

The Journal

Volume 15/3, September 2023

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

Contents

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

Contributed Research Articles

Variety and Mainstays of the R Developer Community	5
SSNbayes: An R Package for Bayesian Spatio-Temporal Modelling on Stream Networks	
26	
C443: An R-package to see a forest for the trees	59
TwoSampleTest.HD: An R Package for the Two-Sample Problem with High-Dimensional	
Data	79
Statistical Models for Repeated Categorical Ratings: The R Package rater	93
GREENeR: An R Package to Estimate and Visualize Nutrients Pressures on Surface	
Waters	119
bayesassurance: An R Package for Calculating Sample Size and Bayesian Assurance .	138
fasano.franceschini.test: An Implementation of a Multivariate KS Test in R.	159
Bayesian Inference for Multivariate Spatial Models with INLA	172
Two-Stage Sampling Design and Sample Selection with the R Package R2BEAT . . .	191
fnets: An R Package for Network Estimation and Forecasting via Factor-Adjusted VAR	
Modelling.	214
Coloring in R's Blind Spot	240
Updates to the R Graphics Engine: One Person's Chart Junk is Another's Chart	
Treasure.	257
'mathml': Translate R Expressions to MathML and LaTeX	277

News and Notes

Changes on CRAN	292
R Foundation News	294
News from Bioconductor	296
R Project Sprint 2023	299

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

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 3 of the R Journal.

We would like to welcome Emi Tanaka to our executive editorial board. Emi has served as an Associate Editor for the last two years and kindly agreed to take the role of an Executive Editor. In addition, we would like to also welcome Ursula Laa, Yanfei Kang and Lucy D'Agostino McGowan to our Associate Editors team.

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

In this issue

News from CRAN, the R Foundation and Bioconductor are included in this issue as well as a report on the R Project Sprint 2023.

This issue features 14 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. Topics covered in this issue are

Graphics and visualization

- Coloring in R's Blind Spot
- Updates to the R Graphics Engine: One Person's Chart Junk is Another's Chart Treasure
- [C443](#): An R Package to See a Forest for the Trees
- [GREENeR](#): An R Package to Estimate and Visualize Nutrients Pressures on Surface Waters

Bayesian inference

- [SSNbayes](#): An R Package for Bayesian Spatio-Temporal Modelling on Stream Networks
- The R Package [rater](#)
- [bayesassurance](#): An R Package for Calculating Sample Size and Bayesian Assurance
- Bayesian Inference for Multivariate Spatial Models with [INLA](#)

Multivariate statistics

- [fasano.franceschini.test](#): An Implementation of a Multidimensional KS Test in R
- [TwoSampleTest.HD](#): An R Package for the Two-Sample Problem with High-Dimensional Data
- [fnets](#): An R Package for Network Estimation and Forecasting via Factor-Adjusted VAR Modelling

Other

- Variety and Mainstays of the R Developer Community
- Two-stage Sampling Design and Sample Selection with the R Package [R2BEAT](#)
- [mathml](#): Translate R Expressions to MathML and LaTeX

Simon Urbanek
University of Auckland

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

Variety and Mainstays of the R Developer Community

by Lijin Zhang, Xueyang Li, and Zhiyong Zhang

Abstract The thriving developer community has a significant impact on the widespread use of R software. To better understand this community, we conducted a study analyzing all R packages available on CRAN. We identified the most popular topics of R packages by text mining the package descriptions. Additionally, using network centrality measures, we discovered the important packages in the package dependency network and influential developers in the global R community. Our analysis showed that among the 20 topics identified in the topic model, *Data Import, Export, and Wrangling*, as well as *Data Visualization, Result Presentation, and Interactive Web Applications*, were particularly popular among influential packages and developers. These findings provide valuable insights into the R community.

1 Introduction

Initially started as a personal project by Ross Ihaka and Robert Gentleman (Ihaka and Gentleman, 1996), R has evolved into one of the most widely used and powerful software packages in the field of data science. It is used across a wide range of academic fields and industries. For example, Lai et al. (2019) showed that 58% of the papers from 30 ecology journals in 2017 reported using R for data analysis. Another example is Bioconductor (Gentleman et al., 2004), a popular R software repository for computational biology and bioinformatics, which has over 2,000 R packages for genetic data analysis. Moreover, *ggplot2* (Wickham, 2011), a well-known R package for data visualization, has been cited more than 30,000 times. Courses teaching R are also in high demand in data science programs (Zhang and Zhang, 2021). Fox and Leanage (2016) analyzed the papers published in the Journal of Statistical Software (JSS) between 1996 and 2016, and found that 75% of the articles were about R, which demonstrated the dominance of R software projects in JSS.

The R developer community plays an important role in maintaining a healthy ecosystem of R packages and increasing R's popularity (Chambers, 2020; Tippmann, 2015). The R ecosystem comprises of the base packages developed by the R Core team and user-contributed packages (German et al., 2013). The Comprehensive R Archive Network (CRAN) is a well-known package repository. Fox (2009) suggested that the CRAN package archive is probably the most important driving force for the growing usage of R. The package system provides essential tools for users to develop packages and promotes the sharing of newly-developed methods and ideas throughout the community. Fox (2009) found that the growth of CRAN packages from 2000 to 2009 was approximately exponential. Since then, the number of new CRAN packages has slowed down a bit, but has otherwise maintained a steady pace (Fox and Leanage, 2016). As of 27th November 2022, there are 18898 packages available on CRAN, more than eight times of the number in 2009.

As the number of CRAN packages increases, the flat organization of these packages (Fox and Leanage, 2016) makes it difficult for user to identify popular and important packages on CRAN. Silge et al. (2018) suggested that the huge size of the CRAN package archive has already made it difficult for users to understand the merits of packages and the relationships among packages. Although some resources such as CRAN Task Views (Zeileis, 2005) group packages related to specific topics, they cannot cover the enormous number of CRAN packages or give an overall view of the various topics covered by the CRAN packages. Moreover, as the number of CRAN packages increases, the newly-developed packages are often built upon other user-contributed packages. It is, therefore, essential to explore the package dependency network and identify the influential developers who lead the R ecosystem.

As such, this study aims to conduct a broad survey of the R developer community. Specifically, the goal of this paper is two-fold: 1) to investigate the popular topics of the R ecosystem, and 2) to explore the influential packages and authors in the R developer community.

The large amount of information on CRAN is a challenge but also a great resource for understanding the R community. To investigate this community, this paper uses text mining and network analysis techniques, with a focus on analyzing package descriptions and the relationships between packages and authors. The paper is structured as follows. First, we analyze the textual data of CRAN package descriptions to identify the various topics of the R ecosystem. Second, a package dependency network, an author collaboration network, and a bipartite network of packages and authors are built to determine which packages and authors play important roles based on the network statistics for the development of the R ecosystem. Finally, we present a detailed discussion of our findings.

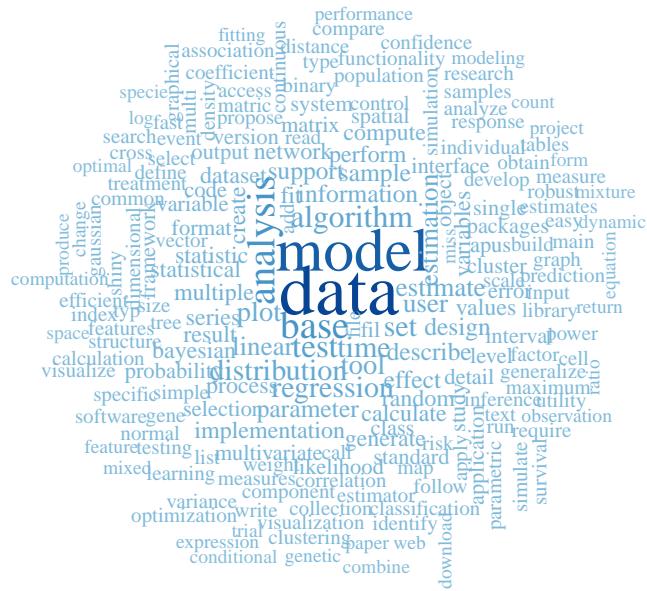


Figure 1: Word cloud of the top 200 frequently used words in the CRAN package descriptions.

2 Identifying the Main Topics of the R Packages

Our first goal is to understand what kind of topics and methods are currently covered in the packages of R. For this purpose, we extracted the descriptions of all 18898 packages from CRAN on 27th November 2022. The mean of the length of package descriptions is 60.15 words, ranging from 1 to 1207 words. Note that data and code of this paper are available at https://github.com/zhanglj37/R_Developer_Community, with which the results in this article can be replicated. Through text analysis technique, we explored the frequently used words and phrases to investigate the focuses of R developers, and conducted topic modeling to identify the main topics of the R ecosystem.

2.1 Data Cleaning

The pre-processing of the package descriptions includes five steps: 1) we converted the upper case letters to lower cases, 2) we deleted the web links, DOI (Digital Object Identifier) links of publications, and numbers (e.g., 1993), and 3) we removed the common stopwords (e.g., *the*, *a*) and some commonly-used words with limited meaning (e.g., *package*, *provide*, *method*). Moreover, to unify different forms of the same root-words, we further 4) singularized plural nouns using the **SemNetCleaner** package (Christensen and Kenett, 2019), and 5) lemmatized verbs using the **spacyr** package (Benoit and Matsuo, 2020).

2.2 Word and Phrase Frequency

In Figure 1, a word cloud depicts the top 200 frequently used words in the package descriptions. Each of these 200 words was used more than 400 times. Four of them appeared more than 4,000 times (*data*, *model*, *analysis*, *base*), and *data* is the only one appeared more than 10,000 times. These words (e.g., *data*, *model*) are related to data analysis and were also commonly used in data science curricula (Zhang and Zhang, 2021).

Figure 2 includes the top 30 frequently used one-, two-, and three-word phrases in the package descriptions, including phrases that are related to statistical models (e.g., *regression model*, *structural equation modeling*), estimation methods (e.g., *least squares*, *maximum likelihood estimation*), different types of datasets (e.g., *high dimensional data*, *gene expression data*), and other common and general terms in data science (e.g., *data analysis*, *variable selection*, *open source*). We also explored the frequently used four-word phrases, and the top phrases include *Markov chain Monte Carlo* (141 times), *genome wide*

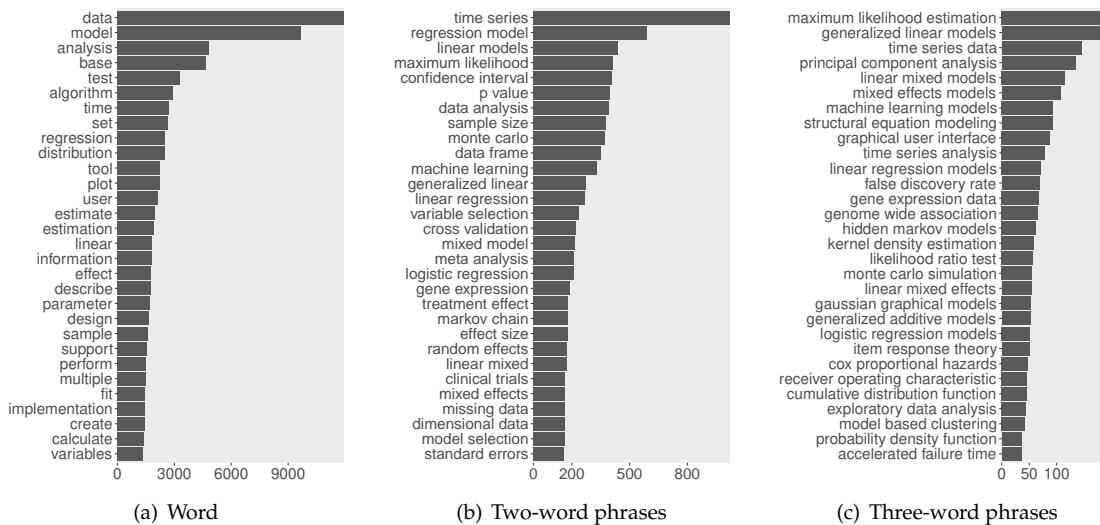


Figure 2: Top 30 frequently used words and phrases in the CRAN package descriptions.

association study (51 times), *single cell RNA sequencing* (43 times), *generalized linear mixed model* (36 times), *cox proportional hazards model* (30 times), and *random effects meta analysis* (15 times). Moreover, we found two informative six-word phrases that represent the journals that were commonly mentioned in the package descriptions: *Journal of the American Statistical Association* (22 times) and *Journal of Computational and Graphical Statistics* (12 times).

2.3 Topic Modeling

To identify relevant thematic features of the CRAN R packages, we conducted topic modeling based on the package descriptions using the [topicmodels](#) (Grün and Hornik, 2011) package. We used the Latent Dirichlet allocation (LDA; Blei et al., 2003) method to reveal the latent structures in R package descriptions. LDA assumes that each document (i.e., the description document of each package) is a mixture of topics, and each topic consists of a mixture of words (Silge and Robinson, 2017).

To fit the model, we first determined the number of topics. For each specified number of topics, ranging from 2 to 30, we built the LDA models with 100 different random number seeds to improve the robustness of results. In each replication, the 5-fold cross-validation technique was employed. We assessed the performance of the LDA models by calculating their perplexity under different numbers of topics. A low perplexity score indicates better model performance (Bao and Datta, 2014). Results suggest that the best fit to the data was a model with 19 topics, followed by the 21- and 20-topic models (Figure 3). In the 100 replications of analysis, the 19-topic model was selected as the best model most frequently, followed by the 21-, 20-, 18-, and 17-topic models. By exploring the meaning of each topic in each model, we found that the model with 20 topics offered the best interpretability. The 21-topic model identified three topics which are hard to distinguish because they were all about mixed models, Bayesian analysis, linear regression, and psychometrics. Compared to the 19-topic model, the 20-topic model was easier to interpret because it clearly separated the topics of *Generalized Linear Modeling and Mixed Models*, and *Psychometrics*. We also examined the inflection points in the performance curve (Figure 3). We found that the 9-topic and 10-topic models could not clearly differentiate many topics, resulting in many similar topics with overlapping components. Therefore, we finally adopted the 20-topic model.

The R package [ctv](#) (CRAN Task Views) introduces the relevant packages of 42 topics (e.g., Bayesian Inference, Chemometrics, Econometrics). We further compared the identified 20 topics with the topics classified in the CRAN Task Views and highlighted their connections in Table 1. In detail, the main topics recovered from the 20-topic LDA model, in no particular order, were listed below.

- 1. Supporting Packages** Keywords of this topic include *data*, *book*, *support*, *ISBN(International Standard Book Number)*, *tool*, and *publication*. This topic pertains to packages that offer supporting functions for other packages (e.g., [iemisc](#)) and/or provide supplementary materials (e.g., datasets) for books, courses, and other packages (e.g., [EnvStats](#), [AER](#), [uwo4419](#), [ACSWR](#), [mosaic](#)).

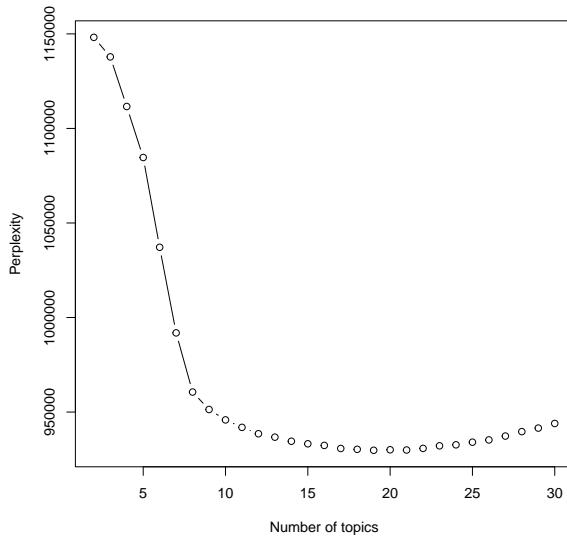


Figure 3: Mean perplexities of topic models with different number of topics.

2. Causal Inference This topic deals with causal inference, and its keywords include *effect*, *treatment*, *causal*, *outcome*, *propensity*, and *intervention*. Packages of this topic are related to the CRAN Task View of [Causal Inference](#). Example packages are [CBPS](#) which contains methods for moments-based propensity score estimation, [BCEE](#) for Bayesian Causal Effect Estimation, and [CausalGAM](#) for estimation of causal effects with generalized additive models.

3. Numerical Mathematics Keywords of this topic include *matrix*, *sparse*, *covariance*, *correlation*, *vector*, *row*, *column*, and *decomposition*. It is related to a CRAN Task View of [Numerical Mathematics](#). Packages identified by this topic include [Matrix](#), a core package for sparse and dense matrix classes and methods, [PRIMME](#), an R interface to a C library for computing eigenvalues, [SparseM](#), a package with sparse linear algebra functions.

