitive choice, labeled `fm2a`, that does not work. In this failed attempt, putting the `Asym` parameter in the model causes the `plinear` algorithm to try to add another term to the model. We believe this is unfortunate, and would like to see a consistent syntax. At the time of writing, we do not foresee a resolution for this issue. In the example, we have not evaluated the commands to save space.

```
DNase1 <- subset(DNase, Run == 1) # select the data
## using a selfStart model - do not specify the starting parameters
fm1 <- try(nls(density ~ SSlogis(log(conc), Asym, xmid, scal), DNase1))
if (!inherits(fm1, "try-error")) summary(fm1)

## using conditional linearity - leave out the Asym parameter
fm2 <- try(nls(density ~ 1 / (1 + exp((xmid - log(conc)) / scal)),
               data = DNase1, start = list(xmid = 0, scal = 1),
               algorithm = "plinear"))
if (!inherits(fm2, "try-error")) summary(fm2)

## without conditional linearity
fm3 <- try(nls(density ~ Asym / (1 + exp((xmid - log(conc)) / scal)),
               data = DNase1,
               start = list(Asym = 3, xmid = 0, scal = 1)))
if (!inherits(fm3, "try-error")) summary(fm3)

## using Port's nl2sol algorithm
fm4 <- try(nls(density ~ Asym / (1 + exp((xmid - log(conc)) / scal)),
               data = DNase1, start = list(Asym = 3, xmid = 0, scal = 1),
               algorithm = "port"))
if (!inherits(fm4, "try-error")) summary(fm4)

## using conditional linearity AND Asym does not work
fm2a <- try(nls(density ~ Asym / (1 + exp((xmid - log(conc)) / scal)),
                 data = DNase1, start = list(Asym=3, xmid = 0, scal = 1),
                 algorithm = "plinear", trace = TRUE))
if (!inherits(fm2a, "try-error")) summary(fm2a)
```

14 Models with indexed parameters

Some models have several common parameters and others that are tied to particular cases. The `man` file for `nls()` includes an example of a situation in which parameters are indexed but which uses the “`plinear`” option as an added complication. Running this example reveals that the answers for the parameters are not indexed as in a vector. That is, we do not see `a[1]`, `a[2]`, `a[3]` but `a1`, `a2`, `a3`. This is no doubt because programming for indexed parameters is challenging. We note that there are capabilities in packages for mixed-effects modeling such as `nlme` (Pinheiro et al. (2013)), `bbmle` (Bolker and Team (2013)), and `lme4` (D. Bates et al. (2015)) for estimating models of such type.

15 Tests and use-case examples

Maintainers of packages need suitable tests and use-case examples in order

- to ensure packages work properly, in particular, giving results comparable to or better than the functions they are to replace.
- to test individual solver functions to ensure they work across the range of calling mechanisms, that is, different ways of supplying inputs to the solver(s);
- to pose “silly” inputs to see if these bad inputs are caught by the programs.

Such goals align with the aims of **unit testing** (e.g., <https://towardsdatascience.com/unit-testing-in-r-68ab9cc8d211>, Wickham (2011), Wickham et al. (2021) and the conventional R package testing tools). In our work, one of us (AB) has developed a working prototype package at <https://github.com/ArkaB-DS/nlsCompare>. A primary design objective of this is to allow the summarization of multiple tests in a compact output. The prototype has a vignette to illustrate its use.

16 Documentation and resources

In our investigation, we built several resources, which are now part of the repository <https://github.com/nashjc/RNonlinearLS/>. Particular items are:

- A BibTex bibliography for use with all documents in this project, but which has wider application to nonlinear least squares projects in general (<https://github.com/nashjc/RNonlinearLS/blob/main/BibSupport/ImproveNLS.bib>).
- MachID.R offers a suggested concise summary function to identify a particular computational system used for tests. A discussion of how it was built and the resources needed across platforms is given in at <https://github.com/nashjc/RNonlinearLS/tree/main/MachineSummary>.
- John C. Nash and Bhattacharjee (2022) is an explanation of the construction of the nls pkg from the nls() code in R-base.
- As the 2021 Summer of Code period was ending, one of us (JN) was invited to prepare a review of optimization in R. Ideas from the present work have been instrumental in the creation of John C. Nash (2022).

17 Future of nonlinear model estimation in R

Given its importance to R, it is possible that nls() will remain more or less as it has been for the past several decades. If so, the focus of discussion should be the measures needed to secure its continued operation for legacy purposes and how that may be accomplished. We welcome an opportunity to participate in such conversations.

To advance the stability and maintainability of R, we believe the program objects (R functions) that are created by tools such as nls() should have minimal cross-linkages and side-effects. The aspects of nls() that concern us follow.

- R tools presume data and parameters needed are available in an accessible environment. This provides a compact syntax to invoke the calculations, but the wrong data can be used if the internal search finds a valid name that is not the object we want, or if subset or na.action settings modify the selection, or weights are applied.
- Mixing of R, C and Fortran code adds to the burden of following the program logic.
- The class nls structure simplifies calls with its rich set of functions, but also adds to the task of understanding what has been done. A design that isolates the setup, solution, and post-solution parts of complicated calculations reduces the number of objects that must be kept in alignment.

Given the existence of examples of good practices such as analytic derivatives, stabilized solution of Gauss-Newton equations, and bounds-constrained parameters, base R tools should be moving to incorporate them. These capabilities are available now by using several tools, but it would be helpful if they were unified.

18 Acknowledgments

Hans Werner Borchers was helpful in developing the GSoC project motivating this article and in comments on this and related work. Heather Turner co-mentored the project and helped guide the progress of the work. Exchanges with Fernando Miguez helped to clarify aspects of selfStart models and instigated the “Introduction to nlsr” vignette. Colin Gillespie (package benchmarkme) has been

helpful in guiding our attempts to succinctly summarize computing environments. We thank one of the referees for pointing out that it is important to use background knowledge and graphs to guide modeling. Simon Urbanek gave some important practical help with preparing files for publication.

References

- Bates, Douglas M. 2012. *NISTnls: Nonlinear least squares examples from NIST*. <https://CRAN.R-project.org/package=NISTnls>.
- Bates, Douglas M., and Donald G. Watts. 1981. "A Relative Offset Orthogonality Convergence Criterion for Nonlinear least Squares." *Technometrics* 23 (2): 179–83.
- . 1988. *Nonlinear Regression Analysis and Its Applications*. Wiley.
- Bates, Douglas, Martin Mächler, Ben Bolker, and Steve Walker. 2015. "Fitting Linear Mixed-Effects Models Using lme4." *Journal of Statistical Software* 67 (1): 1–48. <https://doi.org/10.18637/jss.v067.i01>.
- Baty, Florent, and Marie-Laure Delignette-Muller. 2013. *Nlstools: Tools for Nonlinear Regression Diagnostics*.
- . 2014. *nlsMicrobio: Data Sets and Nonlinear Regression Models Dedicated to Predictive Microbiology*.
- Bolker, Ben, and R Development Core Team. 2013. *Bbmle: Tools for General Maximum Likelihood Estimation*. <http://CRAN.R-project.org/package=bbmle>.
- Chau, Joris. 2023. *gslnls: GSL Nonlinear Least-Squares Fitting*. <https://CRAN.R-project.org/package=gslnls>.
- Duursma, Remko. 2017. *Nlshelper: Convenient Functions for Non-Linear Regression*. <https://CRAN.R-project.org/package=nlshelper>.
- Elzhov, Timur V., Katharine M. Mullen, Andrej-Nikolai Spiess, and Ben Bolker. 2012. *minpack.lm: R interface to the Levenberg-Marquardt nonlinear least-squares algorithm found in MINPACK, plus support for bounds*. R Project for Statistical Computing. <http://CRAN.R-project.org/package=minpack.lm>.
- Galassi, Mark, Jim Davies, James Theiler, Brian Gough, and Gerard Jungman. 2009. *GNU Scientific Library - Reference Manual, Third Edition, for GSL Version 1.12* (3. ed.). Free Software Foundation.
- Gallant, A. Ronald. 1987. *Nonlinear Statistical Models*. Wiley.
- Gentelman, Robert C., Vincent J. Carey, Douglas M. Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, et al. 2004. "Bioconductor: Open Software Development for Computational Biology and Bioinformatics." *Genome Biology* 5 (R80). <https://doi.org/10.1186/gb-2004-5-10-r80>.
- Golub, G. H., and V. Pereyra. 1973. "The Differentiation of Pseudo-Inverses and Nonlinear Least Squares Problems Whose Variables Separate." *SIAM Journal of Numerical Analysis* 10 (2): 413–32.
- Grothendieck, G. 2022. *Nls2: Non-Linear Regression with Brute Force*. <https://CRAN.R-project.org/package=nls2>.
- Hartley, H. O. 1961. "The Modified Gauss-Newton Method for the Fitting of Non-Linear Regression Functions by Least Squares." *Technometrics* 3: 269–80.
- Huet, S., A. Bouvier, M.-A. Poursat, and E. Jolivet. 2004. *Statistical Tools for Nonlinear Regression: A Practical Guide with S-PLUS Examples*, 2nd Edition. Berlin & New York: Springer-Verlag.
- Josef Tvrdík and Ivan Křivý and Ladislav Mišík. 2007. "Adaptive Population-Based Search: Application to Estimation of Nonlinear Regression Parameters." *Computational Statistics & Data Analysis* 52 (2): 713–24. <https://doi.org/10.1016/j.csda.2006.10.014>.
- Levenberg, Kenneth. 1944. "A Method for the Solution of Certain Non-Linear Problems in Least Squares." *Quarterly of Applied Mathematics* 2: 164–68.
- Marquardt, Donald W. 1963. "An Algorithm for Least-Squares Estimation of Nonlinear Parameters." *SIAM Journal on Applied Mathematics* 11 (2): 431–41.
- Miguez, Fernando. 2021. *nlraa: Nonlinear Regression for Agricultural Applications*. <https://CRAN.R-project.org/package=nlraa>.
- Nash, John C. 1977. "Minimizing a Nonlinear Sum of Squares Function on a Small Computer." *Journal of the Institute for Mathematics and Its Applications* 19: 231–37.
- . 1979. *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation*. Bristol: Adam Hilger.
- . 2022. "Function minimization and nonlinear least squares in R." *WIREs Computational Statistics* 14 (e1580). <https://doi.org/https://doi.org/10.1002/wics.1580>.
- Nash, John C., and Arkajyoti Bhattacharjee. 2022. "Making a Package from Base R Files." *R-Bloggers*, January. <https://www.r-bloggers.com/2022/01/making-a-package-from-base-r-files/>.
- Nash, John C., and Duncan Murdoch. 2023. *nlsr: Functions for Nonlinear Least Squares Solutions*.
- Nash, John C., and Ravi Varadhan. 2011. *Optimx: A Replacement and Extension of the optim() Function*. Nash Information Services Inc.; Johns Hopkins University.
- Nash, John C., and Paul Velleman. 1996. "Nonlinear Estimation Combining Visual Fitting with

- Optimization Methods." In *Proceedings of the Section on Physical and Engineering Sciences of the American Statistical Association*, 256–61. American Statistical Association.
- Nash, John C., and Mary Walker-Smith. 1987. *Nonlinear Parameter Estimation: An Integrated System in BASIC*. New York: Marcel Dekker.
- O'Leary, Dianne P., and Bert W. Rust. 2013. "Variable Projection for Nonlinear Least Squares Problems." *Computational Optimization and Applications* 54 (3): 579–93.
- Padfield, Daniel, and Granville Matheson. 2020. *nls.multstart: Robust Non-Linear Regression using AIC Scores*. <https://CRAN.R-project.org/package=nls.multstart>.
- Pinheiro, Jose, Douglas Bates, Saikat DebRoy, Deepayan Sarkar, and R Core Team. 2013. *Nlme: Linear and Nonlinear Mixed Effects Models*.
- Prates, Marcos, Victor Lachos, and Aldo Garay. 2021. *nlsmsn: Fitting Nonlinear Models with Scale Mixture of Skew-Normal Distributions*. <https://CRAN.R-project.org/package=nlsmsn>.
- Ratkowsky, David A. 1983. *Nonlinear Regression Modeling: A Unified Practical Approach*. New York; Basel: Marcel Dekker Inc.
- Rodriguez-Arias, Mariano, Juan Antonio Fernandez, Javier Cabello, and Rafael Benitez. 2020. *Nlstac: An r Package for Fitting Separable Nonlinear Models*. <https://CRAN.R-project.org/package=nlstac>.
- Ross, Gavin J. S. 1990. *Nonlinear Estimation*. New York etc.: Springer-Verlag.
- Seber, G. A. F., and C. J. Wild. 1989. *Nonlinear regression*. New York: Wiley.
- Sokol, Serguei. 2022. *Nlsic: Non Linear Least Squares with Inequality Constraints*. <https://CRAN.R-project.org/package=nlsic>.
- Spiess, Andrej-Nikolai. 2022. *Onls: Orthogonal Nonlinear Least-Squares Regression*. <https://CRAN.R-project.org/package=onls>.
- Torvisco, Juan Antonio Fernandez, Mariano Rodriguez-Arias, and Javier Cabello Sanchez. 2018. "A New Algorithm to Fit Exponential Decays Without Initial Guess." Faculty of Sciences and Mathematics, University of Nis, Serbia. <https://doi.org/10.2298/fil1812233t>.
- Tvrdík, Josef. 2016. *crsnls: Nonlinear Regression Parameters Estimation by 'CRS4HC' and 'CRS4HCE'*. <https://CRAN.R-project.org/package=crsnls>.
- Wickham, Hadley. 2011. "testthat: Get Started with Testing." *The R Journal* 3: 5–10. https://journal.r-project.org/archive/2011-1/RJournal/_2011-1/_Wickham.pdf.
- Wickham, Hadley, Jim Hester, Winston Chang, and Jennifer Bryan. 2021. *devtools: Tools to Make Developing R Packages Easier*. <https://CRAN.R-project.org/package=devtools>.

John C. Nash
retired professor, University of Ottawa
Telfer School of Management
Ottawa ON Canada K1N 6N5
ORCID: 0000-0002-2762-8039
profjcash@gmail.com

Arkajyoti Bhattacharjee
Indian Institute of Technology
Department of Mathematics and Statistics
Kanpur
arkastat98@gmail.com

exvatoools: Value Added in Exports and Other Input-Output Table Analysis Tools

by Enrique Feás

Abstract This article introduces an R package, `exvatoools`, that simplifies the analysis of trade in value added with international input-output tables. It provides a full set of commands for data extraction, matrix creation and manipulation, decomposition of value added in gross exports (using alternative methodologies) and a straightforward calculation of many value added indicators. It can handle both raw data from well-known public input-output databases and custom data. It has a wide sector and geographical flexibility and can be easily expanded and adapted to specific economic analysis needs, facilitating a better understanding and a wider use of the available statistical resources to study globalization.

1 Introduction

The analysis of trade in value added involves the use of international input-output tables and intensive matrix manipulation. Some trade databases, such as the OECD Trade in Value Added Database (TiVA), offer a web interface, but cannot be customized and lack some key indicators developed in recent literature, especially indicators of bilateral trade in value added and participation in global value chains. This makes recourse to raw data almost inevitable and creates the need for software capable of performing complex matrix analysis in a user-friendly environment.

This is the gap `exvatoools` pretends to fill. It has been designed as a package for **R** with a triple purpose: as an international input-output table general analysis tool (with commands to extract raw data and produce and manipulate large matrices), as a way to decompose value added in exports using alternative methodologies, and as a tool to easily produce complex tailor-made value added indicators with flexible sector and geographical customization.

To our knowledge, there are no equivalent software tools available. There are some input-out analysis tools, like `ioanalysis` (Wade and Sarmiento-Barbieri 2020), but they cannot properly handle OECD's ICIO-type data (with some countries divided in industrial areas). The package `decompr` (Quast and Kummritz 2015) produces a decomposition of value added in exports, but only according to the methodology of Wang, Wei, and Zhu (2013).

Outside of the **R** ecosystem, the module `icio` (Belotti, Borin, and Mancini 2021) for the software **Stata** (StataCorp 2021) provides the more modern (and methodologically sounder) decomposition method of Borin and Mancini (2023). However, `icio` is a closed-source module, it does not allow complex sector analysis and customization, direct handling of input-output tables nor detailed decompositions (for instance, distinguishing between value added exported induced by inputs and by final goods).

`exvatoools` can therefore be used as all-purpose tool, both to facilitate the extraction and manipulation of input-output matrices and to obtain detailed and customized indicators of trade in value added, facilitating research on global value chains, globalization, and its economic effects. In the following sections, we will describe the methodological background of `exvatoools`, in particular the international input-output table framework and its use to calculate value added induced by gross exports. Then we will explain the creation of the three basic objects of `exvatoools`: a list of basic input-output tables (the `wio` class), a list with the different matrix components of a decomposition of value added in exports (the `exvadec` class) and a list with a detailed origin and destination of value added (the `exvadir` class). We will then explain the commands to fully exploit the information included in the aforementioned classes. Along the way we will produce examples of use.

`exvatoools` can be installed from the CRAN repository and made available for the current session following the usual procedure:

```
install.packages("exvatoools")
library(exvatoools)
```

2 Background methodology

2.1 The international input-output framework

Analyzing the value added in exports requires the previous extraction of a series of basic input-output matrices in a standardized format. We will consider a typical international input-output framework with G countries and N sectors. Each country s provides goods and services from each sector i to each sector j in country r , and sources its goods and services from the sectors of each country t .

Table 1: Structure of an international Input-Output Table

Input	Output	Intermediate uses				Final uses				Output
		1	2	...	G	1	2	...	G	
Intermediate inputs	1	Z_{11}	Z_{12}	...	Z_{1G}	Y_{11}	Y_{12}	...	Y_{1G}	X_1
	2	Z_{21}	Z_{22}	...	Z_{2G}	Y_{21}	Y_{22}	...	Y_{2G}	X_2

	G	Z_{G1}	Z_{G2}	...	Z_{GG}	Y_{G1}	Y_{G2}	...	Y_{GG}	X_G
Value added		VA_1	VA_2	...	VA_G					
Input		X_1	X_2	...	X_G					

In a typical international input-output table (which is a matrix composed of block matrices) \mathbf{Z} is the matrix of intermediate inputs (dimension $GN \times GN$), with each sub-matrix Z_{sr} (dimension $N \times N$) elements z_{sr}^{ij} representing the deliveries of intermediate inputs from sector i in country s to sector j in country r . \mathbf{Y} is the matrix of final demand, of dimension $GN \times G$ (aggregated by country for practical purposes from an original \mathbf{Yfd} matrix with FD demand components). \mathbf{X} is the production matrix, of dimension $GN \times 1$, and \mathbf{VA} the value added demand, of dimension $1 \times GN$.

2.2 The demand model

The typical demand model, which dates back to Leontief (1936), assumes that the inputs from sector i of country s to sector j of country r are a constant proportion of the output of sector j in country r . From there we obtain a matrix of coefficients \mathbf{A} whose elements are the proportion of intermediate inputs over total production, $a_{sr}^{ij} = z_{sr}^{ij} / x_r^j$. Then the relations in the international input-output table can be expressed as $\mathbf{AX} + \mathbf{Y} = \mathbf{X}$, from where we deduct a relation between production and final demand:

$$\mathbf{X} = (\mathbf{I} - \mathbf{A})^{-1} \mathbf{Y} = \mathbf{BY} \quad (1)$$

Here the matrix \mathbf{B} (inverse of $\mathbf{I} - \mathbf{A}$, where \mathbf{I} is the identity matrix) collects the backward linkages that the final demand \mathbf{Y} induces on production. Each element of \mathbf{B} , b_{sr}^{ij} , can be expressed as the increase of production of sector i in country s , when the final demand of sector j in country r increases by one unit.

2.3 Value added in trade

If the ratio between inputs and production is constant, then the ratio between value added (i.e., the value of production minus the value of inputs) and production can also be considered constant, so we can define vector \mathbf{V} as the value added by unit of output \mathbf{X} and express the value added in terms of global demand as $\mathbf{VX} = \mathbf{VBY}$. More specifically, for a given country s :

$$\mathbf{V}_s \mathbf{X}_s = \mathbf{V}_s \sum_j^G \sum_r^G \mathbf{B}_{sj} \mathbf{Y}_{jr} \quad (2)$$

that we can break down into the value added produced and absorbed in s and the value added produced in s and absorbed abroad:

$$\mathbf{V}_s \sum_j^G \mathbf{B}_{sj} \mathbf{Y}_{js} + \mathbf{V}_s \sum_j^G \sum_{r \neq s}^G \mathbf{B}_{sj} \mathbf{Y}_{jr} \quad (3)$$

The second term in (3) is usually referred to as value added exported \mathbf{VAX}_s (Johnson and Noguera 2012). In aggregated terms, value added absorbed abroad coincides with value added exported, but

this is not true in bilateral terms. In fact, VAX_{sr} is the value added produced in s and absorbed in r , but *regardless of the export destination*. To obtain the value added exported to r *regardless of the absorption country* we need to calculate the value added induced not by final demand, but by the demand of gross exports.

For that, knowing that each column of the matrix product \mathbf{VB} is a linear combination whose sum is a unit vector $\boldsymbol{\iota}$, we can break down the linkage effects over the production of any country s into those derived of domestic inputs and those derived of foreign components.

$$\boldsymbol{\iota} = \sum_t^G \mathbf{V}_t \mathbf{B}_{ts} = \mathbf{V}_s \mathbf{B}_{ss} + \sum_{t \neq s}^G \mathbf{V}_t \mathbf{B}_{ts} \quad (4)$$

If we multiply both terms of equation (4) by the demand of gross exports of country s , \mathbf{E}_s , we obtain a basic decomposition of the content of value added in exports into domestic and foreign content.

$$\boldsymbol{\iota} \mathbf{E}_s = \sum_t^G \mathbf{V}_t \mathbf{B}_{ts} \mathbf{E}_s = \mathbf{V}_s \mathbf{B}_{ss} \mathbf{E}_s + \sum_{t \neq s}^G \mathbf{V}_t \mathbf{B}_{ts} \mathbf{E}_s \quad (5)$$

Expression (5) reflects that the demand of gross exports of s can only be satisfied with domestic value added or with foreign value added (with inputs being value added of other sectors). This is also valid for the bilateral exports of s , \mathbf{E}_{sr} (the sum being in this case the total gross bilateral exports).

We have so far overlooked the sector distribution of exports. If we wanted to preserve the sector information, we should use a diagonalized version of matrix \mathbf{V} , $\hat{\mathbf{V}}$, with the resulting product $\hat{\mathbf{V}}\mathbf{B}$ reflecting the sector of origin of value added. If we opted instead (as it normally the case) to reflect the value-added exporting sector, then the products $\hat{\mathbf{V}}_s \mathbf{B}_{ss}$ and $\hat{\mathbf{V}}_t \mathbf{B}_{ts}$ should be diagonalized as $\widehat{\mathbf{V}_s \mathbf{B}_{ss}}$ and $\widehat{\mathbf{V}_t \mathbf{B}_{ts}}$, respectively.

The decomposition of value added in exports essentially consists in analyzing the elements of Equation (5), extracting the elements that are not really value added (double counted) as well as those that are not really exports (flows eventually re-imported and absorbed domestically).

3 Creating and manipulating input-output matrices

3.1 Usage

The basic matrices of a Leontief demand-induced model can be easily obtained from raw data with the command `make_wio()`:

```
make_wio(wiotype = "icio2023", year = NULL,
         src_dir = NULL, quiet = FALSE)
```

where:

- `wiotype` is a string specifying the type of international input-output table to be used (and therefore the source zip file to be looked for in the source directory `src_dir`).
- `year` is an integer specifying the reference year. If missing, `make_wio()` will use the last available year for that database (e.g., 2020 for "icio2023" or 2014 for "wiod2016").
- `src_dir` is the source directory where the raw data is located. If data is stored in the current working directory, this argument can be omitted.
- `quiet` is a boolean, with `TRUE` opting for a silent output.

3.2 Source data

`exvatoools` produces basic input-output tables from three types of data: public international input-output databases, custom data and sample data.

Public international input-output databases can be directly downloaded from the web pages of their respective institutions. Three sources are currently supported:

- OECD Inter-Country Input-Output (ICIO) tables (OECD 2023), on which the OECD's Trade in Value Added database (TiVA) is based,
- World Input Output Database (WIOD) tables (Timmer et al. 2015).

- FIGARO EU Input-Output Tables (EU IC-SUIOTs) (Remond-Tierrez and Rueda-Cantuche 2019)

The default `wiotype` is "icio2023", corresponding to the OECD ICIO Tables, version 2023 (years 1995 to 2020), available as zip files that can be downloaded from the [ICIO web page](#). Older versions of OECD ICIO ("icio2021", "icio2018" and "icio2016") are provided for backward compatibility (for example, to reproduce examples in literature using those databases). For WIOD Tables, versions included are "wiod2016" (version 2016, years 2000 to 2014), "wiod2013" and "lrwiod2022" (long-run WIOD tables, version 2022, for years 1965 to 2000). All of them are available at the University of Groningen's [Growth and Development Centre web page](#). FIGARO tables, whether industry-by-industry ("figaro2022i") or product-by-product ("figaro2022p") are available at the [Eurostat web page](#).

The advantage of these databases is threefold: they are widely used in the economic literature, they are quite rich in terms of countries and sectors, and they are directly downloadable from the web page of their supporting institutions, typically as zipped files containing comma-delimited files (.csv), Excel files (.xlsx) or R data files (.RData). In any case, `exvatoools` can be easily extended to other available international input-output databases like the Eora database (Lenzen et al. 2013) or the ADB Multi-Regional Input Output (ADB-MRIO) Database (Asian Development Bank 2023).

For instance, if we want to use `exvatoools` with the 2023 edition of the OECD ICIO tables (with data up to 2020), we must first download the source file "ICIO_2016-2020-extended.zip" (92 MB) from the [ICIO web page](#). Then we will use the command `make_wio()`, specifying the edition, the year and the folder where the source zip file is saved (just the directory). For instance, if the file was located in C:\Users\Username\Documents\R and we wanted the year 2020:

```
wio <- make_wio("icio2023", year = 2020,
                  src_dir = "C:/Users/Username/Documents/R")
```

and `exvatoools` will take care of the rest: it will extract the .csv files from the zip file and produce the basic input-output matrices.

Alternatively, `exvatoools` can use custom data to create basic input-output matrices. In this case, we just need as input a numeric matrix or data frame with the intermediate inputs **Z** and the final demand **Yfd**, plus a vector with the names of the countries. In this case, we will use an alternative command, `make_custom_wio()`.

```
wio <- make_custom_wio(df, g_names = c("C01", "C02", "C03"))
```

If we just want to check the features of `exvatoools`, there is no need to download any data. The package includes two sets of fictitious data, `wiotype = "iciotest"` (an ICIO-type data sample, with Mexico and China disaggregated) and `wiotype = "wiodtest"` (a WIOD-type data sample). Both can be directly used in `make_wio()`, with no need to specify year or source directory.

As the dimension of publicly available input-output matrices is big, here we will use here ICIO-type fictitious data made with `make_wio("iciotest")`.

```
wio <- make_wio("iciotest")
```

The newly created `wio` class, which can be easily checked with `summary(wio)`, includes the following elements:

- **Z**, the $GN \times GN$ matrix of intermediate inputs. **Zd** is the matrix of domestic intermediate inputs (diagonal block matrix of **Z**) and **Zm** the matrix of international intermediate inputs, exported or imported (off-diagonal block matrix of **Z**).
- **A**, the $GN \times GN$ coefficient matrix. **Ad** is the coefficient matrix of domestic inputs (diagonal block matrix of **A**) and **Am** the matrix of exported or imported inputs (off-diagonal block matrix of **A**).
- **B**, the $GN \times GN$ global inverse Leontief matrix. **Bd** is the global Leontief matrix for domestic linkage effects (diagonal block matrix) and **Bm** is the global Leontief matrix for international linkage effects (off-diagonal block matrix), in both cases using both domestic and international inputs.
- **Ld**, the $GN \times GN$ local inverse Leontief matrix reflecting domestic linkage effects using only domestic inputs.
- **VA**, the $GN \times 1$ value added vector matrix.
- **X**, the $GN \times 1$ production vector matrix.
- **V**, the $GN \times 1$ vector reflecting the value-added share of production (**VA**/**X**). **W** is the diagonal matrix of **V** (usually expressed as \hat{V}).

- \mathbf{Yfd} , the $GN \times GFD$ full demand matrix (with final demand components by country in columns); \mathbf{Y} , the $GN \times G$ final demand matrix, with aggregated total demand by country in columns; \mathbf{Yd} is the domestic final demand (diagonal block matrix of \mathbf{Y}) and \mathbf{Ym} the foreign final demand (non-diagonal block matrix of \mathbf{Y}).
- \mathbf{EXGR} , the $GN \times G$ total gross bilateral exports matrix. \mathbf{E} is the $GN \times GN$ diagonal matrix of aggregated total exports (also shown sometimes as $\hat{\mathbf{E}}$).

Additionally, the main dimensions are provided in the list `dims`, such as the number of countries G , the number of countries including disaggregated countries GX , the number of sectors N , the number of final demand components FD , or combinations thereof (GN , GXN , GFD). A list `names` is also provided, including the names of countries (ISO codes of 3 characters), the names of sectors, (from D01 to D99 for tables based in ISIC revision 4, and from C01 to C99 for tables based in ISIC revision 3), the names of demand components, and two additional metadata: the type of source database ("icio2023", "wiod2016", etc.) and the year.

3.3 Commands for input-output matrix manipulation

Although `exvatoools` was initially conceived as a trade analysis software, it also includes a series of commands that facilitate the manipulation of international input-output tables for any other purposes. Thus, we can multiply a diagonal matrix by an ordinary one with `dmult()`, an ordinary by a diagonal with `multd()`, or make a block-by-block Hadamard product of matrices with `hmult()`. We can also easily obtain a block diagonal matrix with `bkd()`, a block off-diagonal matrix with `bkoffd()`, or a diagonal matrix with the sums of all columns with `diagcs()`.

Additionally, as `exvatoools` always operates with named rows and columns (names of countries and sectors), several commands are included to consolidate matrices, preserving the matrix format and optionally providing names for the resulting rows or columns: `rsums()` to sum rows, `csums()` to sum columns, `sumnrow()` to sum every n th row of a matrix, `sumncol()` to sum every n th column, `sumgrows()` to sum groups of rows of a particular size, `sumgcols()` to do the same with columns, etc.

On the other hand, the OECD ICIO tables have a particular feature: two big industrial countries, China and Mexico, are broken down into two regions each. Calculations must initially be done with disaggregated data, but later consolidated under the name of the country. The command `meld()` takes care of that.

Let us, for instance, check that the production \mathbf{X} is equivalent to the product of the global Leontief inverse matrix \mathbf{B} and the final demand \mathbf{Y} :

```
BY <- wio$B %*% wio$Y
```

We can sum the rows (with `rsums()`, naming the result as "BY") and check that it coincides with the production vector:

```
BY <- rsums(BY, "BY")
print(cbind(head(BY, 10), head(wio$X, 10)))
```

```
#>          BY      X
#> ESP_01T09 1378.568 1378.568
#> ESP_10T39 1914.607 1914.607
#> ESP_41T98 2113.699 2113.699
#> FRA_01T09 1848.173 1848.173
#> FRA_10T39 1799.486 1799.486
#> FRA_41T98 1608.004 1608.004
#> MEX_01T09    0.000    0.000
#> MEX_10T39    0.000    0.000
#> MEX_41T98    0.000    0.000
#> USA_01T09 1895.742 1895.742
```

Now let us calculate the value added absorbed abroad. For that we need to multiply the value added coefficients matrix $\hat{\mathbf{V}}$ (represented here as \mathbf{W}) by the global inverse matrix \mathbf{B} by the final demand matrix \mathbf{Y} , and then exclude the value added absorbed domestically. This can be easily done with a few commands.

To calculate all value added induced by final demand:

```
VBY <- dmult(wio$W, wio$B) %*% wio$Y
VBY
```

```
#>          ESP      FRA      MEX      USA      CHN      ROW
#> ESP_01T09 33.25239 33.04328 29.13415 39.26315 44.36171 40.26254
#> ESP_10T39 105.86567 99.40932 118.73322 139.90848 115.91495 126.97831
#> ESP_41T98 221.93642 191.16663 158.37563 173.76610 145.22181 168.64254
#> FRA_01T09 48.48628 66.62310 51.62458 49.45643 47.66424 70.97642
#> FRA_10T39 120.80031 104.64030 128.78920 102.79096 97.91708 99.78527
#> FRA_41T98 134.74749 129.14500 99.99938 88.70168 86.75694 101.79354
#> MEX_01T09 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
#> MEX_10T39 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
#> MEX_41T98 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
#> USA_01T09 175.01545 130.46673 128.74728 107.71747 102.20179 117.94400
#> USA_10T39 102.29790 84.03672 79.60012 121.89497 66.60450 100.75615
#> USA_41T98 115.50508 118.18725 129.51719 122.16680 95.86500 111.98916
#> CHN_01T09 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
#> CHN_10T39 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
#> CHN_41T98 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
#> ROW_01T09 82.22487 42.82585 45.01008 60.06683 46.78138 51.62906
#> ROW_10T39 90.44372 97.64510 80.63597 95.00232 91.33130 91.05473
#> ROW_41T98 62.56171 51.01318 42.64560 58.70599 41.40437 50.09897
#> MX1_01T09 173.85507 180.14952 154.43342 149.69601 164.13090 149.43125
#> MX1_10T39 119.84480 74.11099 117.78742 97.93871 142.64384 121.97803
#> MX1_41T98 49.45281 29.23090 35.52473 39.69653 36.67997 31.42076
#> MX2_01T09 96.85254 83.89485 70.55941 97.36493 80.35587 66.40230
#> MX2_10T39 138.50490 86.35272 92.84516 108.88726 99.24637 102.95958
#> MX2_41T98 82.91973 67.68778 62.14849 77.03372 65.06803 75.12146
#> CN1_01T09 109.94070 114.47837 91.18810 152.37369 127.84703 118.37641
#> CN1_10T39 119.64017 127.22095 126.45021 165.70116 164.43470 128.00251
#> CN1_41T98 109.34845 93.91197 106.67385 115.50129 111.77386 119.86482
#> CN2_01T09 84.74417 83.13258 69.33664 86.84078 80.88526 72.00972
#> CN2_10T39 73.17244 54.76049 43.08188 56.11525 70.19744 78.67920
#> CN2_41T98 158.98841 103.45062 108.60598 136.04997 128.63627 109.93866
```

The rows for Mexico and China are disaggregated. We can easily meld them with `meld()`:

```
VBY <- meld(VBY)
VBY

#>          ESP      FRA      MEX      USA      CHN      ROW
#> ESP_01T09 33.25239 33.04328 29.13415 39.26315 44.36171 40.26254
#> ESP_10T39 105.86567 99.40932 118.73322 139.90848 115.91495 126.97831
#> ESP_41T98 221.93642 191.16663 158.37563 173.76610 145.22181 168.64254
#> FRA_01T09 48.48628 66.62310 51.62458 49.45643 47.66424 70.97642
#> FRA_10T39 120.80031 104.64030 128.78920 102.79096 97.91708 99.78527
#> FRA_41T98 134.74749 129.14500 99.99938 88.70168 86.75694 101.79354
#> MEX_01T09 270.70761 264.04436 224.99283 247.06094 244.48677 215.83355
#> MEX_10T39 258.34970 160.46370 210.63258 206.82598 241.89021 224.93762
#> MEX_41T98 132.37254 96.91869 97.67322 116.73025 101.74800 106.54222
#> USA_01T09 175.01545 130.46673 128.74728 107.71747 102.20179 117.94400
#> USA_10T39 102.29790 84.03672 79.60012 121.89497 66.60450 100.75615
#> USA_41T98 115.50508 118.18725 129.51719 122.16680 95.86500 111.98916
#> CHN_01T09 194.68487 197.61095 160.52474 239.21447 208.73229 190.38614
#> CHN_10T39 192.81261 181.98144 169.53209 221.81640 234.63214 206.68171
#> CHN_41T98 268.33686 197.36260 215.27984 251.55127 240.41013 229.80349
#> ROW_01T09 82.22487 42.82585 45.01008 60.06683 46.78138 51.62906
#> ROW_10T39 90.44372 97.64510 80.63597 95.00232 91.33130 91.05473
#> ROW_41T98 62.56171 51.01318 42.64560 58.70599 41.40437 50.09897
```

We just want the value added absorbed abroad. For that we need the block off-diagonal matrix of $\hat{\mathbf{VBY}}$, that we can produce with `bkoffd()`:

```
vax <- bkoffd(VBY)
head(vax, 10)

#>          ESP      FRA      MEX      USA      CHN      ROW
```

```
#> ESP_01T09  0.00000 33.04328 29.13415 39.26315 44.36171 40.26254
#> ESP_10T39  0.00000 99.40932 118.73322 139.90848 115.91495 126.97831
#> ESP_41T98  0.00000 191.16663 158.37563 173.76610 145.22181 168.64254
#> FRA_01T09  48.48628 0.00000 51.62458 49.45643 47.66424 70.97642
#> FRA_10T39  120.80031 0.00000 128.78920 102.79096 97.91708 99.78527
#> FRA_41T98  134.74749 0.00000 99.99938 88.70168 86.75694 101.79354
#> MEX_01T09  270.70761 264.04436 0.00000 247.06094 244.48677 215.83355
#> MEX_10T39  258.34970 160.46370 0.00000 206.82598 241.89021 224.93762
#> MEX_41T98  132.37254 96.91869 0.00000 116.73025 101.74800 106.54222
#> USA_01T09  175.01545 130.46673 128.74728 0.00000 102.20179 117.94400
```

4 Decomposition of value added in exports

4.1 Methodology: extracting double counting and re-imports

Equation (5) showed a basic decomposition of the value of gross exports into its domestic and foreign content. However, matrix \mathbf{B} in that equation is the result of successive rounds of production induced by demand, therefore incurring in double counting. To differentiate between true value added and double counting we must specify a spatial perimeter and a sequential perimeter. The spatial perimeter (or perspective) will delimit the border that has to be crossed a specific number of times for a transaction to be considered as value added, and the the sequential perimeter (or approach) will delimit the number of border crossings for a transaction to be considered as value added.

The most consistent decomposition methods employ for the spatial perimeter the exporting country's border (country perspective) and for the sequential perimeter the first border crossing (source-based approach). As a result, export flows are considered as value added *the first time* they cross *the exporting country's border*, with ulterior flows being considered as double counting.

Once we have identified export flows that are double counted (and therefore do not constitute real value added), we must also differentiate the export flows that eventually return to be absorbed in the exporting country (and therefore do not constitute true exports). The methodology to calculate value added in exports is not univocal, and has led to a considerable amount of discussion in the past few years, following Daudin, Riffart, and Schweisguth (2011); Johnson and Noguera (2012); Foster-McGregor and Stehrer (2013); Wang, Wei, and Zhu (2013); Koopman, Wang, and Wei (2014); Los, Timmer, and Vries (2016) and Los and Timmer (2018); Nagengast and Stehrer (2016); Johnson (2018); Arto et al. (2019); Miroudot and Ye (2021) and Borin and Mancini (2023).

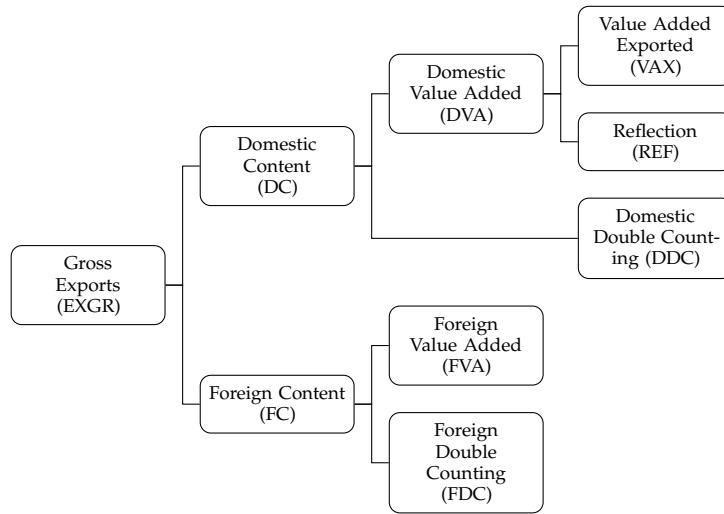
The most recent decomposition methods, including those of Borin and Mancini (2023) or Miroudot and Ye (2021), involve the calculation of a coefficient matrix \mathbf{A} that excludes the linkage effects of exported inputs. Thus, for a given country s we will define an extraction matrix \mathbf{A}^{xs} as a global coefficient matrix whose coefficients corresponding to the exports of inputs of s are equal to zero.

$$\mathbf{A}^{xs} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \cdots & \mathbf{A}_{1s} & \cdots & \mathbf{A}_{1G} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \cdots & \mathbf{A}_{2s} & \cdots & \mathbf{A}_{2G} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \mathbf{A}_{ss} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{G1} & \mathbf{A}_{G2} & \cdots & \mathbf{A}_{Gs} & \cdots & \mathbf{A}_{GG} \end{bmatrix} \quad (6)$$

The global inverse Leontief matrix \mathbf{B}^{xs} derived of this extraction matrix will collect the value added induced by exports excluding intermediate goods. Therefore, the double counting will be the difference between the Leontief inverse global matrix \mathbf{B} and the Leontief inverse global extraction matrix \mathbf{B}^{xs} , from where we can break down the components of equation (5) into the domestic value added $\mathbf{V}_s \mathbf{B}_{ss}^{xs} \mathbf{E}_{sr}$, the double counted domestic value added $\mathbf{V}_s (\mathbf{B}_{ss} - \mathbf{B}_{ss}^{xs}) \mathbf{E}_{sr}$, the foreign value added $\sum_{t \neq s}^G \mathbf{V}_t \mathbf{B}_{ts}^{xs} \mathbf{E}_{sr}$, and the foreign double counting $\sum_{t \neq s}^G \mathbf{V}_t (\mathbf{B}_{ts} - \mathbf{B}_{ts}^{xs}) \mathbf{E}_{sr}$.

Then, after expressing exports in terms of final demand, we we will be able to separate the exports that eventually return home to be absorbed, called reflection. Figure 1 shows the process carried out by `make_exvadec()`, first separating Domestic Content (DC) and Foreign Content (FC), then identifying Domestic Value Added (DVA) and Foreign Value added (FVA), separating them from Domestic Double Counting (DDC) and Foreign Double Counting (FDC), and finally dividing Domestic Value Added between Value Added Exported (VAX) and Reflection (REF).

Using these indicators (or parts thereof) we can obtain additional indicators to measure the participation of countries in global value chains, whether in the form of foreign inputs used in the

Figure 1: Decomposition of the value added embodied in exports

domestic production of exports (backward vertical specialization) or in the form of domestic inputs exported to be used in the production of foreign exports (forward vertical specialization). In case these indicators are available, the denomination will be global value chain participation (GVC), divided in global value chain participation backwards (GVCB) and forward (GVCF).

4.2 Usage

The command `make_exvadec()` (export value added decomposition) provides, from a `wio` object, the full decomposition of the value added in exports for every country or for a particular country or country group, according to different methodologies. The command syntax is as follows:

```
make_exvadec(wio_object, exporter = "all", method = "bm_src",
              output = "standard", quiet = TRUE)
```

with the following arguments:

- `wio_object` is an object of class `wio` obtained through the command `make_wio`.
- `exporter` is a string with the code for the exporting country. Default is "all" (producing an output of dimension $GN \times G$), but can be the code of a country, e.g., "ESP", or country group, e.g., "EU27" (producing an output of dimension $N \times G$).
- `method` is a string specifying the decomposition method.
- `output` is a string specifying the desired output type.
- `quiet` is a boolean indicating whether to produce output silently (default is FALSE).

The available methods and outputs are summarized in Table 2. Selecting "bm_src" will produce a Borin and Mancini (2023) source-based decomposition (from our point of view, the most methodologically sound), "bm_snk" a Borin and Mancini (2023) sink-based decomposition, "my" a Miroudot and Ye (2021) decomposition, "wwz" a Wang, Wei, and Zhu (2013) decomposition, "kww" a Koopman, Wang, and Wei (2014) decomposition and "oeqd" a basic OECD decomposition.

The "standard" output of `make_exvadec()` will include a series of matrices of dimension $GN \times G$ (when `exporter` is "all") or of dimension $N \times G$ (when the exporting country is specified, e.g., "USA"), with breakdown by exporting country and sector, and by importer (country of destination), plus additional metadata:

- **EXGR**, total gross exports by country and exporting sector and by country of destination.
- **DC**, total domestic content in exports by exporting sector and country of destination.
- **DVA**, domestic value added (including value added that returns home).
- **VAX**, or value added effectively exported.
- **REF**, or reflection, value added that is eventually absorbed in the exporting country and therefore does not constitute real exports.
- **DDC**, or domestic double counting, or flows of value added that are reexported after passing a second time for the exporting country and therefore not constituting real value added.
- **FC**, total foreign value added content in exports (including double counting).

Table 2: Output

Decomposition method	Description	Perimeters (approach/perspective)	output options
"bm_src"	Borin and Mancini (2019) source-based	Source-based / various	"basic", "standard", "terms"
"bm_snk"	Borin and Mancini (2019) sink-based	Sink-based / exporting country	"standard", "terms"
"my"	Miroudot and Ye (2021) source-based	Source-based / various	"standard", "terms" "terms2"
"wwz"	Wang et al. (2013)	Mix of both	"standard", "terms", "terms2"
"kww"	Koopman et al. (2014)	Sink-based / mixed	"standard", "terms",
"oecd"	OECD TiVA (not a decomposition)	Not applicable	"standard", "terms", "tiva"

- **FVA**, foreign value added, excluding double counting.
- **FDC**, foreign double content, or flows of foreign value added that have passed more than once by the exporting country, therefore already computed.

In some cases, indicators reflecting the participation in global value chains (vertical specialization) will be provided, mainly foreign value added that eventually takes part in the domestic production of exports (**GVCB** or global value chain participation backwards) or domestic value added that eventually takes part in the foreign production of exports (**GVCF** or global value chain participation forward).

exvadec objects will inherit metadata of the wio object they come from, like dims, names, or source (type), plus the indication of the decomposition method used and, in case of individual decompositions, the exporter. The "standard" output should be enough for most analyses, but alternative outputs can be specified with the argument output. The option "terms" will show all the elements of the decomposition (whose sum is the total value of gross exports), which is useful if we need to distinguish the value added induced by intermediate outputs. The Wang, Wei, and Zhu (2013) decomposition corresponds with the terminology of their Table A2 (pg. 35), but an additional "terms2" is provided (with the terminology of their Table E1, pg. 61).

The "kww" and "wwz" methods are, in fact, a mix of perspectives and approaches. The exporting country perspective and the source approach should probably be considered as the standard, but some alternative approaches (like the sink approach, considering value added all flows prior to the last border crossing) and various tailored perspectives are provided. Thus, the sector, bilateral or bilateral-sector perspectives consider double counting all flows out of those perimeters, and can be calculated for the "bm_src" and the "my" methods (by using the additional arguments partner and sector). Additionally, the "my" method allows a world perspective (using perim = "WLD"), consider as double counting the crossing of *any* border more than once, not only that of the exporting country.

We have included an additional decomposition called "oecd", which is not a true full decomposition method, but represents nevertheless a calculation of several elements of value added in exports. It includes a "tiva" output to show the most typical indicators included in the OECD TiVA database. In this decomposition (unlike in the rest), the bilateral VAX is just VAX absorbed in the partner country.

4.3 Examples

To create a full decomposition of the value added in the exports of Spain using the method of Borin and Mancini (2023), using a exporting country perspective and a source-based approach, we would type:

```
exvadec <- make_exvadec(wio, exporter = "ESP", method = "bm_src")

#> =====
#> DECOMPOSITION OF VALUE ADDED IN EXPORTS OF SPAIN IN 2022
#>      Sector: All sectors
#>      Destination: All countries
#> =====
#> VA_components          USD_MM  Percent
```

```

#> Gross exports of goods and services (EXGR) 4666.96 100.00
#> Domestic Content in VA (DC) 2165.17 46.39
#> Domestic Value Added (DVA) 1880.60 40.30
#> Value Added Exported (VAX) 1624.18 34.80
#> Reflection (REF) 256.41 5.49
#> Domestic Double Counting (DDC) 284.58 6.10
#> Foreign Content in VA (FC) 2501.78 53.61
#> Foreign Value Added (FVA) 2176.21 46.63
#> Foreign Double Counting (FDC) 325.58 6.98
#> Global Value Chain-related trade (GVC) 4034.25 86.44
#> GVC-related trade, backward (GVCB) 2786.36 59.70
#> GVC-related trade, forward (GVCF) 1247.89 26.74
#> =====
#> Method: Borin and Mancini (2023), source-based, standard output
#> Country perspective, source approach

```

If we want to go deeper into the components of this value added, differentiating between final and intermediate exports, we can use:

```

exvadec.terms <- make_exvadec(wio, exporter = "ESP",
                                method = "bm_src", output = "terms")

#> =====
#> DECOMPOSITION OF VALUE ADDED IN EXPORTS OF SPAIN IN 2022
#> Sector: All sectors
#> Destination: All countries
#> =====
#> VA_components USD_MM Percent
#> EXGR (Gross exports of goods and services) 4666.96 100.00
#> T01 VAX1 (DVA, finals) 534.98 11.46
#> T02 VAX2 (DVA, interm. for absorption) 97.73 2.09
#> T03 VAX3 (DVA, interm. for final exports) 314.51 6.74
#> T04 VAX4 (DVA, interm. for reexport) 676.96 14.51
#> T05 REF1 (Reflection, finals) 102.25 2.19
#> T06 REF2 (Reflection, intermediates) 154.17 3.30
#> T07 DDC (Domestic Double Counting) 284.58 6.10
#> T08 FVA1 (FVA, finals) 631.12 13.52
#> T09 FVA2 (FVA, interm. for absorption) 112.29 2.41
#> T10 FVA3 (FVA, interm. for final exports) 478.65 10.26
#> T11 FVA4 (FVA, interm. for reexport) 954.15 20.44
#> T12 FDC (Foreign Double Counting) 325.58 6.98
#> =====
#> Method: Borin and Mancini (2023), source-based, terms output
#> Country perspective, source approach

```

If we want to get a list of common trade indicators (exports, imports, value added, production) similar to those of the TiVA database, we could just use `make_exvadec()` with the method "oecd" and `output = "tiva"`.

```

tiva <- make_exvadec(wio, exporter = "ESP",
                      method = "oecd", output = "tiva")

#> =====
#> DECOMPOSITION OF VALUE ADDED IN EXPORTS OF SPAIN IN 2022
#> Sector: All sectors
#> Destination: All countries
#> =====
#> VA_components USD_MM Percent
#> Gross exports of goods and services (EXGR) 4666.96 100.00
#> Gross exports, finals (EXGR_FNL) 1343.03 28.78
#> Gross exports, intermediates (EXGR_INT) 3323.92 71.22
#> Gross imports (IMGR) 5292.12 113.40
#> Gross imports, finals (IMGR_FNL) 2386.14 51.13
#> Gross imports, intermediates (IMGR_INT) 2905.98 62.27

```

```
#> Domestic absorption (DOM) 739.92 15.85
#> Domestic absorption, finals (DOM_FNL) 224.26 4.81
#> Domestic absorption, interm. (DOM_INT) 515.66 11.05
#> Gross balance (BALGR) -625.17 -13.40
#> Domestic Content in VA (EXGR_DVA) 2165.17 46.39
#> Direct domestic VA content (EXGR_DDC) 1757.25 37.65
#> Indirect domestic VA content (EXGR_IDC) 123.34 2.64
#> Reimported domestic VA content (EXGR_RIM) 284.58 6.10
#> Value Added in final demand (FD_VA) 1985.24 42.54
#> DVA in dom. final dem. (VAD) (DXD_DVA) 361.05 7.74
#> DVA in foreign final dem. (VAX) (FFD_DVA) 1624.18 34.80
#> FVA in dom. final dem. (VAM) (DFD_FVA) 2249.35 48.20
#> Balance of VA (VAX - VAM) (BALVAFD) -625.17 -13.40
#> Foreign VA Content (EXGR_FVA) 2501.78 53.61
#> Backward participation in GVC (DEXFVAP) 2501.78 53.61
#> Forward participation in GVC (FEXDVAP) 3047.87 65.31
#> Value added (VA) 1985.24 42.54
#> Production (PROD) 5406.87 115.85
#> =====
#> Method: OECD (2022), TiVA output
#> Country perspective, source approach
```

5 Direction (origin and destination) of value added

The decomposition of `make_exvadec()` does not distinguish between the different sources of foreign value added. This is where the command `make_exvadir()` might be useful. It provides data on the direction of value added, i.e., details of both the geographical and sectoral origin of the value added incorporated in exports and of the final destination (in gross terms or in terms of final demand). It allows therefore a thorough analysis of where the value added is generated and where it ends up (for instance, how EU services are important for UK's exports of goods, or the role of China as intermediate party in the exports of Russia).

5.1 Methodology

The command `make_exvadir()` produces an output which is the result of multiplying the value added matrix \hat{V} by the global Leontief inverse matrix \hat{B} by a matrix of exports \mathbf{EXGR} , but with a high level of specification of the three matrices. First, the matrix \hat{V} (V diagonalized, to preserve the sector information) will be multiplied by a specific form of \hat{B} : regular \hat{B} to obtain the total value added content, \mathbf{B}_d to obtain the domestic value added content and \mathbf{B}_m to obtain the foreign value added content; or its equivalents in the form of extraction matrices \mathbf{B}^{xs} to obtain the total, domestic or foreign value added excluding double counting, according to the methodology of Borin and Mancini (2023).

Then, depending of the sectoral perspective, the product $\hat{V}\hat{B}$ will be left as it is (sector of origin) or will be summed up by columns and diagonalized, i.e., as \widehat{VB} (default option, exporting sector perspective). The specification of sectors or countries of origin of value added will be done by setting the non-specified values to zero. If the exporter is a group of countries, there is the possibility of considering intra-regional flows as exports or not (by default, intra-regional flows will be excluded).

Finally, the resulting adjusted product \mathbf{VB} will be multiplied by a form of export matrix \mathbf{EXGR} . By default, the ordinary gross export matrix will be considered, but the option will be given to express exports in terms of absorption (i.e., final demand), as \mathbf{EXGRY} , giving in this case the possibility of distinguishing between destination of final goods (\mathbf{Y}_m) and destination of intermediate exports processed as final goods ($\mathbf{A}_m \mathbf{BY}$: this is the only way of calculating the value added induced by intermediate exports).

Additionally, the possibility will be given to consider exports that go via a specific country, i.e., an intermediate importer (for example, Russian exports to the EU that go via China). If this is the case, \widehat{VB} will be multiplied by $\mathbf{Y}_{sr} + \mathbf{A}_{sr}[\mathbf{BY}]_r$, with s being the exporter and r the intermediate importer, and $[\mathbf{BY}]_r$ the rows of product \mathbf{BY} for country r), i.e., the exports of s used in the production of r , that ends up in r or exported elsewhere.

5.2 Usage

The syntax of `make_exvadir()` is:

```
make_exvadir(wio_object, va_type = "TC", flow_type = "EXGR", exporter,
             via = "any", sec_orig = "all", geo_orig = "all",
             intra = FALSE, perspective = "exporter")
```

The arguments are as follows:

- `wio_object` is an object of class `wio` (required).
- `va_type` is a string describing the type of value added: "TC", the default, is the total content in value added (matrix \mathbf{B}), both foreign and domestic, but we can also select "DC" (the domestic value added content, using matrix \mathbf{B}_d) or "FC" (the foreign value added content, using matrix \mathbf{B}_m). "TVA" is the total pure value added, i.e., excluding double counting (matrix \mathbf{B}^{xs}), and we can also get the "DVA" (domestic value added, excluding double counting, matrix \mathbf{B}_d^{xs}) or the "FVA" (foreign value added, excluding double counting, matrix \mathbf{B}_m^{xs}).
- `flow_type` is a string specifying the type of export flow. Default is gross total exports ("EXGR"). Alternatives are exports expressed in terms of final demand: total exports ("EXGRY"), final exports ("EXGRY_FIN") or intermediate exports ("EXGRY_INT"). The latter options will show where value added exported is eventually absorbed, regardless of where it was initially exported.
- `exporter` is a string reflecting the code of the exporting country or group of countries.
- `via` is a string with the code of the intermediate importing country or country group. Default is "any". This option requires flows to be expressed necessarily in terms of final demand.
- `geo_orig` is a string with the code of the country or country group origin of value added. Default is "all".
- `sec_orig` is string with the code of sector of origin of value added (e.g., "AGR", "MANUF", "SRVWC"...). Default is "all".
- `intra` is a boolean to specify whether to include or not intra-region exports (default is FALSE, i.e., EU27 exports will include only extra-EU exports).
- `perspective` shows the sectoral perspective of value added. Default is exporting sector ("exporter") but sector of "origin" can also be specified.

Please note that, compared to `make_exvadec()`, `make_exvadir()` is logically more restricted at decomposing value added, so in principle calculations will be shown in terms of domestic and foreign content (DC/FC) or, at most, domestic and foreign value added (DVA/FVA, excluding double counting), but including reflection (REF). Also note that the total content in value added from all origins and all sectors is, precisely, the value of total gross exports.

5.3 Example

We have seen that the foreign content in Spanish exports amounts to USD 2501.78 million. Where does it come from? The specific geographical and sector origin of the value added in exports can be obtained using the command `make_exvadir()`:

```
exvadir <- make_exvadir(wio, exporter = "ESP", va_type = "FC",
                         flow_type = "EXGR")
head(exvadir$FC, 10)

#>           ESP        FRA        MEX        USA        CHN        ROW
#> ESP_01T09  0  0.00000  0.00000  0.00000  0.00000  0.00000
#> ESP_10T39  0  0.00000  0.00000  0.00000  0.00000  0.00000
#> ESP_41T98  0  0.00000  0.00000  0.00000  0.00000  0.00000
#> FRA_01T09  0 19.45318 26.47295 15.19438 35.51728 24.42115
#> FRA_10T39  0 14.09801 34.93554 23.48504 32.13613 27.09630
#> FRA_41T98  0 15.84674 25.71533 14.12113 17.62330 14.63281
#> MEX_01T09  0 38.41321 52.27477 30.00356 70.13415 48.22319
#> MEX_10T39  0 31.07698 77.01024 51.76931 70.83935 59.72978
#> MEX_41T98  0 33.18999 53.85912 29.57581 36.91088 30.64749
#> USA_01T09  0 23.35813 31.78700 18.24443 42.64685 29.32333
```

Note that the `exvadir` object that we have obtained is different from the `exvadec` object, in the sense that 'exporters' in an `exvadir` object are the different countries and sectors of origin of the value added included in the exports of a specific country (in this case, Spain). We can better understand this by typing `summary(exvadir)`:

```
summary(exvadir)
```

```
#>
#> =====
#> ORIGIN AND DESTINATION OF VALUE ADDED IN EXPORTS OF SPAIN IN 2022
#> =====
#> Value added type: Foreign VA content (FC)
#> In type of flow: Total gross exports (EXGR)
#> That goes via country: any
#> Using inputs from sector: all sectors
#> Of country: all countries
#> With sector perspective: exporter
#> =====
#>
#> Available countries of origin of VA (G): 6
#> ESP, FRA, MEX, USA, CHN, ROW
#>
#> Available sectors of origin of VA (N): 3
#> D01T09, D10T39, D41T98
#>
#> Available destinations of VA (G): 6
#> ESP, FRA, MEX, USA, CHN, ROW
#>
```

Now we will see how to maximize the information given by these objects.

6 Sector and geographic analysis

The commands `make_wio()`, `make_exvadec()` and `make_exvadir()` produce objects that are lists of matrices will a full breakdown by sector and countries of destination. However, most analyses require grouping of those variables. The advantage is that grouping by sector or by destination do not require additional computing, just an aggregation of variables.

To check the information about sectors, it suffices to print `info_sec()`:

```
info_sec("iciotest")

#>
#> =====
#> Test Input Output Table, ICIO-type, 2022 edition
#> =====
#>
#> Individual sectors:
#> PRIMARY: D01T09 (Primary sector), MANUF: D10T39 (Manufacturing),
#> SRVWC: D41T98 (Services, including construction)
#>
#> Sector groups:
#> TOTAL: D01T98 (Total goods and services), GOODSWU: D01T39 (Goods,
#> total, incl. utilities)
```

To check the information about available countries, the command is `info_geo()`:

```
info_geo("iciotest")

#>
#> =====
#> Test Input Output Table, ICIO-type, 2022 edition
#> =====
#>
#> Individual countries:
#> FRA (France), MEX (Mexico), ESP (Spain), USA (United States), CHN
#> (China), ROW (Rest of the world)
#>
#> Groups of countries:
#> WLD (World), EU27 (EU-27), NONEU27 (Non-EU27), NAFTA (NAFTA), USMCA
#> (USMCA)
```

These commands do not require to have a wio in the environment, so we can just check what countries are available in the OECD's ICIO tables, 2023 edition.

```
info_geo("icio2023")

#> =====
#> OECD's Inter-Country Input-Output Table (ICIO), 2023 edition
#> =====
#>
#> Individual countries:
#> AUS (Australia), AUT (Austria), BEL (Belgium), CAN (Canada), CHL
#> (Chile), CZE (Czech Republic), DNK (Denmark), EST (Estonia), FIN
#> (Finland), FRA (France), DEU (Germany), GRC (Greece), HUN (Hungary),
#> ISL (Iceland), IRL (Ireland), ISR (Israel), ITA (Italy), JPN (Japan),
#> KOR (Korea), LVA (Latvia), LTU (Lithuania), LUX (Luxembourg), MEX
#> (Mexico), NLD (Netherlands), NZL (New Zealand), NOR (Norway), POL
#> (Poland), PRT (Portugal), SVK (Slovak Republic), SVN (Slovenia), ESP
#> (Spain), SWE (Sweden), CHE (Switzerland), TUR (Turkey), GBR (United
#> Kingdom), USA (United States), ARG (Argentina), BGD (Bangladesh), BLR
#> (Belarus), BRA (Brazil), BRN (Brunei Darussalam), BGR (Bulgaria), KHM
#> (Cambodia), CMR (Cameroon), CHN (China), COL (Colombia), CRI (Costa
#> Rica), CIV (Côte d'Ivoire), HRV (Croatia), CYP (Cyprus), EGY (Egypt),
#> IND (India), IDN (Indonesia), JOR (Jordan), HKG (Hong Kong, China),
#> KAZ (Kazakhstan), LAO (Laos), MYS (Malaysia), MLT (Malta), MAR
#> (Morocco), MMR (Myanmar), NGA (Nigeria), PAK (Pakistan), PER (Peru),
#> PHL (Philippines), ROU (Romania), RUS (Russia), SAU (Saudi Arabia),
#> SEN (Senegal), SGP (Singapore), ZAF (South Africa), TWN (Chinese
#> Taipei), THA (Thailand), TUN (Tunisia), UKR (Ukraine), VNM (Vietnam),
#> ROW (Rest of the world)
#>
#> Groups of countries:
#> WLD (World), EU28 (EU-28), EU27 (EU-27), OECD (OECD), EMU (EMU),
#> NONEU28 (Non-EU28), NONEU27 (Non-EU27), NONOECD (Non-OECD),
#> EU28NONEMU (EU-28 not EMU), EU27NONEMU (EU-27 not EMU), EURNONEU
#> (Rest of Europe), EUROPE (Europe), AMER (America), NAMER (North
#> America), CSAMER (Central and South America), LATAM (Latin America
#> and Caribbean), AFRI (Africa), ASIA (Asia), OCEA (Oceania), ASIAOC
#> (Asia and Oceania), G20 (G-20), G7 (G7), NAFTA (NAFTA), USMCA
#> (USMCA), EEA (EEA), EFTA (EFTA), APEC (APEC), ASEAN (ASEAN), RCEP
#> (RCEP)
```

Additionally, the commands `get_geo_codes()` and `get_sec_codes()` provide details about the components of the different groups. These commands are also directly applicable for any available input-output table. For instance, for "wiod2016" we would have the following components of NAFTA:

```
get_geo_codes("NAFTA", wiotype = "wiod2016")

#> [1] "CAN|MEX|USA"
```

And for "icio2023" we have the following components of the information services sector (INFO):

```
get_sec_codes("INFO", wiotype = "icio2023")

#> [1] "D58T60|D61|D62T63"
```

Once we know the sector and geographical disaggregation, we can discuss how to take advantage of the information contained in `exvatoools` objects.

7 Breaking down decompositions

Once we have obtained a decomposition, we can play with the results in terms of sectors and countries of destination just using the command `get_exvadec_bkdown()`. For instance, to select the value added in Spanish exports of services (including construction) to the United States, we just have to type:

```

get_exvadec_bkdown(exvadec, exporter = "ESP",
                     sector = "SRVWC", importer = "USA")

#> =====
#> DECOMPOSITION OF VALUE ADDED IN EXPORTS OF SPAIN IN 2022
#>   Sector: Services, including construction (SRVWC)
#>   Destination: United States (USA)
#> =====
#> VA_components          USD_MM Percent
#> Gross exports of goods and services (EXGR) 276.71 100.00
#>   Domestic Content in VA (DC)      159.63 57.69
#>   Domestic Value Added (DVA)      145.95 52.74
#>   Value Added Exported (VAX)      127.28 46.00
#>   Reflection (REF)                18.67  6.75
#>   Domestic Double Counting (DDC)  13.68  4.94
#>   Foreign Content in VA (FC)     117.08 42.31
#>   Foreign Value Added (FVA)      101.53 36.69
#>   Foreign Double Counting (FDC)  15.55  5.62
#> Global Value Chain-related trade (GVC) 221.62 80.09
#> GVC-related trade, backward (GVCB) 130.76 47.26
#> GVC-related trade, forward (GVCF)  90.86 32.84
#> =====
#> Method: Borin and Mancini (2023), source-based, standard output
#> Country perspective, source approach

```

Note that the bilateral VAX in this case shows the value added exported to the United States, regardless of the final absorption country.

We can also produce an exvadec object will all countries and then select any exporting country and any partner or sector:

```
exvadec.all <- make_exvadec(wio, exporter = "all", quiet = TRUE)
```

and then:

```

get_exvadec_bkdown(exvadec.all, exporter = "USA",
                     sector = "MANUF", importer = "CHN")

#> =====
#> DECOMPOSITION OF VALUE ADDED IN EXPORTS OF UNITED STATES IN 2022
#>   Sector: Manufacturing (MANUF)
#>   Destination: China (CHN)
#> =====
#> VA_components          USD_MM Percent
#> Gross exports of goods and services (EXGR) 400.89 100.00
#>   Domestic Content in VA (DC)      164.19 40.96
#>   Domestic Value Added (DVA)      138.36 34.51
#>   Value Added Exported (VAX)      114.29 28.51
#>   Reflection (REF)                24.06  6.00
#>   Domestic Double Counting (DDC)  25.83  6.44
#>   Foreign Content in VA (FC)     236.70 59.04
#>   Foreign Value Added (FVA)      204.59 51.03
#>   Foreign Double Counting (FDC)  32.12  8.01
#> Global Value Chain-related trade (GVC) 379.14 94.58
#> GVC-related trade, backward (GVCB) 262.53 65.49
#> GVC-related trade, forward (GVCF)  116.61 29.09
#> =====
#> Method: Borin and Mancini (2023), source-based, standard output
#> Country perspective, source approach

```

Apart from the console printout, get_exvadec_bkdown() will output a matrix with the results.

8 Extracting data from exvatoools objects

The command `get_data()` takes an exvatoools object (`wio` or `exvadec`) and produces a value or a matrix of values. It allows a quick extraction of data. The main advantage of `get_data()` is that it does not only admit individual sectoral codes ("MANUF") or destination country codes ("USA", "NAFTA"), but also lists of sectors and countries in vector form. Therefore, we can easily produce a matrix of domestic value added with breakdown by sector and by destination groups. The syntax of `get_data()` is as follows:

```
get_data(exvatoools_object, variable, exporter = NULL,
        sector = "TOTAL", importer = "WLD", custom = FALSE)
```

The arguments are as follows:

- `exvatoools_object` (required) is an object of class `wio`, `exvadec` or `exvadir`, and `variable` is a string specifying one of the variables included in the `exvatoools_object`, such as "EXGR", "VAX", "FVA", etc.
- `exporter` is a string vector with one or more codes of exporters or groups of exporters, such as "ESP", "EU27", `c("WLD", "EU27", "NONEU27")`, etc. This will define the rows in the resulting matrix. Specific exclusions can be accepted through the excluding code `x` (lowercase `x`), so "EU27xESP" would be EU countries, excluding Spain, "WLDxEU27" would be total world except EU, and so on. More than one exception can be specified with the "|" element, such as in "WLDxESP|FRA|ITA". Available countries and country group id codes can be checked with the command `info_geo()`. The argument `exporter` is required in all cases except two: in case of a country-specific `exvadec` object, in which the exporter is previously defined, and in case of an `exvadir` object, where, by definition, there is only one exporter. Note that, in the case of an `exvadir` object obtained for country `s`, the argument `exporter` does not refer to country `s` itself, but to the $t = 1 \dots G$ exporters of value added that is used by country `s` to produce its exports. Therefore, if the `exporter` argument is missing, the default behavior of `get_data()` will be different, depending on the case: for a country-specific `exvadec`, it will default to the exporting country, whereas for an `exvadir` object it will default to "WLD" (i.e., sum of all origins of value added).
- `sector` is a string vector with one or more codes of sectors or groups of sectors, such as "MANUF", "SERVS", "TOTAL", `c("TOTAL", "GOODSWU", "SRVWC")`, etc. These will also show as rows in the result. Specific exclusions can be accepted through the excluding code `x`, so "MANUFxPET" would be manufactures, excluding oil products, "SRVWCxBIZSV" would be total services (with construction) except business services, and so on. Available sector id codes can be checked with the command `info_sec()`. Default option is "TOTAL" (sum of all sectors for the specific exporter). The option "all" can be used to specify all sectors.
- `importer` is a character string or vector with one or more codes of importing countries or groups of countries, such as "ESP", "EU27", `c("WLD", "EU27", "NONEU27")`, etc. This defines the columns of the result. Default option is "WLD", i.e, the sum of all importers for the specific exporter (and sector) selection.
- `custom` is a boolean specifying whether custom-made groups of countries or sectors present in the environment should be looked up by `get_data()`. For instance, HITECH could be a specific variable including all high-tech sectors, or LDC could be a list of least-developed countries. Custom variables should be referred to as strings in `get_data()`, so, for instance, `get_data(exva, "VAX", exporter = "LDC", custom = TRUE)` would look for a variable called `LDC` in the environment and would use its codes to extract and group data.

We can use `get_data()` to summarize the foreign content of Spanish exports, with a breakdown between EU and Non-EU origin (specifying a few countries) and also distinguishing between goods (with utilities) and services. We can also break down the destination of those exports between EU and non-EU countries:

```
get_data(exvadir, exporter = c("WLD", "EU27", "FRA",
                               "NONEU27", "USA"),
        sector = c("TOTAL", "GOODSWU", "SRVWC"),
        importer = c("WLD", "EU27", "NONEU27"))

#>          WLD     EU27    NONEU27
#> WLD_TOTAL 2501.78421 367.55781 2134.22640
#> WLD_GOODSWU 1772.66226 236.16941 1536.49285
#> WLD_SRVWC   729.12195 131.38840  597.73355
```

```
#> EU27_TOTAL      340.74928 49.39794 291.35134
#> EU27_GOODSWU   252.80996 33.55120 219.25877
#> EU27_SRVWC     87.93932 15.84674 72.09258
#> FRA_TOTAL      340.74928 49.39794 291.35134
#> FRA_GOODSWU   252.80996 33.55120 219.25877
#> FRA_SRVWC     87.93932 15.84674 72.09258
#> NONEU27_TOTAL 2161.03493 318.15987 1842.87505
#> NONEU27_GOODSWU 1519.85229 202.61821 1317.23408
#> NONEU27_SRVWC  641.18263 115.54166 525.64097
#> USA_TOTAL       433.65389 64.94868 368.70521
#> USA_GOODSWU    286.90179 38.50383 248.39796
#> USA_SRVWC      146.75210 26.44485 120.30725
```

If there is not a specific group in the database of sectors or countries, the user has two options: to create a group and use `get_data()` with the option `custom = TRUE`, or simply to use a combination of countries or sectors with a vertical line " | ".

In the latter case, for instance, if we want to combine ESP and MEX in a single group, we can just type:

```
get_data(exvadec.all, "VAX", exporter = "ESP|MEX",
         sector = c("TOTAL", "MANUF", "SRVWC"),
         importer = "USA")

#>          USA
#> ESP|MEX_TOTAL 756.1258
#> ESP|MEX_MANUF 251.3334
#> ESP|MEX_SRVWC 257.0910
```

If the vertical line " | " is used to join, the exception marker is "x". It allows us, for instance, to calculate NAFTA exports, both intra-regional and extra-regional, employing services and non-services, using as extra-regional "WLDxNAFTA" and as non-services "TOTALxSRVWC"

```
get_data(exvadec.all, "EXGR", exporter = "NAFTA",
         sector = c("TOTAL", "TOTALxSRVWC", "SRVWC"),
         importer = c("WLD", "NAFTA", "WLDxNAFTA"))

#>          WLD    NAFTA WLDxNAFTA
#> NAFTA_TOTAL     13087.740 2602.338 10485.402
#> NAFTA_TOTALxSRVWC 8736.552 1502.340 7234.212
#> NAFTA_SRVWC     4351.188 1099.998 3251.189
```

Let us use `get_data()` to calculate the relative comparative advantage (RCA) in terms of gross exports and compare it with that in terms of VAX. The RCA is the relation between the proportion of exports of sector i in country s to total exports of s (E_{si}/E_s) and the proportion of world exports of sector i to total world exports (E_{wi}/E_w). If RCA is more than 1, it means that country s has a relative specialization (and a comparative advantage) in sector i compared to the world average.

We will create a function to calculate the RCA so it can be used with both gross exports (EXGR) and value added exported (VAX). As we can see, `get_data()` considerably simplifies the calculation of country exports and sector exports.

```
RCA <- function(exva, exvar) {
  Esi <- get_data(exva, exvar, exporter = "all", sector = "all")
  Es <- get_data(exva, exvar, exporter = "all", sector = "TOTAL")
  Es <- rep(as.numeric(Es), each = exva$dim$N)
  Ewi <- get_data(exva, exvar, exporter = "WLD", sector = "all")
  Ewi <- rep(as.numeric(Ewi), exva$dim$G)
  Ew <- as.numeric(get_data(exva, exvar, exporter = "WLD", sector = "TOTAL"))
  rca <- (Esi/Es)/(Ewi/Ew)
  colnames(rca) <- paste0("RCA", ".", exvar)
  return(rca)
}
head(cbind(RCA(exvadec.all, "EXGR"),
           RCA(exvadec.all, "VAX")), 10)
```

```
#>           RCA.EXGR   RCA.VAX
#> ESP_01T09 0.7981206 0.4575154
#> ESP_10T39 1.1110328 1.0953243
#> ESP_41T98 1.0883393 1.3971980
#> FRA_01T09 1.0378051 0.7268141
#> FRA_10T39 1.0686493 1.1512224
#> FRA_41T98 0.8963249 1.0970064
#> MEX_01T09 1.0752922 1.3324652
#> MEX_10T39 0.9908369 1.0324864
#> MEX_41T98 0.9356388 0.6658469
#> USA_01T09 1.0464957 1.2057567
```

We can see that some relative advantage ($RCA > 1$) in terms of gross exports disappear when calculated in terms of VAX, while some other appear.

Another useful application is the calculation of bilateral balances. We can see that there are considerable difference in bilateral balances when calculated in terms of value added compared to the same balances using gross exports.

```
EXGR <- get_data(exvadec.all, "EXGR", "all", importer = "all")
IMGR <- bkt(EXGR)
VAX <- get_data(exvadec.all, "VAX", "all", importer = "all")
VAM <- bkt(VAX)
BALGR <- round(EXGR - IMGR, 0)
BALVA <- round(VAX - VAM, 0)
as.data.frame(cbind(BALGR, " =" , BALVA),
               row.names = exvadec.all$names$g_names)

#>     ESP   FRA   MEX   USA   CHN   ROW     ESP   FRA   MEX   USA   CHN   ROW
#> ESP    0    8 -394 -130 -158   49      0   30 -151 -34  -92  111
#> FRA   -8    0   53 -290 -544  173    -30   0  -28 -127 -253   96
#> MEX  394  -53    0  263  189  459   151   28   0   84   68  279
#> USA  130  290 -263    0 -570  -19    34  127  -84    0 -297   74
#> CHN  158  544 -189  570    0  464   92  253  -68   297    0  319
#> ROW -49 -173 -459   19 -464    0   -111 -96 -279  -74 -319    0
```

9 Other useful commands

exvatoools provides several additional commands, some of them resulting from the mere combination of `make_exvadir()` and `get_data()` with specific default arguments. Of course, it would be easy to create other custom combinations.

9.1 Detailed origin of value added

The command `get_va_exgr()` provides a detailed sector and geographical origin and destination of value added.

```
get_va_exgr(wio_object, va_type = "FC",
            geo_orig = "all", sec_orig = "TOTAL",
            geo_export, sec_export = "TOTAL", as_numeric = TRUE)
```

It allows to analyze, for instance, the percentage of services (both domestic and foreign) embedded in Spanish exports of manufactures, i.e. the so-called ‘servicification’ of Spanish exports:

```
get_va_exgr(wio, va_type = "TC",
            geo_orig = c("ESP", "WLDxESP"), sec_orig = "SRVWC",
            geo_export = "ESP", sec_export = "MANUF",
            as_numeric = FALSE)

#>           WLD
#> ESP_MANUF     84.95594
#> WLDxESP_MANUF 263.71717
```

On the other hand, if we wanted the value added in services of the US incorporated in the Spanish exports of manufactures, i.e., the Spanish dependence of US services to produce exports of manufactures:

```
get_va_exgr(wio, geo_orig = "USA", sec_orig = "SRVWC",
             geo_export = "ESP", sec_export = "MANUF")

#> [1] 51.31605
```

9.2 Detailed final absorption of value added

Sometimes we are not only interested in the origin, but also in the country of final absorption. For that we have the command `get_va_exgry()`. This is equivalent to the OECD's Gross Exports by Origin of Value Added and Final destination (FD_EXGR_VA, FD_EXGRFNLD_VA and FD_EXGRINT_VA), but with much more flexible geographical and sector options. Since the OECD TiVA database no longer provides this indicator in their online version, this command becomes particularly useful.

```
get_va_exgry(wio_object, va_type = "TC", flow_type = "EXGRY",
              geo_orig = "WLD", geo_export, sec_export = "TOTAL",
              geo_fd = "WLD", as_numeric = TRUE)
```

Here `flow_type` are exports (expressed in terms of final demand), whether total ("EXGRY"), final ("EXGRY_FIN") or intermediate ("EXGRY_INT"), and `geo_fd` is the country or region of final demand.

This allows, for instance, to calculate what part of US value added incorporated in China's exports of manufactures ends up absorbed back in the US.

```
get_va_exgry(wio, geo_orig = "USA", geo_export = "CHN",
              sec_export = "MANUF", geo_fd = "USA")

#> [1] 53.81984
```

9.3 Value added induced by final demand

`exvatoools` also provides a simple command to obtain details of both the geographical and sector origin of the value added incorporated in exports induced by final demand.

```
get_va_fd(wio_object, va_type = "TOTAL",
           geo_orig = "WLD", sec_orig = "TOTAL",
           geo_fd = "WLD", sec_fd = "TOTAL", intra = FALSE)
```

This would allow, for instance, the calculation of the Chinese total value added (or GDP) induced by US final demand for manufactures:

```
get_va_fd(wio, geo_orig = "CHN", sec_orig = "TOTAL",
           geo_fd = "USA", sec_fd = "MANUF")

#>          WLD
#> CHN_TOTAL 229.385
```

10 Summary and conclusions

We have presented the package `exvatoools` for decomposition of value added in exports using international-input out tables in R. `exvatoools` provides a convenient tool for calculating and manipulating international input-output matrices, and simplifies the decomposition of value added in exports (using alternative methodologies). It also allows a straightforward calculation of custom value added indicators.