4. Classification This topic is about classification and includes keywords such as *variable*, *split*, *random*, *forest*, *tree*, *category*, and *feature*. Example packages include [randomForest](#) for classification based on a forest of trees, [tree](#) containing functions for classification and regression trees, [rpart](#) for recursive partitioning and regression trees, and [C50](#) for fitting C5.0 classification trees and rule-based models. Some of the packages identified by this topic can be found in the CRAN Task View of [Machine Learning and Statistical Learning](#).

5. Regression Analysis and Regularization This topic is about regression analysis and regularization, with words such as *linear*, *regression*, *model*, *L1*, *lasso*, *ridge*, *shrinkage*, *procedure*, and *regularization* appearing with high probability. It is related to the packages for regularization methods and regression models (e.g., [glmnet](#) for lasso and elastic-net regularized generalized linear models (GLMs), [penalized](#) for applying ridge and lasso regularization in GLMs and Cox proportional hazards models, and [bravo](#) for Bayesian variable selection with ultra-high dimensional linear regression models.).

6. Genetics Keywords of this topic include terms related to genetics, such as *gene*, *RNA (Ribonucleic Acid)*, *DNA (DeoxyriboNucleic Acid)*, *sequence*, *cell*, and *phenotype*. This topic identifies the packages for analyzing biological data, especially genetic data (e.g., [jetset](#), [Seurat](#), [scBio](#)).

7. Datasets High probability words of this topic include *data*, *survey*, *questionnaire*, *sample*, *collection*, *census*, and *report*. This topic is mainly about a special kind of packages for providing datasets (especially survey data) instead of functions for statistical methods. For example, [spiR](#) is for social progress index data, [PakPMICS2018](#) is for survey data of a specific project conducted from 2017 to 2018, [USpopcenters](#) is about united stats centers of population data.

Table 1: Relevant CRAN Task Views of each topic identified in the topic model.

Topic	CRAN Task Views
1. Supporting Packages	Teaching Statistics
2. Causal Inference	Causal Inference
3. Numerical Mathematics	Numerical Mathematics
4. Classification	Machine Learning & Statistical Learning
5. Regression Analysis and Regularization	Machine Learning & Statistical Learning
6. Genetics	Statistical Genetics
7. Datasets	Databases with R
8. Cluster Analysis and Network Analysis	Cluster Analysis & Finite Mixture Models, gRaphical Models in R
9. Machine Learning	Machine Learning & Statistical Learning
10. NHST and Multiple Comparison	-
11. Probability Distributions and Bayesian Analysis	Probability Distributions, Bayesian Inference
12. Color Patterns and R Objects	-
13. Phylogenetics	Phylogenetics
14. Time Series Analysis	Time Series Analysis
15. Data Import, Export, and Wrangling	-
16. Computational Efficiency	Optimization and Mathematical Programming, High-Performance and Parallel Computing with R
17. Experimental Design and Clinical Trials	Clinical Trial Design, Monitoring, and Analysis, Design of Experiments & Analysis of Experimental Data
18. Data Visualization, Result Presentation, and Interactive Web Applications	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization, Reproducible Research Web Technologies and Services
19. Generalized Linear Modeling and Mixed Models	Mixed, Multilevel, and Hierarchical Models in R
20. Psychometrics	Psychometric Models and Methods

Note. NHST = Null Hypothesis Significance Testing.

8. Cluster Analysis and Network Analysis Keywords of this topic are related to two sub-topics. First, *network*, *gaussian*, *graphical*, *node*, *edge*, and *igraph* are commonly used in network analysis. Example packages include *igraph* and *sna*. Second, *cluster*, *class*, *kernel* are relevant to Cluster Analysis. Example packages include *cluster*, *kml* for implementing k-means clustering on longitudinal data, *flexclust* for providing flexible cluster algorithms. Cluster analysis and network analysis were identified in one topic perhaps because there are some packages for clustering based on correlation matrices and network modeling (e.g., *DirectedClustering*, *linkcomm*, *clusterGeneration*).

9. Machine Learning With keywords such as *algorithm*, *machine*, *learning*, *training*, and *prediction*, this topic is mainly about machine learning. The CRAN Task View of *Machine Learning and Statistical Learning* lists and classifies the packages of this topic, for example, packages for neural networks, deep learning (e.g., *RcppDL*, *deepnet*). Some packages are related to both this topic and the topic of regularization (e.g., *bmrm*). Courses of this topic are also very common in data science programs, including *Neural Networks and Deep Learning*, and *Machine Learning and Big Data* (Zhang and Zhang, 2021).

10. Null Hypothesis Significance Testing (NHST) and Multiple Comparison This topic encompasses NHST and Multiple Comparison, and its keywords include *test*, *hypothesis*, *null*, *multiple*, *comparison*, and *significance*. Relevant packages include *onewaytests* for NHST, *PMCMRplus* and *conover.test* for multiple comparisons among multiple groups.

11. Probability Distributions and Bayesian Analysis This topic centers around the word *distribution* and includes other relevant terms such as *probability*, *binomial*, *poisson*, *density*, *Bayesian*, *MCMC* (*Markov chain Monte Carlo*), *chain*, *prior*, *posterior*, and *sampler*. Probability and Bayesian analysis were identified in one topic because they share the main keyword *distribution* (i.e., probability distribution, prior distribution, and posterior distribution). Example packages include *extraDistr* for various univariate and multivariate distributions, *gnorm* for generalized normal or exponential power distributions, *bayesmix* for Bayesian mixture models, and *mcmc*. The CRAN Task Views of *Probability Distributions* and *Bayesian* provided a detailed summary of packages related to this topic.

12. Color Patterns and R Objects Keyword of this topic include *class*, *object*, *s4*, and *r6* which are related to R objects, and *color*, *png*, *image*, and *palettes* which are about visualization. For the R objects, example packages are *R6*, *R6S3*, and *fastdigest*. For the visualization, most packages identified by this topic are designed to provide color patters for plotting (e.g., *colormap*, *ggpattern*). Package

`ggplot2` is also identified by this topic, but its relationship with the topic 18 (Data Visualization, Result Presentation, and Interactive Web Applications) is higher than its relationship with this topic.

13. Phylogenetics This topic is mainly about phylogenetics, with high probability words including *phylogenetic, specie, signal, taxonomic, animal, pedigree, and biodiversity*. Packages related to this topic include `phytools` which provides functions for phylogenetic analysis, `geiger` for fitting macroevolutionary models to phylogenetic trees, and `phylosignal` for exploring the phylogenetic signal. The CRAN Task View `Phylogenetics` also summarized packages related to this topic.

14. Time Series Analysis This topic is mainly about time series analysis. The high probability words associated with this topic include *time, series, change, dynamic, forecast, growth, trend, and lag*. There is a CRAN Task View of this topic that introduces many packages for handling time series data, for example, `tseries` for time series analysis, and `forecast` that provides forecasting functions for time series models.

15. Data Import, Export, and Wrangling This topic includes keywords such as *read, load, import, write, convert, and create*, which are commonly-used verbs for naming R functions for data import, export and wrangling. Example packages include `asciiSetupReader`, which can read fixed-width ASCII data files, `adfExplorer`, which can import and export Amiga disk files, and `dplyr`, "a grammar of data manipulation."

16. Computational Efficiency This topic is related to computational efficiency, with keywords including *run, C++, fast, efficient, Rcpp, efficient, and parallel*. Many packages associated with this topic call compiled C++ code from R to improve performance (e.g., `Rcpp`), and provide utilities for parallel computation (e.g., `future.callr, foreach, doParallel`).

17. Experimental Design and Clinical Trials This topic focuses on the design and data analysis of experiments and clinical trials. High probability words include *design, treatment, control, trial, clinical, sample, size, endpoint, stage, dose, and experimental*. The CRAN Task Views of `Clinical Trial Design` and `Experimental Design` summarize two lists of R packages concerning this topic. Here we listed some example packages such as `TrialSize` for sample size determination in clinical research, and `visit` for phase I dose-escalation study design.

18. Data Visualization, Result Presentation, and Interactive Web Applications Packages and keywords of this topic can be classified into three sub-topics. First, keywords such as `ggplot2, color, graph, image, and plot` are related to data visualization. Example packages include `ggplot2` and `lattice`. The second sub-topic is about presenting results, with keywords such as `table, chart, and figure`. Other keywords include `format, html, document, markdown, rmarkdown, and latex` which are mainly about R markdown tools for generating dynamic reports. Example packages are R markdown-related tools including `knitr` and `bookdown`, and formatting tools such as `styler` and `pander`. The third sub-topic relates to using R to build interactive web applications. Words relevant to this sub-topic include `shiny, widget, web, and XML`, and example packages include `shiny` and `webshot`.

In addition, there are many packages related to more than one sub-topic. For example, `ANOVAShiny` was built based on `rmarkdown` and `shiny`, and `Factoshiny` was developed with `shiny` and `ggplot2` for conducting factorial analysis and drawing graphs with a shiny application. There are also corresponding lists of this topic on CRAN Task Views, and courses in data science programs (e.g., *Data Visualization, Data Presentation and Visualization with R*; [Zhang and Zhang, 2021](#)).

19. Generalized Linear Models and Mixed Models This topic is related to generalized linear modeling, with keywords such as *model, regression, linear, fit, mixed, logistic, and GLM (Generalized Linear Modeling)*. Packages associated with this topic include `dglm` for building double generalized linear models, `brms` for Bayesian Regression Models, `nlme` for linear mixed models (LMMs), and `mbest` for large nested LMMs.

20. Psychometrics The high probability words related to this topic include *factor, item, response, latent, IRT (Item Response Theory), and choice*. There is a detailed list on the CRAN Task Views about this topic. Here we listed some example packages: `mirt` for multidimensional item response theory, `difR` for detecting differential item functioning, and `CDM` for cognitive diagnosis models.

3 Identifying Key Packages and Key Package Developers

To investigate the key packages contributing to the R ecosystem and the notable authors who significantly support the R community, we conducted network analysis. We extracted package dependency and authorship information from CRAN and used `cranly` (Kosmidis, 2019) and `igraph` (Csárdi et al., 2006) to build three networks: 1) a package dependency network, 2) an author collaboration network, and 3) a bipartite network of packages and authors. Influential nodes were identified in these three networks. We first introduce the one-mode network (i.e., network with one type of node) of packages and authors below.

The R package dependency network was built based on the CRAN R packages and their dependent packages, including the base and recommended R packages that are included in the default installation of R. It is a directed and weighted network, and a sub-graph is presented in Figure 4(a). Specifically, the arrow from package A to package B indicates that A depends on, imports, suggests, link to, or is enhanced by B. Table 2 presents the detailed definitions and frequencies of these relationships (R Core Team, 2021). We assigned the weights of edges as 5, 4, 3, 2, and 1 for the relationships of *depends*, *imports*, *suggests*, *links to*, and *enhances*, respectively.

We tested a different weight scheme (3, 2, 1, 1, 1) and found that the results were robust, with correlations of influence scores exceeding .95. The top packages identified were also similar. Hence, we report results based on the (5, 4, 3, 2, 1) weight scheme in this paper.

For the undirected author collaboration network (e.g., Figure 4(b)), the edge between two nodes indicates that they have co-authored in at least one package. The weights of edges denote the number of packages co-authored by the corresponding two authors.

Table 2: Types of Package Dependencies.

Dependency	Definition	Frequency
Depends	B will be attached when attaching A	10523
Imports	The namespace of B will be imported when attaching A	87328
Suggests	B is used on in the examples, tests, or vignettes of A	54991
Links to	A uses the header files in B to compile its C or C++ code	5153
Enhances	B provides methods for classes in A, or helps handling objects in A	558

3.1 Measures of Influence

We focused on the following commonly used measures for evaluating node (a package or an author) influence in the one-mode network (Jain and Sinha, 2020; Morone and Makse, 2015; Salavaty et al., 2020; Wang et al., 2017):

- Local structure information: Degree;
- Global structure information: Betweenness (Freeman, 1978);
- Algorithm based on random walk: Eigenvector centrality (Bonacich, 1972) and PageRank (Page et al., 1999);

Degree and In-degree Degree is the number of direct connections between nodes. For the directed package dependency network, in-degree indicates how many packages this package enhances, or is depended on, imported, suggested, or linked by. For the undirected author collaboration network, degree is the number of collaborators.

Betweenness Centrality Betweenness quantifies how important a node is as a mediator between two other nodes. Given a network, the betweenness centrality of node i can be calculated as (Freeman, 1978):

$$C_{Bet}(i) = \sum_{s \neq i \neq t} \frac{d_{s,t}(i)}{d_{s,t}} \quad (1)$$

where $d_{s,t}$ denotes the number of shortest paths from node s to node t , and $d_{s,t}(i)$ is the number of shortest paths between node s and node t going through node i .

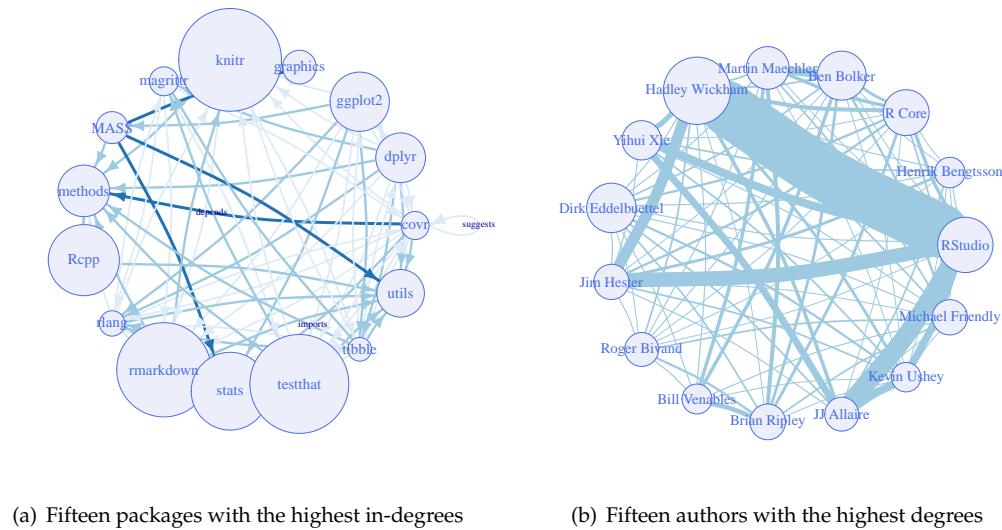


Figure 4: Networks of a subset of nodes with the highest degrees.

Note. The sizes of nodes represent the in-degree / degree of nodes. In the author collaboration network, the edge width reflects the weight. In the package dependency network, the color indicates the weight. "The dark edges represent *depends*, medium dark edges represent *imports*, the lightest edges represent *suggests*. Relationships of *linking to* and *enhances* are not included in this sub-graph."

Eigenvector Centrality A node with a small degree may have high eigenvector centrality (Bonacich, 1972) if it is connected with important nodes. In other words, this index considers the importance of neighbors when evaluating the centrality of a node.

Let $A = (a_{i,j})$ be the adjacency matrix of a graph, where $a_{i,j}$ represents the connection between node i and j . The eigenvector centrality of node i is given by (Bonacich, 2007):

$$C_{Eig}(i) = \frac{1}{\lambda} \sum_j a_{i,j} C_{Eig}(j) \quad (2)$$

where λ is a non-zero eigenvalue. This can also be expressed in the matrix form:

$$\lambda C_{Eig} = C_{Eig} A \quad (3)$$

where C_{Eig} is the eigenvector for the adjacency matrix A given eigenvalue λ . Bonacich (1972) suggested to choose the largest value of λ for measuring centralities.

PageRank PageRank is a variant of eigenvector centrality. It is an algorithm developed by Google to rank web pages (Page et al., 1999), and is primarily used for directed networks. Given a weighted network, it can be calculated as:

$$PR(i) = \alpha \sum_j W_{j,i} PR(j) + (1 - \alpha) PR^0(i) \quad (4)$$

where $PR(i)$ and $PR(j)$ are the PageRank of node i and j , respectively. Besides, $PR^0(i)$ is typically set as $\frac{1}{N}$ where N is the number of nodes, α is the damping factor assigned to the random walk, and $W_{j,i}$ is the normalized edge weight from node j to i :

$$W_{j,i} = \frac{w_{j,i}}{\sum_t w_{j,t}} \quad (5)$$

where $w_{j,i}$ is the edge weight from node j to node i .

3.2 Bipartite Network

A bipartite network of authors and packages was built using the authorship information. The weight of an edge was assigned as 3 if the author is the maintainer of the package, and 1 otherwise (Figure 5).