The purpose of `exvatoools` is to provide the scientific community with tools to take advantage of the valuable statistical resources that are the international input-output tables, facilitating the analysis of the complex interaction between trade in goods and services and between economic sectors in different countries.

References

- Arto, Iñaki, Erik Dietzenbacher, José Manuel Rueda-Cantuche, European Commission, and Joint Research Centre. 2019. "Measuring Bilateral Trade in Terms of Value Added." JRC Technical Report. <https://doi.org/10.2760/639612>.
- Asian Development Bank. 2023. "Multiregional Input-Output Database (ADB-MRIO)." <https://kidb.adb.org/mrio>.
- Belotti, Federico, Alessandro Borin, and Michele Mancini. 2021. "Icio: Economic Analysis with Intercountry Input–Output Tables." *The Stata Journal* 21 (3): 708–55. <https://doi.org/10.1177/1536867X211045573>.
- Borin, Alessandro, and Michele Mancini. 2023. "Measuring What Matters in Value-Added Trade." *Economic Systems Research*, January, 1–28. <https://doi.org/10.1080/09535314.2022.2153221>.
- Daudin, Guillaume, Christine Rifflart, and Danielle Schweisguth. 2011. "Who Produces for Whom in the World Economy?" *Canadian Journal of Economics* 44 (4): 1403–37. <https://doi.org/10.1111/j.1540-5982.2011.01679.x>.
- Foster-McGregor, Neil, and Robert Stehrer. 2013. "Value Added Content of Trade: A Comprehensive Approach." *Economics Letters* 120 (2): 354–57. <https://doi.org/10.1016/j.econlet.2013.05.003>.
- Johnson, Robert C. 2018. "Measuring Global Value Chains." *Annual Review of Economics* 10 (1): 207–36. <https://doi.org/10.1146/annurev-economics-080217-053600>.
- Johnson, Robert C., and Guillermo Noguera. 2012. "Accounting for Intermediates: Production Sharing and Trade in Value Added." *Journal of International Economics* 86 (2): 224–36. <https://doi.org/10.1016/j.jinteco.2011.10.003>.
- Koopman, Robert, Zhi Wang, and Shang-Jin Wei. 2014. "Tracing Value-Added and Double Counting in Gross Exports." *American Economic Review* 104 (2): 459–94. <https://doi.org/10.1257/aer.104.2.459>.
- Lenzen, Manfred, Daniel Moran, Keiichiro Kanemoto, and Arne Geschke. 2013. "Building Eora: A Global Multi-Region Input–Output Database at High Country and Sector Resolution." *Economic Systems Research* 25 (1): 20–49. <https://doi.org/10.1080/09535314.2013.769938>.
- Leontief, Wassily W. 1936. "Quantitative Input and Output Relations in the Economic System of the United States." *Review of Economics and Statistics* 18: 105–25.
- Los, Bart, and Marcel P. Timmer. 2018. "Measuring Bilateral Exports of Value Added: A Unified Framework." NBER Working Paper w24896. NBER. <https://doi.org/10.3386/w24896>.
- Los, Bart, Marcel P. Timmer, and Gaaitsen J. de Vries. 2016. "Tracing Value-Added and Double Counting in Gross Exports: Comment." *American Economic Review* 106 (7): 1958–66. <https://doi.org/10.1257/aer.20140883>.
- Miroudot, Sébastien, and Ming Ye. 2021. "Decomposing Value Added in Gross Exports." *Economic Systems Research* 33 (1): 67–87. <https://doi.org/10.1080/09535314.2020.1730308>.
- Nagengast, Arne J., and Robert Stehrer. 2016. "Accounting for the Differences Between Gross and Value Added Trade Balances." *The World Economy* 39 (9): 1276–306. <https://doi.org/10.1111/twec.12401>.
- OECD. 2023. "OECD Inter-Country Input-Output Database." <http://oe.cd/icio>.
- Quast, Bastiaan, and Victor Kummritz. 2015. "Decompr: Global Value Chain Decomposition in R." CTEI Papers 2015-01. Centre for Trade & Economic Integration. <https://qua.st/decompr/>.
- Remond-Tierrez, Isabelle, and José Manuel Rueda-Cantuche, eds. 2019. *EU Inter-Country Supply, Use and Input-Output Tables: Full International and Global Accounts for Research in Input Output Analysis (FIGARO): 2019 Edition*. LU: EU Publications Office. <https://doi.org/10.2785/008780>.
- StataCorp. 2021. "Stata Statistical Software: Release 17." College Station, TX: StataCorp LLC.
- Timmer, Marcel P., Erik Dietzenbacher, Bart Los, Robert Stehrer, and Gaaitsen J. de Vries. 2015. "An Illustrated User Guide to the World Input-Output Database: The Case of Global Automotive Production." *Review of International Economics* 23 (3): 575–605. <https://doi.org/10.1111/roie.12178>.
- Wade, John, and Ignacio Sarmiento-Barbieri. 2020. "Ioanalysis: Input Output Analysis." <https://cran.r-project.org/web/packages/ioanalysis/ioanalysis.pdf>.
- Wang, Zhi, Shang-Jin Wei, and Kunfu Zhu. 2013. "Quantifying International Production Sharing at the Bilateral and Sector Levels." NBER Working Paper 19677. National Bureau of Economic Research, Inc. <https://econpapers.repec.org/paper/nbrnberwo/19677.htm>.

Enrique Feás

Universidad de Alcalá and Elcano Royal Institute

Universidad de Alcalá, Pl. de San Diego, s/n, 28801 Alcalá de Henares, Madrid, Spain

Elcano Royal Institute, Calle del Príncipe de Vergara, 51. 28006 Madrid, Spain

ORCID: 0000-0002-9431-6051

enrique.feas@edu.uah.es, efeas@rielcano.org

PLreg: An R Package for Modeling Bounded Continuous Data

by Francisco F. Queiroz and Silvia L.P. Ferrari

Abstract The power logit class of distributions is useful for modeling continuous data on the unit interval, such as fractions and proportions. It is very flexible and the parameters represent the median, dispersion and skewness of the distribution. Based on the power logit class, Queiroz and Ferrari (2023b, *Statistical Modelling*) proposed the power logit regression models. The dependent variable is assumed to have a distribution in the power logit class, with its median and dispersion linked to regressors through linear predictors with unknown coefficients. We present the R package **PLreg** which implements a suite of functions for working with power logit class of distributions and the associated regression models. This paper describes and illustrates the methods and algorithms implemented in the package, including tools for parameter estimation, diagnosis of fitted models, and various helper functions for working with power logit distributions, including density, cumulative distribution, quantile, and random number generating functions. Additional examples are presented to show the ability of the **PLreg** package to fit generalized Johnson SB, log-log, and inflated power logit regression models.

1 Introduction

Continuous proportion data frequently appear in areas including medicine, biology, and economics. Some concrete examples are vegetation cover fraction, mortality rate, and body fat percentage. Frequently, the interest lies in predicting or explaining the behaviour of proportions from a set of other variables. A natural approach is to use a regression model in which the response variable takes values on the unit interval. The most frequently employed model for bounded data is the beta regression model (Ferrari and Cribari-Neto, 2004) and its extensions. Other models used for bounded continuous data include, for example, rectangular beta (Bayes et al., 2012), simplex (Barndorff-Nielsen and Jørgensen, 1991; Zhang and Qiu, 2014), log-Lindley (Gómez-Déniz et al., 2014), CDF-quantile (Smithson and Shou, 2017), generalized Johnson SB (GJS; Lemonte and Bazán (2016)). Some of these models are implemented in R. For instance, the beta, simplex, and CDF-quantile regression models can be fitted using the packages **betareg** (Zeileis et al., 2021), **simplexreg** (Zhang et al., 2016), and **cdfquantreg** (Shou and Smithson, 2022), respectively. The **gamlss** (Stasinopoulos and Rigby, 2007) package can also be used to fit beta and simplex regression models.

Recently, Queiroz and Ferrari (2023b) proposed a new class of regression models useful for modeling continuous data with bounded support. The models employ a new class of distributions called power logit (PL), indexed by the median, dispersion and skewness parameters. The PL distributions are constructed from standard symmetric distributions assigned to the power logit transformation of the variable that has support on $(0, 1)$. The power logit transformation is defined in Queiroz and Ferrari (2023b) as $t(y; \lambda) = \log[y^\lambda / (1 - y^\lambda)]$, for $\lambda > 0$ and $y \in (0, 1)$; it reduces to the logit transformation when $\lambda = 1$. The PL distributions may also depend on an extra parameter that indexes the underlying symmetric distribution; for example, the degrees-of-freedom parameter of the Student-t distribution. The extra parameter adds extra flexibility, which can be used, for example, to deal with outliers. The PL distributions are more flexible than two-parameter distributions, such as the beta, simplex, and CDF-quantile distributions. The class of PL regression models has the GJS regression models as a particular case ($\lambda = 1$), with the advantage that the skewness parameter λ provides extra flexibility to fit highly skewed data. Applications in real data presented in Queiroz and Ferrari (2023b) reveal that the PL regression models are helpful for modeling continuous proportions.

The new R package **PLreg** provides a broad set of tools for fitting PL regression models and performing diagnostic analysis. The package is implemented in R and available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/package=PLreg>. This paper describes and illustrates the methods and algorithms implemented in the package. It also presents some examples to demonstrate the ability of the package to fit generalized Johnson SB, log-log, and inflated power logit regression models.

The remaining of this paper is organized as follows. Section 2.2 presents the PL distributions and the associated regression models. Section 2.3 describes the implementation of the PL distributions and the PL regression models in the **PLreg** package. Section 2.4 gives detailed illustrations of the use of the **PLreg** package for modeling continuous bounded data in different scenarios. The paper closes with a brief discussion outlining the features of the **PLreg** package.

2 Power logit regression models

2.1 Power logit distributions

The PL distributions (Queiroz and Ferrari, 2023b) are defined from a transformation of a continuous random variable whose distribution is standard symmetric with probability density function $r(z^2)$, $z \in \mathbb{R}$, in which $r(z) > 0$, for $z \geq 0$, with $\int_0^\infty z^{-1/2}r(z)dz = 1$. The function $r(\cdot)$ is called the density generator function. Let Y be a continuous random variable with support $(0, 1)$ and let:

$$Z = h(Y; \mu, \sigma, \lambda) = \frac{1}{\sigma} \left[\log \left(\frac{Y^\lambda}{1 - Y^\lambda} \right) - \log \left(\frac{\mu^\lambda}{1 - \mu^\lambda} \right) \right],$$

where $0 < \mu < 1$, $\sigma > 0$, and $\lambda > 0$. If Z has a standard symmetric distribution with density generator function $r(\cdot)$, we say that Y has a PL distribution with parameters μ , σ and λ , and density generator function $r(\cdot)$. We write $Y \sim \text{PL}(\mu, \sigma, \lambda; r)$. The density generator function $r(\cdot)$ may depend on an extra parameter, denoted here by ζ . The distribution of Y depends on the distribution chosen for Z . For instance, if Z has a standard normal distribution, then Y has a PL normal distribution; if Z has a standard Student-t distribution with ζ degrees-of-freedom, then Y has a PL Student-t distribution with extra parameter ζ .

The probability density function (pdf) of $Y \sim \text{PL}(\mu, \sigma, \lambda; r)$ is:

$$f_Y(y; \mu, \sigma, \lambda) = \frac{\lambda}{\sigma y (1 - y^\lambda)} r(z^2), \quad y \in (0, 1),$$

where $z = h(y; \mu, \sigma, \lambda)$. The cumulative distribution function (cdf) of Y is $F_Y(y; \mu, \sigma, \lambda) = R(z)$, where $R(\cdot)$ is the cdf of Z .

The GJS class of distributions is a particular case of the PL distributions when $\lambda = 1$. Other particular cases are the logit normal distribution (Johnson, 1949), the L-Logistic distribution (da Paz et al., 2019), and the logit slash distribution (Korkmaz, 2020), obtained by taking $\lambda = 1$ and Z as a standard normal, type II logistic, and slash random variable, respectively.

The PL distributions have some interesting properties. For instance, the parameters μ , σ and λ represent the median, dispersion and skewness of the distributions, and they have as a limiting case when $\lambda \rightarrow 0^+$, the class of log-log distributions, defined in Queiroz and Ferrari (2023b).

2.2 Power logit regression models

The PL regression models are defined as follows. Let Y_1, \dots, Y_n be n independent random variables, where $Y_i \sim \text{PL}(\mu_i, \sigma_i, \lambda; r)$, for $i = 1, \dots, n$, and

$$\begin{aligned} d_1(\mu_i) &= \mathbf{x}_i^\top \boldsymbol{\beta} = \eta_{1i}, \\ d_2(\sigma_i) &= \mathbf{s}_i^\top \boldsymbol{\tau} = \eta_{2i}, \end{aligned} \tag{1}$$

where $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)^\top \in \mathbb{R}^p$, $\boldsymbol{\tau} = (\tau_1, \dots, \tau_q)^\top \in \mathbb{R}^q$ and $\lambda > 0$ are the unknown parameters, which are assumed to be functionally independent and $p + q + 1 < n$; η_{1i} and η_{2i} are the linear predictors; $\mathbf{x}_i = (x_{i1}, \dots, x_{ip})^\top$ and $\mathbf{s}_i = (s_{i1}, \dots, s_{iq})^\top$ are the covariates. We assume that $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]^\top$ and $\mathbf{S} = [\mathbf{s}_1, \dots, \mathbf{s}_n]^\top$ have column rank p and q , respectively. In addition, we assume that the link functions $d_1 : (0, 1) \rightarrow \mathbb{R}$ and $d_2 : (0, \infty) \rightarrow \mathbb{R}$ are strictly monotonic and twice differentiable. Some examples of link functions for the median submodel are: $d_1(\mu) = \log\{\mu/(1 - \mu)\}$ (logit); $d_1(\mu) = \Phi^{-1}(\mu)$ (probit), where $\Phi^{-1}(\cdot)$ is the cdf of a standard normal random variable; $d_1(\mu) = -\log\{-\log \mu\}$ (log-log); and $d_1(\mu) = \log\{-\log(1 - \mu)\}$ (complementary log-log). For the dispersion submodel, the log link, $d_2(\sigma) = \log \sigma$, is the natural choice.

The log-log regression models are a limiting case of the PL regression models when $\lambda \rightarrow 0^+$. The GJS regression models (Lemonte and Bazán, 2016) are obtained by taking $\lambda = 1$.

The estimation of $\boldsymbol{\theta} = (\boldsymbol{\beta}^\top, \boldsymbol{\tau}^\top, \lambda)^\top$ is based on the maximum likelihood approach. The log-likelihood function of $\boldsymbol{\theta}$ for the observed sample y_1, \dots, y_n is:

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^n \ell_i(\mu_i, \sigma_i, \lambda),$$

where $\ell_i = \ell_i(\mu_i, \sigma_i, \lambda) = \log \lambda - \log \sigma_i - \log\{1 - y_i^\lambda\} + \log\{r(z_i^2)\} + c$, $z_i = h(y_i; \mu_i, \sigma_i, \lambda)$, and c does not depend on $\boldsymbol{\theta}$. The maximum likelihood estimate (mle) of $\boldsymbol{\theta}$, denoted by $\hat{\boldsymbol{\theta}}$, can be obtained by

solving simultaneously the nonlinear system of equations $\mathbf{U}(\boldsymbol{\theta}) = \mathbf{0}_{p+q+1}$, which does not have a closed form, where $\mathbf{U}(\boldsymbol{\theta})$ is the score function and $\mathbf{0}_{p+q+1}$ denotes a $(p + q + 1)$ -dimensional vector of zeros. [Queiroz and Ferrari \(2023b\)](#) also proposed a penalized maximum likelihood estimator (pmle), which is recommended when the sample size is small. The pmle, denoted by $\tilde{\boldsymbol{\theta}}$, is computed through numerical optimization as follows.

- i. Compute $\tilde{\lambda}$ such that:

$$\tilde{\lambda} = \underset{\lambda > 0}{\operatorname{argmax}} \ell_p^*(\lambda),$$

where $\ell_p^*(\lambda)$ is the penalized profile log-likelihood for λ ; see [Queiroz and Ferrari \(2023b, Equation 7\)](#).

- ii. Compute $\tilde{\beta}$ and $\tilde{\tau}$ by maximizing $\ell(\boldsymbol{\beta}, \tau, \tilde{\lambda})$.

The extra parameter ζ , if any, is selected by minimizing the overall goodness-of-fit measure Y_ζ , defined as:

$$Y_\zeta = n^{-1} \sum_{i=1}^n |\Phi^{-1}[R(\tilde{z}^{(i)})] - v^{(i)}|,$$

where $\tilde{z}^{(i)}$ is the i th order statistic of \tilde{z} , $v^{(i)}$ is the mean of the i th order statistic in a random sample of size n of the standard normal distribution and $\Phi(\cdot)$ is the cdf of the standard normal distribution. Alternatively, ζ may be selected by maximizing $\ell(\tilde{\boldsymbol{\theta}})$.

Some diagnostic tools for the PL regression models are presented in [Queiroz and Ferrari \(2023b\)](#), including quantile, deviance, and standardized residuals, local influence methods, and a generalized leverage measure. Applications and further details on inference methods are found in [Queiroz and Ferrari \(2023b\)](#).

3 R implementation

The **PLreg** package allows fitting the PL regression models. The package is organized in a similar way to other packages for fitting regression models, such as **betareg** and **simplexreg**. The estimation process is based on the likelihood theory, and two estimators are available: the mle and the pmle. Diagnostic tools for evaluating the fitted model are also implemented. Currently, the package includes methods for computing three types of residuals: quantile, deviance, and standardized residuals. Local influence measures, leverage measures, and goodness-of-fit statistics are also available. Additionally, the package supports PL regression models with the skewness parameter λ fixed, i.e., the package also allows fitting GJS and log-log regression models.

3.1 Power logit distributions in the **PLreg** package

Currently, the **PLreg** package includes seven distributions of the PL class: the PL normal, PL Student-t, PL power exponential, PL slash, PL hyperbolic, PL sinh-normal, and PL type II logistic distributions. **PLreg** provides the **dPL()**, **pPL()**, and **qPL()** functions to compute the probability density function, cumulative distribution function and quantile function of the PL distributions. Also, the **rPL()** function may be used to generate random samples of variables with a PL distribution. The basic usages of these functions are:

```
dPL(x, mu, sigma, lambda, zeta = 2, family, log = FALSE)
pPL(q, mu, sigma, lambda, zeta = 2, family, lower.tail = TRUE, log.p = FALSE)
qPL(p, mu, sigma, lambda, zeta = 2, family, lower.tail = TRUE, log.p = FALSE)
rPL(n, mu, sigma, lambda, zeta = 2, family)
```

The main arguments for these functions are **mu**, **sigma**, **lambda** and **family**, specifying the parameters μ , σ , and λ and the corresponding density generator function $r(\cdot)$, that is, the distribution of the symmetric distribution Z . If **lambda** = 0, those functions provide results for the log-log distributions. If the density generator function depends on an extra parameter, its value must be specified in the **zeta** argument. On the other hand, if it does not depend on an extra parameter, the argument **zeta** is ignored. The arguments **x** and **q** are the vector of quantiles, **p** is a vector of probabilities, and **n** is the number of random numbers to be generated. Other arguments are **log**, **log.p** and **lower.tail**. If **log** = TRUE, then the logarithm of the probability density function will be returned. If **log.p** = TRUE, then the logarithm of the cumulative distribution function will be returned and the quantile function will

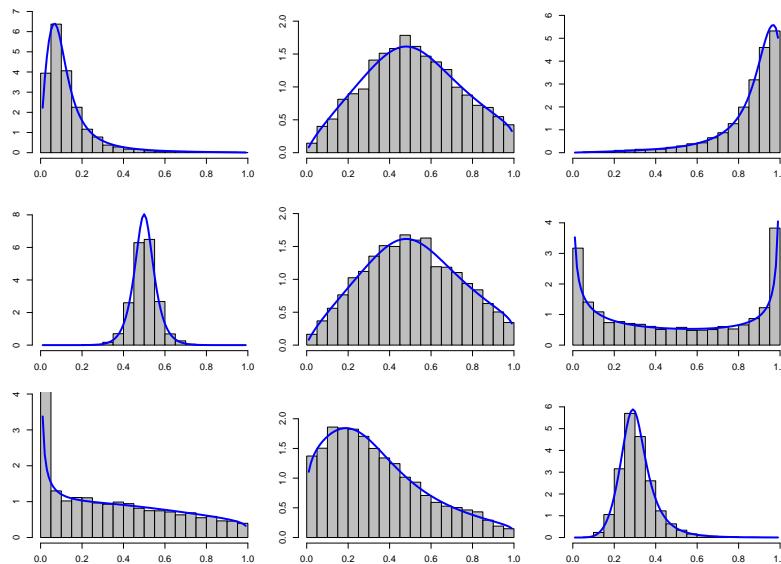


Figure 1: Histograms of random numbers of the PL hyperbolic distributions with $\zeta = 1.2$ with different values for μ , σ , and λ . First line: $\mu = (0.1, 0.5, 0.9)$, $\sigma = 1$, $\lambda = 1.5$; second line: $\mu = 0.5$, $\sigma = (0.2, 1, 3)$, $\lambda = 1.5$; third line: $\mu = 0.3$, $\sigma = 1$, $\lambda = (0.01, 1, 5)$. Solid lines are corresponding to the respective PL hyperbolic density.

be computed for $\exp(p)$. If `lower.tail = FALSE`, then one minus the cumulative distribution function will be returned and the quantile function will be computed for $1 - p$.

In the following, we present the density generator function of all PL distributions implemented in the **PLreg** package:

- PL normal (`family = "NO"`): $r(z) = (2\pi)^{-1/2} \exp(-z/2)$;
- PL Student-t (`family = "TF"`): $r(z) = \zeta^{\zeta/2} B(1/2, \zeta/2)^{-1} (\zeta + z)^{-(\zeta+1)/2}$, $\zeta > 0$ and $B(\cdot, \cdot)$ is the beta function;
- PL type II logistic (`family = "LO"`): $r(z) = \exp\{-z^{1/2}\} (1 + \exp\{-z^{1/2}\})^{-2}$;
- PL power exponential (`family = "PE"`): $r(z) = \zeta / [p(\zeta) 2^{1+1/\zeta} \Gamma(1/\zeta)] \times \exp\left\{-z^{\zeta/2} / (2p(\zeta)^{\zeta})\right\}$, $\zeta > 0$ and $p(\zeta)^2 = 2^{-2/\zeta} \Gamma(1/\zeta) / \Gamma(3/\zeta)$;
- PL slash (`family = "SLASH"`): $r(z) = (\zeta / \sqrt{2\pi}) (z/2)^{-(\zeta+1/2)} G(\zeta + 1/2, z/2)$, for $z > 0$, and $r(z) = 2\zeta / [(2\zeta + 1)\sqrt{2\pi}]$, for $z = 0$, where $\zeta > 0$ and $G(a, x) = \int_0^x t^{a-1} e^{-t} dt$ is the lower incomplete gamma function. When $\zeta = 1$ the slash distribution coincides with the canonical slash distribution;
- PL hyperbolic (`family = "Hyp"`): $r(z) = \exp\{-\zeta \sqrt{1+z}\} / (2\zeta K_1(\zeta))$, with $K_s(\zeta) = \int_0^\infty \frac{x^{s-1}}{2} \times \exp\left\{-\frac{\zeta}{2} \left(x + \frac{1}{x}\right)\right\} dx$, is the modified Bessel function of third-order and index s .
- PL sinh-normal (`family = "SN"`): $r(z) = 1 / (\zeta \sqrt{2\pi}) \cosh(z^{1/2}) \exp\left[-2/\zeta^2 \sinh^2(z^{1/2})\right]$, where $\zeta > 0$ and $\sinh(\cdot)$ and $\cosh(\cdot)$ represent the hyperbolic sine and cosine functions, respectively.

Figure 1 illustrates the use of the `rPL()` and `dPL()` functions showing the distribution of random numbers generated from PL hyperbolic distributions with $\zeta = 1.2$.

3.2 Power logit regression models

The main model-fitting function of the **PLreg** package is `PLreg()`, which is similar to the other functions for implementing regression models in R. The basic usage of the `PLreg()` function is:

```
PLreg(formula, data, subset, na.action,
      family = c("NO", "LO", "TF", "PE", "SN", "SLASH", "Hyp"), zeta = NULL,
      link = c("logit", "probit", "cloglog", "cauchit", "log", "loglog"),
      link.sigma = NULL, type = c("pML", "ML"), control = PLreg.control(...),
      model = TRUE, y = TRUE, x = FALSE, ...)
```

The argument formula may comprise three parts (separated by the symbols “ ” and “ | ”), namely: the observed response variable with values on $(0, 1)$, the linear predictor of the median submodel and the linear predictor of the dispersion submodel (for further details about the formula argument, see [Zeileis and Croissant \(2010\)](#)). For instance, `formula = y ~ x1 + x2 + x3 | z1 + z2` describes y for the response variable, x_1 , x_2 , and x_3 for the median submodel, and z_1 and z_2 for the dispersion submodel. The model is fitted with constant dispersion if the third part of the argument formula is omitted. So, a PL regression model with constant dispersion may be specified either by `formula = y ~ x1 + x2 + x3` or `formula = y ~ x1 + x2 + x3 | 1`. The available link functions for the median submodel are "logit", "probit", "cloglog", "cauchit", and "loglog". For the dispersion submodel, two link functions are allowed: "log" and "sqrt". The default link functions are "logit" for the median submodel and "log" for the dispersion submodel. There are two other important arguments: family and zeta. The argument family specifies the symmetric distribution used for generating the PL model; the currently supported families are "NO", "LO", "TF", "PE", "Hyp", "SN", and "SLASH". For the "TF", "PE", "Hyp", "SN", and "SLASH" families the extra parameter must be specified in the zeta argument.

The estimation process is carried out via `optim()` with control options set in `PLreg.control()`. It is based on the maximum likelihood method. Currently, two estimators are supported: the usual maximum likelihood estimator ("ML") and a penalized maximum likelihood estimator ("pML"); this should be specified in the type argument. If the skewness parameter (λ) is fixed, only the usual maximum likelihood estimator is supported. In this case, a value should be specified in the control argument through the `PLreg.control()` function. For instance, `control = PLreg.control(lambda = 1)` and `control = PLreg.control(lambda = 0)` lead to the GJS and the log-log regression models, respectively; note that $\lambda = 0$ represents $\lambda \rightarrow 0^+$. Also, if `type = "ML"`, `optim()` uses analytical gradients in the iterative process; if `type = "pML"`, analytical gradients are used only in the iterative process to estimate the parameters of the median and dispersion submodels. By default, the starting values are chosen as described in [Queiroz and Ferrari \(2023b\)](#), but they may be user-supplied through the `PLreg.control()` function.

Once the model has been fitted, an object of S3 class 'PLreg' is produced. A list of some of the components of this object is presented in Table 1. The complete list can be obtained in the reference manual of the package ([Queiroz and Ferrari, 2023a](#)). Several methods are available for objects of class 'PLreg'. The `summary()` method presents a standard output, with coefficient estimates, standard errors, partial Wald statistics and p values for the regression coefficients, as well as the overall goodness-of-fit measure (Y_ζ), the pseudo R^2 , and other metrics. The argument type in `summary()` specifies the type of residuals included in the output: "standardized", "quantile" or "deviance". The `plot()` method draws graphs for diagnostic and influence analyses. Table 2 presents a list with all the available functions and methods.

The `extra.parameter()` function can be used to select the extra parameter of some PL models. The basic use is as follows:

```
extra.parameter(object, lower, upper, grid = 10)
```

This function provides a graph of $-2\loglik$ and Y_ζ as functions of ζ , the extra parameter. The object argument is an object of class 'PLreg'; lower and upper are the lower and upper limits of the interval for the extra parameter, respectively; and grid is the number of values of the extra parameter for which the measures are evaluated.

4 Examples using the PLreg package

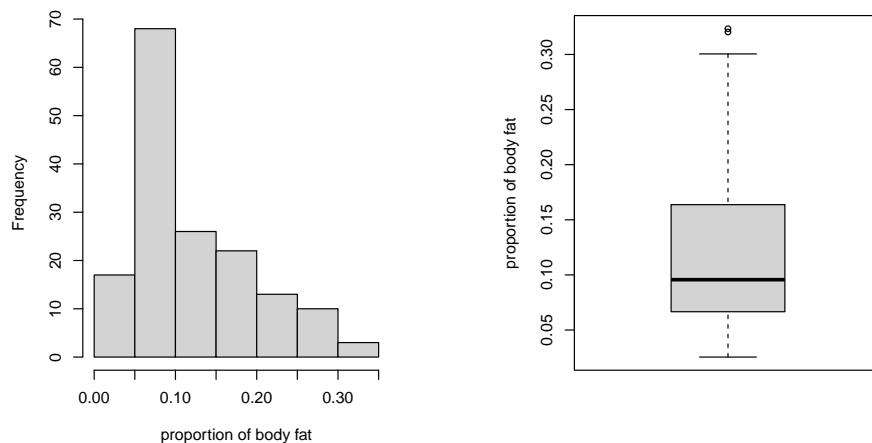
We present some examples to illustrate the features of the `PLreg` package. We use a simulated dataset and three datasets available in the package: `bodyfat_Aeolus`, `Firm`, and `PeruVotes`. These analyses were conducted using R version 4.2.2.

4.1 bodyfat_Aeolus data: IID setting

For a simple illustration of the `PLreg` package, we consider the `bodyfat_Aeolus` data reported in [Cheng et al. \(2019\)](#). The dataset used here has 159 observations and was collected in Aeolus Cave, located in East Dorset, Vermont, in the USA. The bats were sampled during the winter of 2009 (covering the winter season from October 2008 to April 2009) and 2016 (October 2015 to April 2016). Here, the interest lies in modeling the proportion of body fat of little brown bats (percentfat) using the PL distributions. The data can be loaded by:

```
R> data("bodyfat_Aeolus", package = "PLreg")
```

Component	Description
coefficients	list of the fitted model coefficients.
residuals	vector of raw residuals.
fitted.values	vector of the fitted values (fitted median for each observation).
optim	list with the optim() output. For unfixed λ , if type = "pML", the output is based on the iterative process for estimating β and γ ; and, if type = "ML", it is based on the iterative process for estimating the whole parameter vector.
family	character specifying the underlying symmetric distribution.
method	optimization method used in optim(). Default is "BFGS".
control	control arguments passed to optim().
start	vector with the starting values used to initialize the optimization process.
nobs	number of observations.
df.null	residual degrees of freedom in the null model (constant median and dispersion).
df.residual	residual degrees of freedom in the fitted model.
lambda	value of the skewness parameter λ (NULL when λ is not fixed).
loglik	log-likelihood of the fitted model.
loglikp	penalized profile log-likelihood for λ .
vcov	covariance matrix of all the parameters.
pseudo.r.squared	pseudo R-squared value.
Upsilon.zeta	an overall goodness-of-fit measure.
link	a list with elements "median" and "dispersion" containing the link objects for the respective models.
converged	logical value indicating whether the optimization converged successfully.
zeta	a numeric specifying the value of ζ used in the estimation process.
type	a character specifying the estimation method used.
v	a vector with the $v(z)$ values for all the observations; see Queiroz and Ferrari (2023b) for details.

Table 1: List of the components of an object of the 'PLreg' class.**Figure 2:** Histogram (left side) and boxplot (right side) of the response variable – bodyfat_Aeolus data.

and the histogram and boxplot of the response variable are presented in Figure 2. Some summary measures of percentfat are presented below. Note that the distributions of this variable is right-skewed and have some values close to zero, with range 2.5% – 32%.

Function	Description
print()	prints the coefficients estimates.
summary()	output for the fitted model. Returns an object of class "summary.PLreg" containing the relevant information about the fit and has a print() method.
coef()	extracts the coefficients of the fitted model.
vcov()	variance and covariance matrix.
logLik()	extracts the fitted log-likelihood function.
model.matrix()	extracts model matrix of model components.
AIC()	computes information criteria (AIC, BIC, ...).
residuals()	extracts residuals for the fitted model (quantile, standardized and deviance). Default is the standardized residual.
plot()	presents some diagnostic plots. Currently, seven types of plots are available: index plot of residuals, local influence plot based on the case-weight perturbation scheme, scatter plot of the generalized leverage versus the predicted values, scatter plot of the residuals versus the linear predictors, normal probability plot of the residuals, scatter plot of the predicted values versus the observed values, and a scatter plot of the $v(z)$ function versus the residuals (for some PL models, $v(z)$ may be interpreted as weights in the estimation process).
influence()	provides two influence measures and the generalized leverage for PL regression models.
envelope()	returns a normal probability plot with simulated envelopes for the residuals.
extra.parameter()	provides plots for selecting the extra parameter, if any.
CI.lambda()	provides plot of the profile (penalized) likelihood ratio statistics for λ . Used to obtain confidence intervals for λ .
sandwich()	provides an estimate for the asymptotic variance and covariance matrix of the parameter estimators of the PL regression models based on the sandwich estimator.

Table 2: List with the methods and functions of an object of the 'PLreg' class.

```
R> summary(bodyfat_Aeolus$percentfat)

Min. 1st Qu. Median Mean 3rd Qu. Max.
0.02544 0.06658 0.09565 0.12168 0.16371 0.32336
```

We now fit the percentfat variable using the PL normal and PL sinh-normal distributions. For the PL sinh-normal distribution, we first fit the distribution with a fixed value of ζ , e.g., $\zeta = 1$, and then we use the extra.parameter() function to select an optimal value for ζ . In the **PLreg** package, it can be done via:

```
R> PLNO <- PLreg(percentfat ~ 1, data = bodyfat_Aeolus, family = "NO")
R> PLSN.aux <- PLreg(percentfat ~ 1, data = bodyfat_Aeolus, family = "SN",
+                      zeta = 1)
R> extra.parameter(PLSN.aux, lower = 1, upper = 4, grid = 10)
```

```
Estimates for zeta are:
zeta.Ups = 1.67
zeta.loglik = 2
```

```
> PLSN <- PLreg(percentfat ~ 1, data = bodyfat_Aeolus, family = "SN",
+                  zeta = 1.67)
```

The extra.parameter() returns the optimal values for ζ based on two measures and plots these measures as functions of ζ ; see Figure 3. We choose zeta = 1.67.

To select between the PL normal and PL sinh-normal distributions, we compute the Y_ζ and the AIC for both fits:

```
R> PL_NO <- round(c(PLNO$Upsilon.zeta, AIC(PLNO)), 3)
R> PL_SN <- round(c(PLSN$Upsilon.zeta, AIC(PLSN)), 3)
```

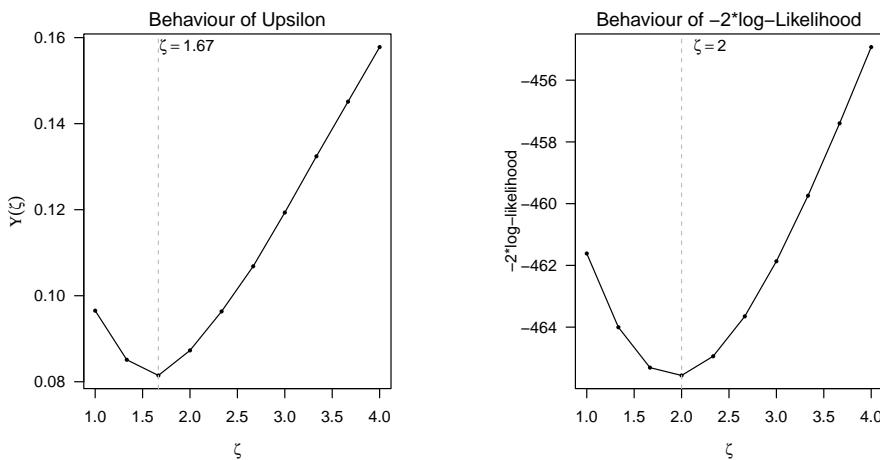


Figure 3: Plot returned by the `extra.parameter()` for selecting an optimal value for ζ of the fit of the PL sinh-normal distribution – `bodyfat_Aeolus` data.

```
R> measures <- rbind(PL_NO, PL_SN)
R> colnames(measures) <- c("Upsilon", "AIC")
R> measures

      Upsilon      AIC
PL_NO    0.114 -443.771
PL_SN    0.082 -453.942
```

Since the values of Υ_ζ and AIC for the PL sinh-normal fit are smaller than those of the PL normal fit, we select the PL sinh-normal distribution. The summary output of the PL sinh-normal fit is presented in the following:

```
R> summary(PLSN)

Call:
PLreg(formula = percentfat ~ 1, data = bodyfat_Aeolus, family = "SN",
      zeta = 1.67)

Standardized residuals:
    Min     1Q   Median     3Q    Max 
-2.4654 -0.8318 -0.2065  0.7453  2.0551 

Coefficients (median model with logit link):
            Estimate Std. Error z value Pr(>|z|)    
(Intercept) -2.11477    0.05416 -39.05  <2e-16 ***

Sigma coefficients (dispersion model with log link):
            Estimate Std. Error z value Pr(>|z|)    
(sigma)    0.2107    0.4506   0.468     0.64    

Lambda coefficient:
            Estimate Std. Error
(lambda)    1.438     0.764
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Family: PL - SN ( 1.67 ) (Power logit sinh-normal)
Estimation method: pML (penalized maximum likelihood)
Log-likelihood:  230 on 3 Df
Upsilon statistic: 0.08151
AIC: -453.9
Number of iterations in BFGS optimization: 9
```

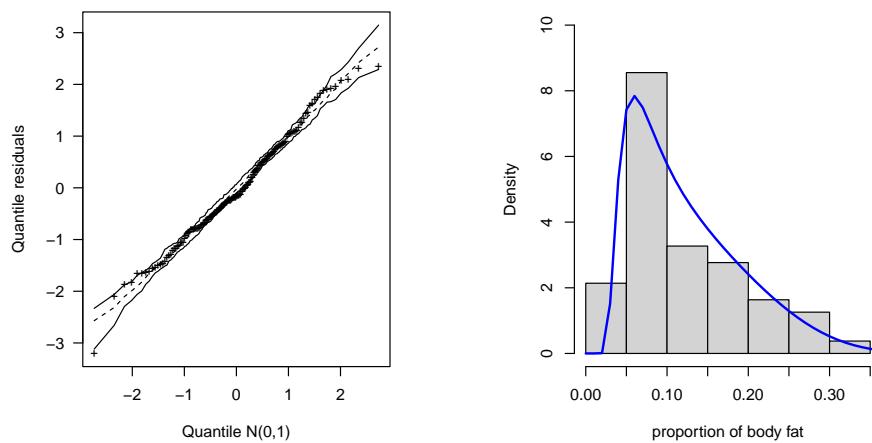


Figure 4: Normal probability plot of the quantile residual with simulated envelope for the PLSN fit and the histogram of percentfat with the estimated density – bodyfat_Aeolus data.