Unlike the one-mode networks, which focus on either packages or authors, the bipartite network captures both the relationships between authors and packages and the collaborations among authors. As shown in Figure 6, the same author relationship configuration could be represented by two different bipartite networks. Therefore, we employed the bipartite network to gain more insights into the influential authors using the BiRank statistic.

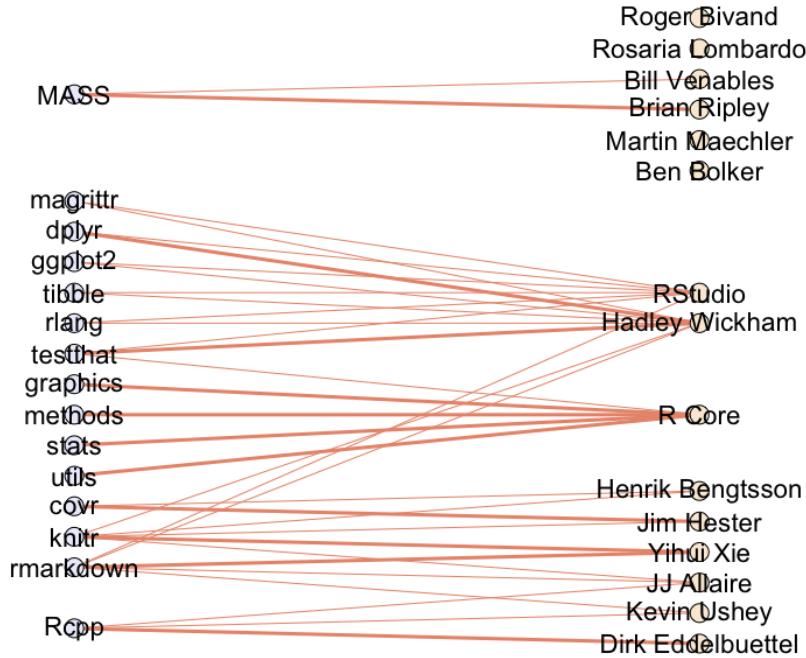


Figure 5: A Bipartite network of selected R packages and authors.

Note. Nodes on the left are 15 packages with the highest in-degrees in the package dependency network, nodes on the right are 15 authors with the highest degrees in the author collaboration network. A bold line indicates that the corresponding author is the maintainer of the corresponding package.

BiRank Aronson et al. (2020) compared different centrality measures for the bipartite network and recommended the BiRank and CoHITS indexes. For our network, the BiRank index exhibited a high correlation with the CoHITS index ($r = .860$). The top 50 authors identified by these two measures are the same, with only minor variations in their ranking. Consequently, we only reported the results of the BiRank index (He et al., 2016). Similar to PageRank, the BiRank index assumes that the ranking of nodes should be related to the ranking of the neighbors. Namely, an author would be ranked high if his or her neighbor packages are important.

Let $A = (a_{i,j})$ be the adjacency matrix of a bipartite graph with two types of nodes $u_i (i = 1, \dots, I)$ and $p_j (j = 1, \dots, J)$, the BiRank value for node u_i is given by:

$$BR(u_i) = \alpha \sum_j a_{i,j} \frac{BR(p_j)}{\sqrt{C_{wDeg}(u_i)} \sqrt{C_{wDeg}(p_j)}} + (1 - \alpha) BR^0(u_i) \quad (6)$$

where $C_{wDeg}(u_i)$ and $C_{wDeg}(p_j)$ are the weighted degrees of node u_i and p_j . Similar to the PageRank index, α is the damping factor, and $BR^0(u_i)$ is the prior belief of the importance of node u_i .

3.3 Results

The centrality measures of these three networks were calculated using the R package **igraph** (Csárdi et al., 2006), **influential** (Salavaty et al., 2020), and **birankr** (Aronson and Yang, 2020).

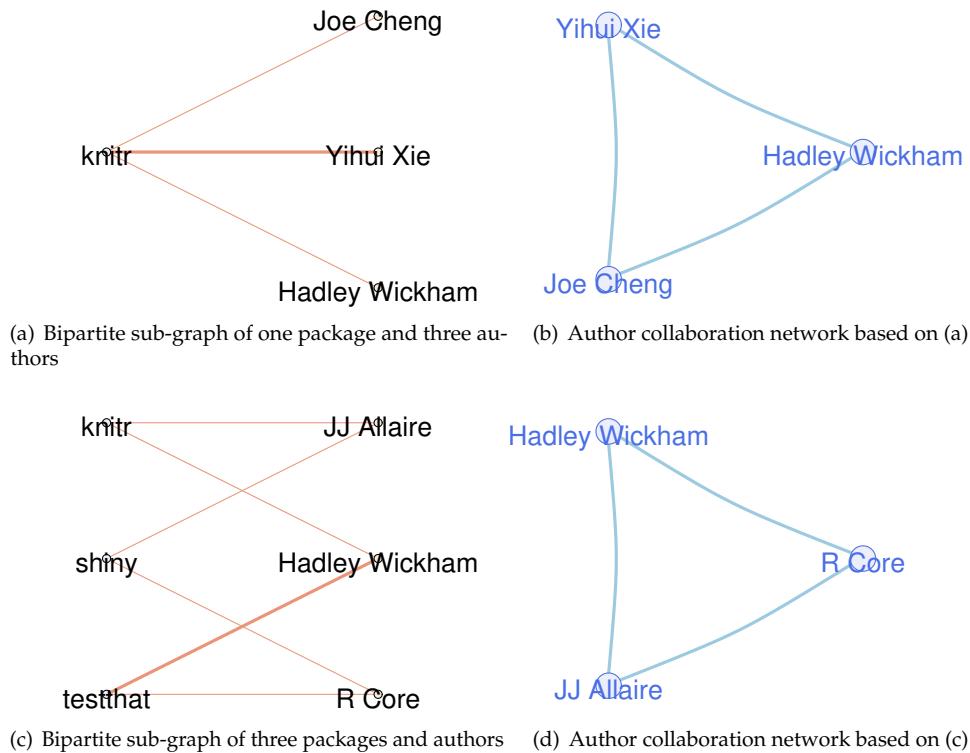


Figure 6: Relationships between the bipartite network and the author collaboration network.

Package Dependency Network Table A1 presents the top 50 packages according to various centrality measures. Results show that the number of packages depending (i.e., depends, imports, suggests, linking to, or reverse enhances) on `knitr` is the greatest. This is probably because `knitr` package can help generate R package vignettes, and is therefore suggested by numerous packages. In other words, these packages are using `knitr` in developing and publishing the package, instead of extending its functionality. Additionally, `knitr` is a comprehensive package related to the topic of *Data Visualization, Result Presentation, and Interactive Web Applications*. It can help generate dynamic reports. Many packages of this topic were built based on `knitr`, such as `bookdown` for writing books and technical documents. The package with the highest betweenness (`ggplot2`) also suggests `knitr`. Compared to `knitr`, `ggplot2` demonstrates greater influence as a mediation package. It suggests and imports many important R packages, and is also depended by numerous packages. Unlike the degree and betweenness centralities, the PageRank index by considering the importance of neighbor nodes prioritizes some packages with relatively low connections. For example, the ranking of `tools` (a base R package) is 55 using in-degree, while 8 using PageRank. This is because `tools` is imported by some influential packages such as `rmarkdown` and `shiny`.

Among these influential packages in Table A1, 54 packages are ranked as top 50 by at least two indexes (Table A2), including 10 out of the 29 base and recommended R packages.

Apart from the base and recommended R packages, other important CRAN packages concentrate mainly on three topics: *Data Import, Export, and Wrangling* (e.g., `dplyr`, `tibble`, `tidyverse`, `readr`), *Data Visualization, Result Presentation, and Interactive Web Applications* (e.g., `knitr`, `ggplot2`, `rmarkdown`, and *Supporting Packages* (e.g., `testthat`, `purrr`)). Other packages are related to topics like *Computational Efficiency* (e.g., `Rcpp`, `RcppArmadillo`), *Probability Distributions* (e.g., `mvtorm`), *Bayesian Estimation and Network Analysis* (e.g., `igraph`), and *Access to Web and Services* (e.g., `httr`). These packages are ranked as top influential packages, likely because they provide basic and comprehensive functions for important topics. A lot of packages have been developed for specific goals based on them. For example, `ggplot2` is a well-known package for data visualization. Many packages were developed based on `ggplot2` to enhance its visualization functionality, like `ggROC` specifically for plotting ROC curve, and `ggbreak` for setting axis breaks.

We also investigated the relationships between the influence scores and the number of downloads of packages. The downloads of CRAN packages on the RStudio mirror from 2021-11-01 to 2022-10-31 were obtained using the `cranlogs` (Csárdi, 2019) package. Note that the base and recommended R packages were not included when measuring downloads since they are part of R installation. The

correlations of the downloads and the five indexes range from .337 – .465.

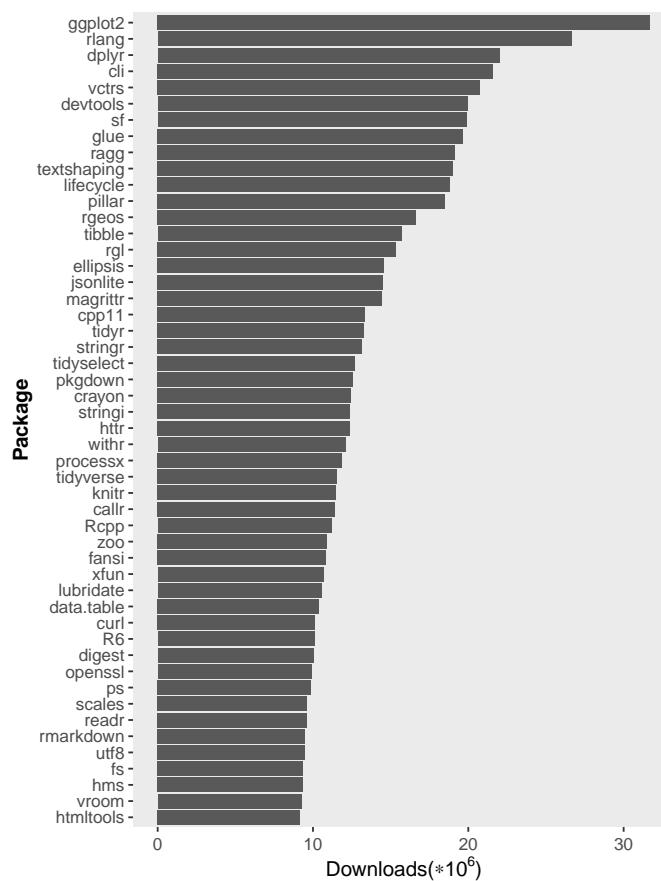


Figure 7: Top 50 packages downloaded on the RStudio mirror from 2021-11-01 to 2022-10-31.

Author Collaboration Network Table A3 lists the top 50 authors ranked by various indexes. RStudio is identified as the most influential author according to eigenvector and PageRank. Hadley Wickham and R Core demonstrated greatest influence based on degree and betweenness, respectively. Table A4 shows that 46 authors are ranked among the top 50 by at least two indexes, including organizations such as R Core, RStudio, and Google Inc., as well as individual scientists affiliated with these organizations. Figure 5 reveals that, unlike the base packages, most organizational authors of CRAN packages are not maintainers or creators. Instead, organizations such as RStudio acts as copyright holders and funders, and the R Core team often works as a contributor.

The results also suggest that different indexes prioritize different aspects of influence. Authors with limited but influential collaborators may not rank among the top 50 based on the degree centrality, but can be revealed using the eigenvector centrality and PageRank (Table A3).

Table A4 presents the top downloaded packages of influential authors. Many packages have been identified as the influential packages listed in Table A1. Figure 8 shows the collaboration heatmap among these authors, revealing intensive collaboration among influential organizations and authors. The heatmap shows three main clusters of author collaboration. Most authors at the upper left corner of the heatmap (e.g., Dirk Eddelbuettel) have the same two packages **DescTools** (an R package for descriptive statistics) and **fortunes** (an R package of collected wisdom from the R community). Their co-authored packages are mainly about two main topics on *Data Import, Export, and Wrangling* (e.g., **data.table**, **MASS**) and *Data Visualization* (e.g., **rgl**, **plotrix**). Authors in the middle of the figure are clustered together because they co-authored the **broom** package, a package for converting objects into tidy tibbles. Authors in the lower-right corner of the matrix co-authored many packages related to the *Data Visualization, Result Presentation, and Interactive Web Applications* topic (e.g., **knitr**, **bookdown**, **shiny**), many of whom are from RStudio. These results align with the finding in the package dependency network that many influential packages are related to the *Data Import, Export, and Wrangling* and *Data Visualization, Result Presentation, and Interactive Web Applications* topics.

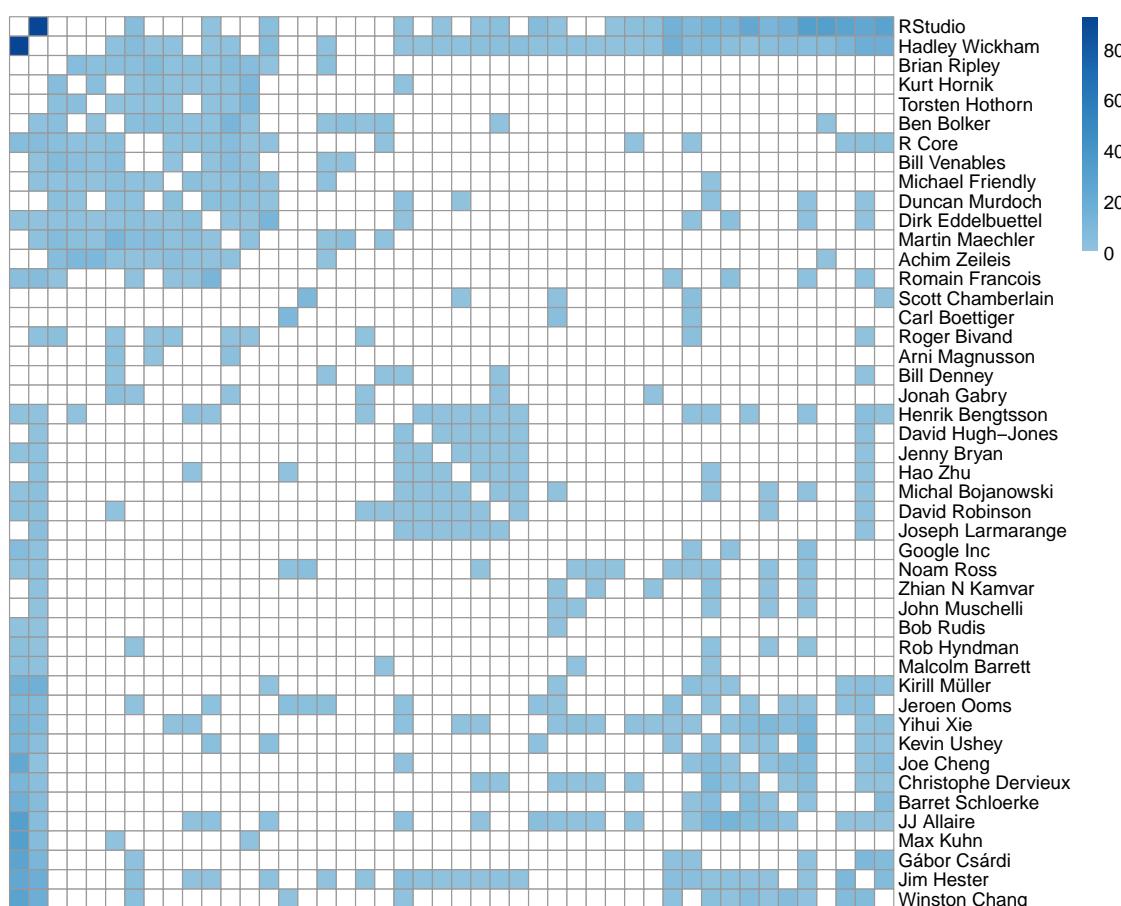


Figure 8: Collaboration heatmap of the influential authors identified in the collaboration network.
Note. Color denotes the number of co-authored packages between two authors.

Bipartite Network The BiRank values demonstrate small to medium correlations with the indexes used in the one-mode author collaboration network ($r = .179 - .427$). The top 50 influential authors and their ranking are different from the results of the author collaboration network. Twenty-three of the top 50 authors are not ranked as top 50 by any index in the collaboration network (Table A5). Compared to the influential authors in the collaboration network, these 23 authors' packages have relatively few number of authors. But the number of their packages is high.

The most influential author identified by BiRank is Dirk Eddelbuettel, followed by Hadley Wickham, RStudio, Stéphane Laurent, and Scott Chamberlain. By contrast, in the one-mode network, Stéphane Laurent is not in the top 50 author lists based on degree, betweenness, and eigenvector. Scott Chamberlain is also not in the top 50 author lists based on degree and eigenvector. This is because many of their packages are solo-authored (e.g., `crul`). These results suggested that the bipartite network can identify productive and influential authors even without a high number of collaborators.