The estimated median of the body fat proportion based on the fit is $\exp(-2.11477) / [1 + \exp(-2.11477)] \approx 10.8\%$, close to the sample median. Figure 4 presents the normal probability plot of the quantile residual with simulated envelope for the PLSN fit and the histogram of percentfat with the estimated density. These plots can be obtained as follows:

```
R> set.seed(180123)
R> envelope(PLSN)
R> hist(bodyfat_Aeolus$percentfat, main = " ",
+       xlab = "proportion of body fat", prob = TRUE, ylim = c(0, 10))
R> curve(dPL(x, PLSN$fitted.values[1], exp(PLSN$coefficients$dispersion),
+             PLSN$coefficients$skewness, zeta = 1.67, family = "SN"), 0 , 1,
+             add = TRUE, lwd = 2, col= "blue")
```

Using the delta method, an approximated 95% confidence interval for the median of the body fat proportion is:

$$\left[\tilde{\mu} \mp 1.96 \times \text{se}(\tilde{\beta}) \frac{\exp(\tilde{\beta})}{[1 + \exp(\tilde{\beta})]^2} \right] = [0.097, 0.118].$$

4.2 Firm data: PL regression model

We now use the Firm data to replicate the application presented in Queiroz and Ferrari (2023b, Section 6.2). The dataset was introduced by Schmit and Roth (1990) and presents information on the risk management practices of 73 firms. The response variable is firmcost, defined as premiums plus uninsured losses as a percentage of the total assets. It is a measure of the firm's risk management cost-effectiveness. Queiroz and Ferrari (2023b) start the analysis with the PL slash regression model with varying dispersion, employing two covariates: sizelog, the logarithm of total assets, and indcost, a measure of the firm's industry risk. This model can be fitted via:

```
R> data("Firm", package = "PLreg")

R> Firm_slash2 <- PLreg(firmcost ~ indcost + sizelog | indcost + sizelog,
+                           data = Firm, family = "SLASH", zeta = 2)
R> extra.parameter(Firm_slash2, lower = 1, upper = 2.5, grid = 30)

Estimates for zeta are:
zeta.Ups = 1.88
zeta.lolik = 1.93

R> Firm_slash <- PLreg(firmcost ~ indcost + sizelog | indcost + sizelog,
+                           data = Firm, family = "SLASH", zeta = 1.88)
R> summary(Firm_slash)
```

```

Call:
PLreg(formula = firmcost ~ indcost + sizelog | indcost + sizelog,
      data = Firm, family = "SLASH", zeta = 1.88)

Standardized residuals:
    Min     1Q   Median     3Q     Max
-2.1220 -0.6253  0.0251  0.6548  4.6194

Coefficients (median model with logit link):
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.8223    1.0196  3.749 0.000178 ***
indcost      2.3117    0.8062  2.867 0.004140 **
sizelog     -0.9082    0.1225 -7.416 1.21e-13 ***

Sigma coefficients (dispersion model with log link):
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.56915   0.78874 -0.722   0.471
indcost      0.36623   0.54062  0.677   0.498
sizelog       0.07455   0.09979  0.747   0.455

Lambda coefficient:
            Estimate Std. Error
(lambda)    2.035     1.196
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Family: PL - SLASH ( 1.88 ) (Power logit slash)
Estimation method: pML (penalized maximum likelihood)
Log-likelihood:  123 on 7 Df
Pseudo R-squared: 0.4177
Upsilon statistic: 0.06723
AIC: -232.1
Number of iterations in BFGS optimization: 10

```

The standard errors presented in the `summary()` output is computed from the observed information matrix. Queiroz and Ferrari (2023b) employ the sandwich matrix to obtain standard errors. Standard errors are computed from the sandwich matrix by using the `sandwich()` function in the `PLreg` package as follows:

```

R> sand.matrix <- sandwich(Firm_slash)
R> se <- sqrt(diag(sand.matrix))
R> se

            (Intercept)           indcost          sizelog (sigma)_-(Intercept)
  1.30397074        1.05026082        0.16419123        0.54569178
  (sigma)_indcost  (sigma)_sizelog        (lambda)
  0.59534023        0.08767362        0.91164845

```

All the covariates are statistically significant for the median submodel but not for the dispersion submodel. Then, the authors fit the PL slash regression model with constant dispersion and select $\zeta = 2.29$. The model can be fitted as follows:

```

R> Firm_slash.CD <- PLreg(firmcost ~ indcost + sizelog,
+                               data = Firm, family = "SLASH", zeta = 2.29)
R> summary(Firm_slash.CD)

```

```

Call:
PLreg(formula = firmcost ~ indcost + sizelog,
      data = Firm, family = "SLASH", zeta = 2.29)

Standardized residuals:
    Min     1Q   Median     3Q     Max
-2.1133 -0.6590  0.0546  0.7168  5.9131

Coefficients (median model with logit link):
            Estimate Std. Error z value Pr(>|z|)

```

```
(Intercept) 3.8668    0.9994   3.869 0.000109 ***
indcost     2.1330    0.5836   3.655 0.000257 ***
sizelog      -0.9053   0.1120  -8.082 6.38e-16 ***

Sigma coefficients (dispersion model with log link):
          Estimate Std. Error z value Pr(>|z|)
(sigma) 0.1333    0.5331   0.25    0.803

Lambda coefficient:
          Estimate Std. Error
(lambda) 1.788     1.01
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Family: PL - SLASH ( 2.29 ) (Power logit slash)
Estimation method: pML (penalized maximum likelihood)
Log-likelihood: 122 on 5 Df
Pseudo R-squared: 0.4162
Upsilon statistic: 0.06448
AIC: -234
Number of iterations in BFGS optimization: 15
```

Normal probability plots of the residuals with simulated envelopes as well as influence plots may be obtained via `envelope()` and `influence()` functions, respectively. For instance, the plots presented in Figure 5 are obtained as follows:

```
R> envelope(Firm_slash.CD, type = "quantile")
R> envelope(Firm_slash.CD, type = "deviance")
R> envelope(Firm_slash.CD, type = "standardized")
R> influence(Firm_slash.CD)
```

Note that one observation is highlighted in almost all the graphics in Figure 5. It is the case #15 and corresponds to a firm with the highest `firmcost` value. Queiroz and Ferrari (2023b) conclude that this observation does not significantly influence the fitted model. In fact, the weight of this observation in the estimation process is close to zero — it may be verified by plotting the weights against the residuals. This plot is presented in Figure 6 and is obtained via:

```
R> plot(Firm_slash.CD, which = 7)
```

The `PLreg` package allows different link functions for the median submodel. In order to illustrate it, we fit the model with the `probit` and `cloglog` link functions; for simplicity, we set `zeta = 2.29`. We compare the fits through the values of the pseudo R^2 and the Y_ζ measure as follows:

```
R> measures <- sapply(c("logit", "probit", "cloglog"),
+                      function(x){
+                          fit <- update(Firm_slash.CD, link = x)
+                          round(c(fit$pseudo.r.squared, fit$Upsilon.zeta),3)
+                      })
R> rownames(measures) <- c("pseudo R-squared", "Upsilon_zeta")
R> measures

      logit probit cloglog
pseudo R-squared 0.416 0.379 0.482
Upsilon_zeta     0.064 0.071 0.069
```

No model simultaneously has the highest pseudo R^2 and the smallest Y_ζ . Note that the values for the fit with the `logit` and `cloglog` link functions are close. The `probit` link function leads to the smallest pseudo R^2 and the highest Y_ζ ; hence it is not recommended.

4.3 PeruVotes data: GJS regression models

Lemonte and Bazán (2016) use the GJS Student-t regression model to model the proportion of blank votes (votes) in the 2006 Peruvian general election of an electoral district as a function of the Human Development Index (HDI). The PeruVotes dataset contains information on 194 electoral districts. Recall that the PL regression models with $\lambda = 1$ reduce to the GJS regression models. The extra parameter ζ of the GJS Student-t regression model may be selected using the `extra.parameter()` function as in

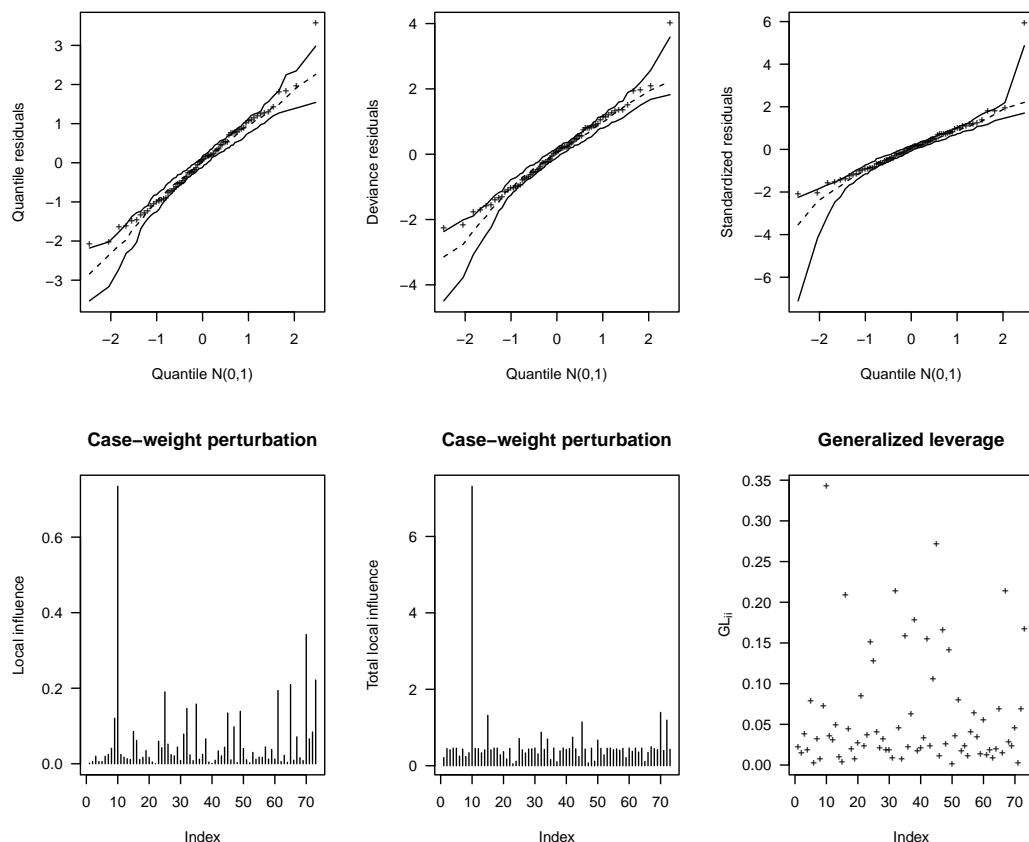


Figure 5: Normal probability plot of the quantile, deviance and standardized residuals with simulated envelope and influence plots for the Firm_slash.CD fit – Firm data.

the previous examples. However, to replicate the analysis in Lemonte and Bazán (2016) we fit the PL Student-t regression model with $\zeta = 4$ (and $\lambda = 1$). Using the control argument in the PLreg() function, we set lambda = 1:

```
R> data("PeruVotes", package = "PLreg")
R> PV_GJSt <- PLreg(votes ~ HDI | HDI, data = PeruVotes, family = "TF",
+                      zeta = 4, control = PLreg.control(lambda = 1))
R> summary(PV_GJSt)

Call:
PLreg(formula = votes ~ HDI | HDI, data = PeruVotes, family = "TF",
      zeta = 4, control = PLreg.control(lambda = 1))

Standardized residuals:
    Min     1Q Median     3Q    Max 
-5.1247 -0.5838  0.0024  0.5821  4.1666 

Coefficients (median model with logit link):
            Estimate Std. Error z value Pr(>|z|)    
(Intercept)  2.3054    0.2086   11.05 <2e-16 ***
HDI        -6.8075    0.3780  -18.01 <2e-16 ***

Sigma coefficients (dispersion model with log link):
            Estimate Std. Error z value Pr(>|z|)    
(Intercept) -2.7560    0.7008  -3.933  8.4e-05 ***
HDI         2.1667    1.2422   1.744   0.0811 . 

Fixed skewness parameter (lambda = 1).
---
```

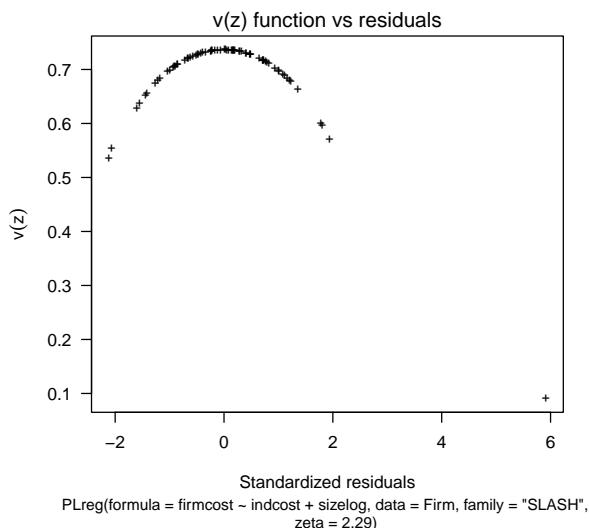


Figure 6: Plot of the $v(z)$ function against the standardized residual for the Firm_slash.CD fit – Firm data.

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Family: PL - TF ( 4 ) (Power logit Student-t)
Estimation method: ML (maximum likelihood)
Log-likelihood: 352.8 on 4 Df
Pseudo R-squared: 0.5822
Upsilon statistic: 0.06836
AIC: -697.5
Number of iterations in BFGS optimization: 9
```

As mentioned before, for fixed λ , the estimation is based on the usual maximum likelihood estimator. The output of the `summary()` function may be used to replicate part of Table 2 of Lemonte and Bazán (2016). The GJS distribution in their paper has a median-precision parameterization, while we use a median-dispersion parameterization. As we are using the logarithmic link function for the dispersion parameter, the estimates for the precision submodel are the negative of those obtained here for the dispersion submodel.

As the GJS regression models are a particular case of the PL regression models when $\lambda = 1$, an inherent question is whether the dataset supports the assumption that $\lambda = 1$. A confidence interval for λ may be constructed using the profile penalized likelihood ratio statistic defined by $W_p^*(\lambda) = 2\{\ell_p^*(\tilde{\lambda}) - \ell_p^*(\lambda)\}$, that is asymptotically distributed as χ_1^2 (Queiroz and Ferrari, 2023b). The `CI.lambda()` function in the `PLreg` package provides a plot of $W_p^*(\lambda)$ against λ and shows the observed confidence interval for λ . As an illustration, we fit the PL Student-t regression model with $\zeta = 4$ and use the `CI.lambda()` function to obtain a 90% confidence interval for λ :

```
R> PV_PLt <- PLreg(votes ~ HDI | HDI, data = PeruVotes, family = "TF",
+                      zeta = 4)
R> coefficients(PV_PLt, conf.coef = 0.9)
```

(Intercept)	HDI (sigma)_Intercept
2.3050765	-6.8065881
(sigma)_HDI	(lambda)
2.0459250	0.9102018

```
R> CI.lambda(PV_PLt)
```

The confidence interval for lambda is: (0, 5.4).

The `CI.lambda()` function provides the plot presented in Figure 7; the horizontal dashed line indicates the 90% confidence interval for λ . Note that the estimated λ is close to one, and the confidence interval contains $\lambda = 1$. A diagnostic analysis not shown here indicates that the GJS Student-t regression model with $\zeta = 4$ suitably fits the data.

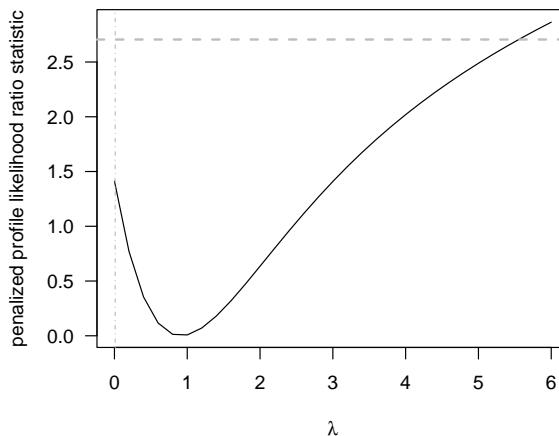


Figure 7: Plot of the profile penalized likelihood ratio statistics for λ based on the PV_PLt fit – PeruVotes data.

4.4 bodyfat_Aeolus data: log-log regression models

We now turn to the bodyfat_Aeolus data, introduced in Section 2.4.1. The interest lies in modelling the proportion of body fat of little brown bats (percentfat) as a function of the year (year, 1 for 2016 and 0 for 2009), sex of the sampled bat (sex, 1 for male and 0 for female) and the hibernation time (days), defined as the number of days since the fall equinox. First, we fit the PL normal regression model and print the estimated skewness parameter:

```
R> bodyf_PL <- PLreg(percentfat ~ days + sex + year | days + sex + year,
+                      data = bodyfat_Aeolus, family = "NO")
R> bodyf_PL$coefficients$skewness
(lambd)
0.0007122085
```

Note that the estimate of λ is close to zero. It may indicate the limiting model when $\lambda \rightarrow 0^+$ may be reasonable. The log-log normal regression model may be fitted by setting `lambda = 0` in the `control` argument of the `PLreg()` function as follows:

```
R> bodyf_loglog <- PLreg(percentfat ~ days + sex + year | days + sex + year,
+                         data = bodyfat_Aeolus, family = "NO",
+                         control = PLreg.control(lambda = 0))
R> summary(bodyf_loglog)

Call:
PLreg(formula = percentfat ~ days + sex + year | days + sex +
    year, data = bodyfat_Aeolus, family = "NO",
    control = PLreg.control(lambda = 0))

Standardized residuals:
      Min     1Q   Median     3Q     Max 
-2.7679 -0.6402  0.0664  0.6834  2.3130 

Coefficients (median model with logit link):
            Estimate Std. Error z value Pr(>|z|)    
(Intercept) -1.1532851  0.0665887 -17.320 <2e-16 ***
days        -0.0094255  0.0005409 -17.427 <2e-16 ***
sexM        -0.0324633  0.0531725 -0.611    0.542    
year2016     0.5039790  0.0581870  8.661 <2e-16 ***

Sigma coefficients (dispersion model with log link):
            Estimate Std. Error z value Pr(>|z|)    

```

```
(Intercept) -1.9668123  0.1663712 -11.822 <2e-16 ***
days         0.0007478  0.0012223   0.612  0.5407
sexM        -0.2873759  0.1145487  -2.509  0.0121 *
year2016     0.1088719  0.1314918   0.828  0.4077

Fixed skewness parameter (limiting case lambda -> 0).
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Family: log-log - NO (log-log normal)
Estimation method: ML (maximum likelihood)
Log-likelihood: 323.4 on 8 Df
Pseudo R-squared: 0.6755
Upsilon statistic: 0.05339
AIC: -630.8
Number of iterations in BFGS optimization: 28
```

Since the log-log normal regression model is more parsimonious than the PL normal regression model, it should be used. One can also consider other models in the log-log class specifying a different family in the family argument.

4.5 Simulated data: Inflated PL regression models

The **PLreg** package requires that the response variable values are all in the open interval $(0, 1)$. It does not allow values at the boundaries, i.e., equal to zero or one. The zero-or-one inflated PL regression models may be employed when the response variable contains values at one of the boundaries. These models may be fitted using the **PLreg** package in conjunction with the `glm()` function.

We say that Y has an inflated PL distribution with parameters $\alpha \in (0, 1)$, $\mu \in (0, 1)$, $\sigma > 0$, and $\lambda > 0$ if $\mathbb{P}(Y = c) = \alpha$, with $c = 0$ or $c = 1$, and, with probability $1 - \alpha$, $Y \sim \text{PL}(\mu, \sigma, \lambda; r)$. In other words, an inflated PL distribution is a mixture of a PL distribution and a degenerate variable in a known value c ($c = 0$ or $c = 1$). If $c = 0$, we have the zero-inflated PL distribution and if $c = 1$, the one-inflated PL distribution. The parameters μ , σ , and λ represent the median, dispersion and skewness of the conditional distribution of Y given that $Y \in (0, 1)$ and α is the mixture parameter. When $\lambda = 1$ the inflated PL distributions reduce to the inflated GJS distributions (Queiroz and Lemonte, 2021). If $\lambda \rightarrow 0^+$, we have the inflated log-log distributions as a limiting case.

In the inflated PL regression models, μ and σ are linked to the covariates through linear predictors with unknown coefficients as in Equation 1. Likewise, the mixture parameter submodel is $d_0(\alpha_i) = z_i^\top \kappa = \eta_{0i}$.

One may use the maximum likelihood approach to estimate the parameters of the model, denoted here by $\theta = (\kappa^\top, \beta^\top, \tau^\top, \lambda)^\top$. The likelihood function of θ factorizes in two terms, one that depends only on κ (discrete part) and the other that depends on the remaining parameters (continuous part). Thus, the inference of the discrete part and the continuous part is performed separately.

We present a brief example with simulated data. We generate 300 observations from the zero-inflated PL normal regression model with a constant dispersion and logit link for the median and mixture parameter submodels:

```
R> n <- 300
R> kappa <- c(-2, 0.5)
R> beta <- c(-1.0, -2.0)
R> sigma <- 0.5
R> lambda <- 2
R> set.seed(25012023)
R> x1 <- runif(n)
R> Z <- X <- matrix(c(rep(1,n), x1), ncol = 2, byrow = FALSE)
R> alpha <- exp(Z%*%kappa)/(1 + exp(Z%*%kappa))
R> mu <- exp(X%*%beta)/(1 + exp(X%*%beta))
R> prob <- runif(n)
R> y <- ifelse((prob <= alpha), 0, rPL(n, mu, sigma, lambda, family = "NO"))
```

The histogram and boxplot of the response variable y are presented in Figure 8. We consider fitting the zero-inflated PL normal regression model in which the parameters α and μ are modeled as a function of $x1$ through the logit link. To estimate the parameters associated with the discrete part, we fit a binomial regression model in which the response variable is equal to one if $y = 0$ and is equal to zero otherwise. The success probability for the i -th observation is α_i . This model is fitted using the `glm()`

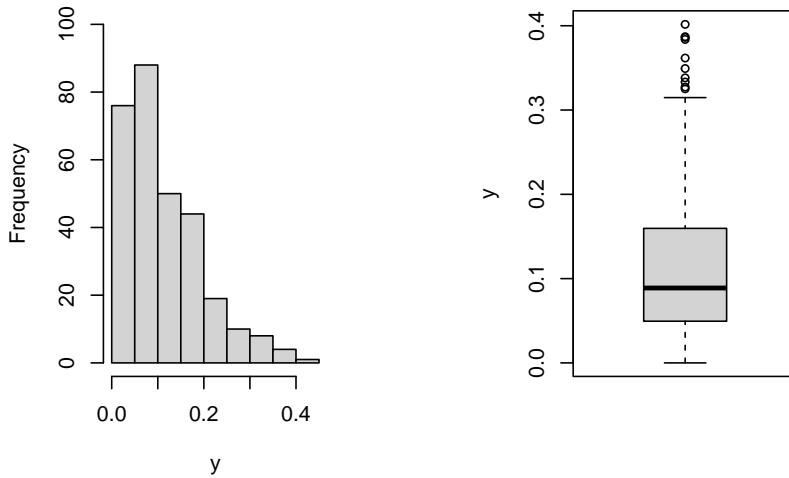


Figure 8: Histogram (left side) and boxplot (right side) of the response variable y – simulated data.

function ([Chambers and Hastie, 1992](#)) of the `stats` package. The continuous part is modeled using the `PLreg` package. The following code shows how the model is fitted:

```
R> Ind <- ifelse(y == 0, 1, 0)
R> fit.glm <- glm(Ind ~ x1, family = binomial())
R> fit.PL <- PLreg(y[Ind == 0] ~ x1[Ind == 0], family = "NO", type = "ML")
```

For estimating the parameters of the discrete part, we consider all the observations. In contrast, to estimate the parameters of the continuous part (PL model), we only consider the observations in $(0, 1)$. The estimated coefficients are obtained as follows:

```
R> coefficients(fit.glm)
(Intercept)          x1
-2.1812682    0.9445324

R> coefficients(fit.PL)
(Intercept) x1[Ind == 0] (sigma)_-(sigma)      (lambda)
-0.9608803   -2.1254449    0.3261416     5.3304235
```

Standard errors and further information may be obtained through the `summary()` function. As expected, the estimates of the parameters are close to those used to generate the data. Diagnostic plots for the discrete and continuous parts may be obtained separately by using the `plot()` method for the `fit.glm` and `fit.PL` fits. The overall adequacy of the fitted model may be investigated using the randomized quantile residual ([Dunn and Smyth, 1996](#)). For the inflated PL regression models, the randomized quantile residuals are defined as:

$$r_i = \begin{cases} \Phi^{-1}(u_i), & y_i = c, \\ \Phi^{-1}\left(\tilde{\alpha}_i \mathbb{I}_{[c, \infty)}(y) + (1 - \tilde{\alpha}_i) F_Y(y_i; \tilde{\mu}_i, \tilde{\sigma}_i, \tilde{\lambda})\right), & y_i \in (0, 1), \end{cases}$$

for $i = 1, \dots, n$, where $c = 0$ or $c = 1$ depending on the case. Also, u_i is a random draw from the uniform distribution on the interval $(0, \tilde{\alpha}_i)$ if $c = 0$, and $(1 - \tilde{\alpha}_i, 1)$ if $c = 1$. As the `PLreg` package provides the `pPL()` function to obtain the cdf of the PL distributions, the randomized quantile residuals can be easily computed. For the data under investigation, the code below provides the computation of the residuals and the plots presented in Figure 9.

```
R> alpha <- fit.glm$fitted.values
R> mu     <- fit.PL$link$median$linkinv(X %*% fit.PL$coefficients$median)
R> sigma  <- fit.PL$link$dispersion$linkinv(fit.PL$coefficients$dispersion)
R> lambda <- fit.PL$coefficients$skewness
R> cdf <- alpha*as.numeric(y >= 0) + (1 - alpha)*pPL(y, mu, sigma, lambda,
```

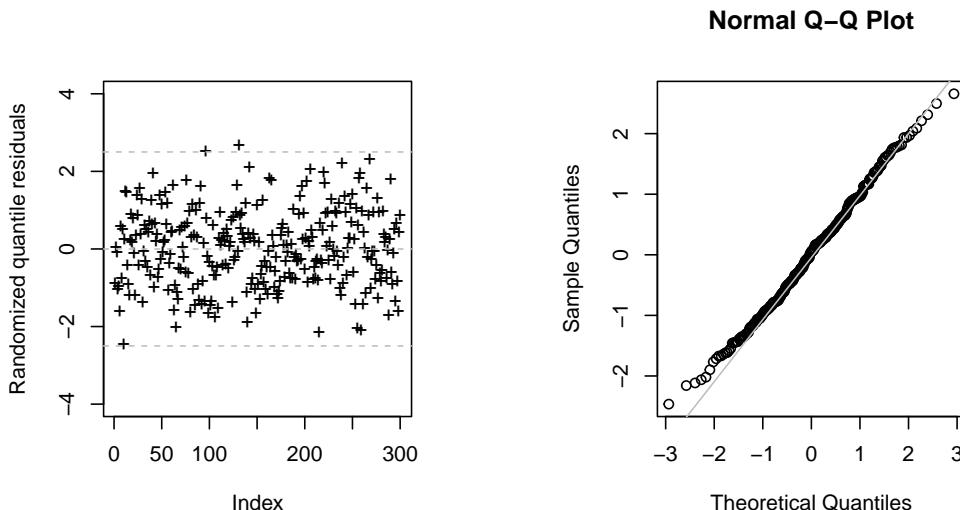


Figure 9: Scatter plot (left side) and quantile-quantile plot (right side) of the randomized quantile residual – simulated data.

```
+      family = "NO")
R> res <- ifelse(y == 0, qnorm(runif(length(y), 0, alpha)), qnorm(cdf))
R> plot(res, ylab = "Randomized quantile residuals", pch = "+", ylim = c(-4, 4))
R> abline(h = 2.5, col = "gray", lty = 2)
R> abline(h = -2.5, col = "gray", lty = 2)
R> abline(h = 0, col = "gray", lty = 2)
R> qqnorm(res)
R> qqline(res, col = "gray")
```

We may also fit inflated GJS and inflated log-log regression models specifying `lambda = 1` and `lambda = 0` in the `control` argument of the `PLreg()` function, respectively.

5 Concluding remarks

This paper presents the R implementation of the PL regression models available in the `PLreg` package. The models are suitable for modeling continuous data observed in the open interval (0,1). The package provides tools for likelihood-based inference and diagnostic analysis. Currently, the package includes seven distributions in the PL class, two types of estimators, profile likelihood-based confidence intervals for the skewness parameter, and procedures for selecting the extra parameter, if any. Different residuals and influence methods for performing diagnostic analysis are implemented. The applications in the previous sections illustrate the ability of the package to fit different PL regression models, including the GJS and log-log models.

The response variable for using the `PLreg` package must lie in the open interval (0,1), as it is an inherent assumption of the PL regression models. A possible approach when the data contain observations in one of the boundaries is to employ the inflated PL regression models, that assume that the response variable has a mixture of a PL distribution and a degenerate distribution at zero or one. A relevant contribution of this paper is to show how the `PLreg` package can be used to fit and perform diagnostic analysis for inflated PL regression models as well as inflated GJS and log-log regression models.

Acknowledgments

We thank the associate editor and the reviewer for their constructive comments on an earlier version of this article. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance Code 001 and by the Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brazil (CNPq). The authors gratefully acknowledge funding provided by CNPq (Grants No. 150976-2022-4 and No. 305963-2018-0).

References

- O. E. Barndorff-Nielsen and B. Jørgensen. Some parametric models on the simplex. *Journal of Multivariate Analysis*, 39(1):106–116, 1991. doi: 10.1016/0047-259X(91)90008-P. [p235]
- C. L. Bayes, J. L. Bazán, and C. García. A new robust regression model for proportions. *Bayesian Analysis*, 7(4):841 – 866, 2012. doi: 10.1214/12-BA728. [p235]
- J. M. Chambers and T. J. Hastie, editors. *Statistical Models in S*. Chapman & Hall, London, 1992. [p250]
- T. L. Cheng, A. Gerson, M. S. Moore, J. D. Reichard, J. DeSimone, C. K. R. Willis, W. F. Frick, and A. M. Kilpatrick. Higher fat stores contribute to persistence of little brown bat populations with white-nose syndrome. *Journal of Animal Ecology*, 88(4):591–600, 2019. doi: 10.1111/1365-2656.12954. [p239]
- R. F. da Paz, N. Balakrishnan, and J. L. Bazán. L-logistic regression models: Prior sensitivity analysis, robustness to outliers and applications. *Brazilian Journal of Probability and Statistics*, 33(3):455 – 479, 2019. doi: 10.1214/18-BJPS397. [p236]
- P. K. Dunn and G. K. Smyth. Randomized quantile residuals. *Journal of Computational and Graphical Statistics*, 5(3):236–244, 1996. doi: 10.1080/10618600.1996.10474708. [p250]
- S. L. P. Ferrari and F. Cribari-Neto. Beta regression for modelling rates and proportions. *Journal of Applied Statistics*, 31(7):799–815, 2004. doi: 10.1080/0266476042000214501. [p235]
- E. Gómez-Déniz, M. A. Sordo, and E. Calderín-Ojeda. The log-lindley distribution as an alternative to the beta regression model with applications in insurance. *Insurance: Mathematics and Economics*, 54: 49–57, 2014. doi: 10.1016/j.inmatheco.2013.10.017. [p235]
- N. L. Johnson. Systems of frequency curves generated by methods of translation. *Biometrika*, 36(1/2): 149–176, 1949. doi: 10.2307/2332539. [p236]
- M. C. Korkmaz. A new heavy-tailed distribution defined on the bounded interval: The logit slash distribution and its application. *Journal of Applied Statistics*, 47(12):2097–2119, 2020. doi: 10.1080/02664763.2019.1704701. [p236]
- A. J. Lemonte and J. L. Bazán. New class of johnson distributions and its associated regression model for rates and proportions. *Biometrical Journal*, 58(4):727–746, 2016. doi: 10.1002/bimj.201500030. [p235, 236, 245, 246, 247]
- F. F. Queiroz and S. L. P. Ferrari. *PLreg: Power Logit Regression for Modeling Bounded Data*, 2023a. URL <https://CRAN.R-project.org/package=PLreg>. R package version 0.4.1. [p239]
- F. F. Queiroz and S. L. P. Ferrari. Power logit regression for modeling bounded data. *Statistical Modelling*, 2023b. doi: 10.1177/1471082X221140157. [p235, 236, 237, 239, 240, 243, 244, 245, 247]
- F. F. Queiroz and A. J. Lemonte. A broad class of zero-or-one inflated regression models for rates and proportions. *Canadian Journal of Statistics*, 49(2):566–590, 2021. doi: 10.1002/cjs.11576. [p249]
- J. T. Schmit and K. Roth. Cost effectiveness of risk management practices. *The Journal of Risk and Insurance*, 57(3):455–470, 1990. [p243]
- Y. Shou and M. Smithson. *cdfquantreg: Quantile Regression for Random Variables on the Unit Interval*, 2022. URL <https://CRAN.R-project.org/package=cdfquantreg>. R package version 1.3.1-1. [p235]
- M. Smithson and Y. Shou. Cdf-quantile distributions for modelling random variables on the unit interval. *British Journal of Mathematical and Statistical Psychology*, 70(3):412–438, 2017. doi: 10.1111/bmsp.12091. [p235]
- D. M. Stasinopoulos and R. A. Rigby. Generalized additive models for location scale and shape (GAMLSS) in r. *Journal of Statistical Software*, 23(7):1–46, 2007. doi: 10.18637/jss.v023.i07. [p235]
- A. Zeileis and Y. Croissant. Extended model formulas in r: Multiple parts and multiple responses. *Journal of Statistical Software*, 34(1):1–13, 2010. doi: 10.18637/jss.v034.i01. [p239]
- A. Zeileis, F. Cribari-Neto, B. Gruen, and I. Kosmidis. *betareg: Beta Regression*, 2021. URL <https://CRAN.R-project.org/package=betareg>. R package version 3.1-4. [p235]
- P. Zhang and Z. Qiu. Regression analysis of proportional data using simplex distribution. *Science China Mathematics (Chinese Version)*, 44(1):89–104, 2014. doi: 10.1360/012013-200. [p235]

P. Zhang, Z. Qiu, and C. Shi. *simplexreg: Regression Analysis of Proportional Data Using Simplex Distribution*, 2016. URL <https://CRAN.R-project.org/package=simplexreg>. R package version 1.3. [p235]

Francisco F. Queiroz
Department of Statistics, University of São Paulo
Rua do Matão, 1010
05508-090, São Paulo, Brazil
E-mail: email:felipeq@ime.usp.br

Silvia L.P. Ferrari
Department of Statistics, University of São Paulo
Rua do Matão, 1010
05508-090, São Paulo, Brazil
E-mail: email:silviaferrari@usp.br

Inference for Network Count Time Series with the R Package PNAR

by Mirko Armillotta, Michail Tsagris, and Konstantinos Fokianos

Abstract We introduce a new R package useful for inference about network count time series. Such data are frequently encountered in statistics and they are usually treated as multivariate time series. Their statistical analysis is based on linear or log-linear models. Nonlinear models, which have been applied successfully in several research areas, have been neglected from such applications mainly because of their computational complexity. We provide R users the flexibility to fit and study nonlinear network count time series models which include either a drift in the intercept or a regime switching mechanism. We develop several computational tools including estimation of various count Network Autoregressive models and fast computational algorithms for testing linearity in standard cases and when non-identifiable parameters hamper the analysis. Finally, we introduce a copula Poisson algorithm for simulating multivariate network count time series. We illustrate the methodology by modeling weekly number of influenza cases in Germany.

1 Introduction

Many data examples are frequently observed as multivariate counting processes recorded over a time-span and with known relations among observations (e.g epidemiological data with geographical distances between different areas). An important objective then is to study the effect of a known network to the observed data. This motivates a great amount of interest to network time series models; see Zhu et al. (2017) who developed continuous Network Autoregressive models (abbreviated as NAR). For these models, the observed variable Y , for the node i at time t , is denoted by $Y_{i,t}$, and it is assumed to depend on the past value of the variable for the node itself, say $Y_{i,t-1}$, and on the historical averages of its neighboring variables. The unknown parameters of the model are estimated by Least Squares estimation (LS). This work was advanced by Armillotta and Fokianos (2023a) who developed linear and log-linear Poisson Network Autoregression model (PNAR) for multivariate count distributed data. The joint dependence among different variables is specified by a copula construction (Fokianos et al., 2020, Sec. 2). In addition, Armillotta and Fokianos (2023a) have further established parametric estimation under the framework of quasi maximum likelihood inference (see Wedderburn (1974) Gourieroux et al. (1984)) and associated asymptotic theory when the network dimension increases. In the context of epidemiology, related applied work has been developed by Held et al. (2005) for the linear model only, and it was extended by Paul et al. (2008); Paul and Held (2011); Held and Paul (2012); Meyer and Held (2014) and Bracher and Held (2020).

The previous contributions impose linearity (or log-linearity) of the model, which can be a restrictive assumption for real world applications. For example, existence of different underlying states (e.g. exponentially expanding pandemic / dying out pandemic) implies that different regime switching data generating processes should be applied and this fits the framework we consider. Recently, Armillotta and Fokianos (2023b) specified a general nonlinear Network Autoregressive model for both continuous and discrete-valued processes, establishing also related theory. In addition, the authors study testing procedures for examining linearity of NAR model against specific nonlinear alternatives by means of a quasi score test statistic. This methodology was developed with and without the presence of identifiable parameters under the null hypothesis.

Even though there exists sufficient statistical methodology for PNAR models, there has been a lack of up-to-date software for implementing their analyses. The aim of this work is to fill this gap by introducing the new package **PNAR** (Tsagris et al., 2023) and to demonstrate its usefulness for count network data analysis. Related R packages do not provide tools for estimating nonlinear models and applying associated testing procedures. The package **GNAR** (Leeming et al., 2023; Knight et al., 2020) studies Generalized NAR models (GNAR); this is a linear NAR model which takes into account the effect of several connection layers between the network nodes. This package deals with continuous-valued time series and does not contain tools for testing linearity. **PNAR** complements **GNAR** as it provides additional methodology for testing and inference about nonlinear discrete-valued network models.

Package **surveillance** (Höhle et al., 2022; Meyer et al., 2017) fits only linear models for spatial-temporal disease counts with Poisson or Negative Binomial distribution and with an autoregressive network effect. The package does accommodate various structural break-point tests but it does not contain functions for testing linearity and for log-linear model fitting. Moreover, standard errors of estimated parameters are computed by considering the quasi-likelihood as the true likelihood of the

model (Paul et al., 2008, Sec. 2.3). However, if the count time series are cross-sectional dependent, as it is usually the case, the likelihood function is misspecified and the obtained standard errors are not consistently estimated.

The **PNAR** package provides several advancements to the state of the art software: i) efficient estimation of (log-)linear models with proper robust standard errors accounting for possible model misspecification; ii) appropriate functions for testing linearity when a parameter is either identifiable or non-identifiable, under then null hypothesis, by providing appropriate p -value bounds and bootstrap approximations; iii) new algorithms for generating (log-)linear and nonlinear network count time series models.

The paper is organized as follows. The next section introduces linear and log-linear network time series autoregressive models for count data. Details about inference for unknown model parameters are provided. An application to estimation of weekly number of influenza A & B cases from two Southern German states is given and some further model aspects are discussed. Then we focus on non-linear models and associated testing theory. Results concerning score tests for testing linearity in NAR models are discussed and applied to influenza data. We also address the issue of computational speed. A short section discussing simulation of network count time series shows the usefulness of this methodology. The paper concludes with a short discussion.

2 Poisson network models

Consider a known network with N nodes, indexed by $i = 1, \dots, N$. The neighborhood structure of such a network is completely described by its adjacency matrix, say $A = (a_{ij}) \in \mathbb{R}^{N \times N}$ where $a_{ij} = 1$, if there is a directed edge from i to j , say $i \rightarrow j$, and 0 otherwise. Undirected graphs are allowed ($A = A'$), which means that the edge between two nodes, i and j , has no specific direction (say $i \sim j$). This is common in geographical and epidemic networks (e.g. district i shares a border with district j , patient i has a contact with patient j). Self-relationships are excluded i.e. $a_{ii} = 0$ for any $i = 1, \dots, N$.

Let Y be a count variable measured on each node of the network ($i = 1, \dots, N$), over a window of time ($t = 1, \dots, T$). The data is a N -dimensional vector of time series $Y_t = (Y_{1,t}, \dots, Y_{i,t}, \dots, Y_{N,t})'$, which is observed over the domain $t = 1, 2, \dots, T$; in this way, a univariate time series is observed for each node, say $Y_{i,t}$, with corresponding conditional expectation $\lambda_{i,t}$. Denote by $\lambda_t = E(Y_t | \mathcal{F}_{t-1})$ with $\lambda_t = (\lambda_{1,t}, \dots, \lambda_{i,t}, \dots, \lambda_{N,t})'$ the conditional expectation vector of the counts with respect to their past history \mathcal{F}_{t-1} . The following linear autoregressive network model takes into account the known relations between nodes

$$Y_{i,t} | \mathcal{F}_{t-1} \sim \text{Poisson}(\lambda_{i,t}), \quad \lambda_{i,t} = \beta_0 + \beta_1 n_i^{-1} \sum_{j=1}^N a_{ij} Y_{j,t-1} + \beta_2 Y_{i,t-1}, \quad (1)$$

where $n_i = \sum_{j \neq i} a_{ij}$ is the total number of connections starting from the node i , such that $i \rightarrow j$; called out-degree. We call (1) linear Poisson Network Autoregression of order 1, abbreviated by PNAR(1); (Armillotta and Fokianos, 2023a). From the left hand side equation of (1), we observe that the process $Y_{i,t}$ is assumed to be marginally Poisson but the joint process depends upon a copula function described in simulations at the end of the paper. Note that $\beta_0, \beta_1, \beta_2 > 0$ since the conditional mean of the Poisson is positive. Model (1) postulates that, for every single node i , the marginal conditional mean of the process is regressed on:

- the average count of the other nodes $j \neq i$ which have a connection with i ; the parameter β_1 is called network effect, as it measures the average impact of node i 's connections;
- the past count of the variable itself for i ; the coefficient β_2 is called autoregressive effect because it provides an estimator for the impact of past count $Y_{i,t-1}$.

Model (1) implies that only nodes directly followed by the focal node i (i.e. $i \rightarrow j$), possibly, have an impact on its mean process of counts. It is a reasonable assumption in many applications; for example, in a social network the activity of node k , which satisfies $a_{ik} = 0$, does not affect node i . Hence, (1) measures the effect of a network to the observed multivariate count time series. Moreover, the model accommodates different types of network connectivity i.e. $a_{i,j}$ does not necessarily take the values 1-0 (connected-not connected). For example, $a_{i,j} = 1/d_{i,j}$ where $d_{i,j}$ is some measure of distance between node i and node j and $a_{i,i} = 0$. In this way the network effect becomes a spatial network component; see the last paragraph of Knight et al. (2020, p.3) for a discussion about a similar set of weights.

More generally, the counts $Y_{i,t}$ can be assumed to depend on the last p lagged values and q

covariates. Then consider the PNAR(p, q) model

$$\lambda_{i,t} = \beta_0 + \sum_{h=1}^p \beta_{1h} \left(n_i^{-1} \sum_{j=1}^N a_{ij} Y_{j,t-h} \right) + \sum_{h=1}^p \beta_{2h} Y_{i,t-h} + \sum_{l=1}^q \delta_l Z_{i,l}, \quad (2)$$

where $\beta_0, \beta_{1h}, \beta_{2h} \geq 0$, for all $h = 1, \dots, p$, $\delta_l \geq 0$, $l = 1, 2, \dots, q$ and $Z_{i,l}$ are non-negative covariates measured for each node $i = 1, \dots, N$. If $p = 1$ and $q = 0$ set $\beta_{11} = \beta_1$, $\beta_{21} = \beta_2$ to obtain (1). Model (2) is stationary if $\sum_{h=1}^p (\beta_{1h} + \beta_{2h}) < 1$ (Armillotta and Fokianos, 2023a).

The linear form of (2) offers a great advantage interpreting the parameters but it accommodates positive covariates. A real valued covariate enters (2) through suitable transformations that ensure positivity (e.g. include $\exp(Z)$ instead of directly Z). This restriction is bypassed by the log-linear model (Armillotta and Fokianos, 2023a):

$$\nu_{i,t} = \beta_0 + \sum_{h=1}^p \beta_{1h} \left(n_i^{-1} \sum_{j=1}^N a_{ij} \log(1 + Y_{j,t-h}) \right) + \sum_{h=1}^p \beta_{2h} \log(1 + Y_{i,t-h}) + \sum_{l=1}^q \delta_l Z_{i,l}, \quad (3)$$

where $\nu_{i,t} = \log(\lambda_{i,t})$ and the observation are still marginally Poisson, $Y_{i,t} | \mathcal{F}_{t-1} \sim \text{Poisson}(\exp(\nu_{i,t}))$, for every $i = 1, \dots, N$. Then the model parameters are real-valued since $\nu_{i,t} \in \mathbb{R}$ and the covariates can take any real values. The stationarity condition turns out to be $\sum_{h=1}^p (|\beta_{1h}| + |\beta_{2h}|) < 1$. Moreover, the interpretation of coefficients is similar to the case of linear model (1) but on the log-scale.

2.1 Inference

Model (2), or (3), depends on the m -dimensional vector of unknown parameters $\theta = (\beta_0, \beta_{11}, \dots, \beta_{1p}, \beta_{21}, \dots, \beta_{2p}, \delta_1, \dots, \delta_q)'$, with $m = 1 + 2p + q$. We use of quasi-maximum likelihood methodology for estimation of θ ; see Wedderburn (1974) and Gourieroux et al. (1984). The Quasi Maximum Likelihood Estimator (QMLE) is the vector of parameters $\hat{\theta}$ maximizing the function

$$l_T(\theta) = \sum_{t=1}^T \sum_{i=1}^N \left(Y_{i,t} \log \lambda_{i,t}(\theta) - \lambda_{i,t}(\theta) \right), \quad (4)$$

which is the so called pooled Poisson log-likelihood (up to a constant). Note that (4) is not necessarily the *true* log-likelihood of the process but it serves as an approximation. In particular, (4) is the log-likelihood function that would have been obtained if all time series were contemporaneously independent. However, the QMLE is not computed under the assumption of independence because (4) is simply a *working* log-likelihood function. The choice of maximizing (4) is justified for several reasons: i) full likelihood based on the joint process is complex (see the last section); ii) the optimization of (4) guarantees consistency and asymptotic normality of QMLE for the *true* parameter vector θ_0 ; iii) the QMLE is asymptotically equivalent to the MLE if the true probability mass function belongs to the linear exponential family (Gourieroux et al., 1984); iv) simplified computations entailing increased speed for estimation. Robustness of the QMLE in finite samples has been verified by Armillotta and Fokianos (2023a) through extensive simulation studies.

When considering the linear model (2), the score function is

$$S_T(\theta) = \sum_{t=1}^T \sum_{i=1}^N \left(\frac{Y_{i,t}}{\lambda_{i,t}(\theta)} - 1 \right) \frac{\partial \lambda_{i,t}(\theta)}{\partial \theta} = \sum_{t=1}^T s_t(\theta). \quad (5)$$

Define $\partial \lambda_t(\theta) / \partial \theta'$ the $N \times m$ matrix of derivatives, $D_t(\theta)$ the $N \times N$ diagonal matrix with elements equal to $\lambda_{i,t}(\theta)$, for $i = 1, \dots, N$ and $\xi_t(\theta) = Y_t - \lambda_t(\theta)$ is the error sequence. Then, the empirical Hessian and conditional information matrices are given, respectively, by

$$H_T(\theta) = \sum_{t=1}^T \sum_{i=1}^N \frac{Y_{i,t}}{\lambda_{i,t}^2(\theta)} \frac{\partial \lambda_{i,t}(\theta)}{\partial \theta} \frac{\partial \lambda_{i,t}(\theta)}{\partial \theta'}, \quad B_T(\theta) = \sum_{t=1}^T \frac{\partial \lambda'_t(\theta)}{\partial \theta} D_t^{-1}(\theta) \Sigma_t(\theta) D_t^{-1}(\theta) \frac{\partial \lambda_t(\theta)}{\partial \theta'}, \quad (6)$$

where $\Sigma_t(\theta) = E(\xi_t(\theta) \xi_t'(\theta) | \mathcal{F}_{t-1})$ is the conditional covariance matrix evaluated at θ . Under suitable assumptions, Armillotta and Fokianos (2023a) proved that $\sqrt{NT}(\hat{\theta} - \theta_0) \xrightarrow{d} N(0, H^{-1} B H^{-1})$, when $N \rightarrow \infty$ and $T \rightarrow \infty$, where H and B are the theoretical limiting Hessian and information matrices, respectively, evaluated at the true value $\theta = \theta_0$. Then, a suitable estimator for the standard errors of θ is the square-rooted main diagonal of the empirical "sandwich" covariance matrix, i.e. $SE(\hat{\theta}) = \{\text{diag}[H_T(\hat{\theta})^{-1} B_T(\hat{\theta}) H_T(\hat{\theta})^{-1}]\}^{1/2}$. Closely related works to ours have employed (4) for inference; Paul et al. (2008) and Paul and Held (2011), among others. However, in such works (4) is

viewed as the true log-likelihood of the model and standard errors are computed by using the naive approach $SE_H(\hat{\theta}) = \{\text{diag}[H_T(\hat{\theta})^{-1}]\}^{1/2}$ which underestimates the real source of variation of the parameters when cross-section dependence among counts is present; see (6) which depends on the conditional covariance matrix of the process Y_t . The package **PNAR** returns robust standard errors as independence among counts is not assumed for their calculation. Similar theory holds for the log-linear model (3); details can be found in the aforementioned works.

2.2 Influenza data

To illustrate the use of **PNAR** we apply the methodology to the dataset `fluBYBW` from the **surveillance** package (Meyer et al., 2017). This dataset includes information about the weekly number of influenza A & B cases in the 140 districts of the two Southern German states Bavaria and Baden-Wuerttemberg, for the years 2001 to 2008 (416 time points). The response variable $Y = (Y_1, \dots, Y_t, \dots, Y_T)'$ is then a 416×140 matrix of collective disease counts. Figure 1 illustrates the data in these two regions during 2007. We model these data by a linear PNAR model as we discuss next.

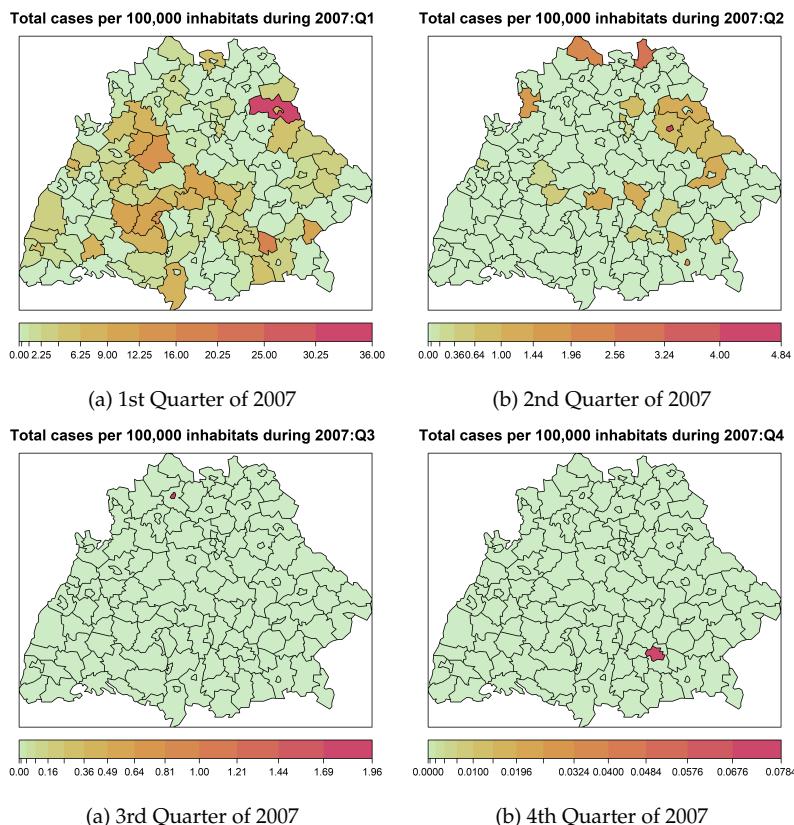


Figure 1: Quarterly flu cases in two Southern German states Bavaria and Baden-Wuerttemberg for 2007.

```
library(PNAR)
library(surveilance)
data(fluBYBW)
flu <- fluBYBW@observed
A_flu <- fluBYBW@neighbourhood
pop <- as.matrix(t(fluBYBW@populationFrac)[,1])
```

After loading **PNAR** we load **surveillance** for obtaining the 140×416 matrix of collective disease counts `flu`. The network adjacency matrix `A_flu` of dimension 140×140 has been obtained by linking two districts if they share (at least) a border. A covariate vector consisting of fraction of population in each district is introduced (`pop`). Model estimation for (2) when $p = 1$ and $p = 2$ are obtained below by using the function `lin_estimnarpq()` as follows:

```
est1.z <- lin_estimnarpq(y = flu, W = A_flu, p = 1, Z = pop)
```

```
est2.z <- lin_estimnarpq(y = flu, W = A_flu, p = 2, Z = pop)
```

Any type of non-negative matrix with zero main diagonal can be used as a valid adjacency matrix W . For instance consider weighted networks or inverse distance matrices, etc; see the last section for some alternatives when generating data. Optimization of (4) is implemented under the non-negativity constraint of coefficients satisfying the stationary condition. This is a nonlinear constrained optimization problem solved by means of a Sequential Quadratic programming (SQP) gradient-based algorithm (Kraft, 1994) of package **nloptr** (Ypma and Johnson, 2022). By default, the optimization is constrained in the stationary region; this can be removed by setting the option `uncons = TRUE` although this is not suggested because large sample properties of the estimators have been developed within the stationary region. The function `lin_estimnarpq()` has three additional features:

- `maxeval`: the maximum number of iterations for the optimization (the default is 100);
- `xtol_rel`: relative tolerance for the optimization termination condition (the default is 1e-8);
- `init`: starting value of the optimization (the default is NULL).

When `init = NULL` starting values are computed internally by ordinary LS using a lower barrier value of 0.01 because the regression coefficients cannot assume negative values. However, users can provide their own initial values through the argument `init`.

The function `lin_estimnarpq` returns as output a list consisting of the estimated coefficients, their associated standard errors, a z-test statistics with p -values, the score function evaluated at the optimum, the maximized log-likelihood, and the usual Akaike and Bayesian Information Criteria (AIC, BIC) accompanied by the Quasi IC (QIC) (Pan, 2001) which takes into account the fact that the log-likelihood (4) is a quasi log-likelihood; see Table 1 for the results, which are obtained in less than a second (see also Table 2). The score computed at the optimum values is of order 1e-5, on average, indicating successful convergence of the algorithm.

Table 1: Estimation of linear PNAR model (2) for $p = 1, 2$ and $Z = \text{pop}$. Standard errors of coefficients are given in parentheses.

p	β_0	$\beta_{1,1}$	$\beta_{1,2}$	$\beta_{2,1}$	$\beta_{2,2}$	δ	AIC	BIC	QIC
1	0.0118 (0.0022)	0.2862 (0.0204)	-	0.6302 (0.0345)	-	2.0027 (0.4475)	-6041.20	-6025.08	-5886.16
2	0.0081 (0.0018)	0.2303 (0.0218)	0.0136 (0.006)	0.5459 (0.0379)	0.1445 (0.0183)	1.7609 (0.3998)	-7447.48	-7423.30	-7240.56

All the estimated coefficients are positive and significantly different from 0. The autoregressive effect $\beta_{2,h}$ shows higher magnitude with respect to the network effects $\beta_{1,h}$ since past counts of the same district are, in general, more informative than the neighboring cases. Both network and autoregressive parameters when $p = 1$ have a larger magnitude when compared to the corresponding coefficients at $p = 2$. This can be explained since influenza has an incubation period of only 1-4 days (with an average of 2 days) and a patient is still contagious for no more than 5-7 days after becoming sick¹ so the case counts at first lag are more informative than the ones at second lag which are still important. The population covariate is significant with positive effect. Standard errors are computed by using the sandwich estimator.

To select the model order p , for the PNAR model (2), **PNAR** includes a function for estimating model parameters for a range of lag values. By default $p \in \{1, 2, \dots, 10\}$. This function returns the scatter plot of any IC (default is QIC) versus the lag order, for example

```
lin_ic_plot(y = flu, W = A_flu, p = 1:10, Z = pop, ic = "AIC")
```

Figure 2 shows the output of `lin_ic_plot()` for the case of AIC. Plots for the cases of BIC and QIC are similar and not shown. All information criteria point to the model with $p = 9$. However, from the corresponding estimation results reported in the Appendix, almost all β coefficients, which correspond to lags $p \geq 3$, are close to zero and non significant. Therefore, we decide to retain $p = 2$, for parsimony.

We compare estimation of standard errors after fitting the linear PNAR model to influenza data using **PNAR** and **surveillance** packages (covariate Z is excluded). The latter follows the standard error estimation according to the approach described in Paul et al. (2008) and Paul and Held (2011). The estimation of model (1) with **PNAR** gives

```
est1 <- lin_estimnarpq(y = flu, W = A_flu, p = 1)
summary(est1)
```

¹<https://www.cdc.gov/flu/about/disease/spread.htm>

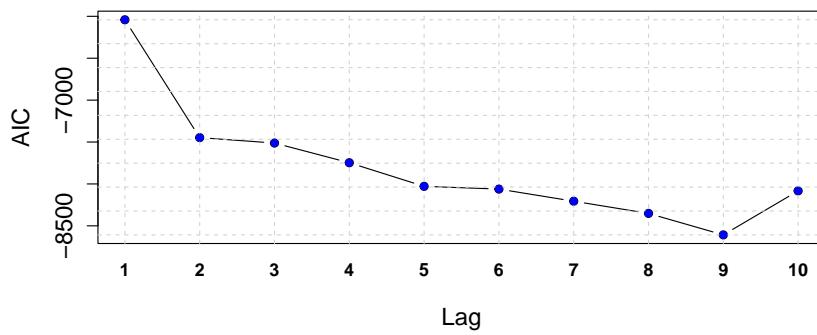


Figure 2: Scatter plot of AIC for PNAR(p) model versus p .

```
Coefficients:
            Estimate Std. Error   z value Pr(>|z|)
beta0  0.02460691 0.002722673 9.037777 1.598906e-19 ***
beta11 0.28952683 0.020393106 14.197289 9.522500e-46 ***
beta21 0.63082409 0.034462519 18.304642 7.598940e-75 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Using **surveillance**, the same model is fitted by the following lines of code:

```
sts0bj <- fluBYBW
ni <- rowSums(neighbourhood(sts0bj))
control <- list(ar = list(f = ~ 1),
ne = list(f = ~ 1, weights = neighbourhood(sts0bj) == 1,
offset = matrix(1/ni, nrow(sts0bj), ncol(sts0bj), byrow=TRUE)),
end = list(f = ~ 1),
family = "Poisson", keep.terms = TRUE)
fit1 <- hhh4(sts0bj, control)
coefSE <- coef(fit1, idx2Exp = TRUE, se = TRUE)
coefSE

            Estimate Std. Error
exp(ar.1)  0.63082409 0.0066281675
exp(ne.1)  0.28952683 0.0052551614
exp(end.1) 0.02460691 0.0007619529
```

Comparing the above outputs, note that estimated coefficients are identical (subject to rounding errors) but standard errors, obtained by **surveillance** package, underestimate their value as it was explained at the end of the inference section. In addition, **PNAR** is more user-friendly for fitting (2) and it is almost twice as fast. Indeed, the average estimation time over 10 calls of the same function is 0.42 seconds in comparison to 0.22 by the **PNAR** package (see Table 2).

The usefulness of the **PNAR** package is not limited to epidemiological data. For example, the dataset **crime**, which is already built in the package, contains monthly number of burglaries within the census blocks on the south side of Chicago during 2010-2015 and includes a network matrix, called **crime_W**, connecting two blocks sharing a border. The documentation of the functions **lin_estimnarpq()** and **log_lin_estimnarpq()** provides an example of PNAR models applied to crime data.

3 Extending Linearity

In this section, we give some motivating examples of nonlinear models for time series of networks as introduced by Armillotta and Fokianos (2023b). In addition, we provide testing procedures for

testing the linearity assumption. For ease of presentation, denote by $X_{i,t} = n_i^{-1} \sum_{j=1}^N a_{ij} Y_{j,t}$ the average neighbor network mean. When a drift in the intercept term of (2) is introduced, the following nonlinear Intercept Drift model, ID-PNAR(p, q), is obtained

$$\lambda_{i,t} = \frac{\beta_0}{(1 + X_{i,t-d})^\gamma} + \sum_{h=1}^p (\beta_{1h} X_{i,t-h} + \beta_{2h} Y_{i,t-h}) + \sum_{l=1}^q \delta_l Z_{i,l}, \quad (7)$$

where $\gamma \geq 0$. Model (7) shares similar to a linear model when the parameter γ takes small values and for $\gamma = 0$ reduces to (2). Instead, when γ takes values away from zero, model (7) introduces a perturbation, deviating from the linear model, which depends on the network mean at lag $(t-d)$, where d is an additional delay parameter such that $d = 1, 2, \dots, p$. Model (7) is directly applicable when the baseline effect, β_0 , varies over time as a function of the network.

Another interesting class of nonlinear models are regime switching models, i.e. models allowing for the dynamics of the count process to depend on different regimes (e.g. exponentially expanding pandemic/ dying out pandemic). We give two such examples of models whose specification is based either on a smooth distortion or an abrupt transition. With the same notation as before, the Smooth Transition PNAR model, ST-PNAR(p, q) assumes a smooth transition between two regimes and it is defined by

$$\lambda_{i,t} = \beta_0 + \sum_{h=1}^p \left(\beta_{1h} X_{i,t-h} + \beta_{2h} Y_{i,t-h} + \alpha_h e^{-\gamma X_{i,t-d}^2} X_{i,t-h} \right) + \sum_{l=1}^q \delta_l Z_{i,l}, \quad (8)$$

where $\gamma \geq 0$ and $\alpha_h \geq 0$, for $h = 1, \dots, p$. This models introduces a smooth regime switching behavior of the network effect making it possible to vary smoothly from $\beta_{1,h}$ to $\beta_{1,h} + \alpha_h$, as γ varies from large to small values. The additional delay parameter d determines the time of nonlinear transition that can be chosen. When $\alpha_h = 0$, for $h = 1, \dots, p$ in (8), the linear PNAR model (2) is recovered. In some applications the transition between regimes may be abrupt (e.g. financial market crashes). For this reason, we consider the Threshold PNAR model, T-PNAR(p, q), which is defined by

$$\lambda_{i,t} = \beta_0 + \sum_{h=1}^p \left(\beta_{1h} X_{i,t-h} + \beta_{2h} Y_{i,t-h} + (\alpha_0 + \alpha_{1h} X_{i,t-h} + \alpha_{2h} Y_{i,t-h}) I(X_{i,t-d} \leq \gamma) \right) + \sum_{l=1}^q \delta_l Z_{i,l}, \quad (9)$$

where $I(\cdot)$ is the indicator function and $\gamma \geq 0$ is now threshold parameter. Moreover, $\alpha_0, \alpha_{1,h}, \alpha_{2,h} \geq 0$, for $h = 1, \dots, p$. When $\alpha_0 = \alpha_{11} = \dots = \alpha_{2p} = 0$, model (9) reduces to (2).

For all nonlinear models, estimation of the unknown parameters is based on QMLE, following the discussion in the inference section. Therefore, analogous conclusions about estimation of regression parameters, their standard errors and model selection apply to the case of nonlinear models (7)–(9). Further details can be found in Armillotta and Fokianos (2023b).

3.1 Standard implementation of testing linearity

Testing linearity against several specific alternatives offers guidance about the type of nonlinear model to be fitted. Moreover, in certain cases where the linear model is nested within a nonlinear model, some nonlinear parameters may be inconsistently estimated (see the case ST-PNAR and T-PNAR models below) so testing linearity prevents incorrect estimation.

Consider model (7) and the hypothesis testing problem $H_0 : \gamma = 0$ vs. $H_1 : \gamma > 0$ which is a hypothesis between the linear PNAR (2) null assumption versus ID-PNAR alternative model (7). Consider the vector of all the parameters of ID-PNAR model (7), $\theta = (\beta_0, \beta_{11}, \dots, \beta_{2p}, \delta_1, \dots, \delta_q, \gamma)'$. Define the partition of the parameters $\theta = (\theta^{(1)}', \theta^{(2)}')'$, where $\theta^{(1)} = (\beta_0, \beta_{11}, \dots, \beta_{2p}, \delta_1, \dots, \delta_q)'$ is the sub-vector of parameters associated with the linear part of model. Let $\theta^{(2)}$ be the sub-vector of θ which corresponds to nonlinear parameters; for model (7), $\theta^{(2)} = \gamma$. Denote further by $S_T(\theta) = (S_T^{(1)'}(\theta), S_T^{(2)'}(\theta))'$ the corresponding partition of the quasi score function (5). We develop a quasi score test statistic based on the quasi log-likelihood (4). This is a convenient choice, because the score test requires estimation of model under the null hypothesis, i.e. under the linear model. Then the restricted estimator is denoted by $\tilde{\theta} = (\tilde{\beta}_0, \tilde{\beta}_{11}, \dots, \tilde{\beta}_{2p}, \tilde{\delta}_1, \dots, \tilde{\delta}_q)'$ and it is usually simpler to compute. The quasi score test statistic is given by

$$LM_T = S_T^{(2)'}(\tilde{\theta}) \Sigma_T^{-1}(\tilde{\theta}) S_T^{(2)}(\tilde{\theta}), \quad (10)$$

with $\Sigma_T(\tilde{\theta}) = \tilde{J} H_T^{-1}(\tilde{\theta}) \tilde{J}' \left(\tilde{J} H_T^{-1}(\tilde{\theta}) B_T(\tilde{\theta}) H_T^{-1}(\tilde{\theta}) \tilde{J}' \right)^{-1} \tilde{J} H_T^{-1}(\tilde{\theta}) \tilde{J}'$, where $\tilde{J} = (O_{m_2 \times m_1}, I_{m_2})$, I_s is a $s \times s$ identity matrix and $O_{a \times b}$ is a $a \times b$ matrix of zeros. $\Sigma_T(\tilde{\theta})$ is the estimator for the unknown covariance matrix $\Sigma = \text{Var}[S_T^{(2)}(\tilde{\theta})]$. It can be proved that the quasi score test (10) converges, asymptotically, to a standard normal distribution under the null hypothesis.

totically, to a $\chi^2_{m_2}$ distribution, where m_2 is the number of nonlinear parameter tested (Armillotta and Fokianos, 2023b). Then, in case of model (7), $m_2 = 1$ and we can compute the p -value of the test statistic (10) by $p = P(\chi^2_1 \geq M)$, where M is the observed value of the test statistic LM_T .

3.2 Non-standard implementation of testing linearity

Suppose now we wish to test linearity against (8). Then, considering the hypotheses $H_0 : \alpha_1 = \dots = \alpha_p = 0$ versus $H_1 : \alpha_h \neq 0$ for some $h = 1, \dots, p$, we note that this testing problem is non-standard because it is not possible to estimate the value of γ under H_0 . Note that the parameter γ exists in the partition of the score function (5) related to nonlinear parameters, in particular in $\partial\lambda_{i,t}(\theta)/\partial\alpha_h$, and its associated covariance matrix. Hence, all relevant quantities for computing (10) are functions of γ ; that is $S_T^{(2)}(\tilde{\theta}, \gamma)$, $\Sigma_T(\tilde{\theta}, \gamma)$ and $LM_T(\gamma)$. The model is then subject to non-identifiable parameter γ under the null. So if the true model is the linear PNAR model but a ST-PNAR is estimated instead the smoothing parameter γ will not be consistently estimated. Analogous conclusions hold for testing linearity against the T-PNAR model (9), where the threshold parameter γ is not identifiable under the null. When this issue arises, the standard theory does not apply and a chi-square type test is not suitable any more; see Davies (1987), Hansen (1996) and Armillotta and Fokianos (2023b), among others. It is clear that the value of the test changes by varying $\gamma \in \Gamma$, where Γ is some domain. A summary function of the test, computed under different values of γ , is then routinely employed in applications; a typical choice is $g_T = \sup_{\gamma \in \Gamma} LM_T(\gamma)$; see Armillotta and Fokianos (2023b) who established the convergence of g_T to $g = \sup_{\gamma \in \Gamma} LM(\gamma)$, where g is a function of a chi-square process, $LM(\gamma)$. The values of the latter asymptotic distribution cannot be tabulated, as they depends on unknown values of γ . Hence, we give methodology for computing p -values of such sup-type test statistic since they cannot be obtained otherwise.

Davies' bound. Since the space $\Gamma = [\gamma_L, \gamma_U]$ is usually assumed to be a closed interval, in practice, we take $\Gamma_F = (\gamma_L, \gamma_1, \dots, \gamma_l, \gamma_U)$ i.e. a grid of values for the non-identifiable parameter γ , and the sup-test is obtained as the maximum of the tests $LM_T(\gamma)$ computed over Γ_F . Davies (1987) showed that the p -value of the sup-test is approximately bounded by

$$P \left[\sup_{\gamma \in \Gamma_F} LM(\gamma) \geq M \right] \leq P(\chi^2_{m_2} \geq M) + VM^{\frac{1}{2}(m_2-1)} \frac{\exp(-\frac{M}{2})2^{-\frac{m_2}{2}}}{\Gamma(\frac{m_2}{2})}, \quad (11)$$

where M is the value of the sup-test statistic computed in the available sample, $\Gamma(\cdot)$ is the gamma function, and V is the approximated total variation

$$V = |LM_T^{\frac{1}{2}}(\gamma_1) - LM_T^{\frac{1}{2}}(\gamma_L)| + |LM_T^{\frac{1}{2}}(\gamma_2) - LM_T^{\frac{1}{2}}(\gamma_1)| + \dots + |LM_T^{\frac{1}{2}}(\gamma_U) - LM_T^{\frac{1}{2}}(\gamma_l)|.$$

Equation (11) shows how to approximate the p -values of the sup-type test in a straightforward way. Indeed, by adding to the tail probability of a chi-square distribution a correction term, which depends on the total variation of the process, we obtain the desired bound. This method is attractive for its simplicity and speed even when the dimension N of the network is large. However, the method approximate p -values with their bound (11) leading to a conservative test. In addition, (11) cannot be applied to the T-PNAR model (9), because the total variation requires differentiability of the asymptotic distribution $LM(\gamma)$ under the null hypothesis (Davies, 1987, p. 36), a condition that is not met for the case of T-PNAR models.

Bootstrapping the test statistic. Based on the previous arguments, we suggest an alternative p -value approximation of the test statistic employing stochastic permutations (Hansen, 1996; Armillotta and Fokianos, 2023b)-see Algorithm 1.

An approximation of the p -values is obtained from step 10 of Algorithm 1, where g_T is the value of the sup-test statistic computed on the available sample. When the number of bootstrap replications J is large enough, p_T^J provides a good approximation to the unknown p -values of the test. Then, the null hypothesis H_0 is rejected if p_T^J is smaller than a given significance level.

3.3 Revisiting the influenza data

We now apply the testing methodology described in the previous sections to influenza data. Consider testing linearity of the PNAR(2) model against the nonlinear ID-PNAR(2) (7) with $d = 1$. Analogous

Algorithm 1 Score bootstrap

```

1: Obtain the constrained QMLE of the linear model (2), say  $\tilde{\theta}$ 
2: for  $j = 1, \dots, J$  do
3:   for  $t = 1, \dots, T$  do
4:     Generate  $v_{t,j} \sim N(0, 1)$ 
5:   end for
6:   Compute  $S_T^{v_j}(\tilde{\theta}, \gamma) = \sum_{t=1}^T s_t(\tilde{\theta}, \gamma)v_{t,j}$ 
7:   Compute the test  $LM_T^{v_j}(\gamma) = S_T^{v_j(2)'}(\tilde{\theta}, \gamma)\Sigma_T^{-1}(\tilde{\theta}, \gamma)S_T^{v_j(2)}(\tilde{\theta}, \gamma)$ 
8:   Optimize  $LM_T^{v_j}(\gamma)$  for  $\gamma$  and take  $g_T^j = \sup_{\gamma \in \Gamma} LM_T^{v_j}(\gamma)$ 
9: end for
10: Compute  $p_T^J = J^{-1} \sum_{j=1}^J I(g_T^j \geq g_T)$ 

```

results have been obtained for $d = 2$ and therefore are omitted. The quasi score test (10) is computed by

```

id2.z <- score_test_nonlinpq_h0(b = est2.z$coefs[, 1], y = flu, W = A_flu,
p = 2, d = 1, Z = pop)
id2.z

Linearity test against non-linear ID-PNAR(p) model

data: coefficients of the PNAR(p, q)/time series data/order/lag/covariates/
chi-square-test statistic = 7.2318, df = 1, p-value = 0.007162
alternative hypothesis: True gamma parameter is greater than 0

```

where the first argument requires estimates under the null hypothesis $H_0 : \gamma = 0$ which have been obtained already. The rest of arguments follow previous syntax. For this testing problem the test is asymptotically chi-square distributed with 1 degree of freedom. The output lists the test statistic value and its corresponding p -value. There is strong indication to reject the linear model in favour of model (7).

Next consider testing linearity against the ST-PNAR(2) alternative (8). In this case, the test behaves in a non-standard way so we will be using the Davies' bound p -value (DV) for the sup-type test (11) and the bootstrap p -value approximation. First, the DV is computed by calling the following function

```

dv2.z <- score_test_stnarpq_DV(b = est2.z$coefs[, 1], y = flu, W = A_flu,
p = 2, d = 1, Z = pop)

```

Extreme values for the range of $\gamma \in \Gamma_F$ are computed internally in such a way that γ_U and γ_L are those values of γ where, on average, the smoothing function $\exp(-\gamma X_{i,t-d}^2)$ is equal to 0.1 and 0.9, respectively. In this way, during the optimization procedure, the extremes of the function domain are excluded. For more details see the **PNAR** manual ([Tsagris et al., 2023](#)). The user can specify different values for γ_L and γ_U using the arguments `gama_L`, `gama_U` and the number of grid values `len` (the default is 100). Results suggest again a deviation from linearity of the model, i.e.

```

dv2.z

Test for linearity of PNAR(p) versus the non-linear ST-PNAR(p)

```

```

data: coefficients of the PNAR(p, q)/time series data/order/lag/covariates
/lower gamma/upper gamma/length
chi-square-test statistic = 35.074, df = 2, p-value = 9.076e-08
alternative hypothesis: At least one coefficient of the non-linear component
is not zero

```

Next, we apply Algorithm 1 to compute the bootstrap p -values for the sup-type test statistic. In this case, the observed value of $\sup_{\gamma \in \Gamma} LM(\gamma)$ has to be computed. Initially, perform a global optimization of the $LM_T(\gamma)$ for the ST-PNAR(p) model, with respect to the nuisance scale parameter γ by using Brent's algorithm ([Brent, 1973](#)) in the interval [`gama_L` to `gama_U`], (see previous discussion for their computation). To ensure global optimality, the optimization is performed on runs at `len-1` consecutive

equidistant sub-intervals and the global optimum is determined by the maximum over those sub-intervals. The default value for `len` is 10. Then using the function `global_optimise_LM_stnarpq` with the same arguments we obtain the optimal γ value and the corresponding value of the test statistic:

```
go1.z2 <- global_optimise_LM_stnarpq(b = est2.z$coefs[, 1], y = flu, W = A_flu,
p = 2, d = 1, Z = pop)
go1.z2$gama
[1] 8.387526
go1.z2$supLM
[1] 35.07402
```

This information is used as follows

```
boot1.z2 <- score_test_stnarpq_j(supLM = go1.z2$supLM, b = est2.z$coefs[, 1],
y = flu, W = A_flu, p = 2, d = 1, Z = pop,
J = 499, ncores = 7, seed = 1234)
```

which implements Algorithm 1 using $J = 499$ bootstrap replicates. The function uses a parallel processing option; the user can set the number of cores `ncores` (the default is no parallel). The seed for random number generation assures reproducibility of the results.

```
boot1.z2$pJ
[1] 0.002004008
boot1.z2$cpJ
[1] 0.004
```

The above output gives (among other information) p_T^J of step 10 of Algorithm 1 and an alternative corrected unbiased estimator for the p -value which is $cp_T^J = (J+1)^{-1} \left[\sum_{j=1}^J I(g_T^j \geq g_T) + 1 \right]$. Like in the case of DV p -value, linearity is rejected.

We work analogously for testing the PNAR(2) model versus the T-PNAR(2) model (9). Note that optimization of $LM_T(\gamma)$, in this case, is based on `gama_L` and `gama_U` which are obtained (by default) as the mean over $i = 1, \dots, N$ of 20% and 80% quantiles of the empirical distribution of the network mean $X_{i,t}$ for $t = 1, \dots, T$. In this way, during the optimization process, the indicator function $I(X_{i,t-d} \leq \gamma)$ avoids values close to 0 or 1. Alternatively, their value can be supplied by the user. The functions used are analogous to the functions used for the ST-PNAR(2).

```
tgo1.z2 <- global_optimise_LM_tnarpq(b = est2.z$coefs[, 1], y = flu, W = A_flu,
p = 2, d = 1, Z = pop)
tgo1.z2$gama
[1] 0.1257529
tgo1.z2$supLM
[1] 49.06505

tboot1.z2 <- score_test_tnarpq_j(supLM = tgo1.z2$supLM, b = est2.z$coefs[, 1],
y = flu, W = A_flu, p = 2, d = 1, Z = pop,
J = 499, ncores = 7, seed = 1234)
tboot1.z2$pJ
[1] 0.3907816
tboot1.z2$cpJ
[1] 0.392
```

The test does not reject the null hypothesis of linearity, in the case of a threshold model. Overall, the analysis shows that the linear PNAR model may not be a suitable model to fit such epidemic data and nonlinear alternatives should be considered. In particular, evidence of a nonlinear drift in the intercept and of a regime switching mechanism is detected. In addition, it appears that a smooth regime switching mechanism might be more appropriate for the data.

3.4 Computational Speed

Package `PNAR` is quite efficient in terms of computational speed, especially for estimation problems. We have employed the `Rfast` and `Rfast2` packages (Papadakis et al., 2023a,b) wherever possible to ensure computational speed. We run each function 10 times and compute the average time required to be executed. We use a laptop computer equipped with Intel Core i7 processor (3.00GHz) and 16 GB of RAM. Results are given in Table 2. Estimation of linear PNAR model and standard testing (ID-PNAR model) is fast. Computations of p -values, in the case of non-identifiable parameters, requires several

evaluations of the test statistic on a grid of values for γ (Davies' bound) or global optimization of the test statistic (bootstrap). However both tasks are executed in a satisfactory amount of time. The bootstrap approximation Algorithm 1 runs slower but it still executed within satisfactory time limits. Computational speed, for both Davies' bound and bootstrap p -values, can be further increased by reducing the length of the γ grid for the former and the number of bootstrap replications for the latter. Increasing the number of cores will provide faster bootstrap approximated p -values.

Table 2: Average computation times (in seconds) for the functions called in the text.

PNAR Estimation					
p	1	2	4	9	
No cov.	0.22	-	-	-	
Cov.	0.39	0.57	0.89	1.26	
Tests					
Models	χ^2_1	DV	Global opt.	Bootstrap	
ID-PNAR	0.43	-	-	-	
ST-PNAR	-	12.88	3.74	74.60	
T-PNAR	-	-	1.30	75.20	

4 Simulating network count time series

In this last section we present a further novel implementation of the **PNAR** package which can be used to simulate network count time series from linear and nonlinear models with multivariate copula Poisson distribution, as we explain next.

Equation (1) does not include information about the joint dependence structure of the PNAR(1) model. Following Fokianos et al. (2020) the joint multivariate distribution of the vector count time series Y_t is defined as $Y_t = N_t(\lambda_t)$ where, $\{N_t\}$ is a sequence of copula-Poisson processes, that is $N_t(\lambda_t)$ is a sequence of N -dimensional IID marginally Poisson count processes, with intensity 1, counting the number of events in the interval of time $[0, \lambda_{1,t}] \times \dots \times [0, \lambda_{N,t}]$, and whose structure of dependence is modeled through a copula function $C(\rho)$ on their associated exponential waiting times random variables. The algorithm is described below for model (1).

Consider a network matrix A and a set of values $(\beta_0, \beta_1, \beta_2)'$ for model (1). Moreover, define a starting mean vector at time $t = 0$, say $\lambda_0 = (\lambda_{1,0}, \dots, \lambda_{N,0})'$.