4 Discussion

The growing usage of R in data science is tightly associated with the increasing growth of the R ecosystem. With the rich amount of information on CRAN, this paper shows the main trends among the R developers and their packages.

Based on the package descriptions, we investigated the popular topics in the R ecosystem. Results identified a wide range of topics including various statistical methods (e.g., Cluster Analysis, Machine Learning, Bayesian Estimation, and Network Analysis) across different fields (e.g., Genetics, Environmetrics, Multivariate Analysis, and Psychometrics). Moreover, there were also packages for providing data sets, helping access to websites and online services, and improving computational efficiency. Some of these 19 topics are similar to the topics identified in the data science curricula (Zhang and Zhang, 2021) including *Machine Learning* and *Data Visualization, Result Presentation, and Interactive Web Applications*.

Using network analysis, we explored the influential packages and authors based on package dependencies and authorships. The results highlighted the crucial role of the base and recommended R packages in developing R packages, with 12 out of the 29 base and recommended packages ranked in the top 50 by at least two indexes. In addition to the packages shipped alongside base R, we also identified some important CRAN packages that support the development of R in various topics. Many influential packages were related to *Data Visualization, Result Presentation, and Interactive Web Applications* and *Data Import, Export, and Wrangling* topics. These results were similar to the finding that influential authors co-developed many important packages of these two topics.

Our findings also indicated that the contributors of R, developing from the R Core team, now have become a worldwide community, and some of them are organizations. Many of these influential authors are from RStudio or the R Core team and lead the development across many topics. For example, Yihui Xie, the creator of `knitr` and many R markdown-related packages, facilitates the development of reproducible research. While some of these influential contributors are from central positions in the author collaboration network, we also identified many productive authors with relatively few collaborators through the bipartite network of packages and authors.

4.1 Limitations and Future Directions

There are a few of limitations in the current work. First, we investigated the influential packages from the perspective of the developer community rather than the user community. To illustrate, we focused on identifying which packages are used by other packages (i.e., in the package dependency network). However, packages with high influence for developers may differ from those for the user community. Future studies are needed to explore important packages based on the usage data (e.g., citations, number of times mentioned on Twitter). Second, package dependency relationships cannot distinguish between dependencies required for publishing the package, such as vignette generation, and those needed for developing utilities. To better understand the relationships of packages based on their functionality, future study can explore the similarity between packages based on text analysis of reference manuals or vignettes. Third, we only included the R packages on CRAN or embedded in the R source code. There has already been a trend of releasing packages on GitHub and R-universe (a personal R package repository). Numerous GitHub packages and their information (e.g., contributors, commits, stars, forks) could be a rich resource for future research to understand the development and importance of R packages. Forth, as suggested by a referee, future studies can compare R and other software communities (e.g., Python). Fifth, this paper was limited by the timing of data extraction and did not include the state of the R community in the past. Future research are suggested to conduct longitudinal analysis of the R community to understand how it developed over time.

This paper also provides directions for the development of an R package recommendation system. As Silge et al. (2018) suggested, with the growth of the R ecosystems, it becomes difficult for the users to search for specific packages on CRAN. The flat organization of R packages, as well as the sorting system based on name or the date of publication, pose a big challenge for searching. It has been proposed to add keywords or tags for packages to help organize and search packages (Silge et al., 2018). Based on the results of the current paper, future research is suggested to develop a package recommendation system using the information on CRAN. Specifically, the topic modeling can help identify packages related to different topics and could also be an effective way for automatically tagging. Network analysis can also help understand the relationships among packages. Methods based on network analysis such as latent space modeling may help quantify the closeness of two packages and provide recommendations.

4.2 Concluding Remarks

This paper depicts a general picture of the R developer community based on information on CRAN. Results identified the popular topics of the R ecosystem. Investigation of influential packages and authors also contributes to the understanding of the development of the R community. It is the efforts from these experts with various backgrounds that lead to the prosperity of the R ecosystem. Moreover, results of the current study highlight the direction for developing a package recommendation system.

4.3 Acknowledgment

This work is supported by a grant from the Department of Education (R305D210023) and the Notre Dame International at the University of Notre Dame. However, the contents of the study do not necessarily represent the policy of the Department of Education, and you should not assume endorsement by the Federal Government.

1 Appendix

Table A1: Top 50 packages identified based on different network statistics of the package dependency network.

Rank	In-degree	Betweenness	Eigenvector	PageRank
1	knitr	ggplot2	knitr	testthat
2	testthat	knitr	rmarkdown	utils
3	rmarkdown	broom	testthat	methods
4	stats	dplyr	stats	stats
5	Rcpp	emmeans	ggplot2	knitr
6	ggplot2	testthat	dplyr	patchSyncTeX
7	methods	shiny	methods	rmarkdown
8	dplyr	stats	utils	tools
9	utils	rmarkdown	Rcpp	covr
10	graphics	survival	rlang	stringr
11	MASS	bayestestR	magrittr	graphics
12	magrittr	sf	tibble	mapmisc
13	covr	gap	tidyR	Rcpp
14	rlang	MASS	graphics	tis
15	tibble	targets	covr	grDevices
16	tidyR	caret	stringr	rlang
17	stringr	multcomp	purrr	MASS
18	grDevices	Hmisc	MASS	ggplot2
19	purrr	texreg	grDevices	magrittr
20	parallel	insight	data.table	withr
21	Matrix	parameters	jsonlite	jsonlite
22	data.table	enrichwith	Matrix	parallel
23	jsonlite	cops	shiny	htmltools
24	RcppArmadillo	mlr	parallel	scales
25	shiny	zoo	httr	fastcluster
26	httr	earth	glue	enrichwith
27	mvtnorm	tibble	scales	cops
28	survival	AER	sf	QuasiSeq
29	foreach	lme4	readr	tibble
30	scales	jsonlite	lubridate	dplyr
31	plyr	robustbase	igraph	lattice
32	igraph	plotmo	reshape2	plyr
33	reshape2	quantreg	withr	R6
34	lubridate	effects	gridExtra	vctrs
35	doParallel	rgl	foreach	reshape
36	grid	nnet	cli	glue
37	sp	nlme	tidyselect	digest
38	gridExtra	sp	plyr	tinytest
39	readr	doParallel	xml2	RUnit
40	spelling	surveillance	sp	httr
41	lattice	mice	plotly	Matrix
42	glue	Matrix	grid	xml2
43	RColorBrewer	data.tree	survival	shiny
44	sf	car	lme4	curl
45	xml2	spelling	vctrs	rstudioapi
46	raster	metafor	ifecycle	cli
47	zoo	marginaleffects	broom	yaml
48	markdown	gtools	mvtnorm	survival
49	R6	partykit	RcppArmadillo	markdown
50	glmnet	hunspell	doParallel	htmlwidgets

Table A2: Packages identified by at least two indexes.

Package	Title	Maintainer
knitr	A General-Purpose Package for Dynamic Report Generation in R	Yihui Xie
ggplot2	Create Elegant Data Visualisations Using the Grammar of Graphics	Thomas Lin Pedersen
testthat	Unit Testing for R	Hadley Wickham
rmarkdown	Dynamic Documents for R	Yihui Xie
utils*	The R Utils Package	R Core
broom	Convert Statistical Objects into Tidy Tibbles	Simon Couch
methods*	Formal Methods and Classes	R Core
stats*	The R Stats Package	R Core
dplyr	A Grammar of Data Manipulation	Hadley Wickham
Rcpp	Seamless R and C++ Integration	Dirk Eddelbuettel
shiny	Web Application Framework for R	Winston Chang
covr	Test Coverage for Packages	Jim Hester
grDevices*	The R Graphics Devices and Support for Colours and Fonts	R Core
survival	Survival Analysis	Terry M Therneau
rlang	Functions for Base Types and Core R and 'Tidyverse' Features	Lionel Henry
stringr	Simple, Consistent Wrappers for Common String Operations	Hadley Wickham
MASS*	Support Functions and Datasets for Venables and Ripley's MASS	Brian Ripley
magrittr	A Forward-Pipe Operator for R	Lionel Henry
sf	Simple Features for R	Edzer Pebesma
tibble	Simple Data Frames	Kirill Müller
tidyverse	Tidy Messy Data	Hadley Wickham
graphics*	The R Graphics Package	R Core
purrr	Functional Programming Tools	Lionel Henry
parallel*	Support for Parallel Computation	R Core
data.table	Extension of 'data.frame'	Matt Dowle
withr	Run Code 'With' Temporarily Modified Global State	Lionel Henry
Matrix*	Sparse and Dense Matrix Classes and Methods	Martin Maechler
jsonlite	A Simple and Robust JSON Parser and Generator for R	Jeroen Ooms
enrichwith	Methods to Enrich R Objects with Extra Components	Ioannis Kosmidis
cops	Cluster Optimized Proximity Scaling	Thomas Rusch
RcppArmadillo	Rcpp' Integration for the 'Armadillo' Templated Linear Algebra Library	Dirk Eddelbuettel
scales	Scale Functions for Visualization	Hadley Wickham
zoo	S3 Infrastructure for Regular and Irregular Time Series (Z's Ordered Observations)	Achim Zeileis
httr	Tools for Working with URLs and HTTP	Hadley Wickham
glue	Interpreted String Literals	Jennifer Bryan
mvtnorm	Multivariate Normal and t Distributions	Torsten Hothorn
foreach	Provides Foreach Looping Construct	Folashade Daniel
lme4	Linear Mixed-Effects Models using 'Eigen' and S4	Ben Bolker
readr	Read Rectangular Text Data	Jennifer Bryan
lubridate	Make Dealing with Dates a Little Easier	Vitalie Spinu
plyr	Tools for Splitting, Applying and Combining Data	Hadley Wickham
igraph	Network Analysis and Visualization	Tamás Nepusz
lattice*	Trellis Graphics for R	Deepayan Sarkar
reshape2	Flexibly Reshape Data: A Reboot of the Reshape Package	Hadley Wickham
R6	Encapsulated Classes with Reference Semantics	Winston Chang
gridExtra	Miscellaneous Functions for "Grid" Graphics	Baptiste Auguie
vctrs	Vector Helpers	Lionel Henry
doParallel	Foreach Parallel Adaptor for the 'parallel' Package	Folashade Daniel
grid*	The Grid Graphics Package	R Core
cli	Helpers for Developing Command Line Interfaces	Gábor Csárdi
sp	Classes and Methods for Spatial Data	Edzer Pebesma
xml2	Parse XML	Hadley Wickham
spelling	Tools for Spell Checking in R	Jeroen Ooms
markdown	Render Markdown with 'commonmark'	Yihui Xie

Note. *Base and recommended R packages are part of R source code.

Table A3: Top 50 authors identified based on different network statistics of the author collaboration network.

Rank	Degree	Betweenness	Eigenvector	PageRank
1	Hadley Wickham	R Core	RStudio	RStudio
2	RStudio	Hadley Wickham	Hadley Wickham	Hadley Wickham
3	Dirk Eddelbuettel	Dirk Eddelbuettel	Jim Hester	R Core
4	Ben Bolker	Martin Maechler	JJ Allaire	Martin Maechler
5	R Core	RStudio	Winston Chang	Ben Bolker
6	Martin Maechler	Ben Bolker	Yihui Xie	Dirk Eddelbuettel
7	Yihui Xie	Kurt Hornik	Joe Cheng	Yihui Xie
8	Brian Ripley	Brian Ripley	Max Kuhn	Achim Zeileis
9	Michael Friendly	Roger Bivand	Gábor Csárdi	JJ Allaire
10	Jim Hester	Achim Zeileis	Lionel Henry	Kurt Hornik
11	Roger Bivand	Scott Chamberlain	Kirill Müller	Brian Ripley
12	JJ Allaire	Jeroen Ooms	Kevin Ushey	Roger Bivand
13	Henrik Bengtsson	Yihui Xie	Christophe Dervieux	Jim Hester
14	Bill Venables	Zhian N Kamvar	Barret Schloerke	Jeroen Ooms
15	Kevin Ushey	Michael Friendly	Jennifer Bryan	Scott Chamberlain
16	Achim Zeileis	John Muschelli	Jeroen Ooms	Michael Friendly
17	Kurt Hornik	Rob Hyndman	Carson Sievert	Zhian N Kamvar
18	Max Kuhn	Tyler Rinker	Javier Luraschi	Winston Chang
19	Romain Francois	Bill Denney	Romain Francois	Joe Cheng
20	Duncan Murdoch	John Wiseman	Daniel Falbel	Henrik Bengtsson
21	Zhian N Kamvar	Thomas Lumley	PBC	Bob Rudis
22	David Robinson	Jim Hester	R Core	Kevin Ushey
23	Hao Zhu	Sahir Bhatnagar	Henrik Bengtsson	Duncan Murdoch
24	Michał Bojanowski	Henrik Bengtsson	Ben Bolker	Max Kuhn
25	Joe Cheng	Toby Hocking	David Robinson	Bill Venables
26	Vilmantas Gegzna	Carl Boettiger	Michał Bojanowski	John Muschelli
27	Christophe Dervieux	Kevin Ushey	Thomas Lin Pedersen	Gábor Csárdi
28	Bill Denney	Kirill Müller	Davis Vaughan	Rob Hyndman
29	Adrian Baddeley	Bob Rudis	JooYoung Seo	Torsten Hothorn
30	Jeroen Ooms	Cleve Moler	Atsushi Yasumoto	Kirill Müller
31	Hong Ooi	JJ Allaire	Yixuan Qiu	Noam Ross
32	Noam Ross	Noam Ross	Joseph Larmarange	Romain Francois
33	David Hugh-Jones	Bill Venables	Dirk Eddelbuettel	Christophe Dervieux
34	Joseph Larmarange	Max Kuhn	Malcolm Barrett	Carl Boettiger
35	John Muschelli	Ben Goodrich	Noam Ross	David Robinson
36	Jonah Gabry	Jonah Gabry	Jeff Allen	Jonah Gabry
37	Malcolm Barrett	Christophe Dutang	Hao Zhu	Maëlle Salmon
38	Greg Snow	Torsten Hothorn	Michael Friendly	Michael Sumner
39	Jim Lemon	Winston Chang	Jenny Bryan	Barret Schloerke
40	Alessandro Gasparini	Kosuke Imai	Google Inc	Stéphane Laurent
41	Eduard Szoecs	Apache Foundation	David Hugh-Jones	Bill Denney
42	Tal Galili	Arni Magnusson	jQuery Foundation	Edzer Pebesma
43	Jenny Bryan	Romain Francois	Roger Bivand	Hao Zhu
44	Torsten Hothorn	Joe Cheng	Hiroaki Yutani	Indrajeet Patil
45	Matthieu Stigler	Duncan Murdoch	Martin Maechler	Michel Lang
46	Garrick Aden-Buie	Steven G Johnson	Alex Hayes	Michał Bojanowski
47	Dieter Menne	Yuan Tang	Mango Solutions	John Fox
48	Frank E Harrell Jr	Jacob Bien	Richard Iannone	Google Inc
49	Rolf Turner	Lampros Mouselimis	Simon Couch	Arni Magnusson
50	Michael Chirico	Yi Yang	Frederik Aust	Uwe Ligges

Table A4: Authors identified by at least two indexes.

Author	His or Her top downloaded packages
Hadley Wickham	ggplot2, rlang, dplyr, cli, vctrs
R Core	rgl, fansi, testthat, pkgload, car
RStudio	ggplot2, rlang, dplyr, cli, vctrs
Dirk Eddelbuettel	rgl, Rcpp, data.table, digest, nloptr
Jim Hester	devtools, glue, cpp11, withr, knitr
Ben Bolker	rgl, broom, lme4, car, gtools
Martin Maechler	lme4, car, bitops, MatrixModels, mvtnorm
JJ Allaire	knitr, Rcpp, rmarkdown, rstudioapi, shiny
Winston Chang	ggplot2, devtools, withr, processx, callr
Yihui Xie	knitr, xfun, rmarkdown, htmltools, yaml
Kurt Hornik	digest, colorspace, nloptr, e1071, tseries
Joe Cheng	knitr, rmarkdown, htmltools, bslib, sass
Brian Ripley	nloptr, car, bit, quantreg, MASS
Max Kuhn	broom, generics, caret, recipes, hardhat
Achim Zeileis	zoo, colorspace, car, lmtest, quantreg
Michael Friendly	knitr, car, maptools, DescTools, vcd
Roger Bivand	sf, rgeos, broom, sp, maptools
Gábor Csárdi	cli, rgl, crayon, processx, callr
Scott Chamberlain	plotly, crul, httpcode, jqr, bibtex
Kirill Müller	dplyr, cli, sf, pillar, tibble
Jeroen Ooms	sf, rgl, jsonlite, curl, openssl
Kevin Ushey	withr, Rcpp, data.table, rmarkdown, rstudioapi
Henrik Bengtsson	knitr, digest, broom, globals, matrixStats
Christophe Dervieux	knitr, xfun, rmarkdown, sass, tinytex
Bill Venables	MASS, raster, polynom, gplots, plotrix
Zhian N Kamvar	knitr, yaml, ggrepel, adegenet, rticles
Barret Schloerke	rmarkdown, htmltools, evaluate, sass, shiny
John Muschelli	knitr, officer, rticles, riskRegression, rscopus
Rob Hyndman	rmarkdown, forecast, fracdiff, tsfeatures, tsibble
Romain Francois	tibble, knitr, Rcpp, readr, RcppEigen
Bill Denney	digest, broom, classInt, janitor, rio
Duncan Murdoch	rgl, knitr, digest, car, kableExtra
Bob Rudis	viridisLite, viridis, ggthemes, hrbrthemes, flexdashboard
David Robinson	knitr, broom, reprex, tidytext, broom.mixed
Hao Zhu	knitr, broom, kableExtra, skimr, REDCapR
Michał Bojanowski	knitr, broom, labelled, bookdown, alluvial
Carl Boettiger	rticles, EML, taxize, drat, RNXML
Torsten Hothorn	lmtest, mvtnorm, ipred, multcomp, TH.data
Noam Ross	knitr, viridisLite, viridis, bookdown, data.tree
Joseph Larmarange	knitr, lubridate, broom, GGally, labelled
David Hugh-Jones	knitr, broom, covr, maxLik, huxtable
Malcolm Barrett	rmarkdown, broom, ggrepel, usethis, bayesplot
Jonah Gabry	broom, rstan, StanHeaders, loo, rstantools
Jenny Bryan	knitr, broom, tidyxl
Google Inc	rmarkdown, gargle, s2, odbc, tensorflow
Arni Magnusson	xtable, gplots, coda, gdata, DescTools

Table A5: Authors identified in the bipartite network and not in the author collaboration network.

Author	His or Her top downloaded packages
Jan Wijffels	imager, udpipe, cronR, taskscheduleR, opencv
Robin K S Hankin	gsl, hypergeo, elliptic, contrfrac, magic
Simon Urbanek	rgl, digest, base64enc, uuid, rJava
Kartikeya Bolar	STAT, ANOVAIREVA, ANOVASHINY, ANOVASHINY2, BLRShiny
Richard Cotton	withr, knitr, assertive.properties, assertive.base, assertive
Muhammad Yaseen	PakPMICS2014HL, PakPMICS2014Wm, DiallelAnalysisR, bayesammi, baystability
Guangchuang Yu	yulab.utils, gggfun, apilot, ggplotify, tidytree
Florian Schwendinger	Rglpk, lpSolveAPI, ROI, scs, Rsymphony
Paul Murrell	colorspace, lattice, polyclip, gridGraphics, gridBase
Kisung You	Rdimtools, maotai, mclustcomp, ADMM, ROptSpace
Przemyslaw Biecek	survminer, DALEX, iBreakDown, ingredients, eurostat
Joe Thorley	chk, yesno, nlist, term, universals
Philippe Grosjean	Rcmdr, sfsmisc, pastecs, tcltk2, svDialogs
Mohamed El Fodil Ihaddaden	ggeasy, ralger, BARIS, batata, bubblyr
Karline Soetaert	shape, inline, DescTools, deSolve, rootSolve
Emil Hvitfeldt	tidytext, yardstick, parsnip, prismatic, themis
Kevin R Coombes	ClassDiscovery, oompaBase, PCDimension, oompaData, Polychrome
Matthias Kohl	DescTools, distr, distrEx, MKmisc, MKinfer
Michael Hahsler	dbscan, arules, arulesSequences, seriation, TSP
Michael Kleinsasser	AEenrich, covid19india, SEIRfansy, eSIR, FEprovideR
Robrecht Cannoodt	ggrepel, rttiles, proxyC, princurve, diffusionMap
Pablo Sanchez	facebookadsR, googleadsR, pinterrestadsR, linkedInadsR, campaignmanageR
Paul R Rosenbaum	sensitivityfull, sensitivitymw, sensitivitymv, sensitivitymult, exteriorMatch

References

- B. Aronson and K.-C. Yang. *birankr: Ranking Nodes in Bipartite and Weighted Networks*, 2020. URL <https://CRAN.R-project.org/package=birankr>. R package version 1.0.1. [p13]
- B. Aronson, K.-C. Yang, M. Odabas, Y.-Y. Ahn, and B. L. Perry. Comparing measures of centrality in bipartite social networks: A study of drug seeking for opioid analgesics. *SocArXiv*, 2020. doi: 10.31235/osf.io/hazvs. [p13]
- Y. Bao and A. Datta. Simultaneously discovering and quantifying risk types from textual risk disclosures. *Management Science*, 60(6):1371–1391, 2014. [p7]
- K. Benoit and A. Matsuo. *spacyr: Wrapper to the 'spaCy' 'NLP' Library*, 2020. URL <https://CRAN.R-project.org/package=spacyr>. R package version 1.2.1. [p6]
- D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003. [p7]
- P. Bonacich. Factoring and weighting approaches to status scores and clique identification. *Journal of Mathematical Sociology*, 2(1):113–120, 1972. [p11, 12]
- P. Bonacich. Some unique properties of eigenvector centrality. *Social Networks*, 29(4):555–564, 2007. [p12]
- J. M. Chambers. S, R, and Data Science. *The R Journal*, 12(1):462–476, 2020. doi: 10.32614/RJ-2020-028. URL <https://doi.org/10.32614/RJ-2020-028>. [p5]
- A. P. Christensen and Y. N. Kenett. Semantic network analysis (SemNA): A tutorial on preprocessing, estimating, and analyzing semantic networks. *PsyArXiv*, 2019. doi: 10.31234/osf.io/eht87. [p6]
- G. Csárdi, T. Nepusz, et al. The igraph software package for complex network research. *InterJournal, complex systems*, 1695(5):1–9, 2006. [p11, 13]
- G. Csárdi. *cranlogs: Download Logs from the 'RStudio' 'CRAN' Mirror*, 2019. URL <https://CRAN.R-project.org/package=cranlogs>. R package version 2.1.1. [p14]
- J. Fox. Aspects of the Social Organization and Trajectory of the R Project. *The R Journal*, 1(2):5–8, 2009. [p5]

- J. Fox and A. Leanage. R and The Journal of Statistical Software. *Journal of Statistical Software*, 73(1):1–13, 2016. [p5]
- L. C. Freeman. Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239, 1978. [p11]
- R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, et al. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology*, 5(10):1–16, 2004. [p5]
- D. M. German, B. Adams, and A. E. Hassan. The evolution of the R software ecosystem. In 2013 17th European Conference on Software Maintenance and Reengineering, pages 243–252. IEEE, 2013. [p5]
- B. Grün and K. Hornik. topicmodels: An R package for fitting topic models. *Journal of Statistical Software*, 40(13):1–30, 2011. doi: 10.18637/jss.v040.i13. [p7]
- X. He, M. Gao, M.-Y. Kan, and D. Wang. Birank: Towards ranking on bipartite graphs. *IEEE Transactions on Knowledge and Data Engineering*, 29(1):57–71, 2016. [p13]
- R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996. [p5]
- S. Jain and A. Sinha. Identification of influential users on twitter: A novel weighted correlated influence measure for covid-19. *Chaos, Solitons & Fractals*, 139:110037, 2020. [p11]
- I. Kosmidis. *cranly: Package Directives and Collaboration Networks in CRAN*, 2019. URL <https://CRAN.R-project.org/package=cranly>. R package version 0.5.4. [p11]
- J. Lai, C. J. Lortie, R. A. Muenchen, J. Yang, and K. Ma. Evaluating the popularity of R in ecology. *Ecosphere*, 10(1):e02567, 2019. [p5]
- F. Morone and H. A. Makse. Influence maximization in complex networks through optimal percolation. *Nature*, 524(7563):65–68, 2015. [p11]
- L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999. [p11, 12]
- R Core Team. *Writing R Extensions*. R Foundation for Statistical Computing, Vienna, Austria, 2021. URL <https://cran.r-project.org/doc/manuals/r-release/R-exts.html>. [p11]
- A. Salavaty, M. Ramialison, and P. D. Currie. Integrated value of influence: an integrative method for the identification of the most influential nodes within networks. *Patterns*, 1(5):100052, 2020. [p11, 13]
- J. Silge and D. Robinson. *Text mining with R: A tidy approach*. O'Reilly Media, Inc., 2017. [p7]
- J. Silge, J. C. Nash, and S. Graves. Navigating the R Package Universe. *The R Journal*, 10(2):558–563, 2018. doi: 10.32614/RJ-2018-058. URL <https://doi.org/10.32614/RJ-2018-058>. [p5, 18]
- S. Tippmann. Programming tools: Adventures with R. *Nature News*, 517(7532):109, 2015. [p5]
- Z. Wang, C. Du, J. Fan, and Y. Xing. Ranking influential nodes in social networks based on node position and neighborhood. *Neurocomputing*, 260:466–477, 2017. [p11]
- H. Wickham. ggplot2. *Wiley Interdisciplinary Reviews: Computational Statistics*, 3(2):180–185, 2011. [p5]
- A. Zeileis. Cran task views. *R News*, 5(1):39–40, 2005. [p5]
- Z. Zhang and D. Zhang. What is Data Science? An Operational Definition based on Text Mining of Data Science Curricula. *Journal of Behavioral Data Science*, 1(1):1–16, 2021. [p5, 6, 9, 10, 17]

Lijin Zhang
Graduate School of Education, Stanford University
United States
ORCID: 0000-0002-4222-8850
lijinzhang@stanford.edu

Xueyang Li
Department of Computer Science and Engineering, University of Notre Dame

United States
xli34@nd.edu

Zhiyong Zhang
Department of Psychology, University of Notre Dame
United States
ORCID: [0000-0003-0590-2196](https://orcid.org/0000-0003-0590-2196)
zzhang4@nd.edu

SSNbayes: An R Package for Bayesian Spatio-Temporal Modelling on Stream Networks

by Edgar Santos-Fernandez, Jay M. Ver Hoef, James McGree, Daniel J. Isaak, Kerrie Mengersen, and Erin E. Peterson

Abstract Spatio-temporal models are widely used in many research areas from ecology to epidemiology. However, a limited number of computational tools are available for modeling river network datasets in space and time. In this paper, we introduce the R package **SSNbayes** for fitting Bayesian spatio-temporal models and making predictions on branching stream networks. **SSNbayes** provides a linear regression framework with multiple options for incorporating spatial and temporal autocorrelation. Spatial dependence is captured using stream distance and flow connectivity while temporal autocorrelation is modelled using vector autoregression approaches. **SSNbayes** provides the functionality to make predictions across the whole network, compute exceedance probabilities, and other probabilistic estimates, such as the proportion of suitable habitat. We illustrate the functionality of the package using a stream temperature dataset collected in the Clearwater River Basin, USA.

1 Introduction

Rivers and streams are of vital ecological and economic importance (Vörösmarty et al., 2010) but are under pressure from anthropogenic impacts such as climate change, pollution, water extractions, and overfishing. In the past, data describing critical characteristics such as nutrients, sediments, pollutants and stream flow tended to be sparse in space and/or time. However, recent developments in in-situ sensor technology have revolutionized ecological research and natural resource monitoring. These new data sets create exciting opportunities to measure, learn about, and manage the spatio-temporal dynamics of abiotic (e.g. temperature, water chemistry, habitat characteristics) and biotic processes (e.g. migration, predation, and competition). However, the unique spatial relationships found in stream data (e.g. branching network structure, longitudinal (upstream/downstream) connectivity, water flow volume and direction) and high-frequency of sampling create analytical challenges that make it difficult to gain meaningful insights from these datasets. We attempt to overcome these challenges through the **SSNbayes** package which provides convenient and practical tools to undertake Bayesian inference in complex spatial and temporal stream network settings.

1.1 Motivating dataset: repeated measures from *in-situ* sensor locations in a river

Consider the river network in the Clearwater River Basin, USA shown in Fig 1. Water temperature data were collected at fixed time intervals using in-situ sensors placed at 18 unique locations throughout the network (Isaak et al., 2018). We want to use these data to address several research questions and goals. Firstly, we would like to analyse water temperature to assess the impact of covariates such as air temperature. Secondly, we aim to make predictions with uncertainty at other locations throughout the network (approximately every 1 km). Also, we want to impute missing data e.g. the most downstream point in Fig 1 (location 12), and determine regions of the network which remain suitable habitats over time to sustain fish species such as bull trout which typically preferred water temperatures of < 13 °C. Throughout this paper, we show how the **SSNbayes** package can be used to analyse these spatial and temporal data, and address these motivating research questions.

1.2 A brief review

A number of R software packages for spatial stream-network modelling have been developed over the last few decades (Ver Hoef et al., 2014; Skoien et al., 2014; Rushworth, 2017). These packages account for unique spatial relationship found in stream data. For example, the R packages **SSN** and **SSN2** (Ver Hoef et al., 2014; Dumelle et al., 2023) fits spatial regression models for stream networks, with autocorrelation in time only possible by using random effects as repeated measures, which induces equal correlation for all times at a location. Similarly, spatial additive models can be fitted using the package **smnet** (Rushworth, 2017). However, these models are not designed to simultaneously account for the temporal variability that often accompanies spatial variation in new data sets derived from modern sensor arrays.

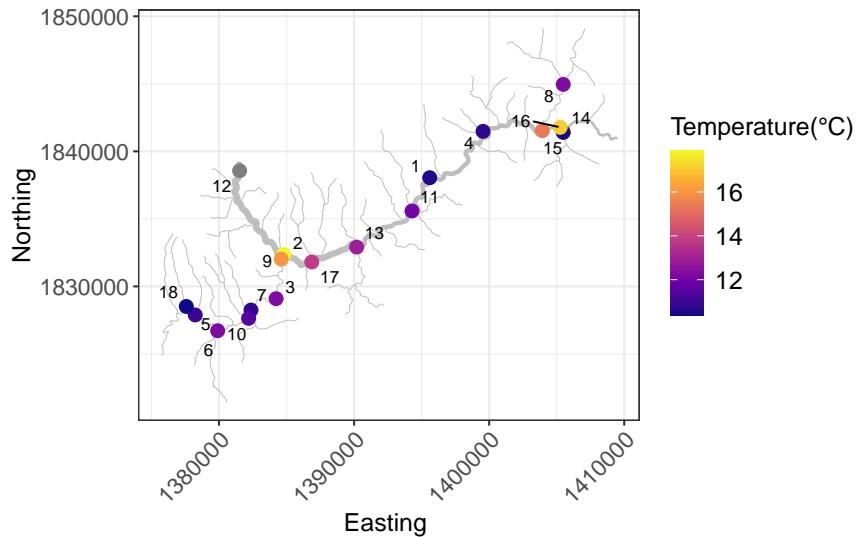


Figure 1: Mean daily water temperature in degrees Celsius on August 1, 2012, at 18 different spatial locations in the Clearwater stream network in the US. The temperature values are represented by a color scale, with cooler temperatures shown in blue and warmer temperatures shown in yellow. Each spatial location is labeled with a unique identifier (locID). The plot highlights the variation in water temperature across the different locations in the network.

There are several R packages for spatio-temporal modelling that are described in the Space-time CRAN Task View (Pebesma, 2021). For example, spatial/temporal dependence can be incorporated via the `nlme` package (Pinheiro and Bates, 2000; Pinheiro et al., 2020). Other packages such as `spBayes` (Finley et al., 2015) allow random effects modelling for point-referenced data. `CARBayes` (Lee, 2013) contains useful tools for implementing Bayesian spatial models using random effects via conditional autoregressive (CAR) priors. Similarly, spatial process data can be represented using kernels e.g. using the package `RandomFields` (Schlather et al., 2015). Interpolation of spatial data and Kriging can be done via tools from the package `geoR` (Ribeiro Jr et al., 2020). One of the most popular implementations among practitioners is the `R-INLA` package (Lindgren and Rue, 2015), which uses approximate Bayesian inference via integrated nested Laplace approximations. Multiple latent Gaussian spatio-temporal models can be fit in `R-INLA`. `FRK` (Zammit-Mangion and Cressie, 2021) makes use of spatial basis functions and discrete areal units with a focus on large datasets. In the same line, new packages harnessing the power of `rstan` (Carpenter et al., 2017) are also emerging. For instance, `bmstdr` (Sahu et al., 2022) implements several spatio-temporal approaches for point referenced and areal unit datasets, and `geostan` (Donegan, 2022) fits spatial models for areal data. However, none of these packages are specifically designed to account for the unique spatial relationships found in data collected on streams.