1. Let $U_l = (U_{1,l}, \dots, U_{N,l})'$, for $l = 1, \dots, K$ a sample from a N -dimensional copula $C(u_1, \dots, u_N; \rho)$, where $U_{i,l}$ follows a Uniform(0,1) distribution, for $i = 1, \dots, N$.
2. The transformation $E_{i,l} = -\log U_{i,l} / \lambda_{i,0}$ follows the exponential distribution with parameter $\lambda_{i,0}$, for $i = 1, \dots, N$.
3. If $E_{i,1} > 1$, then $Y_{i,0} = 0$, otherwise $Y_{i,0} = \max \left\{ k \in [1, K] : \sum_{l=1}^k X_{i,l} \leq 1 \right\}$, by taking K large enough. Then, $Y_{i,0} | \lambda_0 \sim \text{Poisson}(\lambda_{i,0})$, for $i = 1, \dots, N$. So, $Y_0 = (Y_{1,0}, \dots, Y_{N,0})'$ is a set of (conditionally) marginal Poisson processes with mean λ_0 .
4. By using the model (1), λ_1 is obtained.
5. Return back to step 1 to obtain Y_1 , and so on.

In applications, choose K large, e.g. $K = 100$; its values generally depends on the magnitude of observed data. Moreover, the copula $C(\rho)$ depends on one or more unknown parameters, say ρ , which capture the contemporaneous correlation among the variables. The proposed algorithm ensures that all marginal distributions of $Y_{i,t}$ are univariate Poisson, conditionally to the past, as described in (1), while it introduces an arbitrary dependence among them in a flexible and general way by the copula construction through the parameter ρ . An analogous process is employed for generating log-linear and nonlinear count network models by suitable modifications.

Multivariate Poisson-type distributions have typically complicated form and their covariance matrix might not be appropriate (Fokianos et al., 2020); this inspired the adoption of this simulation methodology. Imposing a copula directly on Poisson marginals can lead to identifiability issues (Genest and Nešlehová, 2007). For further details see Fokianos et al. (2020), Armillotta and Fokianos (2023a) and the recent review in Fokianos (2022).

PNAR allows to generate multivariate count times from well-known network models like the Erdős-Rényi Model (Erdős and Rényi, 1959), with the function `adja_gnp()`, or the Stochastic Block Model (SBM) (Wang and Wong, 1987) with the function `adja()`. Such functions are based on the `igraph` package (Csardi and Nepusz, 2006); see Tsagris et al. (2023) for details.

```
set.seed(1234)

W_SBM <- adja(N = 10, K = 2, alpha = 0.7, directed = TRUE)

sim1 <- poisson.MODpq(b = c(0.2, 0.2, 0.4), W = W_SBM, p = 1, TT = 100, N = 10,
copula = "gaussian", corrtype = "equicorrelation", rho = 0.5)
sim1$y
```

The first function randomly generates an adjacency matrix from the directed SBM model with 10 nodes and 2 groups. The second function generates a 100×10 time series matrix object of network counts from the linear PNAR(1) model (1) where the joint dependence in the data generating process is modeled by a Gaussian copula with $\rho = 0.5$. The "equicorrelation" option generates a correlation matrix for the Gaussian copula where all the off-diagonal entries equal ρ . Another type of correlation matrix which can be used is the "toeplitz" option that returns a correlation matrix whose generic off-diagonal (i, j) -element is $\rho^{|i-j|}$. Moreover, other copula functions can also be chosen as the t or the Clayton copula. Some of the 10 simulated time series are plotted in Figure 3, for illustration. Analogous functions are provided for generating synthetic data from log-linear or nonlinear Poisson network model (Table 4 in the Appendix).

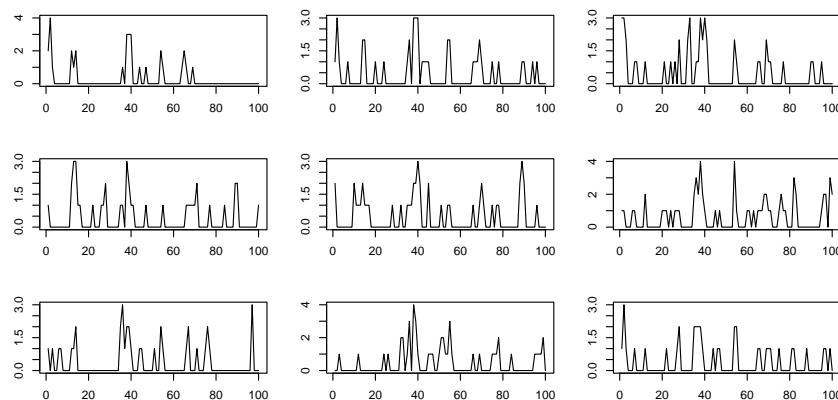


Figure 3: Simulated count time series from the linear PNAR(1) model.

5 Conclusion

There exists R software for fitting Network Autoregressive models (**GNAR**, **surveillance**). However, no published package includes functions for inference with nonlinear Network Autoregressive models. **PNAR** fills this gap by providing users tools for efficient estimation of (log-)linear models with proper robust standard errors, test statics and computational algorithms for testing model linearity and new simulation methodology for generating (log-)linear and nonlinear network count time series models. We showed that all these tasks are executed with a minimal computational effort.

There are a number of possible developments for **PNAR**. One possibility is to include several other nonlinear models and develop related linearity tests. In addition, developing a negative binomial quasi-likelihood estimation method offers more flexibility to model fitting. Alternative ways to compute p -values, for example employing different bootstrap approximation, may also be considered. Further simulation methods for generating network count time series are easily accommodated by suitable modification of the copula Poisson algorithm. All these extensions provide users with new set of tools for inference in the broad framework of multivariate discrete-valued time series models.

6 Appendix

Output from estimation of PNAR model (2) with lag $p = 9$ and population covariate:

```

est9.z <- lin_estimnarpq(y = flu, W = A_flu, p = 9, Z = pop)
summary(est9.z)

Coefficients:
            Estimate   Std. Error      z value    Pr(>|z|)
beta0  8.046948e-03 0.0018543147 4.339581e+00 1.427548e-05 ***
beta11 2.291637e-01 0.0222504112 1.029930e+01 7.097436e-25 ***
beta12 1.219436e-02 0.0058931749 2.069234e+00 3.852414e-02 *
beta13 2.880415e-08 0.0034601665 8.324498e-06 9.999934e-01
beta14 6.840207e-08 0.0047642557 1.435735e-05 9.999885e-01
beta15 4.994822e-08 0.0022529259 2.217038e-05 9.999823e-01
beta16 6.335964e-07 0.0023712114 2.672037e-04 9.997868e-01
beta17 1.096226e-06 0.0022836178 4.800390e-04 9.996170e-01
beta18 1.094862e-07 0.0018014328 6.077731e-05 9.999515e-01
beta19 1.230050e-07 0.0017010627 7.231070e-05 9.999423e-01
beta21 5.462136e-01 0.0389427608 1.402606e+01 1.079843e-44 ***
beta22 1.361396e-01 0.0179869627 7.568794e+00 3.767043e-14 ***
beta23 1.404481e-02 0.0053121667 2.643894e+00 8.195825e-03 **
beta24 1.764756e-08 0.0023065543 7.651051e-06 9.999939e-01
beta25 1.874096e-06 0.0016152031 1.160285e-03 9.990742e-01
beta26 2.036961e-05 0.0023094311 8.820187e-03 9.929626e-01
beta27 1.492721e-06 0.0019472654 7.665731e-04 9.993884e-01
beta28 6.414357e-08 0.0018167793 3.530620e-05 9.999718e-01
beta29 8.818047e-07 0.0009314276 9.467238e-04 9.992446e-01
delta1 1.412017e+00 0.3536615653 3.992566e+00 6.536216e-05 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Log-likelihood value: 4324.057
AIC: -8608.114 BIC: -8527.5 QIC: -8362.176

```

Table 3: A brief overview of the main functions in PNAR.

Function	Description
lin_estimnarpq()	Fitting the linear PNAR(p, q) model.
log_lin_estimnarpq()	Fitting the log-linear PNAR(p, q) model.
score_test_nonlinpq_h0()	Score test of linear PNAR versus ID-PNAR model.
score_test_stnarpq_DV()	Score test of linear PNAR versus ST-PNAR model by (11).
global_optimise_LM_stnarpq()	Maximize test statistic of ST-PNAR for nuisance parameters.
score_test_stnarpq_j()	Bootstrap score test of linear PNAR versus ST-PNAR model.
global_optimise_LM_tnarpq()	Maximize test statistic of T-PNAR for nuisance parameters.
score_test_tnarpq_j()	Bootstrap score test of linear PNAR versus T-PNAR model.

Table 4: A brief overview of functions simulating network count time series models in PNAR.

Function	Description
poisson.MODpq()	Generation from a linear PNAR(p) model with covariates.
poisson.MODpq.log()	Generation from a log-linear PNAR(p) model with covariates.
poisson.MODpq.nonlin()	Generation from a Intercept Drift PNAR(p) model with covariates.
poisson.MODpq.stnar()	Generation from a Smooth Transition PNAR(p) model with covariates.
poisson.MODpq.tnar()	Generation from a Threshold PNAR(p) model with covariates.

7 Acknowledgments

Part of this work was done while M. Armillotta was with the Department of Mathematics & Statistics, University of Cyprus. We cordially thank R. Hyndman and two anonymous reviewers for several constructive comments that improved an earlier version of the manuscript. In addition, we thank M. Papadakis for his help with the S3 methods (print() and summary() functions). M. Armillotta acknowledges financial support from the EU Horizon Europe programme under the Marie Skłodowska-Curie grant agreement No.101108797.

References

- M. Armillotta and K. Fokianos. Count network autoregression. *Journal of Time Series Analysis*, jtsa.12728, 2023a. URL <http://doi.org/10.1111/jtsa.12728>. [p254, 255, 256, 264]
- M. Armillotta and K. Fokianos. Nonlinear Network Autoregression. *The Annals of Statistics*, 51(6): 2526–2552, 2023b. URL <https://doi.org/10.1214/23-AOS2345>. [p254, 259, 260, 261]
- J. Bracher and L. Held. Endemic-epidemic models with discrete-time serial interval distributions for infectious disease prediction. *International Journal of Forecasting*, 38:1221–1233, 2020. [p254]
- R. Brent. *Algorithms for Minimization Without Derivatives*. Prentice-Hall, 1973. [p262]
- G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*:1695, 2006. URL <https://igraph.org>. [p265]
- R. Davies. Hypothesis testing when a nuisance parameter is present only under the alternative. *Biometrika*, 74:33–43, 1987. [p261]
- P. Erdős and A. Rényi. On random graphs I. *Publicationes Mathematicae Debrecen*, 6, 1959. [p265]
- K. Fokianos. Multivariate count time series modelling. *To appear in Econometrics and Statistics*, 2022. [p264]
- K. Fokianos, B. Støve, D. Tjøstheim, and P. Doukhan. Multivariate count autoregression. *Bernoulli*, 26: 471–499, 2020. [p254, 264]
- C. Genest and J. Nešlehová. A primer on copulas for count data. *Astin Bulletin*, 37:475–515, 2007. [p264]
- C. Gourieroux, A. Monfort, and A. Trognon. Pseudo maximum likelihood methods: Theory. *Econometrica*, 52:681–700, 1984. [p254, 256]
- B. Hansen. Inference when a nuisance parameter is not identified under the null hypothesis. *Econometrica*, 64:413–430, 1996. [p261]
- L. Held and M. Paul. Modeling seasonality in space-time infectious disease surveillance data. *Biometrical Journal*, 54:824–843, 2012. [p254]
- L. Held, M. Höhle, and M. Hofmann. A statistical framework for the analysis of multivariate infectious disease surveillance counts. *Statistical Modelling*, 5:187–199, 2005. [p254]
- M. Höhle, S. Meyer, and M. Paul. *surveillance: Temporal and Spatio-Temporal Modeling and Monitoring of Epidemic Phenomena*, 2022. URL <https://CRAN.R-project.org/package=surveillance>. R package version 1.21.1. [p254]
- M. Knight, K. Leeming, G. Nason, and M. Nunes. Generalized network autoregressive processes and the GNAR package. *Journal of Statistical Software*, 96:1–36, 2020. URL <https://www.jstatsoft.org/v096/i05>. [p254, 255]
- D. Kraft. Algorithm 733: TOMP–Fortran modules for optimal control calculations. *ACM Transactions on Mathematical Software (TOMS)*, 20:262–281, 1994. [p258]
- K. Leeming, G. Nason, M. Nunes, and J. Wei. *GNAR: Methods for Fitting Network Time Series Models*, 2023. URL <https://CRAN.R-project.org/package=GNAR>. R package version 1.1.2. [p254]
- S. Meyer and L. Held. Power-law models for infectious disease spread. *The Annals of Applied Statistics*, 8:1612–1639, 2014. [p254]
- S. Meyer, L. Held, and M. Höhle. Spatio-temporal analysis of epidemic phenomena using the R package surveillance. *Journal of Statistical Software*, 77:1–55, 2017. [p254, 257]
- W. Pan. Akaike’s information criterion in generalized estimating equations. *Biometrics*, 57:120–125, 2001. [p258]
- M. Papadakis, M. Tsagris, M. Dimitriadis, S. Fafalios, I. Tsamardinos, M. Fasiolo, G. Borboudakis, J. Burkhardt, C. Zou, K. Lakiotaki, and C. Chatzipantsiou. *Rfast: A Collection of Efficient and Extremely Fast R Functions*, 2023a. URL <https://CRAN.R-project.org/package=Rfast>. R package version 2.0.8. [p263]

- M. Papadakis, M. Tsagris, S. Fafalios, and M. Dimitriadis. *Rfast2: A Collection of Efficient and Extremely Fast R Functions II*, 2023b. URL <https://CRAN.R-project.org/package=Rfast2>. R package version 0.1.5.1. [p²⁶³]
- M. Paul and L. Held. Predictive assessment of a non-linear random effects model for multivariate time series of infectious disease counts. *Statistics in Medicine*, 30:1118–1136, 2011. [p^{254, 256, 258}]
- M. Paul, L. Held, and A. M. Toschke. Multivariate modelling of infectious disease surveillance data. *Statistics in Medicine*, 27:6250–6267, 2008. [p^{254, 255, 256, 258}]
- M. Tsagris, M. Armillotta, and K. Fokianos. *PNAR: Poisson Network Autoregressive Models.*, 2023. URL <https://CRAN.R-project.org/package=PNAR>. R package version 1.6. [p^{254, 262, 265}]
- Y. J. Wang and G. Y. Wong. Stochastic blockmodels for directed graphs. *Journal of the American Statistical Association*, 82:8–19, 1987. [p²⁶⁵]
- R. W. Wedderburn. Quasi-likelihood functions, generalized linear models, and the Gauss-Newton method. *Biometrika*, 61:439–447, 1974. [p^{254, 256}]
- J. Ypma and S. G. Johnson. *nloptr: R Interface to NLOpt*, 2022. URL <https://CRAN.R-project.org/package=nloptr>. R package version 2.0.3. [p²⁵⁸]
- X. Zhu, R. Pan, G. Li, Y. Liu, and H. Wang. Network vector autoregression. *The Annals of Statistics*, 45:1096–1123, 2017. [p²⁵⁴]

Mirko Armillotta
Vrije Universiteit Amsterdam
Department of Econometrics and Data Science
Netherlands
Tinbergen Institute
Netherlands
ORCID: 0000-0002-0548-6957
m.armillotta@vu.nl

Michail Tsagris
University of Crete
Department of Economics
Greece
ORCID: 0000-0002-2049-3063
mtsagris@uoc.gr

Konstantinos Fokianos
University of Cyprus
Department of Mathematics and Statistics
Cyprus
ORCID: 0000-0002-0051-711X
fokianos@ucy.ac.cy

SUrival Control Chart EStimation Software in R: the success Package

by Daniel Gomon, Marta Fiocco, Hein Putter, and Mirko Signorelli

Abstract Monitoring the quality of statistical processes has been of great importance, mostly in industrial applications. Control charts are widely used for this purpose, but often lack the ability to monitor survival outcomes. Recently, inspecting survival outcomes has become of interest, especially in medical settings where outcomes often depend on risk factors of patients. For this reason many new survival control charts have been devised and existing ones have been extended to incorporate survival outcomes. The package `success` allows users to construct risk-adjusted control charts for survival data. Functions to determine control chart parameters are included, which can be used even without expert knowledge on the subject of control charts. The package allows to create static as well as interactive charts, which are built using `ggplot2` (Wickham 2016) and `plotly` (Sievert 2020).

1 Introduction

Inspecting the quality of a survival process is of great importance, especially in the medical field. Many of the methods currently used to inspect the quality of survival processes in a medical setting, such as funnel plots (Spiegelhalter 2005), Bernoulli Cumulative sum (CUSUM) charts (Steiner et al. 2000) and exponentially weighted moving average (EWMA) charts (Cook, Coory, and Webster 2011) work only with binary outcomes, and are thus not appropriate for survival outcomes. These charts require the continuous time outcome to be dichotomized, often leading to delays when trying to detect problems in the quality of care. To overcome this limitation, (Biswas and Kalbfleisch 2008) developed a continuous time CUSUM procedure (which we call the BK-CUSUM), that can be used to inspect survival outcomes in real time. (Gomon et al. 2022) proposed a generalization of the BK-CUSUM chart called the Continuous Time Generalized Rapid Response CUSUM (CGR-CUSUM). The CGR-CUSUM allows to estimate some of the parameters involved in the construction of the chart, overcoming the need for the user to correctly specify parameters that the BK-CUSUM procedure relies on. Recently other procedures for the continuous time inspection of survival outcomes have been developed, such as the improved Bernoulli CUSUM (Keefe et al. 2017), uEWMA chart (Steiner and Jones 2009) and STRAND chart (Grigg 2018). To the best of our knowledge there are no publicly available software implementations of these methods.

When constructing control charts in continuous time, not only the time to failure of a subject is of interest, but also the information provided by the survival up until current time is crucial. Many quality control methods cannot incorporate continuous time (survival) outcomes, requiring the continuous time outcome to be dichotomized (e.g. 30-day survival). The resulting binary data is called discrete time data. We provide an overview of some of the existing R packages which can be used for constructing control charts. The package `qcc` (Scrucca 2004) for discrete time data contains functions to construct many types of Shewhart, binary CUSUM, EWMA and other charts. The packages `qcr` (Flores 2021), `qicharts` (Anhoej 2021) and `ggQC` (Grey 2018) also allow for the construction of discrete time control charts, but differ in their graphical possibilities and their intended application area such as medicine, industry and economics. It is possible to assess some of the discrete time control charts by means of their zero/steady state average run length using the packages `spc` (Knoth 2021) and `vlad` (Wittenberg and Knoth 2020). The package `funnelR` (Kumar 2018) allows for the construction of funnel plots for proportion data, a method often used in medical statistics to visualise the difference in proportion over a time frame. The package `cusum` (Hubig 2019) can be used to monitor hospital performance using a Bernoulli CUSUM, also allowing users to easily determine control limits for continuously inspecting hospitals. The package however requires the input to be presented in a binary format.

Whereas many packages exist allowing for the construction of quality control charts on discrete time data, to the best of our knowledge there are currently few statistical software packages allowing for the construction of quality control charts on survival data and no R packages allowing for the continuous time inspection of survival outcomes.

The main contribution of this article is to present the R package `success` (SUrival Control Chart EStimation Software), a tool for constructing quality control charts on survival data. With this package, we aim to fill the gap in available open source software for the construction of control charts on survival data. The primary goal of `success` is to allow users to easily construct the BK- and CGR-CUSUM; moreover, `success` can also be used to construct the discrete time funnel plot (Spiegelhalter 2005) and the Bernoulli CUSUM chart (Steiner et al. 2000) on survival data without manually dichotomizing the

outcomes. This way, users can determine the possible gain in detection speed by using continuous time quality control methods over some popular discrete time methods.

The article is structured as follows. In Section [Theory and models](#) we briefly describe the theory that underlines the control charts presented in the package. Section [The R package success](#) shows how to prepare the input data and describes the main functions and their arguments. For the reader not interested in technical details, a helper function is described in Section [The parameter-assist function](#). The control charts in the package are then applied to data based on a clinical trial for breast cancer in the [Application](#) section. Finally, the article ends with a [Discussion](#) about the methods presented.

2 Theory and models

Throughout this article, we are interested in comparing institutional (hospital) performance for survival after a medical procedure (surgery). Even though our focus is on medical applications, the methods can be applied to any data set containing survival outcomes.

In this section we introduce the funnel plot, Bernoulli CUSUM, BK-CUSUM and CGR-CUSUM implemented in the package [success](#). The goal of each of the methods is to detect a deviation from a certain target performance measure and discover hospitals with increased mortality rates as quickly as possible.

Reading guide

This section consists of two parts. Section [Terminology](#) summarizes the minimal required knowledge in layman's terms. Section [Mathematical notation](#) delves into the mathematical details and assumptions. Further sections introduce the mathematical theory of the methods available in [success](#).

2.1 Terminology

We assume that hospitals/units have a constant and steady stream of patients/subjects coming in for a treatment of interest (e.g. surgery). In survival quality control, we are interested in determining whether failure/death rates after treatment at a certain hospital deviate significantly from a certain target measure. A target measure defines the acceptable failure rate. This measure can be set or estimated from a large set of (historical) data. Most of the time, we are only interested in detecting an increase in failure rate as this indicates that the hospital in question is performing worse than expected, and that corrective interventions may be necessary to improve the quality of care at such a hospital.

The process is defined to be *in-control* when failures occur according to the target level. It is *out-of-control* if failures happen at a higher rate than expected. Hospitals may have in-control periods followed by out-of-control periods. The goal is to detect when observations at a hospital start going out-of-control as soon as possible, so action can be taken quickly.

To continuously inspect the quality of the process, we construct a control chart to monitor the process failure rate from the start of the study. Some charts only change value when an outcome is observed (discrete time), while others change value at each time point (continuous time). When the control chart exceeds a pre-defined *control limit*, a signal is produced, indicating that the hospital in question is performing worse/better than the target measure. The time it takes for a control chart to exceed the control limit is called the *run length* of the chart. For some control charts it is necessary to fix the expected increase in failure rate in advance. This is done by specifying a parameter θ , where e^θ indicates the expected increase in the failure odds (discrete time) or expected increase in the failure rate (continuous time).

Not all patients have an equal probability of failure at any given point in time. For example, people who smoke may have a larger risk of failure than non-smokers. Therefore, patient's risk can be incorporated into the control charts by using a risk-adjustment model. Relevant characteristics (called covariates) are then used to determine the increase/decrease in the risk of failure for each patient.

2.2 Mathematical notation

Consider a single hospital. For each patient ($i = 1, 2, \dots$) let A_i and X_i be the chronological entry time and survival/failure time from the time of entry respectively. The chronological time of failure is then given by $T_i = A_i + X_i$. Assume that patients arrive (enter the study) at the hospital according to a Poisson process with rate ψ , and that each patient i has a set of p covariates Z_i . Let $h_i(x) = h_0(x)e^{Z_i\beta}$ be the subject-specific hazard rate obtained from the Cox proportional hazards model (Cox 1972). Let

$Y_i(t) = \mathbb{1}\{A_i \leq t \leq \min(T_i, R_i)\}$ indicate whether a patient is *at risk* (of failure) at time t , where R_i denotes the right censoring time of patient i .

Define by $N_i(t)$ the counting process indicating whether patient i experiences a failure at or before time t . Let $N(t) = \sum_{i \geq 1} N_i(t)$ be the total number of observed failures at the hospital at or before time t . Define the cumulative intensity of patient i as $\Lambda_i(t) = \int_0^t Y_i(u)h_i(u)du$ and $\lambda_i(t) = Y_i(t)h_i(t)$. Note that $\lambda_i(t)$ is equal to zero when patient i is not at risk. Similarly, let $\Lambda(t) = \sum_{i \geq 1} \Lambda_i(t)$ be the total cumulative intensity at the hospital at time t .

For some methods, we will be interested in detecting a fixed increase in the cumulative intensity at the hospital from $\Lambda(t)$ to $\Lambda^\theta(t) := e^\theta \Lambda(t)$. In a similar fashion, denote $h_i^\theta := e^\theta h_i(t)$. The corresponding density and distribution functions are denoted by f_i^θ and F_i^θ . We call e^θ the *true hazard ratio*. When $e^\theta = 1$ (similarly $\theta = 0$), we say that the failure rate is *in-control*. Alternatively, when $e^\theta > 1$ we say that the failure rate is *out-of-control*.

For a chart $K(t)$ with changing values over time define the *average run length* for a given control limit h as $\mathbb{E}[\tau_h]$, where $\tau_h = \inf\{t > 0 : K(t) \geq h\}$.

2.3 Funnel plot

The risk-adjusted funnel plot (Spiegelhalter 2005) is a graphical method used to compare performance between hospitals over a fixed period of time. The general structure of the data is as follows: there are k centers/hospitals ($j=1\dots k$) with n_j treated patients in hospital j . For each patient we observe a binary variable $X_{i,j}$:

$$X_{i,j} = \begin{cases} 1, & \text{if patient } i \text{ at hospital } j \text{ had an adverse event within } C \text{ days,} \\ 0, & \text{if patient } i \text{ at hospital } j \text{ otherwise.} \end{cases} \quad (1)$$

We model $X_{i,j} \sim \text{Ber}(p_j)$, with p_j the probability of failure at hospital j within C days. Consider the hypotheses:

$$H_0 : p_j = p_0 \quad H_1 : p_j \neq p_0 \quad (2)$$

with p_0 some baseline (in-control) failure proportion. The proportion of failures observed at hospital j is then given by $\gamma_j = \frac{\sum_{i=1}^{n_j} X_{i,j}}{n_j}$. The asymptotic distribution of γ_j under H_0 is $\gamma_j \Big|_{H_0} \sim \mathcal{N}\left(p_0, \frac{p_0(1-p_0)}{n_j}\right)$. We can then signal an increase or decrease in the failure proportion of hospital j with confidence level $1 - 2\alpha$ when

$$\gamma_j \notin \left[p_0 + \xi_\alpha \sqrt{\frac{p_0(1-p_0)}{n_j}}, p_0 - \xi_\alpha \sqrt{\frac{p_0(1-p_0)}{n_j}} \right], \quad (3)$$

with ξ_α the α -th quantile of the standard normal distribution.

It is often desirable to determine patient specific failure probabilities using some of their characteristics. A risk-adjusted funnel plot procedure can then be performed by modelling the patient specific failure probability using a logistic regression model: $p_i = \frac{1}{1+e^{-\beta_0+\mathbf{Z}_i\boldsymbol{\beta}}}$, where \mathbf{Z}_i is the vector of p covariates for patient i . The expected number of failures at hospital j is then given by

$$E_j = \mathbb{E} \left[\sum_{i=1}^{n_j} X_{j,i} \right] = \sum_{i=1}^{n_j} p_i = \sum_{i=1}^{n_j} \frac{1}{1+e^{-\beta_0+\mathbf{Z}_i\boldsymbol{\beta}}}. \quad (4)$$

Let O_j be the observed number of failures at hospital j , the risk-adjusted proportion of failures at hospital j is given by $\gamma_j^{\text{RA}} = \frac{O_j}{E_j} \cdot p_0$. The quantity γ_j^{RA} is then used in Equation (3) instead of γ_j .

The funnel plot can be used to compare hospital performance over a fixed time period. The funnel plot is often used for monitoring the quality of a process by repeatedly constructing funnel plots over different time intervals. We advocate against such an inspection scheme, as it introduces an increased probability of a type I error due to multiple testing. We recommend to only use the funnel plot as a graphical tool to visually inspect the proportion of failures at all hospitals over a time frame. The funnel plot is a discrete time method and can therefore only be used to compare overall performance over a time span. To determine whether a hospital was performing poorly during the time of interest, one of the following CUSUM charts should be used.

2.4 Bernoulli cumulative sum (CUSUM) chart

The Bernoulli CUSUM chart (Steiner et al. 2000) can be used to sequentially test whether the failure rate of patients at a single hospital has changed starting from some patient $v \geq 1$. Consider a hospital with patients $i = 1, \dots, v, \dots$ and a binary outcome:

$$X_i = \begin{cases} 1, & \text{if patient } i \text{ had an undesirable outcome within } C \text{ days,} \\ 0, & \text{if patient } i \text{ had a desirable outcome within } C \text{ days.} \end{cases} \quad (5)$$

We model $X_i \sim \text{Ber}(p_i)$ with p_i the failure probability within C days for patient i . The Bernoulli CUSUM can be used to test the hypotheses of an increased failure rate starting from some patient v :

$$H_0 : X_1, X_2, \dots \sim \text{Ber}(p_0) \quad H_1 : \begin{array}{l} X_1, \dots, X_{v-1} \sim \text{Ber}(p_0) \\ X_v, X_{v+1}, \dots \sim \text{Ber}(p_1) \end{array} , \quad (6)$$

where $v \geq 1$ is not known in advance, $p_0 < p_1$, and patient outcomes are ordered according to the time of entry into the study A_i .

The Bernoulli CUSUM statistic is given by:

$$S_n = \max(0, S_{n-1} + W_n), \quad (7)$$

with $W_n = X_n \ln\left(\frac{p_1(1-p_0)}{p_0(1-p_1)}\right) + \ln\left(\frac{1-p_1}{1-p_0}\right)$. Alternatively, it is possible to reformulate the chart in terms of the Odds Ratio $OR = \frac{p_1(1-p_0)}{p_0(1-p_1)} =: e^\theta$. In that case, $W_n = X_n \ln(e^\theta) + \ln\left(\frac{1}{1-p_0+e^\theta p_0}\right)$. The null hypothesis is rejected when the value of the chart exceeds a control limit h .

A risk-adjusted procedure may be performed by modelling patient-specific failure probability ($p_{0,i}$) using a logistic regression model. The risk-adjusted Bernoulli CUSUM can be used as a sequential quality control method for binary outcomes. Dichotomizing the outcome (survival time) can lead to delays in detection. When survival outcomes are available, it can therefore be beneficial to construct one of the CUSUM charts described in the following sections.

2.5 Biswas and Kalbfleisch CUSUM (BK-CUSUM)

The BK-CUSUM chart can be used to continuously test whether the failure rate of patients at the hospital has changed at some point in time. Consider a hospital with patients $i = 1, 2, \dots$ and assume that the patient specific hazard rate is given by $h_i(x) = h_0(x)e^{\mathbf{Z}_i\beta}$. The BK-CUSUM chart can be used to test the hypotheses that the baseline hazard rate of all active patients has increased from $h_0(x)$ to $h_0(x)e^{\theta_1}$ at some point in time $s > 0$ after the start of the study:

$$H_0 : X_i \sim \Lambda_i(t), i = 1, 2, \dots \quad H_1 : \begin{array}{l} X_i \sim \Lambda_i(t) \mid t < s, i = 1, 2, \dots \\ X_i \sim \Lambda_i^{\theta_1}(t) \mid t \geq s, i = 1, 2, \dots \end{array} , \quad (8)$$

where $\theta_1 > 0$ is the user's estimate of the true hazard ratio θ and $s > 0$ is the unknown time of change in hazard rate.

The likelihood ratio chart associated with the hypotheses in (8) is given by:

$$BK(t) = \max_{0 \leq s \leq t} \left\{ \theta_1 N(s, t) - (e^{\theta_1} - 1) \Lambda(s, t) \right\}, \quad (9)$$

where the estimated hazard ratio $e^{\theta_1} > 1$ has to be prespecified, $N(s, t) = N(t) - N(s)$ and $\Lambda(s, t) = \Lambda(t) - \Lambda(s)$. The null hypothesis is rejected when the value of the chart exceeds the control limit.

The BK-CUSUM chart can lead to faster detection speeds than the Bernoulli CUSUM chart as the hypotheses can be tested at any point in time, rather than just at the (dichotomized) times of outcome. Unfortunately the chart requires users to specify θ_1 to estimate θ , which is not known a priori in most practical applications. Misspecifying this parameter can lead to large delays in detection (Gomon et al. 2022).

2.6 Continuous time Generalized Rapid response CUSUM (CGR-CUSUM)

The CGR-CUSUM chart can be used to test the following hypotheses:

$$H_0 : X_i \sim \Lambda_i, i = 1, 2, \dots \quad H_1 : \begin{array}{l} X_i \sim \Lambda_i, i = 1, 2, \dots, v-1 \\ X_i \sim \Lambda_i^\theta, i = v, v+1, \dots \end{array} , \quad (10)$$

where e^θ and ν do not need to be prespecified. The CGR-CUSUM chart is then given by

$$CGR(t) = \max_{1 \leq \nu \leq n} \{ \hat{\theta}_{\geq \nu}(t) N_{\geq \nu}(t) - (\exp(\hat{\theta}_{\geq \nu}(t)) - 1) \Lambda_{\geq \nu}(t) \}, \quad (11)$$

where the subscript “ $\geq \nu$ ” stands for all subjects after the potential change point ν : $N_{\geq \nu}(t) = \sum_{i \geq \nu} N_i(t)$, $\Lambda_{\geq \nu}(t) = \sum_{i \geq \nu} \Lambda_i(t)$ and $\hat{\theta}_{\geq \nu}(t) = \max(0, \log(\frac{N_{\geq \nu}(t)}{\Lambda_{\geq \nu}(t)}))$. The null hypothesis is rejected when the value of the chart exceeds the control limit.

In contrast to the BK-CUSUM where an estimate of e^θ had to be specified in advance, the CGR-CUSUM uses the maximum likelihood estimate $\hat{\theta}$ to estimate the true hazard ratio e^θ from the data. This means that when e^{θ_1} is misspecified in the BK-CUSUM, the CGR-CUSUM can lead to quicker detections. In practice the real hazard ratio e^θ is never known in advance and may vary over time. The maximum likelihood estimator can alleviate this problem, therefore the CGR-CUSUM is generally the preferred chart. The major difference between the two charts is that the BK-CUSUM tests for a sudden change in the failure rate of all patients currently at risk of failure, while the CGR-CUSUM tests for a sudden change in the failure rate of all patients currently at risk of failure who have entered the hospital after a certain time. A drawback of the CGR-CUSUM is that the computation of the MLE $\hat{\theta}$ requires sufficient information (in the form of survival times/failures) to converge to the true value. This means that the chart can be unstable at the beginning of the study and might not provide reliable values for hospitals with low volumes of patients.

2.7 Choosing control limits

For the funnel plot in Section [Funnel plot](#), it is sufficient to choose a confidence level to determine which hospitals are performing worse/better than the baseline. For the CUSUM charts, instead, it is necessary to choose a control limit h , so that a signal is produced when the value of the chart exceeds h . The most common ways to choose this control limit are to either restrict the in-control average run length (ARL) of the chart, or to restrict the type I error over a certain time period. With the first method, one could choose to restrict the in-control ARL to approximately 5 years, so that on average we would expect a hospital which performs according to the target to produce a false signal (detection) once every 5 years. Using the second method, one could choose the control limit such that at most a proportion α of the in-control hospitals yields a signal (false detection) within a period of 5 years. For the (risk-adjusted) Bernoulli CUSUM plenty of results exist allowing for the numerical estimation of the ARL, most of which are implemented in the packages [spc](#) and [vlad](#). For continuous time control charts such results are lacking, therefore in the [success](#) package we choose to determine control limits by restricting the type I error probability α over a chosen time frame. We estimate these control limits by means of a simulation procedure.

3 The R package success

The package [success](#) can be used by both laymen and experts in the field of quality control charts. In Section [Input data](#) we describe the general data structure to be used for constructing control charts by means of an example data set. In Section [The parameter-assist function](#), a function is described that can be used to determine all necessary parameters for the construction of control charts for the user not interested in technical details. In Sections [Manual risk-adjustment - The CGR-CUSUM function](#) we present the functions that can be used to compute the control charts described in Section [Theory and models](#).

3.1 Input data

All methods in this package require the user to supply a `data.frame` for the construction of control charts and the estimation of a benchmark failure rate. We show how to use the [success](#) package by means of an enclosed data set.

Example data set

The data frame `surgerydat` contains 32529 survival times, censoring times and covariates (age, BMI and sex) of patients from 45 hospitals with 15 small, medium and large hospitals (0.5, 1 and 1.5 patients per day). Patients enter the hospitals for 2 years (730 days) after the start of the study. Survival times were generated using a risk-adjusted Cox proportional hazards model using inverse transform sampling (Austin 2012), with coefficient vector $\beta = c(\text{age} = 0.003, \text{BMI} = 0.02, \text{sexmale} = 0.2)$

Table 1: Overview of the enclosed ‘surgerydat’ data set.

entrytime	survtime	censorid	unit	exptheta	psival	age	sex	BMI
5	21	0	1	1.887204	0.5	70	male	29.52
9	19	1	1	1.887204	0.5	68	male	24.06
10	64	1	1	1.887204	0.5	101	female	20.72
20	64	1	1	1.887204	0.5	67	female	24.72
21	0	1	1	1.887204	0.5	44	male	27.15

and exponential baseline hazard rate $h_0(t, \lambda = 0.01)e^\theta$. Hospitals are numbered from 1 to 45 with hazard ratio θ sampled from a normal distribution with mean 0 and standard deviation 0.4. This means that some hospitals are performing better or equal to baseline ($\theta \leq 0$) and some are performing worse ($\theta > 0$).

An overview of the data set can be found in Table 1.

```
library(success)
data("surgerydat", package = "success")
head(surgerydat, 5)
```

Each row represents a patient. The first 2 columns (entrytime and survtime) are crucial for the construction of control charts. These columns have to be present in the data. The time scale of entrytime and survtime must be the same. They can both only be supplied in a “numeric” format (dates are not allowed). In the example data, the time scale is in days; entrytime is the number of days since the start of the study and survtime is the survival time since the time of entry. The column censorid is a censoring indicator, with 0 indicating right-censoring and 1 that the event has occurred. If censorid is missing, a column of 1’s will automatically be created, assuming that no observations were right-censored. All relevant functions in the package will print a warning when this happens. The column unit (indicating the hospital number) is required for the construction of a funnel plot, but not for the CUSUM charts. This is because CUSUM charts are constructed separately for every unit, requiring the user to manually subset the data for each unit. The columns exptheta and psival indicate the parameters e^θ and ψ used to create the simulated data set. The last three columns age, sex and BMI are the covariates of each individual. In a user supplied data.frame these can of course take any desired name.

3.2 The parameter-assist function

Readers not interested in technical details can use the parameter_assist function to determine most of the parameters required for constructing the control charts in this package.

The parameter_assist function can be used to determine control limits for all control charts described in Section [Theory and models](#). The function guides users through the following 3 steps:

- Step 1: Specify arguments to parameter_assist.
- Step 2: Determine control limit(s) for the required control chart(s) by feeding the output of Step 1 to one of the *_control_limit functions.
- Step 3: Construct the desired control chart by feeding the output of Step 1 to the function, as well as the control limit from Step 2.

Step 2 can be skipped for the funnel_plot function, as funnel plots do not require control limits.

The parameter_assist function has the following arguments:

- **baseline_data (required):** a data.frame in the format described in Section [Input data](#). It should contain the data to be used to determine the target performance metric (both for discrete and continuous time charts). This data is used to determine what the “acceptable” failure rate is, as well as how much risk-adjustment variables influence the probability of observing an event. Preferably this should be data of subjects that were known to fail at acceptable rates, or historical data which we want to compare with. Usually all available data is used instead, resulting in the average performance being selected as the target;
- **data (required):** a data.frame in the format described in Section [Input data](#). It should contain the data used to construct a control chart. For this data we want to know whether it adheres to the target determined in baseline_data or not. For example: the data from one hospital;
- **formula (optional):** a formula indicating in which way the linear predictor of the risk-adjustment models should be constructed in the underlying generalized linear model or Cox proportional

hazards model. Only the right-hand side of the formula will be used. If left empty, no risk-adjustment procedure will be performed;

- **followup (required for discrete time charts):** a numeric value which has to be in the same unit as entrytime and survtime and should specify how long after entrytime we consider the binary outcome (failure/non-failure) of the patient. This argument can be left empty when the user does not want to construct discrete time charts;
- **theta (recommended for Bernoulli and BK-CUSUM):** the expected increase in the log-odds of failure/hazard rate. Default is $\log(2)$, meaning the goal will be a detection of a doubling in the failure rate;
- **time (recommended):** time interval for type I error to be determined. Default is the largest entrytime of subjects in baseline_data;
- **alpha (recommended):** required type I error to control over the time frame specified in time. Default is 0.05.