Here, we describe the `SSNbayes` package which has been designed to address many of the limitations of the current software tools for spatio-temporal modelling on stream networks. More specifically, `SSNbayes` can fit spatio-temporal stream-network models and produce predictions in space and time, with associated estimates of uncertainty. It uses the Bayesian inference machinery implemented using the probabilistic programming language Stan.

The rest of the paper is organised as follows: we introduce the relevant statistical models in the Methods section, followed by an overview of the package structure and functions. We then demonstrate how the `SSNbayes` package can be used to explore, analyse and draw conclusions from a stream temperature data set collected in the western United States (USA). A second reproducible example is provided in the Appendix to help users adapt the R code for their own data. Finally, we conclude with a discussion of the benefits and challenges in using the `SSNbayes` package and Bayesian spatio-temporal models on stream networks, as well as potential extensions to the methods and improvements to the package.

2 Methods

Data collected on stream networks often exhibit complex patterns of spatial autocorrelation resulting from ecosystem processes occurring within branching stream networks, as well as between the aquatic and terrestrial environments (Peterson et al., 2013). It is therefore common for streams data to exhibit both Euclidean and in-stream patterns of spatial autocorrelation at multiple scales (Peterson et al., 2006; Peterson and Ver Hoef, 2010). Thus, we start this section with a description of spatial models based on Euclidean distance and then describe methodological extensions to stream networks. These models provide the foundation to describe the Bayesian hierarchical spatio-temporal model variations implemented in the **SSNbayes** package.

General space-time model

Consider the general spatio-temporal linear model:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{v} + \boldsymbol{\epsilon}, \quad (1)$$

where the response variable $\mathbf{y} = [\mathbf{y}_1', \mathbf{y}_2', \dots, \mathbf{y}_T']'$ is a stacked vector of length $n = S \times T$ for S spatial locations and T time points. We order data such that vector \mathbf{y}_t contains the observations at the S spatial locations at time t for $t = 1, \dots, T$. Let \mathbf{X}_t be an $S \times p$ design matrix of p covariates for the t th time. We construct a stacked matrix of covariates $\mathbf{X} = [\mathbf{X}_1', \mathbf{X}_2', \dots, \mathbf{X}_T']'$ with dimensions $n \times p$. We then define $\boldsymbol{\beta}$ as a $p \times 1$ vector of regression coefficients. We let $\mathbf{v} = [\mathbf{v}_1', \mathbf{v}_2', \dots, \mathbf{v}_T']'$ be a stacked vector of n spatial random effects, where \mathbf{v}_t is a vector of length S for each t , and all \mathbf{v}_t have the same spatial dependence model (shared locations and parameters across all times t). For example, \mathbf{v}_t can be modelled using a Gaussian process (Banerjee et al., 2014), but we also use spatial stream-network models that we describe below for each \mathbf{v}_t . The final step in our construction is to use vector autoregressive models to add the temporal components which we develop below. The vector $\boldsymbol{\epsilon}$ is the independent unstructured random error term, where $\text{var}(\boldsymbol{\epsilon}) = \sigma_0^2 \mathbf{I}$. The parameter σ_0^2 is called the nugget effect and \mathbf{I} is an $n \times n$ identity matrix.

Euclidean distance models

A typical modelling approach to capture spatial dependence is via the second moment of \mathbf{v} from Eq 1 where the amount of autocorrelation decays with the Euclidean distance. Some of the most common covariance functions are the exponential, Gaussian and spherical (Cressie and Wikle, 2015; Banerjee et al., 2014):

$$\text{exponential model, } C_{ED}(d | \boldsymbol{\theta}) = \sigma_e^2 e^{-3d/\alpha_e}, \quad (2)$$

$$\text{Gaussian model, } C_{ED}(d | \boldsymbol{\theta}) = \sigma_e^2 e^{-3(d/\alpha_e)^2}, \quad (3)$$

and

$$\text{spherical model, } C_{ED}(d | \boldsymbol{\theta}) = \sigma_e^2 \left(1 - \frac{3d}{2\alpha_e} + \frac{d^3}{2\alpha_e^3} \right) \mathbb{1}(d/\alpha_e \leq 1), \quad (4)$$

where $\alpha_e \in (0, \infty)$, $\sigma_e^2 > 0$, and d is the Euclidean distance between two locations s_i and s_j . The vector $\boldsymbol{\theta}$ represents the spatial parameters (α_e, σ_e^2) , where σ_e^2 is the partial sill, α_e is the spatial range parameter and $\mathbb{1}(\cdot)$ is the indicator function. The partial sill is the resulting variance after accounting for the nugget effect (sill minus nugget effect). Negligible spatial correlation is assumed between points located at a distance greater than the spatial range parameter.

2.1 Spatial models for stream networks

The stream network data shown in Fig. 2 represents repeated measures at several time points t from four spatial locations (s_1 to s_4). Stream distance is defined as the separation distance between two locations when movement is restricted to the network. The direction of the water flow is also shown in the figure (from north to south) and the stream outlet (i.e. most downstream point on the stream network) is below location s_4 . The watershed (i.e. drainage basin) includes the land that contributes water flow to a discrete downstream location in the stream network. Thus, the watershed for the stream outlet in Fig. 2 includes all of the upstream regions (r_1-r_4). Spatial locations s_1 and s_3 are considered flow-connected because the water flows from the upstream location s_1 to the downstream

location s_3 . In contrast, s_1 and s_2 are considered flow-unconnected because they reside on the same stream network, but do not share flow.

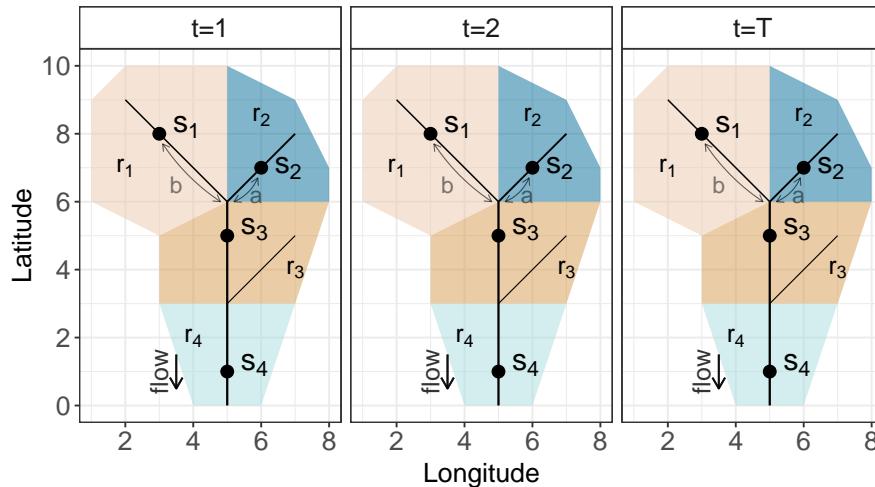


Figure 2: Visualization of a stream network at multiple time points, with water flowing from top to bottom. The plot displays four spatial locations ($s_1 - s_4$) and four regions ($r_1 - r_4$), with the colors indicating the regions. This visualization allows for an understanding of the spatial and temporal dynamics of the stream network.

Multiple covariance models have been proposed for stream networks that describe the unique spatial dependence related to network structure and stream flow. These models are based on moving average constructions that use stream distance, and were introduced by Ver Hoef et al. (2006) and Cressie et al. (2006) (tail-up models) and Ver Hoef and Peterson (2010) and Garreta et al. (2010) (tail-down models); several models of each class are given below. These spatial covariance matrices have been extensively employed in numerous water quality modelling frameworks and applications (Money et al., 2009a,b; Isaak et al., 2014; McManus et al., 2020; Jackson et al., 2018; Rodríguez-González et al., 2019) and are described in more detail below.

Tail-up models

Tail-up covariance models were developed by convolving a moving average function with white noise (Ver Hoef et al., 2006). As the name suggests, the moving average function points in the upstream direction from a stream location in a tail-up model, which restricts correlation to flow-connected locations only. In addition, the function must be split at upstream junctions using spatial weights, which maintain stationary variances by controlling the proportion allocated to each upstream segment.

Given a pair of sites s_i and s_j , the tail-up covariance matrix is defined as:

$$C_{TU}(s_i, s_j | \boldsymbol{\theta}) = \begin{cases} 0 & \text{if } s_i, s_j \text{ are flow-unconnected,} \\ C_u(h | \boldsymbol{\theta}) W_{ij} & \text{if } s_i, s_j \text{ are flow-connected,} \end{cases}$$

where $C_u(h | \boldsymbol{\theta})$ is an unweighted tail-up covariance between two locations based on the total stream distance between them, h . W_{ij} represents the spatial weights between sites i and j . C_u can take a variety of forms including:

$$\text{Tail-up exponential model, } C_u(h | \boldsymbol{\theta}) = \sigma_u^2 e^{-3h/\alpha_u}, \quad (5)$$

$$\text{Tail-up linear-with-sill model, } C_u(h | \boldsymbol{\theta}) = \sigma_u^2 (1 - h/\alpha_u) \mathbb{1}(h/\alpha_u \leq 1), \quad (6)$$

$$\text{Tail-up spherical model, } C_u(h | \boldsymbol{\theta}) = \sigma_u^2 \left(1 - \frac{3h}{2\alpha_u} + \frac{h^3}{2\alpha_u^3} \right) \mathbb{1}(h/\alpha_u \leq 1) \quad (7)$$

where σ_u^2 is the partial sill and α_u is the range parameter.

The spatial weights used in the tail-up model are generated for flow-connected locations

$$W_{ij} = \sqrt{\frac{\Omega(s_j)}{\Omega(s_i)}},$$

where $\Omega(s_i)$ and $\Omega(s_j)$ are the additive function values for the upstream and downstream site, respectively. Additive function values can be derived from any variable available for every line segment in a stream network, but are often based on an ecologically meaningful variable (e.g. stream flow) or a surrogate (e.g. watershed area), which is thought to represent relative influence in a stream network (Frieden et al., 2014). Additive function values can be generated in **SSN** using the `additive.function()` or in the STARS ArcGIS custom toolset using the SegmentPI, Additive Function - Edges, and Additive Function - Sites tools (Peterson and Ver Hoef, 2014). For additional information about how the additive function values are constructed, please see Ver Hoef et al. (2006) or Appendices A in Santos-Fernandez et al. (2022) or Peterson and Ver Hoef (2010).

Tail-down models

Tail-down models are also based on a moving average construction (Ver Hoef and Peterson, 2010), but in this case the function points in the downstream direction. This allows the models to describe spatial correlation between both flow-connected and flow-unconnected locations. In addition, the moving average functions converge downstream and do not need to be split at junctions using spatial weights.

There is a distinction between flow-connected and flow-unconnected relationships in a tail-down model. When pairs of sites are flow-connected, distance is based on the total stream distance between them, h . When sites are flow-unconnected (e.g. s_1 and s_2 in Fig. 2), we define a and b as the stream distance from s_1 and s_2 to their first common downstream junction, so that $a \leq b$.

Tail-down exponential model,

$$C_{TD}(a, b, h|\boldsymbol{\theta}) = \begin{cases} \sigma_d^2 e^{-3h/\alpha_d} & \text{if flow-connected,} \\ \sigma_d^2 e^{-3(a+b)/\alpha_d} & \text{if flow-unconnected,} \end{cases}$$

Tail-down linear-with-sill model,

$$C_{TD}(a, b, h|\boldsymbol{\theta}) = \begin{cases} \sigma_d^2(1 - \frac{h}{\alpha_d}) \mathbb{1}(\frac{h}{\alpha_d} \leq 1) & \text{if flow-connected,} \\ \sigma_d^2(1 - \frac{b}{\alpha_d}) \mathbb{1}(\frac{b}{\alpha_d} \leq 1) & \text{if flow-unconnected,} \end{cases}$$

Tail-down spherical model,

$$C_{TD}(a, b, h|\boldsymbol{\theta}) = \begin{cases} \sigma_d^2(1 - \frac{3h}{2\alpha_d} + \frac{h^3}{2\alpha_d^3}) \mathbb{1}(\frac{h}{\alpha_d} \leq 1) & \text{if flow-connected,} \\ \sigma_d^2(1 - \frac{3a}{2\alpha_d} + \frac{b}{2\alpha_d})(1 - \frac{b}{\alpha_d}) \mathbb{1}(\frac{b}{\alpha_d} \leq 1) & \text{if flow-unconnected,} \end{cases}$$

where σ_d^2 is the partial sill, α_d is the range parameter, and $\mathbb{1}(\cdot)$ is the indicator function, equal to 1 if its argument is true, otherwise it is zero.

2.2 Mixed models

A mixture of Euclidean, tail-up and tail-down covariance matrices is often used to capture the complex patterns of spatial dependency found in stream data. In Eq 1, \mathbf{v}_t for a purely spatial case is a vector of dimension S corresponding to the spatial locations, with covariance matrix $\boldsymbol{\Sigma}$.

$$\boldsymbol{\Sigma} = COV(\mathbf{v}) = \mathbf{C}_{ED} + \mathbf{C}_{TU} + \mathbf{C}_{TD} = \sigma_e^2 \mathbf{R}_e(\alpha_e) + \sigma_u^2 \mathbf{R}_u(\alpha_u) + \sigma_d^2 \mathbf{R}_d(\alpha_d), \quad (8)$$

where σ_e^2 , σ_u^2 , and σ_d^2 are the partial sills for Euclidean, tail-up and tail-down functions, respectively. The correlation matrices $\mathbf{R}_u(\alpha_u)$, $\mathbf{R}_d(\alpha_d)$ and $\mathbf{R}_e(\alpha_e)$ are obtained as a function of the range parameters α_u , α_d and α_e (Ver Hoef et al., 2014).

For space-time applications, we can use the same spatial covariance matrix or we can build a dynamic model with spatial parameters that are time-specific. We opted for the first approach in the **SSNbayes**, since this reduces the number of parameters to be estimated from the model and is less computationally demanding. We return to this point in the Discussion.

2.3 Spatio-temporal stream network models

Following the above discussion, consider the stream network in Fig. 2, that evolves across discrete time points $t = 1, 2, \dots, T$. Let a response variable \mathbf{y}_t be an $S \times 1$ vector of random variables at unique and fixed spatial locations of $s = 1, 2, \dots, S$. We start with the following conditional spatio-temporal model:

$$[\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_T] = \prod_{t=2}^T [\mathbf{y}_t | \mathbf{y}_{t-1}, \boldsymbol{\theta}, \mathbf{X}_t, \mathbf{X}_{t-1}, \boldsymbol{\beta}, \boldsymbol{\Phi}, \boldsymbol{\Sigma}] \quad (9)$$

where \mathbf{y}_1 is the process at $t = 1$, and

$$[\mathbf{y}_t | \mathbf{y}_{t-1}, \boldsymbol{\theta}, \mathbf{X}_t, \mathbf{X}_{t-1}, \boldsymbol{\beta}, \boldsymbol{\Phi}, \boldsymbol{\Sigma}] \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma} + \sigma_0^2 \mathbf{I}), \quad (10)$$

Here, $\boldsymbol{\Sigma} = \text{COV}(\mathbf{v}_t)$ is an $S \times S$ spatial covariance matrix of the form given in Eq 8. The mean $\boldsymbol{\mu}_t$ can be expressed as a vector autoregressive process of order one VAR(1) (Hamilton, 1994):

$$\boldsymbol{\mu}_t = \mathbf{X}_t \boldsymbol{\beta} + \boldsymbol{\Phi}(\mathbf{y}_{t-1} - \mathbf{X}_{t-1} \boldsymbol{\beta}), \quad (11)$$

The square transition matrix, $\boldsymbol{\Phi}$ of dimension $S \times S$, has elements ϕ_{ij} , which describe the conditional temporal cross-correlation between two spatial locations i and j .

2.4 Vector autoregressive model variations

We use a vector autoregressive (VAR) approach to simultaneously model time series from multiple spatial locations in the network. This stochastic approach allows capturing and incorporating temporal dependence across multiple time series. Two variations of the vector autoregressive spatial process have been implemented in the **SSNbayes** package.

Case 1 (AR)

The simplest case considers the same temporal autocorrelation for all spatial locations. Therefore all the diagonal elements of $\boldsymbol{\Phi}$ are equal to ϕ and all the off-diagonal elements are set to zero, which assumes negligible cross-correlation between time series. That is:

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi & 0 & \cdots & 0 \\ 0 & \phi & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \phi \end{bmatrix}. \quad (12)$$

Under this model, the temporal autocorrelation for a fixed spatial location is an AR1 model with autocorrelation parameter ϕ , and the joint spatio-temporal autocovariance matrix is the separable model,

$$\text{var}(\mathbf{y}) = \mathbf{C}_{OO} = \frac{1}{1-\phi^2} \begin{bmatrix} 1 & \phi & \cdots & \phi^{T-1} \\ \phi & 1 & \cdots & \phi^{T-2} \\ \vdots & \vdots & \ddots & \vdots \\ \phi^{T-1} & \phi^{T-2} & \cdots & 1 \end{bmatrix} \otimes (\boldsymbol{\Sigma} + \sigma_0^2 \mathbf{I}), \quad (13)$$

where \otimes is the Kronecker product.