The first two arguments should always be specified. Depending on the number of additional arguments specified, different functions in the package can be used. Most arguments have default values, but these may not always be suitable for the desired inspection scheme. Risk-adjusted procedures can only be constructed if at least formula is specified.

An example where all arguments are specified is provided below, but specifying only baseline_data and data is sufficient to construct a CGR-CUSUM without risk-adjustment.

```
assisted_parameters <- parameter_assist(
  baseline_data = subset(surgerydat, entrytime < 365),
  data = subset(surgerydat, entrytime >= 365 & unit == 1),
  formula = ~age + sex + BMI,
  followup = 30,
  theta = log(2),
  time = 365,
  alpha = 0.05)
```

We use data on patients arriving in the first year (entrytime < 365) to determine the target performance measure. We then construct control charts on the first hospital (unit == 1) using information on all patients arriving after the first year. Risk-adjustment is performed using the 3 available covariates. We choose to consider patient followup 30 days after surgery and want to detect a doubling of failure rate. The last two arguments were chosen such that the type I error of the procedure is restricted to 0.05 within 1 year (on average 1 in 20 hospitals performing according to baseline will be detected within 1 year).

The parameter_assist function then returns a list of arguments to supply to other functions in this package:

```
names(assisted_parameters)

#> [1] "call"          "data"          "baseline_data" "glmmmod"
#> [5] "coxphmod"     "theta"         "psi"           "time"
#> [9] "alpha"         "maxtheta"     "followup"      "p0"
```

The user can manually feed the determined parameters to other functions in this package. Conversely, it is possible to feed the output of the parameter_assist function to the following functions directly:

- funnel_plot
- bernoulli_control_limit and bernoulli_cusum
- bk_control_limit and bk_cusum
- cgr_control_limit and cgr_cusum

Step 1 was performed in the code above. For Step 2 we feed the output of parameter_assist to determine control limits for the Bernoulli, BK- and CGR-CUSUM charts.

```
bernoulli_control <- bernoulli_control_limit(assist = assisted_parameters)
bk_control <- bk_control_limit(assist = assisted_parameters)
cgr_control <- cgr_control_limit(assist = assisted_parameters)
```

The determined control limits can then be fed to the control chart functions to finish Step 3.

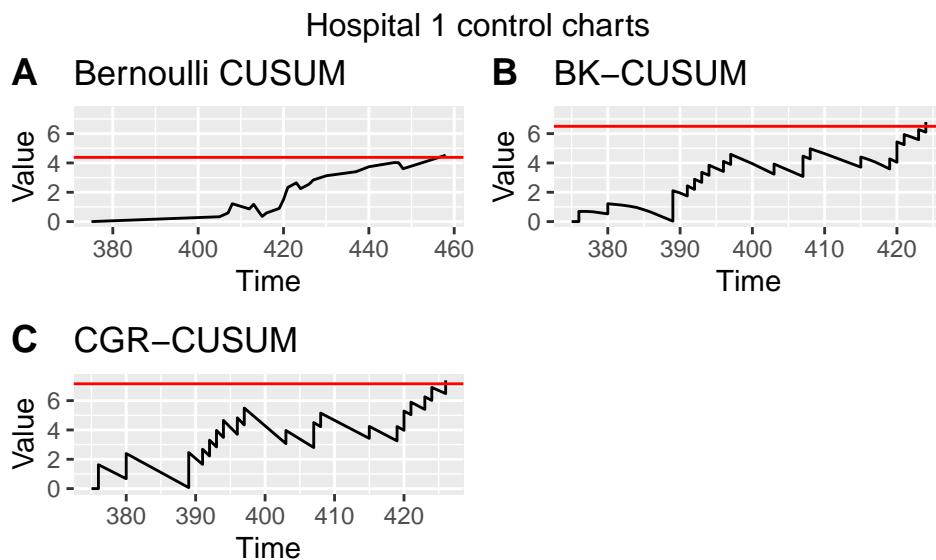


Figure 1: Bernoulli, BK- and CGR-CUSUM charts for hospital 1 in the surgery data set starting from 1 year after the start of the study.

```
ber�oulli_assist <- ber�oulli_cusum(assist = assisted_parameters,
                                         h = ber�oulli_control$h)
bk_assist <- bk_cusum(assist = assisted_parameters, h = bk_control$h)
cgr_assist <- cgr_cusum(assist = assisted_parameters, h = cgr_control$h)
```

We plot the control charts using the `plot` function (see Figure 1). The Bernoulli CUSUM jumps upward every time a failure is observed 30 days after patient entry and downward every time no failure is observed at that point. The BK- and CGR-CUSUM make an upward jump directly when a failure has been observed, and slope downward as long as no failures are happening. The BK- and CGR-CUSUM cross their respective control limits (the red lines) at approximately the same time after the start of the study, producing a signal. The Bernoulli CUSUM does so a little while later, producing a delayed signal.

The run length of the charts (time until control limit is reached) can then be determined using the `runlength` function.

```
runlength(ber�oulli_assist, h = ber�oulli_control$h)

#> [1] 83
```

Note that the run length of the charts are determined from the smallest time of entry of subjects into specified data. The study therefore starts at the moment the first subject has a surgery (in this case, at day 375).

The funnel plot does not require control limits, therefore steps 2 and 3 can be skipped. We use the `plot` function to display the funnel plot.

```
funnel_assist <- funnel_plot(assist = assisted_parameters)
plot(funnel_assist, label_size = 2) + ggtitle("Funnel plot of surgerydat")
```

The resulting plot can be seen in Figure 2. The blue shaded regions indicate the 95 and 99 percent prediction intervals. Each dot represents a hospital, with the colour representing the prediction limits at which this hospital would be signaled. When a hospital falls outside of a prediction interval, it will be signaled at that level.

3.3 Manual risk-adjustment

Risk-adjustment models should be estimated on a data set known to have in-control failures, as this allows the coefficients to be determined as precisely as possible. In real life applications it is not known in advance which hospitals have had in-control failures. It is therefore common practice to use all available data to determine risk-adjustment models.

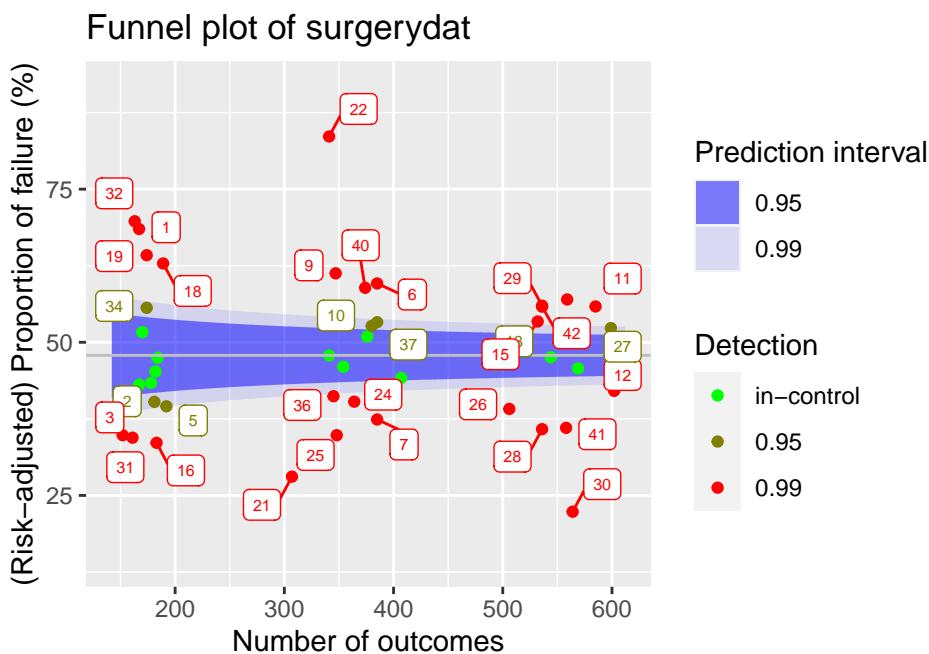


Figure 2: Funnel plot for hospitals over the first year of the surgery data set. Target performance determined as average over the considered data.

We consider a logistic model to use for risk-adjustment in the discrete time methods (funnel plot and Bernoulli CUSUM), using all available data of patients with surgeries in the first year of the study. We use 30 days mortality as outcome for these charts.

```
baseline_data <- subset(surgerydat, entrytime <= 365)
followup <- 30
glm_risk_model <- glm(survtime <= followup) & (censorid == 1) ~ age + sex + BMI,
                      data = baseline_data, family = binomial)
```

Then we estimate a Cox proportional hazards model to use for risk-adjustment in the continuous time BK- and CGR-CUSUM, using the same baseline data. For this we use the functions `Surv` and `coxph` from the package `survival` (Terry M. Therneau and Patricia M. Grambsch 2000).

```
require(survival)
coxph_risk_model <- coxph(Surv(survtime, censorid) ~ age + sex + BMI,
                           data = baseline_data)
```

Conversely, we can manually specify a risk-adjustment model:

```
RA_manual <- list(formula = ~ age + sex + BMI,
                    coefficients = c(age = 0.003, BMI = 0.02,
                                     sexmale = 0.2))
```

This is useful for users who do not want to use the package `survival` for the estimation of the models.

3.4 The funnel plot function

The `funnel_plot` function can be used to construct the funnel plot described in Section [Funnel plot](#). The code below constructs a funnel plot over the first 1 year (`ctime = 365`) on the simulated data set. By not specifying `ctime` the funnel plot is constructed over all data (2 years). By leaving the parameter `p0` empty, the average failure proportion within 30 days is used as baseline failure probability.

```
funnel <- funnel_plot(data = surgerydat, ctime = 365,
                       glmmmod = glm_risk_model, followup = 30)
```

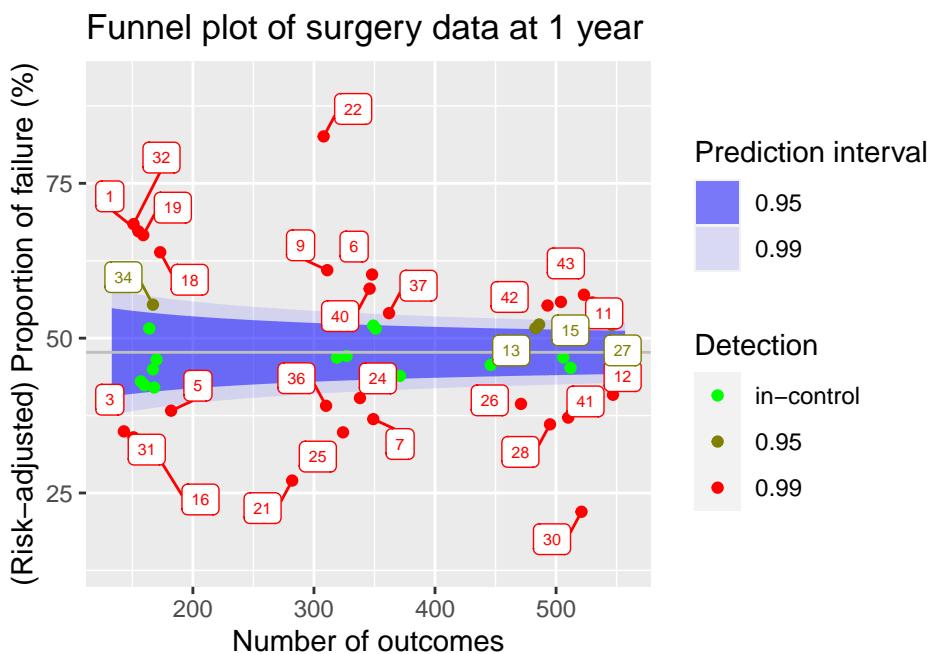


Figure 3: Funnel plot of hospitals in the surgery data, using all information known after 1 year of the study.

Table 2: Summary statistics for the funnel plot.

unit	observed	expected	numtotal	p	0.95	0.99
1	105	74.49454	155	0.6724872	worse	worse
2	71	80.60038	168	0.4202816	in-control	in-control
3	50	68.25903	143	0.3494854	better	better
4	76	80.64515	167	0.4496292	in-control	in-control
5	70	87.22662	182	0.3828848	better	better

The function creates an object of class 'funnelplot'. The plot function can be used on classes in the `success` package. This creates a 'gg' object, whose graphical parameters can further be edited by the user using the package `ggplot2` (Wickham 2016).

The resulting plot can be seen in Figure 3. By default, the `funnel_plot` function will display 95% and 99% prediction intervals using blue shaded regions. The intervals and fill colour can be changed using the `predlim` and `col_fill` arguments respectively. Additionally, the unit label for detected units will be displayed in the plot. In the `plot` function, the size of the labels can be adjusted using the `label_size` argument or they can be disabled by setting the `unit_label` argument to FALSE.

A summary of the funnel plot can be obtained by using the `summary` function.

```
head(summary(funnel), 5)
```

The resulting statistics can be found in Table 2.

3.5 Control limits for CUSUM functions

All CUSUM charts considered in this article require control limits to signal changes in the failure rate. When the value of a chart exceeds this control limit, a signal is produced. The `success` package can be used to determine control limits such that the type I error of the CUSUM procedure is restricted over some desired time frame. This is achieved by using the `*_control_limit` functions. We briefly discuss the underlying three steps of the simulation procedure, meanwhile explaining some key parameters that are shared across the functions.

- Step 1: Generate `n_sim` units (hospitals), with subjects at each unit arriving according to a Poisson process with rate `psi` over a certain time frame specified by the parameter `time`. We therefore expect each unit to represent approximately $\psi \times \text{time}$ subjects. If specified, sample covariate values for subjects from `baseline_data`. Generate binary outcomes/survival times for

each patient according to the specified baseline (risk-adjustment) model (for details see [Manual risk-adjustment](#)). Note that the simulated units represent the in-control situation.

- Step 2: For each simulated unit, determine the CUSUM chart over the considered time.
- Step 3: Determine the largest possible control limit such that at most a proportion alpha of the n_sim in-control units would be detected using this control limit. This value then represents a simulation estimate of the control limit for a procedure with a type I error of alpha over time.

Increasing n_sim will increase the accuracy of the determined control limit, at the cost of increased computation time. Similarly, increasing time and psi will also increase the computation time. These parameters however are usually determined by the underlying inspection problem. Changing alpha does not influence computation time.

All *_control_limit functions have a seed argument that can be used to obtain reproducible simulation results by setting an initial state for pseudorandom number generation. This makes sure that the procedure in Step 1 is reproducible, as the final two steps do not involve randomness. The boolean parameter pb can be used to display a progress bar for Step 2, which can be useful if the control limit has to be determined at a high accuracy. The h_precision argument can be used to specify the required number of significant digits in determining the control limit. Choosing a high value for h_precision is only useful when the constructed CUSUM charts show only very minor fluctuations or when n_sim is very high, warranting a very accurate determination of the control limit. The default of two significant digits will suffice in most situations.

3.6 The Bernoulli CUSUM function

The bernoulli_cusum function can be used to construct the Bernoulli CUSUM detailed in Section [Bernoulli cumulative sum \(CUSUM\) chart](#). The Bernoulli CUSUM uses the same dichotomized outcome as the funnel plot. For this reason, the syntax of bernoulli_cusum is quite similar to that of funnel_plot. In this section we will construct a Bernoulli CUSUM for the ninth hospital in the simulated data set, again using 30 day post operative survival as outcome and aiming to detect an increase of the odds ratio to 2.

Determining control limits

The Bernoulli CUSUM produces a signal when the value of the chart exceeds a value h called the control limit. The bernoulli_control_limit function can be used to determine a control limit such that the type I error of the Bernoulli CUSUM procedure is restricted over some desired time frame. Suppose we want to restrict the type I error of the procedure to 0.05 over the time frame of 1 year at a hospital with an average of 1 patient per day undergoing surgery. We determine the control limit as follows:

```
bern_control <- bernoulli_control_limit(
  time = 365, alpha = 0.05, followup = 30, psi = 1,
  glmmmod = glm_risk_model, baseline_data = surgerydat, theta = log(2))
```

The determined control limit h can then be retrieved by:

```
bern_control$h
#> [1] 5.56
```

By default, the control limit is determined using 200 simulated units (hospitals). As the Bernoulli CUSUM is not very computationally intensive, it is usually possible to determine the control limit with higher precision by increasing the n_sim argument.

Constructing the chart

After determining the control limit, we can construct the control chart. The bernoulli_cusum function requires the user to specify one of the following combinations of parameters:

- glmmmod & theta
- p0 & theta
- p0 & p1

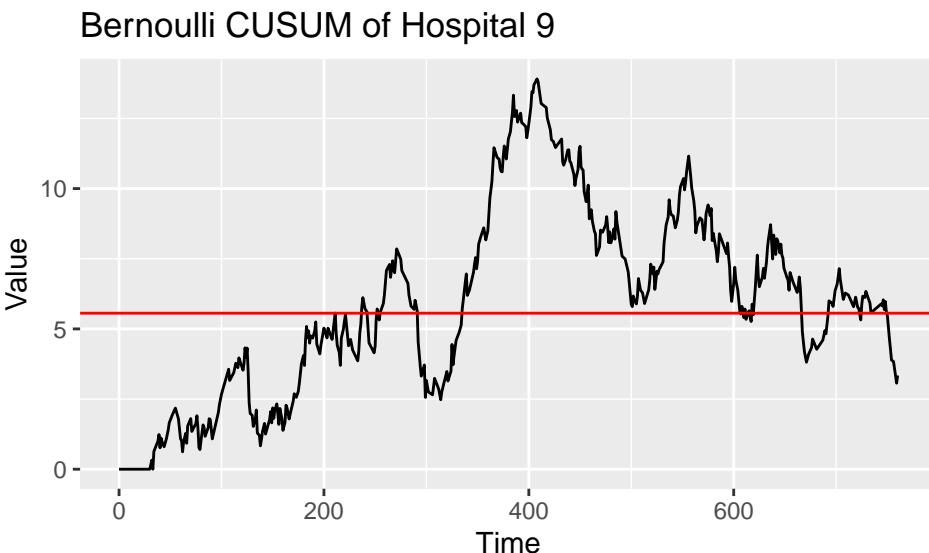


Figure 4: Bernoulli CUSUM for hospital 9 over the study duration. Target performance estimated from the failure rate of patients treated in the first year of the study.

Only the first option allows for risk-adjustment. The difference between these parametrizations is described in Section [Bernoulli cumulative sum \(CUSUM\) chart](#). We construct the Bernoulli CUSUM on the data of the ninth simulated hospital, aiming to detect whether the odds ratio of failure for patients is 2 ($\theta = \log(2)$). Using the plot function we obtain a 'gg' object (see Figure 4).

```
Bernoulli <- bernoulli_cusum(
  data = subset(surgerydat, unit == 9),
  glmmmod = glm_risk_model, followup = 30, theta = log(2))
plot(Bernoulli, h = bern_control$h) +
  ggtitle("Bernoulli CUSUM of Hospital 9")
```

The run length of the chart, in Figure 4 visible as the time at which the chart first crosses the red line, can be found using the runlength function.

3.7 The BK-CUSUM function

The `bk_cusum` function can be used to construct the BK-CUSUM chart presented in Section [Biswas and Kalbfleisch CUSUM \(BK-CUSUM\)](#). The chart is no longer constructed using dichotomized outcomes, therefore leading to faster detections on survival data than discrete time methods.

Determining control limits

The BK-CUSUM produces a signal when the value of the chart exceeds a value h called the control limit. The `bk_control_limit` function can be used to determine a control limit such that the type I error of the BK-CUSUM procedure is restricted over some desired time frame. Suppose we want to restrict the type I error of the procedure to 0.05 over the time frame of 1 year for a hospital with an average of 1 patient per day undergoing surgery. The control limit is determined as follows:

```
BK_control <- bk_control_limit(
  time = 365, alpha = 0.05, psi = 1, coxphmod = coxph_risk_model,
  baseline_data = surgerydat, theta = log(2))
```

By default, the control limit is determined on a simulated sample of 200 in-control hospitals. The BK-CUSUM is not very computationally expensive. It is therefore usually possible to determine the control limit with higher precision by increasing the `n_sim` argument.

Constructing the chart

We construct the BK-CUSUM on the data of the ninth hospital aiming to detect a doubling in the hazard rate of patients ($\theta = \log(2)$).

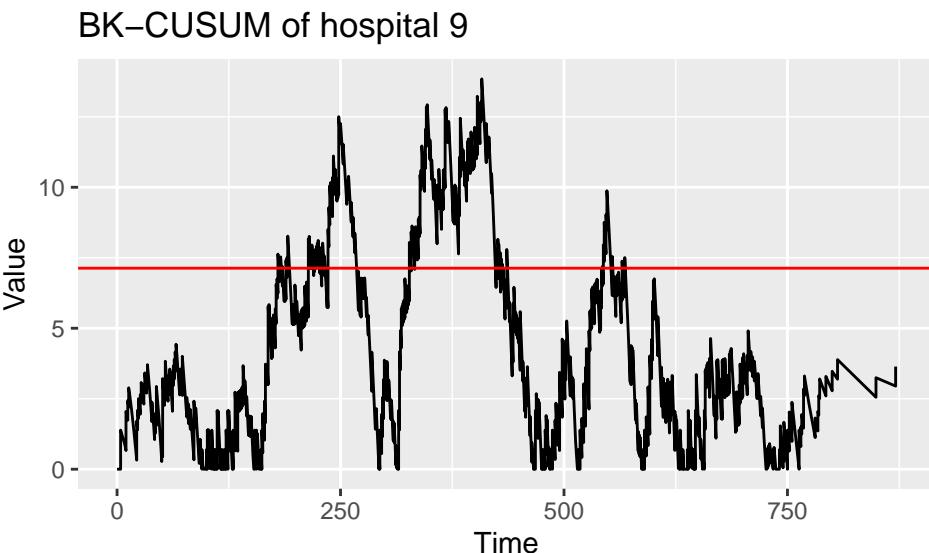


Figure 5: BK-CUSUM for hospital 9 over the study duration. Target performance estimated from the failure rate of patients treated in the first year of the study.

```
BK <- bk_cusum(data = subset(surgerydat, unit == 9), theta = log(2),
                 coxphmod = coxph_risk_model)
plot(BK, h = BK_control$h) + ggtitle("BK-CUSUM of hospital 9")
```

The resulting plot is presented in Figure 5. The run length of the chart, visible as the time at which the chart first crosses the red line, can be found using the runlength function.

When the cumulative baseline hazard is not specified through the argument cbaseline, but a Cox Risk-adjustment model coxphmod as obtained from coxph is provided, the cumulative baseline hazard will automatically be determined from this Cox model. When the argument ctimes is left empty, the chart will only be determined at the times of patient failures, as this is sufficient for detection purposes and saves computation time. A control limit h can be specified, so that the chart is only constructed until the value of the chart exceeds the value of the control limit. This is very convenient when monitoring the quality of care at a hospital. Sometimes it is desirable to only construct the chart up until a certain time point, for this the argument stoptime can be used. The argument C can be used to only consider patient outcomes up until C time units after their surgery. Originally the BK-CUSUM (Biswas and Kalbfleisch 2008) was proposed with $C = 365$, considering patient outcomes only until 1 year post surgery. Finally, a progress bar can be added using the argument pb.

Suppose a decrease in the failure rate is of interest, we can then construct a lower-sided BK-CUSUM by specifying a theta value smaller than 0. For example, for detecting a halving of the hazard rate we can take $\theta = -\log(2)$, such that $e^\theta = \frac{1}{2}$.

```
BK_control_lower <- bk_control_limit(
  time = 365, alpha = 0.05, psi = 1, coxphmod = coxph_risk_model,
  baseline_data = surgerydat, theta = -log(2))

BK_control_lower <- bk_control_limit(
  time = 365, alpha = 0.05, psi = 1, coxphmod = coxph_risk_model,
  baseline_data = surgerydat, theta = -log(2))

BKlower <- bk_cusum(data = subset(surgerydat, unit == 9),
                      theta = -log(2), coxphmod = coxph_risk_model)
plot(BKlower, h = BK_control_lower$h) +
  ggtitle("BK-CUSUM of hospital 9 (lower sided)")
```

The resulting plot can be found in Figure 6 A.

Similarly, when both an increase and decrease of the failure rate are of interest the argument twosided = TRUE can be used. This produces a two-sided BK-CUSUM (see Figure 6 B). For the lower-sided BK-CUSUM the control limit must be determined separately.

```
BKtwosided <- bk_cusum(data = subset(surgerydat, unit == 9),
                        theta = log(2), coxphmod = coxph_risk_model, twosided = TRUE)
plot(BKtwosided, h = c(BK_control_lower$h, BK_control$h))
```

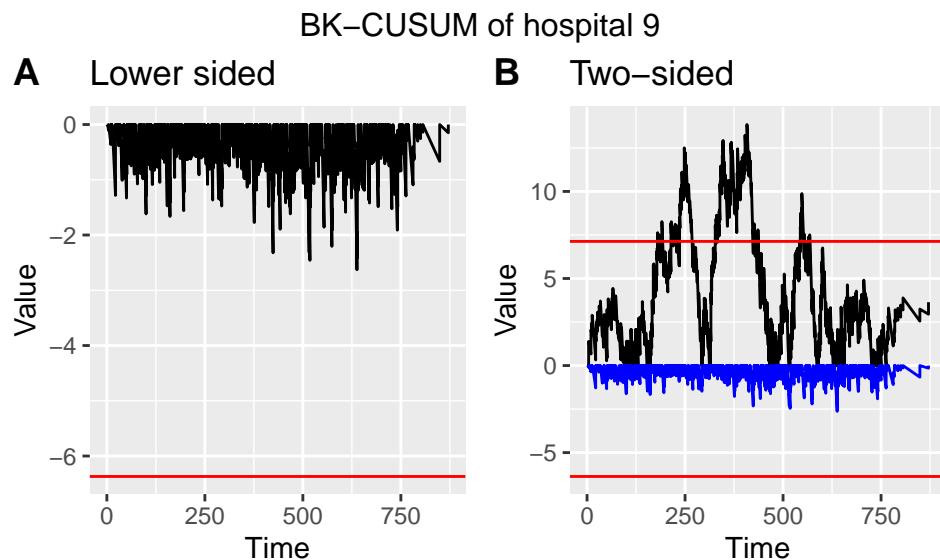


Figure 6: Lower sided (left) and two-sided (right) Bernoulli CUSUM for hospital 9 over the study duration. Target performance estimated from the failure rate of patients treated in the first year of the study.

3.8 The CGR-CUSUM function

The `cgr_cusum` function can be used to construct the CGR-CUSUM detailed in Section [Continuous time Generalized Rapid response CUSUM \(CGR-CUSUM\)](#). This function has almost the same syntax as the `bk_cusum` function. The difference is that the procedure estimates a suitable value for theta through maximum likelihood estimation, instead of requiring the users to specify such a value a priori.

The maximum likelihood estimate in the CGR-CUSUM can be unstable at early time points, when not much information is available about subject failure. For this reason, the value of the maximum likelihood estimate is restricted to $e^{\hat{\theta}} \leq 6$ by default. This comes down to believing that the true hazard ratio at any hospital is always smaller than or equal to 6 times the baseline. To change this belief, the user can supply the `maxtheta` parameter to the `cgr_cusum` and `cgr_control_limit` functions.

Determining control limits

Similarly to the `bk_control_limit` function used for the BK-CUSUM, the `cgr_control_limit` function can be used to determine the control limit for the CGR-CUSUM chart as follows:

```
CGR_control <- cgr_control_limit(
  time = 365, alpha = 0.05, psi = 1, coxphmod = coxph_risk_model,
  baseline_data = surgerydat)
```

By default the control limit for the CGR-CUSUM is determined on only 20 simulated samples (due to the computational intensity of the procedure), but we recommend to increase the number of samples by using the argument `n_sim` to get a more accurate control limit. This will greatly increase the computation time. To speed up the procedure it is possible to parallelize the computations of the CUSUM charts in Step 2 of the simulation procedure (see [Control limits for CUSUM functions](#)) by specifying the number of cores to use to the argument `ncores`. Additionally, as the simulation procedure for the CGR-CUSUM can take a lot of time, it is possible to display individual progress bars for each constructed CUSUM chart by specifying `chartpb = TRUE`.

Constructing the chart

The CGR-CUSUM for the ninth hospital is created with:

```
CGR <- cgr_cusum(data = subset(surgerydat, unit == 9),
  coxphmod = coxph_risk_model)
plot(CGR, h = CGR_control$h) + ggtitle("CGR-CUSUM of hospital 9")
```

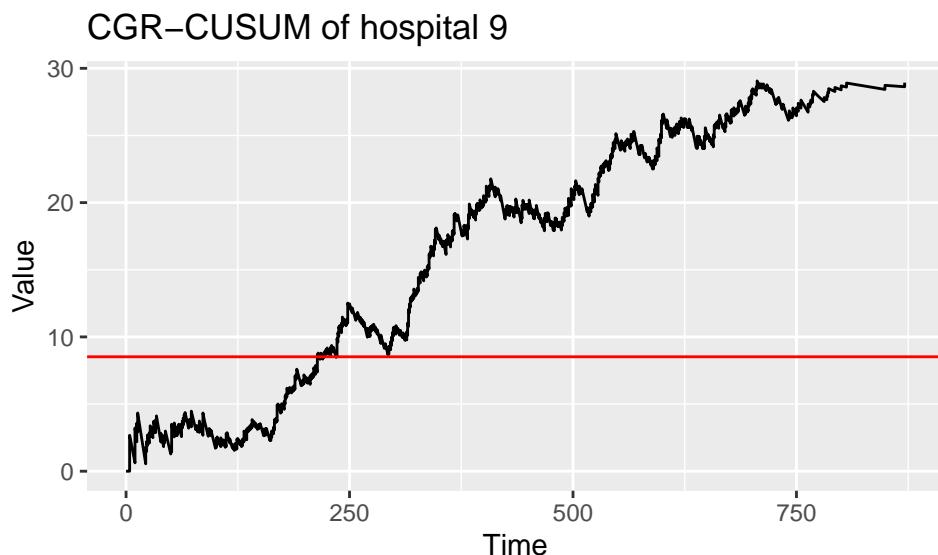


Figure 7: CGR-CUSUM for hospital 9 over the study duration. Target performance estimated from the failure rate of patients treated in the first year of the study.

The resulting plot can be seen in Figure 7. To determine the run length of the procedure (time at which chart crosses the control limit), use the function `runlength`.

To construct the control chart up until the time of detection, the parameter `h` can be specified in the `cgr_cusum` function. This allows for continuous inspection, as well as reducing computation time.

Since the CGR-CUSUM is time consuming when the value has to be computed at many time points, we recommend to leave `ctimes` unspecified, so that the CGR-CUSUM will only be determined at the times necessary for detection purposes.

To reduce computing time, we allow users to parallelize the computations across multiple cores. This can be easily done through the `ncores` argument. The calculation of the CGR-CUSUM proceeds through 2 steps. First, the contributions to the cumulative intensity of each subject are determined at every time point of interest and are stored in a matrix. Afterwards, the value of the chart is computed by performing matrix operations. When `ncores > 1`, both steps are automatically parallelized using functions from the `pbapply` package (Solymos and Zawadzki 2021). When a value for the control limit has been specified, only the first step can be parallelized. For small data sets and/or short runs `cmethod = "CPU"` can be chosen, thereby recalculating the value of the chart at every desired time point but not requiring a lot of initialization. For small hospitals and/or short detection times it could be the preferred method of construction. As the CGR-CUSUM can take long to construct, it is recommended to display a progress bar by specifying `pb = TRUE`.

```
CGR_multicore <- cgr_cusum(data = subset(surgerydat, unit == 9),
                           coxphmod = coxph_risk_model, ncores = 3, pb = TRUE)
```

3.9 The interactive plot function

The `interactive_plot` function can be used to plot multiple CUSUM charts together in one figure, while allowing the user to interact with the plot. This is achieved by using the package `plotly` (Sievert 2020). We show how to use these features by plotting some of the CUSUM charts from the previous Sections together in one figure. We first combine all CUSUM charts into a list, together with the control limits.

```
Bernoulli$h <- bernoulli_control$h
BK$h <- BK_control$h
CGR$h <- CGR_control$h
cusum_list <- list(Bernoulli, BK, CGR)
interactive_plot(cusum_list, unit_names = rep("Hosp 9", 3), scale = TRUE)
```

As charts have different control limits, it is preferable to scale their values by their respective control limits by specifying `scale = TRUE`. After scaling, the control limit will be $h = 1$ for all CUSUM charts. The resulting plot can be seen in Figure 8. By choosing `highlight = TRUE`, the user can

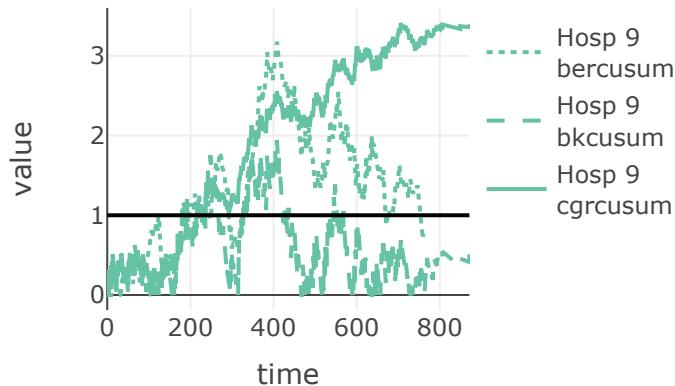


Figure 8: Interactive plot of CUSUM charts for hospital 9.

highlight CUSUM charts by hovering over them. The `plotly` package allows for many interactive capabilities with the plot.

4 Application

In Section [The R package success](#) we employed a simulated data set to show how to use the `success` package. In this Section, we illustrate the use of the `success` package on a data set based on a clinical trial for breast cancer conducted by the European Organisation for Research and Treatment of Cancer (EORTC). Covariates for 2663 patients over 15 treatment centres are available, with patients having surgery over a span of 61 time units. In addition, the chronological time of surgery and time since surgery until a combined endpoint are known.

To analyse the data with the `success` package, we first arrange them in the format presented in Section [The R package success](#). The outcome of interest is event-free survival. For patients who did not experience an event during the study period, the observations were censored at the last time the patients were known to be event-free. We consider the start of the study as the time when the first patient had surgery. The resulting data was stored in a data.frame called `breast` which can be loaded using the `data` function.

To determine the risk-adjustment models, we consider 36 time units post surgery followup as outcome. We fit a logistic model to be used for risk adjustment in the funnel plot and Bernoulli CUSUM, and a Cox model for risk adjustment in the BK- and CGR-CUSUM:

```
glmmmodEORTC <- glm((survtime <= 36) & (censorid == 1) ~ var1 + var2 +
  var3 + var4 + var5 + var6 + var7,
  data = breast, family = binomial)
phmodEORTC <- coxph(Surv(survtime, censorid) ~ var1 + var2 +
  var3 + var4 + var5 + var6 + var7, data = breast)
```

We then construct the funnel plot and Bernoulli, BK- and CGR-CUSUM for each of the 15 centres in the data. To make the continuous time charts visually interesting, we determine their values at every time unit from the start of the study using the argument `ctimes`.

We then estimate the Poisson arrival rate for each center in the data using the `arrival_rate` function.

```
arrival_rate <- arrival_rate(breast)
arrival_rate

#>      1       2       3       5       4       6       7
#> 0.1568693 0.7127849 0.7922508 0.9657104 0.9676291 1.3067964 1.3870253
#>      8       9      10      11      12      13      14
#> 1.4589466 1.6650555 3.0589682 3.6197173 6.8752701 7.5733827 11.0912301
#>     15
#> 18.2107083
```

Based on the estimated arrival rate $\hat{\psi}$ (per time unit), we group the centres into 3 categories:

- Small: Centres 1-5: $\hat{\psi} \approx 0.9$;

Table 3: Control limits for the EORTC data. Estimated using a simulation procedure so that the probability of a type I error within 36 time units is approximately 0.05.

	Ber	BK	CGR
0.9	2.29	3.23	4.90
2.1	2.93	3.89	5.51
11	4.71	6.43	6.49

- Medium: Centres 6-11: $\hat{\psi} \approx 2.1$;
- Large: Centres 12-15: $\hat{\psi} \approx 11$;

For each category, we determine the control limits to use in the CUSUM charts, using the *_control_limit functions. For this, we restrict the simulated type I error over 60 time units to 0.05.

```

h <- matrix(0, nrow = 3, ncol = 3,
            dimnames = list(c(0.9, 2.1, 11), c("Ber", "BK", "CGR")))
psi <- c(0.9, 2.1, 11)
for(i in 1:3){
  h[i,1] <- bernoulli_control_limit(time = 60, alpha = 0.05, followup = 36,
                                       psi = psi[i], n_sim = 300, theta = log(2), glmmmod = glmmmodEORTC,
                                       baseline_data = breast)$h
  h[i,2] <- bk_control_limit(time = 60, alpha = 0.05, psi = psi[i], n_sim = 300,
                               theta = log(2), coxphmod = phmodEORTC, baseline_data = breast)$h
  h[i,3] <- cgr_control_limit(time = 60, alpha = 0.05, psi = psi[i],
                                n_sim = 300, coxphmod = phmodEORTC, baseline_data = breast)$h
}

```

The resulting control limits can be found in Table 3.

The columns represent the chart and the rows represent the estimated arrival rate. Using these control limits, we determine the times of detection for the 15 centres using the runlength function.

```

times_detection <- matrix(0, nrow = 3, ncol = 15,
                           dimnames = list(c("Ber", "BK", "CGR"), 1:15))
for(i in 1:5){
  times_detection[1,i] <- runlength(EORTC_charts[[i]]$ber, h = h[1,1])
  times_detection[2,i] <- runlength(EORTC_charts[[i]]$bk, h = h[1,2])
  times_detection[3,i] <- runlength(EORTC_charts[[i]]$cgr, h = h[1,3])
}
for(i in 6:11){
  times_detection[1,i] <- runlength(EORTC_charts[[i]]$ber, h = h[2,1])
  times_detection[2,i] <- runlength(EORTC_charts[[i]]$bk, h = h[2,2])
  times_detection[3,i] <- runlength(EORTC_charts[[i]]$cgr, h = h[2,3])
}
for(i in 12:15){
  times_detection[1,i] <- runlength(EORTC_charts[[i]]$ber, h = h[3,1])
  times_detection[2,i] <- runlength(EORTC_charts[[i]]$bk, h = h[3,2])
  times_detection[3,i] <- runlength(EORTC_charts[[i]]$cgr, h = h[3,3])
}

```

We determine the centres which were detected by any of the charts, and compare their detection times.

```

ceiling(times_detection[,colSums(is.infinite(times_detection)) != 3])

#>      3   5   9 10 11 14
#> Ber  50 Inf  41 45 47 Inf
#> BK   Inf  92 Inf 21 25 127
#> CGR  21 112 Inf 16 23 Inf

```

The columns represent the centre numbers, while the rows represent the CUSUM charts. We find that the detections by the considered charts do not coincide perfectly with the centres detected by the

funnel plot (10, 11) at a 5 percent significance level. Comparing the continuous time methods, we see that centre 5 is detected faster by the BK-CUSUM, while centres 10 and 11 are signaled faster by the CGR-CUSUM. Centre 14 is only detected by the BK-CUSUM while centre 9 is only detected by the Bernoulli CUSUM.

An important consideration when comparing detection times between discrete and continuous time methods is that the discrete time charts inspect the survival probability of patients after 36 time units, while the continuous time charts inspect overall survival. This means that the Bernoulli CUSUM might detect a centre with high post operative failure proportions in the 36 time units after surgery. However, it does not mean that patients necessarily experience failures faster than expected at this centre as the Bernoulli CUSUM makes no distinction between a patient who has failed 1 time unit or 10 time units post treatment. This could explain why only the Bernoulli CUSUM detects centre 9.

For the continuous time charts it is important to keep in mind that multiple consecutive failures cause the BK-CUSUM to jump up by $\log(2)$ for every failure, independent of the probability of failure of the patients at that point in time. This can lead to fast detections when many failures are clustered. The CGR-CUSUM can make smaller or larger jumps, depending on the failure probability for each patient at the time of death. This could explain why only the BK-CUSUM detects Centre 14. For centres with low volumes of patients, the maximum likelihood estimate in the CGR-CUSUM might not converge quickly to an appropriate value therefore causing a delay in detection times. In contrast, a wrong choice of θ_1 in the BK-CUSUM may negatively influence detection times (Gomon et al. 2022).