Spatial locations in large river networks often have different elevations, climatic conditions, or local flow regimes and this can affect the amount of temporal autocorrelation in observations. Hence, the assumption that there is a common ϕ for all locations may not always be appropriate and this motivates Case 2.

Case 2 (VAR)

In the second method, rather than a shared ϕ across all locations, each location gets its own temporal autocorrelation parameter ϕ_s for $s \in 1, \dots, S$. This is known as the autoregressive shock model (Wikle et al., 1998), which can be defined through $\boldsymbol{\Phi}$ as follows:

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi_1 & 0 & \cdots & 0 \\ 0 & \phi_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \phi_S \end{bmatrix}. \quad (14)$$

Other VAR structures consider ϕ as a linear combination of spatial covariates and cross-correlation between time series (Santos-Fernandez et al., 2022). These variations are not currently implemented in **SSNbayes** but are under development.

2.5 Bayesian inference and specification of prior distributions

Formulating this model in a Bayesian framework requires sampling from the following posterior distribution:

$$[\boldsymbol{\beta}, \boldsymbol{\Phi}, \sigma_0^2, \sigma_u^2, \alpha_u, \sigma_d^2, \alpha_d, \sigma_e^2, \alpha_e \mid \mathbf{y}, \mathbf{X}]. \quad (15)$$

SSNbays uses Stan (Carpenter et al., 2017) to efficiently produce samples from this distribution via Hamiltonian Monte Carlo (HMC) methods.

We also need to define prior distributions for the parameters of interest. Currently, diffuse prior distributions are the only option in `ssnbayes()`, but the functionality to include other prior distributions may be included in future package versions. The implemented prior distributions are the following:

$\phi \sim \text{Uniform}(-1, 1)$	# uniform prior on the autoregressive parameter
$\alpha \sim \text{Uniform}(0, 4 \times \max(h))$	# diffuse prior on the spatial range
$\sigma \sim \text{Uniform}(0, 100)$	# diffuse prior on the square root of the partial sill
$\sigma_0 \sim \text{Uniform}(0, 100)$	# diffuse prior on the square root of the nugget effect
$\beta \sim \mathcal{N}(0, 1000)$	# prior on the regression coefficients (intercept and slope)

For the autoregressive parameter (ϕ), a uniform prior defined from -1 to 1 is used to ensure the process is stationary. The upper limit for the spatial range parameter is set to four times the maximum stream distance between observation locations on the network.

2.6 Prediction

There are two ways of making predictions in **SSNbays**. By default, predictions are produced for missing values (NA) in the response variable in the observation dataset used to fit the model via the posterior predictive distribution:

$$p(\hat{\mathbf{y}} \mid \mathbf{y}, \mathbf{X}, \hat{\mathbf{X}}) = \int p(\hat{\mathbf{y}} \mid \eta, \hat{\mathbf{X}}) p(\eta \mid \mathbf{y}, \mathbf{X}) d\eta, \quad (16)$$

where $\hat{\mathbf{X}}$ are the covariates for $\hat{\mathbf{y}}$. This Eq 16 gives the probability distribution of a new observation, $\hat{\mathbf{y}}$, given the observed data \mathbf{y} and covariates/predicted covariates.

However, this approach is not recommended when making predictions at a large number of spatial locations (e.g. predicting over an extensive branching stream network) since it would involve operations with large matrices which can be very inefficient.

The second approach uses the fitted model produced using `ssnbayes()` to generate predictions using the Kriging predictor in a prediction dataset (Banerjee et al., 2014; Gelfand et al., 2019). This produces estimates as a weighted average of observations.

$$\hat{\mathbf{y}}_P = \mathbf{X}_P \boldsymbol{\beta} + \mathbf{C}_{OP}' \mathbf{C}_{OO}^{-1} (\mathbf{y}_O - \mathbf{X}_O \boldsymbol{\beta}), \quad (17)$$

where subscripts O and P indicate the observation and prediction locations, respectively. The stacked vector $\hat{\mathbf{y}}_P$ contains the predictions at the P spatial locations across all the time points T . The observations are represented in the stacked vector \mathbf{y}_O , which contains all of the observations across the T time points. \mathbf{X}_P and \mathbf{X}_O are space-time design matrices of covariates for the observations and predictions, respectively, while $\boldsymbol{\beta}$ is a vector of regression coefficients.

The separable square matrix \mathbf{C}_{OO} of dimension $n = S \times T$ (defined in Eq 1), contains the covariances between observations at all time points, where O and T are the number of observations and time points respectively. Similarly, \mathbf{C}_{OP} is a $O \times T$ by $P \times T$ rectangular separable matrix of covariances between observation and prediction locations at all time points with the same structure as \mathbf{C}_{OO} . That is, if \mathbf{C}_{OO} was obtained from an AR exponential tail-down model with parameters ϕ , σ_{td} and α_{td} , these same parameters are used to construct \mathbf{C}_{OP} .

The covariance matrix of observations (\mathbf{C}_{OO}) must be inverted at each MCMC iteration when making predictions (Eq. 17) and this quickly becomes computationally challenging for large datasets. However, the separability of (13) significantly reduces the computational burden in these cases (Wikle et al., 2019)

$$\mathbf{C}_{OO}^{-1} = \boldsymbol{\Sigma}_{ar1}^{-1} \otimes \boldsymbol{\Sigma}_{OO}^{-1},$$

where Σ_{OO} is the spatial covariance matrix defined in Eq 8, and Σ_{ar1} is the temporal covariance matrix of an AR(1) process (13) which has an analytical inverse that is tridiagonal.

3 The **SSNbayes** package

3.1 Data pre-processing

Detailed spatial, topological, and attribute data are needed to fit spatial stream network models, including those implemented in **SSNbayes**. There are currently two software packages that can be used to generate this information: 1) the Spatial Tools for the Analysis of River Systems (STARS) custom toolset (Peterson and Ver Hoef, 2014) for ArcGIS version ≥ 10.1 (ESRI, 2019) or 2) the R package **openSTARS** (Kattwinkel et al., 2020). When the pre-processing is complete, both tools create a new directory with the extension .ssn (i.e. a .ssn object), which contains shapefiles of the stream network, observed locations, and prediction locations (optional). It also contains the response, covariates (optional), and the information needed to generate stream distances and spatial weights between observed and prediction locations. Several unique identifiers are also assigned to observed and prediction locations to denote unique locations (locID) and unique measurements in space and time (pid). If real data are not being used, the `createSSN()` and `SimulateOnSSN()` functions found in **SSN** can be used to generate artificial .ssn objects that meet these requirements. It is important to note, however, that the **SSN** package has been archived on CRAN. In the absence of a .ssn object, **SSNbayes** can still be used to fit models based solely on Euclidean covariance models.

3.2 Installation

The **SSNbayes** package for R statistical software ($\geq 3.3.0$) extends the models implemented in the **SSN2** package. **SSNbayes** is based on the R package **rstan** and the C++ toolchain is required. See <https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started>.

The **SSNbayes** package can be found at CRAN and Github (<https://github.com/EdgarSantos-Fernandez/SSNbayes>) and installed using:

```
if(!require('SSNbayes')) install.packages("SSNbayes", dependencies = T)
```

Or:

```
remotes::install_github("EdgarSantos-Fernandez/SSNbayes", dependencies = T)
```

For this demonstration we will need the companion R package **SSNdata** (Santos-Fernandez, 2022).

```
remotes::install_github("EdgarSantos-Fernandez/SSNdata",
  ref = "HEAD", upgrade_dependencies = F, dependencies = F)
```

SSNbayes describes both spatial and temporal autocorrelation using Bayesian inference. Table 1 shows a summary of the main functions included in the package and a description of the most important arguments. The function `collapse()` is used to extract line features from an spatial stream network (ssn) object in a format suitable for visualisation using packages such as **ggplot2**. The function `ssnbayes()` is the core function in **SSNbayes**, providing the functionality to fit linear spatio-temporal regression models (Santos-Fernandez et al., 2022), while the `predict()` function is used to perform spatio-temporal prediction from a stanfit object generated from `ssnbayes()`. Generic functions such as `summary()` and `plot` can be used to generate summary statistics and visualize model outcomes. In addition, the helper functions `dist_weight_mat()` and `dist_weight_mat_preds()` provide the functionality to extract a list of Euclidean and stream distance matrices, a spatial weight matrix, and an indicator matrix for flow connectivity between sites.

3.3 Modelling stream temperatures

We use the dataset described in the Introduction to illustrate how the **SSNbayes** package can be used to explore, analyse and draw conclusions from a Bayesian spatio-temporal model. The .ssn object for these data is part of the **SSNdata** package.

```
if(!require('SSNdata')) remotes::install_github("EdgarSantos-Fernandez/SSNdata")
```

Table 1: A description of functions in the **SSNbayes** package and the most important arguments.

function	arguments	description
collapse()	t	Path to a .ssn object
ssnbayes()	formula data path space_method time_method iter warmup chains addfuncoll	A formula object, as used in lm() A long data.frame containing the locations, dates, covariates and response Path with the name of the .ssn object A list defining whether a spatial stream network (ssn) object is used and the spatial corr A list specifying the temporal structure Number of iterations Number of warm up samples Number of chains (optional) Column name for site additive function
predict()	formula obs_data stanfit pred_data nsamples addfuncoll locID_pred chunk_size	A formula object, as used in lm() A long data.frame containing the locations, dates, covariates and response A stanfit object returned from ssnbayes() A long df of predictions with the locations, dates, covariates and the response The number of samples to draw from the posterior distributions (optional) Column name for site additive function (optional) the location id for the predictions (optional) the number of locID to make prediction from
dist_weight_mat() dist_weight_mat_preds()	t t	Path to a .ssn object Path to a .ssn object

For reproducibility, we also created a Kaggle notebook containing the example from this section (<https://www.kaggle.com/edsans/ssnbayes>). For completeness, similar analyses were performed on simulated data and the results are presented in the Appendix.

The .ssn object was generated using the STARS custom toolset (Peterson and Ver Hoef, 2014) and contains 18 observation and 60 prediction locations spaced at 1km intervals along the stream network. Hourly temperature recordings were taken at the observation sites but these were averaged to mean daily values for the two-year period of the data set. The data residing within the .ssn object are imported into R, converted to an ssn object and pair-wise distances are calculated for all observed sites, observed and prediction sites, and prediction sites using the following **SSN2** package functions:

```
path <- system.file("extdata/clearwater.ssn", package = "SSNdata")
n <- SSN2::ssn_import(path, predpts = "preds", overwrite = TRUE)
SSN2::ssn_create_distmat(n,
                         predpts = "preds" ,
                         overwrite = TRUE,
                         among_predpts = TRUE)
```

We also read in a data frame containing the response and covariate data:

```
clear <- readRDS(system.file("extdata/clear_obs.RDS", package = "SSNdata"))
```

In the data.frame clear, the response variable (temp) is the mean daily stream temperature measured at 18 observation sites. Here we focus on a subsample of longitudinal response data consisting of 24 time points at those sites over two years (Figure 3). We randomly split the dataset, with 2/3 used for training the model and 1/3 for testing the out-of-sample prediction accuracy. This training/testing split was performed once for illustration purposes but for complex datasets we recommend using leave-one-out cross-validation.

Stream temperature is strongly influenced by topography and climate variables (Isaak et al., 2017). The following covariates were available for the observation and prediction locations across all the time points: stream slope, elevation, watershed area (Isaak et al., 2017), and air temperature (e.g. Bal et al., 2014). In addition, we included the first pair of harmonic covariates (Fourier terms) for the time periods (\sin_t and \cos_t) (Hyndman and Khandakar, 2008).

3.4 Visualizing stream network data in space and time

We begin by extracting the streams from the ssn object so that we can visualise the data in **ggplot2**.

```
n.df <- SSNbayes::collapse(n)
```

The data.frame n.df contains data describing the spatial location of individual stream segments, along with the additive function column. The spatial and space-time data can then be visualised using **ggplot2** (Fig 1).

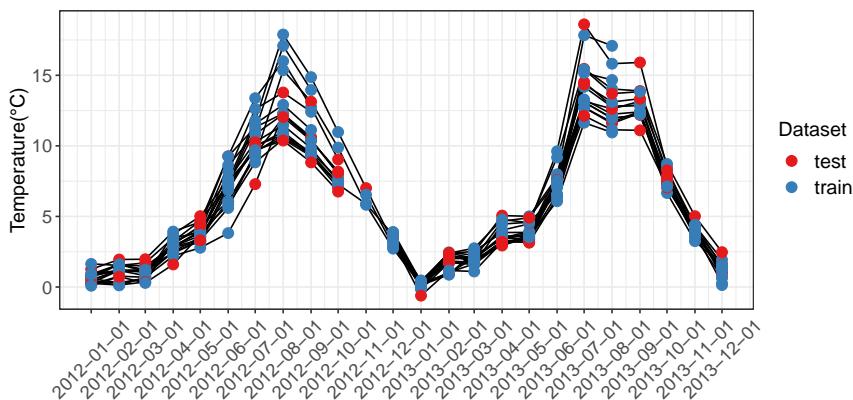


Figure 3: Time series of stream temperatures at multiple locations. Each line represents the time series for a unique observation location. The x-axis represents date, while the y-axis represents the water temperature in $^{\circ}\text{C}$. Observations (training dataset) are displayed as blue points and the predictions (testing set) are displayed in red. The plot reveals the periodic changes in water temperatures over time at different locations in the stream network.

3.5 Fitting spatio-temporal linear models

The next step is to fit a linear spatio-temporal regression model using the function `ssnbayes()`. We specify the following linear regression model using the covariates in the observed dataset:

$$X'_{(t)}\beta = \beta_0 + \beta_1 * \text{SLOPE} + \beta_2 * \text{elev} + \beta_3 * \text{h2o_area} + \beta_4 * \text{air_temp}_{(t)} + \beta_5 * \sin_{(t)} + \beta_6 * \cos_{(t)}, \quad (18)$$

and fit the model to the observed temperature data using the `ssnbayes()` function.

```
fit_ar <- ssnbayes(formula = temp ~ SLOPE + elev + h2o_area + air_temp + sin + cos,
                     data = clear,
                     path = path,
                     space_method = list("use_ssn", "Exponential.taildown"),
                     time_method = list("ar", "date"),
                     iter = 3000,
                     warmup = 1500,
                     chains = 3,
                     net = 2,
                     addfuncoll='afvArea',
                     refresh = max(iter/100,1))
```

Running this function takes several minutes and the progress of the sampler is shown during the execution. We stored the fitted model within **SSNdata**, which can be accessed using the code below if the reader wants to skip fitting the model.

```
fit_ar <- readRDS(system.file("extdata//fit_ar.rds", package = "SSNdata"))
```

The reader is referred to the Appendix for a second reproducible example using simulated data.

In the `ssnbayes()` function call shown above, the argument `formula` describes the regression model and is defined in the same way as in other model-fitting functions such as `lm`. We also pass a `data.frame` using the `data` argument, which must contain all of the variables specified in the `formula` argument. This `data.frame` should be in long format, with one row for each unique observation in space and time, which are also defined using `locID` and `pid`. In addition, the data set needs to be structured such that each spatial location has the same number of temporal observations. Such observations can be missing but should be denoted as such via "NA". In such cases, Bayesian imputation is used to obtain a complete data set. In other words, the `data.frame` has to contain all the combinations S and T . This can be done e.g. using the `tidy::complete()` function.

The `space_method` argument is a list containing information about the spatial modelling component. The first element specifies whether the topological information is stored in a `ssn` object or not ("use_ssn" or "no_ssn"), while the second list element specifies which spatial correlation model(s) to use. Options include tail-up ("Exponential.tailup", "LinearSill.tailup", "Spherical.tailup"),

tail-down ("Exponential.taildown", "LinearSill.taildown", "Spherical.taildown") and Euclidean ("Exponential.Euclid") models. It is possible to have more than one spatial covariance function per family (tail-up, tail-down and Euclidean distance). For instance: `space_method = list('use_ssn', c("Exponential.tailup", "Spherical.taildown"))`. However, care should be taken in this case to ensure identifiability of the model. If the user specifies `use_ssn` as the first element and the second element in the list is missing, then an "Exponential.tailup" model will be used by default. When a tail-up covariance function is specified, an additional column containing the additive function values used to compute the spatial weights must also be specified (e.g. `addfunccol ='afvArea'`).