We take a closer look at the disparities between detection times by visualising the funnel plot as well as CUSUM charts for all centres in the EORTC data.

```

unnames <- paste(rep("Centre", 15), 1:15)
ber_EORTC <- lapply(EORTC_charts, FUN = function(x) x$ber)
bk_EORTC <- lapply(EORTC_charts, FUN = function(x) x$bk)
cgr_EORTC <- lapply(EORTC_charts, FUN = function(x) x$cgr)
for(i in 1:5){
  ber_EORTC[[i]]$h <- h[1,1]
  bk_EORTC[[i]]$h <- h[1,2]
  cgr_EORTC[[i]]$h <- h[1,3]
}
for(i in 6:11){
  ber_EORTC[[i]]$h <- h[2,1]
  bk_EORTC[[i]]$h <- h[2,2]
  cgr_EORTC[[i]]$h <- h[2,3]
}
for(i in 12:15){
  ber_EORTC[[i]]$h <- h[3,1]
  bk_EORTC[[i]]$h <- h[3,2]
  cgr_EORTC[[i]]$h <- h[3,3]
}
cols <- palette.colors(6, "Set2")
col_manual <- rep("lightgrey", 15)
col_manual[c(3,5,9,10,11,14)] <- cols
t1 <- interactive_plot(ber_EORTC, unit_names = unnames,
                       scale = TRUE, group_by = "type",
                       manual_colors = col_manual)
t2 <- interactive_plot(bk_EORTC, unit_names = unnames,
                       scale = TRUE, group_by = "type",
                       manual_colors = col_manual)
t3 <- layout(interactive_plot(cgr_EORTC, unit_names = unnames,
                           scale = TRUE, group_by = "type",
                           manual_colors = col_manual))
t0 <- ggplotly(plot(EORTC_funnel))

```

The resulting plots are shown in Figure 9.

All hospitals detected during the time of the study by the CUSUM charts are highlighted, the remaining centres are shown in gray. We can clearly see the 36 time unit delay in the Bernoulli CUSUM charts. The CGR-CUSUM charts have high initial spikes when the first failures are observed. This happens due to the instability of the maximum likelihood estimate $\hat{\theta}(t)$ when only few failures have been observed. After a sufficient amount of patients have been observed, the CGR-CUSUM makes a clear distinction between centres with respect to their performance. Part of the centres retain a value close to zero while for others the value increases over time. In contrast, the BK-CUSUM charts appear

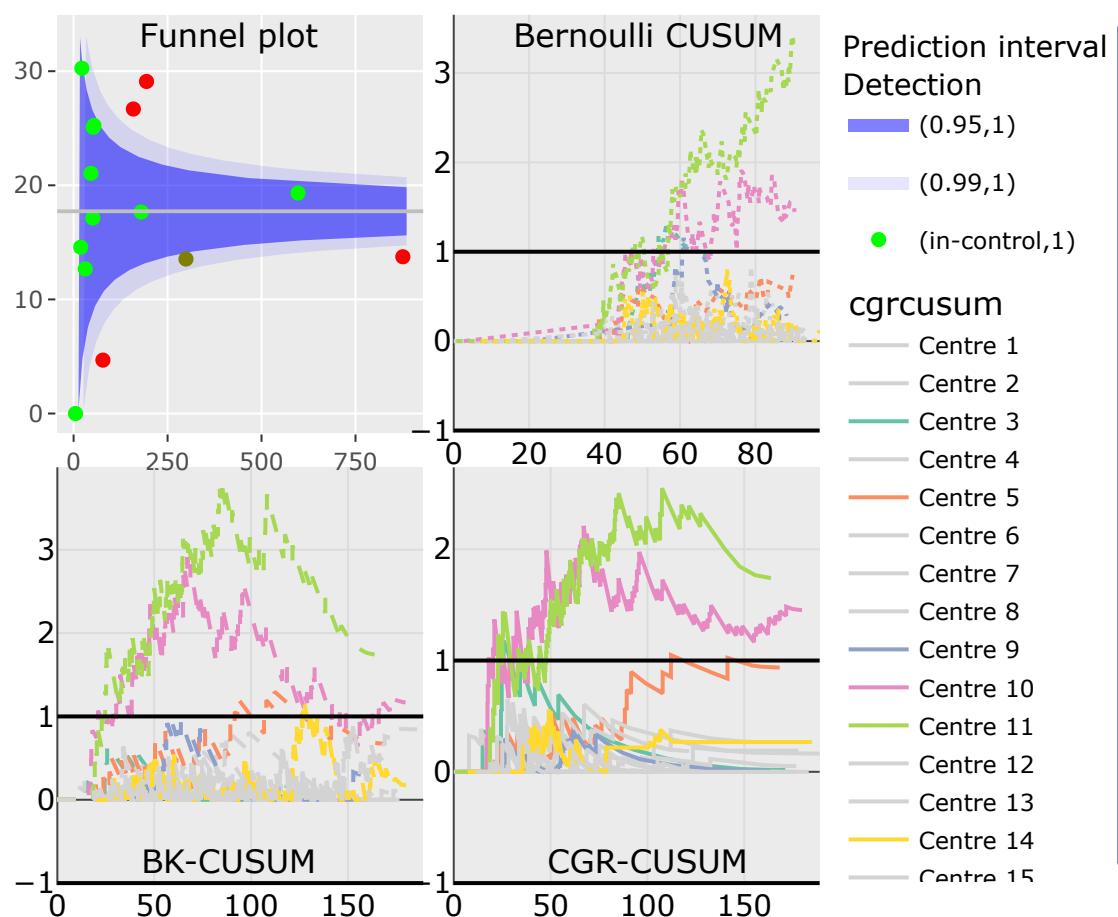


Figure 9: (Top left) Funnel plot of breast data. (Top right) Bernoulli, (Bottom left) BK- and (Bottom right) CGR-CUSUM charts of all 15 centres. Centres not detected by any of the charts are greyed out.

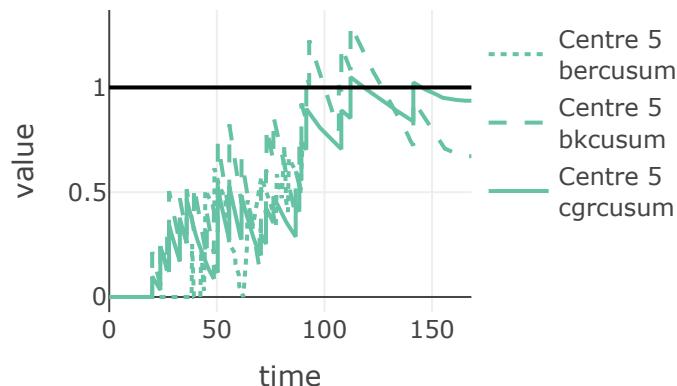


Figure 10: Bernoulli, BK- and CGR-CUSUM for centre 5 of the breast data.

to be less stable, with centres almost hitting the control limit over the period of the study multiple times. This could indicate that the value of $\theta = \ln(2)$ is not suitable for these centres. Only centres 10 (purple) and 11 (lightgreen) were detected by all charts. From the value of the continuous time charts we can presume that centre 10 had a cluster of failures at the beginning of the study, followed by a period of (slightly) above average failures. The Bernoulli CUSUM does not provide such insights, as failures come in 36 time units after surgery. It seems that centre 11 had a high rate of failure from the start until the end of the study. Centre 9 (dark blue) was only detected by the Bernoulli CUSUM. This disparity between discrete and continuous time charts mostly happens when the failure proportions at 36 time units post surgery are relatively large, but many patients fail at reasonable times (e.g. around 30 time units post surgery). From a visual inspection of the charts, this seems to be the case for Centre 9. Finally, Centre 5 was only detected by the continuous time CUSUM charts. We display all 3 CUSUM charts for Centre 5 in Figure 10. The Bernoulli CUSUM only incorporates the information provided by survival 36 time units after surgery. Because of this, the Bernoulli CUSUM can only be calculated up to 36 time units after the last patient had surgery (in this case, until the 89th time unit). The continuous time charts can incorporate failures of patients at any point in time, therefore producing signals at later times. While the BK-CUSUM always rises by $\ln(2)$ whenever a failure is observed, the CGR-CUSUM can make jumps of different sizes, depending on the risk of failure of the observed patient. This causes a disparity in the times of detection and also in interpretation of the values of the charts. As no patients had surgery later than 60 time units after the start of the study, the arrival rate after 60 time units is $\psi = 0$ for all centres, meaning detections at that point should be taken with a grain of salt.

5 Discussion

The `success` package implements three CUSUM methods for the inspection of the failure rate in survival data in continuous time and the funnel plot. Using the `parameter_assist` function, quality control charts can also be constructed by users unfamiliar with control chart and survival theory.

We would like to highlight the different type of outcome and purpose of the control charts in this package. The funnel plot should not be used for the continuous inspection of survival data, as it can only be used to test for a difference in failure proportion at a fixed point in time. The Bernoulli CUSUM is closest to the funnel plot, as it uses the same outcome to determine chart values. The Bernoulli CUSUM is suitable for continuous inspection, but the followup time has a great impact on the resulting conclusions as well as the choice of the expected increase in hazard ratio theta. In contrast, the BK-CUSUM can incorporate patient failures at any point in time, but also requires the specification of theta a priori. The CGR-CUSUM does not have this downside, and as the increase in failure rate is never known in advance in practical applications, it can lead to quicker detection times. Finally, the CUSUM charts test different hypotheses, nuancing their interpretation even further: the Bernoulli and CGR-CUSUM charts can be used to test for a change in the failure rate starting from some patient, while the BK-CUSUM can be used to test for a sudden change in the failure rate of all patients.

The funnel plot, Bernoulli CUSUM and BK-CUSUM do not require a lot of computational power, whereas the computation of the CGR-CUSUM is more sophisticated and can require more time. For this reason, we provide the user the option to parallelize the computation of this chart. A key part of CUSUM charts are their control limits, which are mostly determined using simulation studies due to the lack of analytical results. This can be done using the `*_control_limit` functions in the `success` package. The time required to compute control limits depends on the value of the arrival rate psi and the proportion of failures in the data. A higher value of psi means more patients have to be accounted

for, and a higher failure proportion means chart values need to be calculated more often. For the breast cancer data, determining control limits took approximately 10 minutes in total (using a consumer grade laptop), on a simulated sample of 300 in-control centres.

It is important to determine appropriate control limits for the inspection of survival processes using CUSUM charts. The chosen value of ψ_i greatly influences the value of the control limit. Heuristically, this can be compared with the prediction intervals in the funnel plot. As the number of outcomes in a centre increases, the prediction intervals become narrower. For the CUSUM charts, this is expressed in the control limit.

6 Acknowledgements

The authors thank the European Organization for Research and Treatment of Cancer for permission to apply our methods on data based on an EORTC study. The contents of this publication and methods used are solely the responsibility of the authors and do not necessarily represent the official views of the EORTC.

References

- Anhoej, Jacob. 2021. *qicharts: Quality Improvement Charts*. <https://CRAN.R-project.org/package=qicharts>.
- Austin, P. C. 2012. "Generating Survival Times to Simulate Cox Proportional Hazards Models with Time-Varying Covariates." *Statistics in Medicine* 31 (29): 3946–58. <https://doi.org/10.1002/sim.5452>.
- Biswas, P., and J. D. Kalbfleisch. 2008. "A Risk-adjusted CUSUM in Continuous Time Based on the Cox Model." *Statistics in Medicine* 27: 3452–52. <https://doi.org/10.1002/sim.3216>.
- Cook, David, M Coory, and R Webster. 2011. "Exponentially Weighted Moving Average Charts to Compare Observed and Expected Values for Monitoring Risk-Adjusted Hospital Indicators." *BMJ Quality and Safety* 20 (May): 469–74. <http://dx.doi.org/10.1136/bmjqqs.2008.031831>.
- Cox, D. R. 1972. "Regression Models and Life-Tables." *Journal of the Royal Statistical Society. Series B (Methodological)* 34 (2): 187–220. <https://doi.org/10.1111/j.2517-6161.1972.tb00899.x>.
- Flores, Miguel. 2021. *qcr: Quality Control Review*. <https://CRAN.R-project.org/package=qcr>.
- Gomon, Daniel, Hein Putter, Rob G. H. H. Nelissen, and Stéphanie van der Pas. 2022. "CGR-Cusum: A Continuous Time Generalized Rapid Response Cumulative Sum Chart." *Biostatistics*. <https://doi.org/10.1093/biostatistics/kxac041>.
- Grey, Kenneth. 2018. *ggQC: Quality Control Charts for 'Ggplot'*. <https://CRAN.R-project.org/package=ggQC>.
- Grigg, O. A. 2018. "The STRAND Chart: A Survival Time Control Chart." *Statistics in Medicine* 38 (9): 1651–61. <https://doi.org/10.1002/sim.8065>.
- Hubig, Lena. 2019. *cusum: Cumulative Sum (CUSUM) Charts for Monitoring of Hospital Performance*. <https://CRAN.R-project.org/package=cusum>.
- Keefe, Matthew J., Justin B. Loda, Ahmad E. Elhabashy, and William H. Woodall. 2017. "Improved Implementation of the Risk-Adjusted Bernoulli CUSUM Chart to Monitor Surgical Outcome Quality." *International Journal for Quality in Health Care* 29 (3): 343–48. <https://doi.org/10.1093/intqhc/mzx036>.
- Knoth, Sven. 2021. *spc: Statistical Process Control – Calculation of ARL and Other Control Chart Performance Measures*. <https://CRAN.R-project.org/package=spc>.
- Kumar, Matthew. 2018. *funnelR: Funnel Plots for Proportion Data*. <https://CRAN.R-project.org/package=funnelR>.
- Scrucca, Luca. 2004. "Qcc: An R Package for Quality Control Charting and Statistical Process Control." *R News* 4/1: 11–17. <https://CRAN.R-project.org/package=qcc>.
- Sievert, Carson. 2020. *Interactive Web-Based Data Visualization with R, Plotly, and Shiny*. Chapman; Hall/CRC. <https://plotly-r.com>.
- Solymos, Peter, and Zygmunt Zawadzki. 2021. *pbapply: Adding Progress Bar to 'Apply' Functions*. <https://CRAN.R-project.org/package=pbapply>.
- Spiegelhalter, D. J. 2005. "Funnel Plots for Comparing Institutional Performance." *Statistics in Medicine* 24 (8): 1185–1202. <https://doi.org/10.1002/sim.1970>.
- Steiner, S. H., R. J. Cook, V. T. Farewell, and T. Treasure. 2000. "Monitoring Surgical Performance Using Risk-Adjusted Cumulative Sum Charts." *Biostatistics* 1 (4): 441–52. <https://doi.org/10.1093/biostatistics/1.4.441>.
- Steiner, S. H., and M. Jones. 2009. "Risk-Adjusted Survival Time Monitoring with an Updating Exponentially Weighted Moving Average (EWMA) Control Chart." *Statistics in Medicine* 29 (4): 444–54. <https://doi.org/10.1002/sim.3788>.

- Terry M. Therneau, and Patricia M. Grambsch. 2000. *Modeling Survival Data: Extending the Cox Model*. New York: Springer.
- Wickham, Hadley. 2016. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. <https://ggplot2.tidyverse.org>.
- Wittenberg, Philipp, and Sven Knoth. 2020. *vlad: Variable Life Adjusted Display and Other Risk-Adjusted Quality Control Charts*. <https://CRAN.R-project.org/package=vlad>.

Daniel Gomon
Leiden University
Mathematical Institute
Niels Bohrweg 1
2333CA Leiden, the Netherlands
<https://github.com/d-gomon>
ORCID: 0000-0001-9011-3743
d.gomon@math.leidenuniv.nl

Marta Fiocco
Leiden University
Mathematical Institute
Niels Bohrweg 1
2333CA Leiden, the Netherlands
ORCID: 0000-0001-5588-0277

Hein Putter
Leiden University
Mathematical Institute
Niels Bohrweg 1
2333CA Leiden, the Netherlands
ORCID: 0000-0001-5395-1422

Mirko Signorelli
Leiden University
Mathematical Institute
Niels Bohrweg 1
2333CA Leiden, the Netherlands
ORCID: 0000-0002-8102-3356

Changes in R

by Tomas Kalibera and Sebastian Meyer

Abstract We present selected changes in the development version of R (referred to as R-devel, to become R 4.4) and provide some statistics on bug tracking activities in 2023.

1 Selected changes in R-devel

R 4.4.0 is due to be released around April 2024. The following gives a selection of changes in R-devel, which are likely to appear in the new release. The summaries below include text contributed by authors of some of the changes: Peter Dalgaard, Martyn Plummer, Brian Ripley, Deepayan Sarkar and Luke Tierney.

- The `anova()` function is used for analysis of variance for linear models and analysis of deviance for generalized linear models (GLMs). Previously the `anova()` function behaved differently for GLMs: it would not show test statistics and p-values by default, instead relying on the user to specify the required test statistic. Thanks to changes to "family" objects already included in R 4.3.0, the `anova()` function can now determine an appropriate default test for comparing two GLMs ("LRT" for families with a fixed dispersion parameter and "F" for families with free dispersion) and will show this along with the associated p-value.
- As part of the process of allowing the use of Rao's score test in connection with `glm()`, the `confint()` method for "glm" objects now allows `test = "Rao"`, as does the underlying `profile()` method. To enable this, the code for these functions, and also the corresponding `plot()` and `pairs()` methods, was copied from the **MASS** package to the R sources before modification. The `pairs()` method has also been revised to better handle the case where only a subset of parameters have been profiled.
- R 4.4.0 will include support for producing single-page HTML reference manuals for an entire package, similar to the PDF reference manuals currently hosted on CRAN package pages. It will also include support for a table of contents in HTML help pages, which is controlled by `options("help.htmltoc")`.
- R 4.3.0 added support for experimenting with alternate object systems by providing the `chooseOpsMethod()` generic for resolving method selection for `Ops` group generics, and the `nameOfClass()` generic to allow more flexible class representations to be used in `inherits()`. In addition, `@` became an internal generic, `@<-` already was. R 4.4.0 will add internal support for bare objects by renaming the `S4SXP` type to `OBJSXP` and having `typeof()` return "object" for generic bare objects. For now, generic bare `S4` objects are distinguished by having a special bit set; it is hoped that this can eventually be dropped.
- R relies on the system `libiconv` for encoding conversions, especially from UTF-8. Apple replaced completely its `libiconv` in macOS 14 with substantial revisions in 14.1 and 14.2: rather than reporting errors when an exact conversion is not possible, it in almost all cases attempts 'transliteration' so for example permille ("‰") is rendered as "o/oo".
musl (as used by Alpine Linux) has long substituted "", but we now faced converted strings growing in length. Issues were particularly seen when plotting on `pdf()` devices and it became clear many package authors had never looked at their graphical output. That suggested that transliteration was a safer route, and now R transliterates if the system `libiconv` has not got there first and so (except in rare cases and under musl) R will give the same PDF output on all platforms.
- `Rprof()`, the sampling profiler in R, now supports profiling in "elapsed" time (a.k.a. wall-clock time, real-time) on Unix in addition to "cpu" time. When profiling in elapsed time, the time advances also while R is waiting on I/O, so it may be preferred for some kinds of analysis in I/O intensive applications. Also, elapsed time profiling is the only one currently supported on Windows, so it is good to have a matching option on Unix.
- R gained initial support for 64-bit ARM hardware on Windows (macOS and Linux machines are already supported). It is already possible to build R and recommended packages from source and they pass their automated checks. Testing and porting of other CRAN packages has been started, with a number of patches contributed to package maintainers. This effort uses an experimental LLVM-based toolchain with the new flang compiler, which has been added to Rtools. In addition to actually supporting 64-bit ARM Windows machines, which are still rare but emerging, this effort also drives portability improvements of R and R packages. Previously, a lot of this code explicitly or implicitly assumed GCC compilers and Intel CPUs on Windows.

- The R CMD check utility for package development performs some additional checks on R documentation (Rd) files. The most prominent addition (in the sense that over 3000 CRAN packages were affected) is a new note about “lost braces”. In (LaTeX-like) Rd syntax, braces are used to mark arguments and otherwise group tokens; they must be escaped as `\{` and `\}` to be included literally in normal text. The new check tries hard to report relevant mistakes, for example:
 - `code{...}`: missing backslash in front of the macro name
 - `{1, 2}`: in-text set notation, where the braces need escaping or the whole expression needs to be put inside a math `\eqn{}`
 - `\itemize{ ... \item{label}{description} ... }`: Rd code meant as a description list with initial labels; this needs `\describe` instead of `\itemize`, otherwise the element becomes “labeldescription” because an `\itemize \item` does not take any arguments.
- A new binary infix operator `%||%` is defined in **base**. This is the so-called *null coalescing operator*: `x %||% y` expresses “use `x` if not `NULL`, otherwise use `y`”.
- `is.atomic(NULL)` now returns `FALSE` and thus behaves according to the R language definition of an atomic vector (`RShowDoc("R-lang")`, Section 2.1.1), which covers the six basic types `logical`, `integer`, `double`, `complex`, `character` and `raw`. For historical reasons (compatibility with S), `is.atomic(NULL)` gave `TRUE` in R < 4.4.0, treating `NULL` loosely as “any vector of size 0”. Similarly, `NCOL(NULL)` returned 1 but now gives 0.
- There is a new startup option `--max-connections` to set the maximum number of connections for the R session. It defaults to 128 as before. Values up to 4096 are allowed, but resource limits may in practice restrict to smaller values. This enables advanced users to configure R in environments where a large number of connections (e.g., network) is needed.
- R 4.4.0 on recent Windows will use the new Segment Heap allocator provided by the system. This new allocator has slightly better performance on some applications than the default Low Fragmentation Heap allocator, with the hope that it would be further improved in future versions of Windows.
- R makes use of a system libdeflate library if available, in preference to the system libz library. This can speed up decompressing R objects in lazy-loading databases and other operations.

See the NEWS.Rd file in the R sources for a more complete list; nightly rendered versions are available at <https://CRAN.R-project.org/doc/manuals/r-devel/NEWS.html> with RSS feeds at <https://developer.R-project.org/RSSfeeds.html>.

2 Bug statistics for 2023

Summaries of bug-related activities over the past year were derived from the database underlying R’s [Bugzilla system](#). Overall, 186 new bugs or requests for enhancements were reported, 204 reports were closed, and 942 comments were added by a total of 120 contributors. The numbers of new reports and contributors were comparable to 2022, but comments increased by 8% and closures by 20%. Higher activity in 2023 was driven by a dedicated effort in reviewing and discussing open reports during the R Project Sprint at the University of Warwick, UK, 30 August to 1 September (Turner and Becker 2023).

Figure 1 shows the monthly numbers of new reports, closures and comments in 2023. Comment activity was relatively low in July and peaked in September due to the sprint.

The top 5 components reporters have chosen for their reports were “Low-level”, “Misc”, “Language”, “Documentation”, and “Accuracy”. 9% of the reports were suggestions for enhancements that were submitted either in the “Wishlist” component or in a specific component but with severity level set to “enhancement”.

References

- Turner, Heather, and Gabriel Becker. 2023. “R Project Sprint 2023.” *The R Journal* 15: 299–305.
<https://journal.R-project.org/news/RJ-2023-3-sprint>.

Tomas Kalibera
 R Core Team
 Prague, Czechia
 ORCID: 0000-0002-7435-734X
Tomas.Kalibera@R-project.org

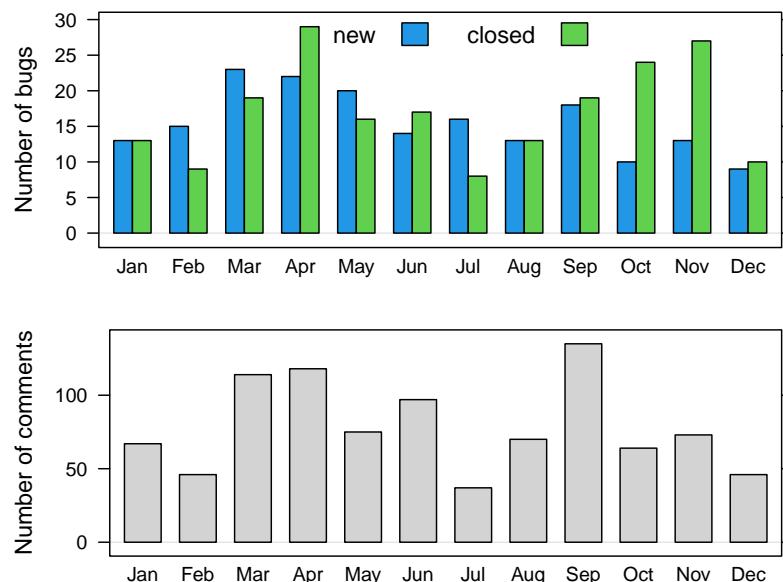


Figure 1: Bug tracking activity by month in 2023.

Sebastian Meyer

Friedrich-Alexander-Universität Erlangen-Nürnberg

Erlangen, Germany

ORCID: 0000-0002-1791-9449

Sebastian.Meyer@R-project.org

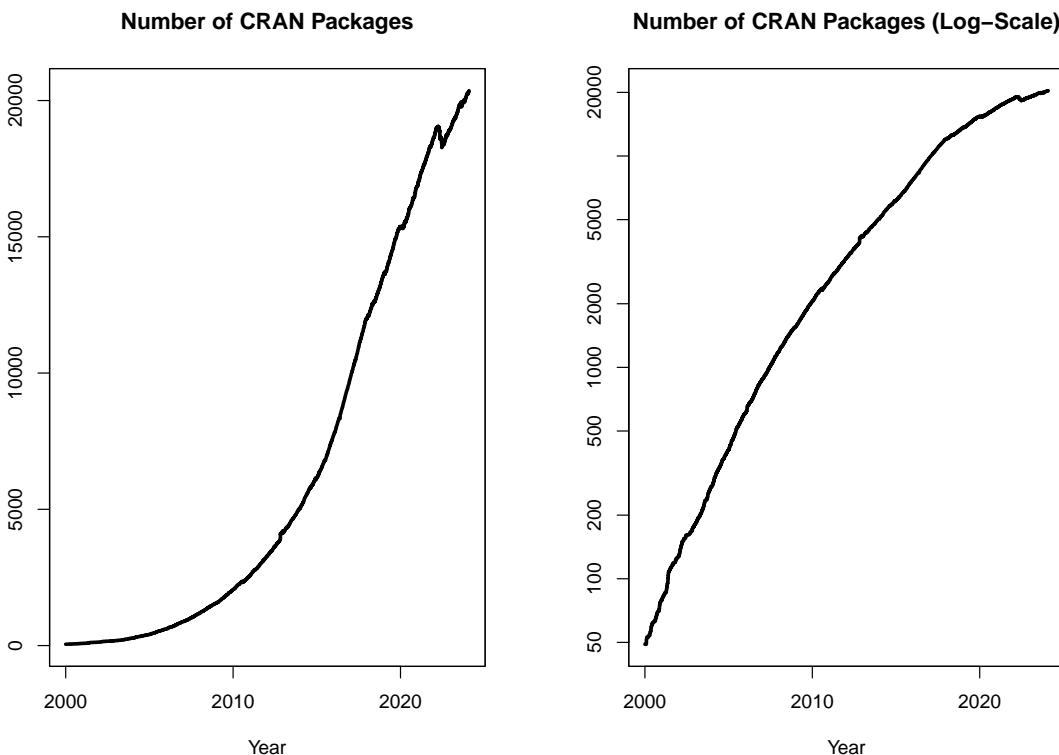
Changes on CRAN

2023-10-01 to 2023-12-31

by Kurt Hornik, Uwe Ligges, and Achim Zeileis

1 CRAN growth

In the past 3 months, 455 new packages were added to the CRAN package repository. 217 packages were unarchived, 362 were archived and 1 had to be removed. The following shows the growth of the number of active packages in the CRAN package repository:



On 2023-12-31, the number of active packages was around 20249.

2 CRAN package submissions

From October 2023 to December 2023 CRAN received 7408 package submissions. For these, 12015 actions took place of which 8598 (72%) were auto processed actions and 3417 (28%) manual actions.

Minus some special cases, a summary of the auto-processed and manually triggered actions follows:

	archive	inspect	newbies	pending	pretest	publish	recheck	waiting
auto	2489	749	1627	18	0	2352	842	521
manual	1178	73	58	206	75	1380	379	68

These include the final decisions for the submissions which were

	archive	publish
auto	2394 (32.9%)	2045 (28.1%)
manual	1162 (16.0%)	1681 (23.1%)

where we only count those as *auto* processed whose publication or rejection happened automatically in all steps.

3 CRAN mirror security

Currently, there are 94 official CRAN mirrors, 77 of which provide both secure downloads via ‘`https`’ and use secure mirroring from the CRAN master (via `rsync` through ssh tunnels). Since the R 3.4.0 release, `chooseCRANmirror()` offers these mirrors in preference to the others which are not fully secured (yet).

4 CRAN Task View Initiative

Currently there are 44 task views (see <https://CRAN.R-project.org/web/views/>), with median and mean numbers of CRAN packages covered 104 and 123, respectively. Overall, these task views cover 4516 CRAN packages, which is about 22% of all active CRAN packages.

Kurt Hornik
WU Wirtschaftsuniversität Wien
Austria
ORCID: [0000-0003-4198-9911](https://orcid.org/0000-0003-4198-9911)
Kurt.Hornik@R-project.org

Uwe Ligges
TU Dortmund
Germany
ORCID: [0000-0001-5875-6167](https://orcid.org/0000-0001-5875-6167)
Uwe.Ligges@R-project.org

Achim Zeileis
Universität Innsbruck
Austria
ORCID: [0000-0003-0918-3766](https://orcid.org/0000-0003-0918-3766)
Achim.Zeileis@R-project.org

News from the Forwards Taskforce

by Heather Turner

Abstract [Forwards](#) is an R Foundation taskforce working to widen the participation of under-represented groups in the R project and in related activities, such as the *useR!* conference. This report rounds up activities of the taskforce during 2023.

0.1 Accessibility

Di Cook, along with Mitchell O'Hara Wild, co-mentored Abhishek Ulayil on a Google Summer of Code (GSoC) 2023 project [converting past R Journal articles to HTML](#) continuing from work started in GSoC 2022. Further improvements were made to the R packages written for the conversion, enabling HTML versions to be created for all articles in the archive.

Di Cook, Heather Turner and Jonathan Godfrey are leading an R Consortium funded project to begin adding alt text to the figures in the converted HTML articles, to make them fully accessible.

At R Project Sprint 2023, Jonathan Godfrey collaborated with participants on a couple of accessibility issues, now incorporated [BrailleR](#). Work with Deepayan Sarkar led to improvement in the ability to extract content of graphics devices, see `summary.recodedplot()`. Work with Gabriel Becker led to a working solution to extract recent console output, see the `ShowMe()`, `SessionLog()`, and `GrabLast()` functions.

0.2 Community engagement

Kevin O'Brien, Ella Kaye and Heather Turner attended SatRdays London 2023. This was an opportunity to catch up with community organizers, including Tuli Amutenya and Emmanuel Olawale Olamijuwon, from Namibia and Eswatini R User Groups respectively, now working in UK. Ella and Heather gave a talk on [Sustainability and EDI in the R Project](#), giving an overview of their work as part of Heather's research fellowship on this topic.

[RainbowR](#), led by Ella Kaye and Hanne Oberman, have increased the frequency of their online meetups and are now meeting monthly. At the end of November, they launched a [pilot buddy scheme](#) to foster stronger connections between community members, with a plan to pair people up with buddies every three months.

Also in November, Kevin O'Brien started a monthly community call for organizers of R User Groups - upcoming meetings can be found on the [Global R User Group](#) meetup.

0.3 Conferences

Julie Josse is on the program committee for *useR! 2024* and Forwards have been involved in suggesting people for keynotes and the organizing/program committees.

Yanina Bellini Saibene co-chaired [LatinR 2023](#) along with Natalia da Silva and Riva Quiroga. This was the first in-person LatinR since going online in 2020 and the first time R experts from outside Latin America attended as keynotes and instructors. The conference was attended by 300 people from 14 different countries and of the 82% who reported their gender, 48% identified themselves as women. Thanks to sponsor support, scholarships were awarded to 25 participants. As in previous years, content was presented in Spanish, Portuguese, and English. In future, the conference will alternate between in-person and online format.

0.4 R Contribution

Forwards members continue to play an active role in the R Contribution Working Group (RCWG).

Heather Turner and Ella Kaye, have been regular facilitators of the monthly office hours for R contributors, along with Gabriel Becker. Thanks to the R Consortium, these office hours are now advertised on the [R Contributors Meetup](#), along with other RCWG events, which has help to sustain good attendance.

Saranjeet Kaur Bhogal continued work on the [R Development Guide](#), with funding remaining from the Google Season of Docs (GSoD) 2022. This work improved structure and content, based on reviews by the steering committee.

Several Forwards members were involved in R Project Sprint 2023 (a full report of this event was published in the R Journal Volume 15/3). One important outcome of the sprint was a new section in the R Dev Guide on [How to contribute new translations](#) using the Weblate interface, which only existed as a prototype when the translation chapter was added as part of the GSoD 2022 project.

Heather Turner, along with James Tripp, mentored Atharva Shirdhankar on a Google Summer of Code 2023 project creating the [R Dev Container](#) a GitHub Codespace providing a containerised development environment for editing and compiling the R source code. The prototype proved useful at the R Project Sprint and further development is planned to improve on the first version.

At LatinR 2023, Pao Corrales gave a lightning talk with María Nanton on contributing translations to R and they co-organized a translation space. The Spanish translation coverage increased from 40% to 42% during the event. Heather Turner gave a keynote on [Contributing to R](#) at the II Conference of R in Barcelona in November, where the audience were happy to hear that a Catalan translation has been started, with initial translations due to be added to R 4.4.0.

0.5 Social Media

Zane Daz, Ella Kaye, gwynn gebeyehu and Heather Turner gave the Forwards website a long overdue update, switching from the previous Hugo/blogdown framework to Quarto. This should be easier to maintain and we hope to add some fresh content in 2024.

0.6 Changes in Membership

Previous members

The following members have stepped down:

- Teaching team: Emily Dodwell, Ritwik Mitra
- Community team: Sam Toet
- Conferences team: Yanina Bellini Saibene (co-leader), Noa Tamir (co-leader)
- Surveys team: Andrea Sánchez-Tapia (co-leader), Claudia Huaylla
- On-ramps team: Jyoti Bhogal, Allison Vuong

We thank them for their contribution to the taskforce.

Heather Turner
University of Warwick
Coventry, United Kingdom
<https://warwick.ac.uk/heatherturner>
ORCID: 0000-0002-1256-3375
heather.turner@r-project.org

R Foundation News

by Torsten Hothorn

1 Donations and members

Membership fees and donations received between
2023-10-20 and 2024-04-12.

2 Donations

b-data GmbH (Switzerland)
SAS EUREKA MER (France)
Gilberto Camara (Brazil)
Keith Chamberlain (United States)
Giles Dickenson-Jones (Australia)
Shalese Fitzgerald (United States)
Roger Koenker (United Kingdom)
Flavio Lombardo (Switzerland)
Rudolph Martin (United States)
Rees Morrison (United States)
Quintessa Ltd (United Kingdom)
Kem Phillips (United States)
Alexandra Pippitt (United States)
Bruno Rodrigues (Luxembourg)
David Smith (United States)
Rav Vaid (United States)
Alejandro Verri Kozlowski (Argentina)
Yihui Xie (China)
ilustat, Lisbon (Portugal)
Statistik Aargau, Aarau (Switzerland)

3 Supporting institutions

Departement Klinische Forschung, Basel (Switzerland)
Ef-prime, Inc., Tokyo (Japan)
Institute of Botany of the Czech Academy of Sciences, Průhonice (Czechia)
oikostat GmbH, Ettiswil (Switzerland)

4 Supporting members

Richard Abdill (United States)
Douglas Adamoski (Brazil)
Mohammed Almozini (Saudi Arabia)
Tim Appelhans (Germany)
Tim Arbogast (United States)
Michael Blanks (United States)

Emmanuel Blondel (France)
Gordon Blunt (United Kingdom)
Riccardo Bonfichi (Italy)
Tom Boulay (United States)
Tamara Bozovic (New Zealand)
Keith Chamberlain (United States)
Cédric Chambru (Switzerland)
John Chandler (United States)
Michael Chirico (United States)
Tom Clarke (United Kingdom)
Gerard Conaghan (United Kingdom)
Robin Crockett (United Kingdom)
Alistair Cullum (United States)
Brandon Dahl (United States)
Robert Daly (Australia)
Kevin DeMaio (United States)
Anna Doizy (Réunion)
Fraser Edwards (United Kingdom)
Anthony Alan Egerton (Malaysia)
Isaac Florence (United Kingdom)
Neil Frazer (United States)
David Freedman (United States)
Bernd Fröhlich (Germany)
Sven Garbade (Germany)
Jan Marvin Garbuszus (Germany)
Eduardo García Galea (Spain)
Gabriel Gersztein (Brazil)
SUJOY GHOSH (United States)
Anne Catherine Gieshoff (Switzerland)
Pavel Goriacko (United States)
Brian Gramberg (Netherlands)
Spencer Graves (United States)
Krushi Gurudu (United States)
Frank Hafner (United States)
Hlynur Hallgrímsson (Iceland)
Joe Harwood (United Kingdom)
Bela Hausmann (Austria)
Kieran Healy (United States)
Philippe Heymans Smith (Costa Rica)
Adam Hill (United States)
Alexander Huelle (Germany)
Heidi Imker (United States)
Sebastian Jeworutzki (Germany)
JUNE KEE KIM (Korea, Republic of)
Ziyad Knio (United States)

Sebastian Koehler (Germany)
Chris Kuty (United States)
Luca La Rocca (Italy)
Vishal Lama (United States)
Thierry Lecerf (Switzerland)
Thomas Levine (United States)
Eric Lim (United Kingdom)
Baoxiao Liu (Netherlands)
Joseph Luchman (United States)
Mehrad Mahmoudian (Finland)
Gilles Marodon (France)
Daniel McNichol (United States)
Philippe MICHEL (France)
Bogdan-Alexandru Micu (Luxembourg)
Ernst Molitor (Germany)
David Monterde (Spain)
Stefan Moog (Germany)
Keon-Woong Moon (Korea, Republic of)
Steffen Moritz (Germany)
yoshinobu nakahashi (Japan)
Tsubasa Narihiro (Japan)
Maciej Nasinski (Poland)
Dan Orsholits (Switzerland)
Antonio Paez (Canada)
Sermet Pekin (Turkey)
Elgin Perry (United States)
PierGianLuca Porta Mana (Norway)
Fergus Reig Gracia (Spain)
Peter Ruckdeschel (Germany)
Ingo Ruczinski (United States)
Choonghyun Ryu (Korea, Republic of)
John Schmitt (United States)
Raoul Schorer (Switzerland)
Dejan Schuster (Germany)
Ivan Scotti (France)
David Sides (United States)
Rachel Smith-Hunter (United States)
Matteo Starri (Italy)
Harald Sterly (Germany)
Tobias Strapatsas (Germany)
Kai Streicher (Switzerland)
ROBERT Szabo (Sweden)
Jan Tarabek (Czechia)
Koray Tascilar (Germany)
Chris Toney (United States)

Robert van den Berg (Austria)
Marcus Vollmer (Germany)
Petr Waldauf (Czechia)
Jaap Walhout (Netherlands)
Sandra Ware (Australia)
Fredrik Wartenberg (Sweden)
Sam Waters (United States)
Dieter Wilhelm (Germany)
Nan Xiao (United States)
Matti Zemack (Sweden)
Vaidotas Zemlys-Balevičius (Lithuania)
Lim Zhong Hao (Singapore)
Guangyu Zeng (China)

Torsten Hothorn
Universität Zürich
Switzerland
ORCID: [0000-0001-8301-0471](https://orcid.org/0000-0001-8301-0471)
Torsten.Hothorn@R-project.org