The argument `net` specifies the network identifier when multiple networks are found within the same `ssn` object. Much less information is needed to fit traditional Euclidean covariance models and so a `ssn` object is not needed. Instead, the columns containing the spatial coordinates (e.g. longitude and latitude) must be included as a third element in the list: `space_method = list("no_ssn", "Exponential.Euclid", c("lon", "lat"))`.

The temporal part of the model is defined in a similar fashion using a list `time_method = list("method", "date")`. The first element defines the temporal model and options include an autoregressive model, "`ar`", defined in Eq 12 or a vector autoregression model, "`var`", defined in Eq 14). The second element is the variable defining the time points in the observation `data.frame`, which must be a discrete numeric variable. They should also be spaced at regular intervals, as expected in many time series models.

In `SSNbayes` the number of chains (`chains`), iterations (`i_iter`), and burn-in samples (`warmup`) can be specified. By default, `chains = 3`, `i_iter = 3000`, `warmup = 1500`. Thinning is also possible using the argument `thin`. Optionally, the `seed` parameter can be set to ensure reproducibility.

The `SSNbayes` package depends on `rstan`, which does not allow missing values in the data. Missing values in the response variable (left hand side element in the formula argument) will be automatically imputed in the `ssnbayes()` function. However, missing values in the covariates (right hand side elements in the formula argument) are not allowed. Instead, they must be imputed by the user before fitting the model. Many options for imputation can be found in <https://cran.r-project.org/web/views/MissingData.html>.

The `ssnbayes()` function shows the progress of the model fit and will be updated based on the number of samples specified using the `refresh` argument. At every iteration, the inverse of the spatial covariance matrix has to be computed, which takes a substantial amount of time for a large number of spatial locations and time points. Fitting this dataset using the `ssnbayes()` function took approximately 10 minutes on a laptop with an Intel Core i7-8650U CPU @ 1.90GHz and 16 Gb of memory.

3.6 Exploring results

The output from `ssnbayes()` is a `stanfit` object, which contains information about the fitted model and the MCMC chains for the parameters of interest. It can be summarized and visualized using generic functions such as `summary()` and `plot()`, or functions in the `ggplot2` package. We can also visualize the posterior distributions in the parameters of interest using the `mcmc_dens_overlay()` function from the R package `bayesplot` (Gabry and Mahr, 2018). The regression coefficients across three chains are shown in Figure 4).

```
library('bayesplot')
mcmc_dens_overlay(
  fit_ar,
  pars = paste0("beta[",1:7,"]"),
  facet_args = list(nrow = 1))
```

Apart from `h2o_area` (β_3), all of the estimated regression coefficients for covariates are substantially different from zero. The bulk of the posterior distribution of the autoregressive parameter (ϕ) was also far away from zero, suggesting a strong temporal dependence (Figure 5).

The `mcmc_dens_overlay()` function can also be used to visualise the posterior distributions of the spatial model parameters (σ_{TD}^2 and α_{TD}) and the nugget effect (σ_0^2) (Figure 6).

```
mcmc_dens_overlay(
  fit_ar,
  pars = c("var_td", "alpha_td", "var_nug"),
  facet_args = list(nrow = 1))
```

Notice that the median of the spatial range α_{TU} is approximately 200,000 m, indicating that spatial autocorrelation exists between locations that are less than 200 km apart.

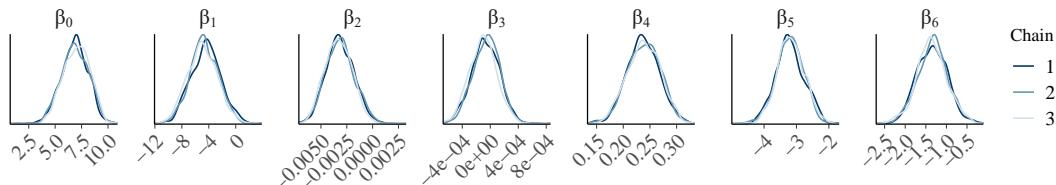


Figure 4: Posterior distributions of the regression coefficients for the linear model with seven covariates, including intercept (β_0), stream slope (β_1), elevation (β_2), watershed area (β_3), air temperature (β_4), sin of day of year (β_5), and cos of day of year (β_6). The distributions are shown for chains 1, 2, and 3, providing insights into the uncertainty and variability of the model coefficients.



Figure 5: Boxplot of the posterior distribution for the autoregression parameter, ϕ , estimated from the stream temperature time series data. The box represents the interquartile range (IQR). The distribution shows moderate uncertainty in estimating the value of ϕ .

3.7 Predictions

Ecological and environmental monitoring on stream networks generally produces data at discrete locations, which represent only a small section of the stream network. However, it is often desirable to make predictions in areas where data have not been collected to create spatially continuous maps (Isaak et al., 2017). In this section, we illustrate (1) how the model imputes missing values producing predictions, and (2) how to use the fitted model to predict in unsampled locations using a Kriging approach.

As mentioned previously, the `ssnbayes()` function imputes missing values in the response variable. The time series corresponding to the 18 spatial locations are shown in Fig 7. The model captures the periodic patterns in stream temperatures well, even in locations where most of the observations were missing (e.g. 8 and 12, Fig 7). We also compared the predictions produced by the model with the true latent hold-out data (Fig 8). If the model predictions were perfect we would expect points to fall on the diagonal line. The results suggest that the Bayesian model produces predictions that are similar to the true latent values. Most of the predictions (96%) were included within the 95% highest density interval, showing appropriate coverage of the predictions. The root mean square prediction error (RMSPE) between the true temperature values and the predictions was 0.510 °C, which is small considering the magnitude of variation in temperature values.

Additionally, in our case study, we want to produce temperature predictions at 60 locations generated using a systematic design ($\approx 1\text{km}$ apart). The function `predict()` produces predictions using information contained in the `stanfit` object obtained from `ssnbayes()`. The argument `nsamples` specifies the number of random samples to select from the posterior distributions and it must be smaller than or equal to the number of iterations `iter` specified in `ssnbayes()`.

```
# reading the prediction data
clear_preds <- readRDS(system.file("extdata/clear_preds.RDS", package = "SSNdata"))
pred <- predict(path = path,
                 obs_data = clear,
                 stanfit = fit_ar,
```

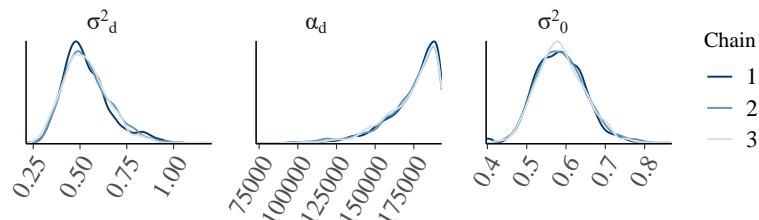


Figure 6: Posterior distributions of the spatial parameters, including the nugget effect (σ_0^2) and the spatial dependence (σ_d^2) in $^{\circ}\text{C}$, and the range parameter (α_d) in meters.

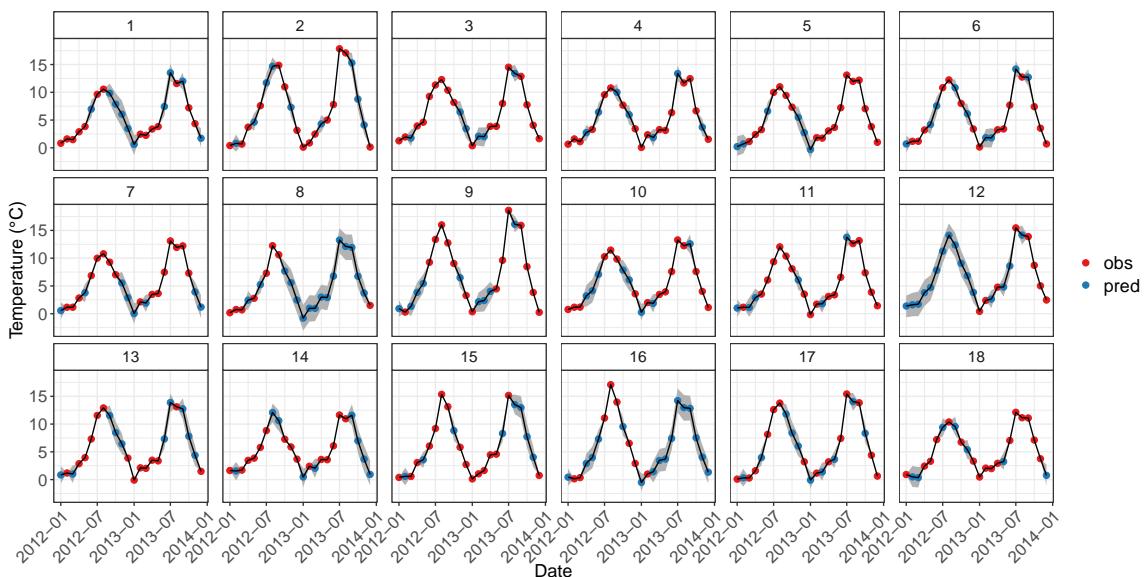


Figure 7: Time series of stream temperatures at 18 spatial locations. The observed points are represented in red, and the imputed or predicted values are shown in blue. The gray areas represent the 95% posterior credible intervals, providing an indication of the model’s uncertainty.

```

pred_data = clear_preds,
net = 2,
nsamples = 100, # number of samples to use from the posterior
addfunccol = 'afvArea', # additive function values
locID_pred = locID_pred,
chunk_size = 60)

```

The observation and prediction data frames (`data_obs`, `data_pred`, respectively) must be specified and must contain all of the covariates and the response variable specified in the “formula” argument in `ssnbayes()`.

Generally, producing subsets of predictions on the stream network is more efficient for big datasets, and can be parallelized. The argument `chunk_size` is used to define the size of the subsets. For instance, predictions in the case study in Santos-Fernandez et al. (2022) consist of more than 6000 locations. Performing matrix operations with a such large number of sites is not feasible. Instead, predictions were run in parallel using `chunk_size = 100`. `locID_pred` also allows the user to define a subset of prediction locations where predictions should be generated, as demonstrated in the example below. Similarly, the argument `seed` allows the user to set a seed so that the results are reproducible.

Figure 9 shows the predicted time series for the observation and prediction locations. The patterns in the prediction time series captured the seasonality in the observed data well. Figure 10 visualizes the predictions’ posterior mean temperature on the stream network. As expected, higher temperature values are obtained in the main stream channel, compared to predictions in small streams which generally are found at higher elevations.

Network exceedance probability

One advantage of using Bayesian inference is the ability to easily obtain various probabilistic estimates based on the model posterior predictive samples. In this example, we use the function `melt` from the R package `reshape2` (Wickham, 2007) to generate exceedance probabilities based on a critical thermal threshold of 13 °C for bull trout, a cold-water fish species that is sensitive to increased temperatures.

```

ys <- reshape2::melt(pred, id.vars = c('locID0', 'locID', 'date'), value.name ='y')
ys$iter <- gsub("[^0-9.-]", "", ys$variable)
ys$variable <- NULL
# network exceedance probability
limit <- 13
ys$exc <- ifelse(ys$y > limit , 1, 0)
ys <- data.frame(ys) %>% dplyr::group_by(date, locID, locID0) %>

```

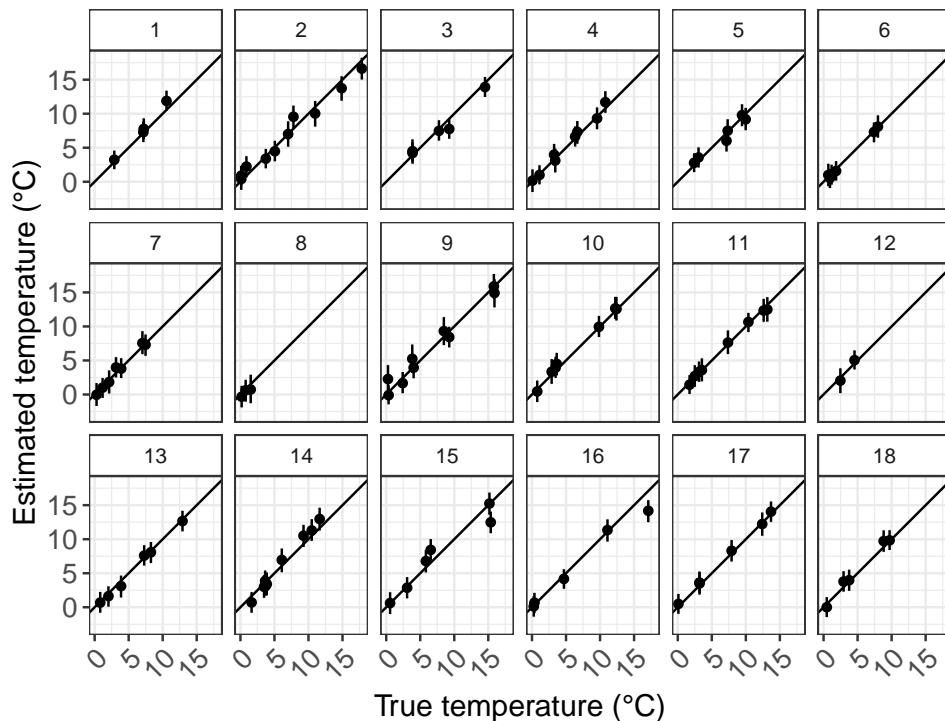


Figure 8: Scatterplot of predicted temperature versus the true latent values at the 18 spatial locations in the test dataset. The x and y axes represent the predicted and true temperature values, respectively. The vertical bars represent the 95% posterior credible intervals.

```
dplyr::summarise(sd = sd(y, na.rm=T),
                  y_pred = mean(y, na.rm=T),
                  prop = mean(exc, na.rm=T)) %>%
dplyr::arrange(ys, locID)
clear_preds <- clear_preds %>% left_join(ys, by = c('locID', 'date' ))
```

Figure 11 shows the exceedance probabilities for all 60 prediction locations on two dates, obtained from the posterior predictive distributions. Knowledge about when and where biologically relevant thermal thresholds are likely to be exceeded provides critical information for the management of threatened and endangered freshwater species (Isaak et al., 2016).

Other useful functions

Users often want to extract distance matrices and spatial weights so that they can be analysed and/or visualised in other R packages or external software e.g. McGuire et al. (2014). The `dist_weight_mat()` and `dist_weight_mat_preds()` produce a list of distance and weight matrices with the following elements:

1. `e`: Euclidean distance matrix containing the distances between locations
2. `D`: Downstream distance.
3. `H`: Total stream distance.
4. `w.matrix`: spatial weights for flow connected locations. This matrix is used in the tail-up models.
5. `flow.con.mat`: flow connected matrix. Indicates whether two locations in the network are connected by flow.

The `dist_weight_mat()` function produces matrices of the distances and weights between observation locations, with dimensions equal to the number of observation locations ($n_o \times n_o$). The `dist_weight_mat_preds()` function produces the same information for observed and prediction locations, with $n_o n_p \times n_o n_p$ dimensions. Details and detailed descriptions of the computation of these matrices can be found in Peterson and Ver Hoef (2010), Ver Hoef et al. (2014), and Santos-Fernandez et al. (2022).

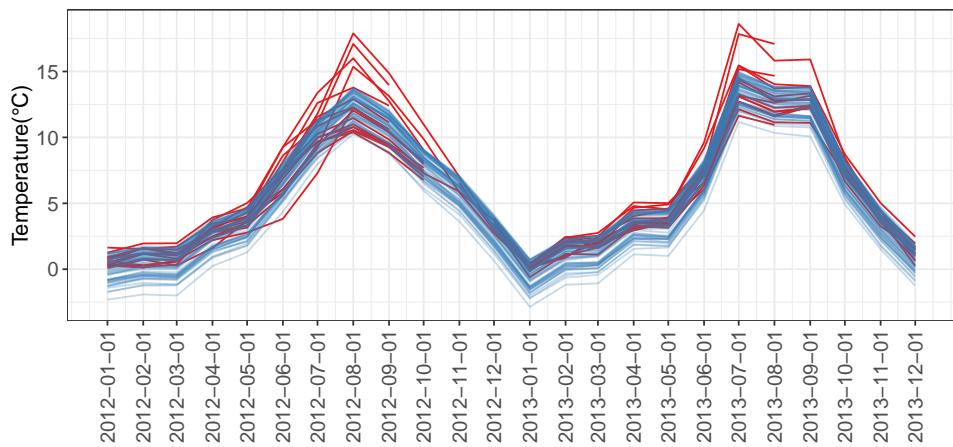


Figure 9: Time series plot comparing predicted (blue lines) and observed (red lines) temperature values over time.

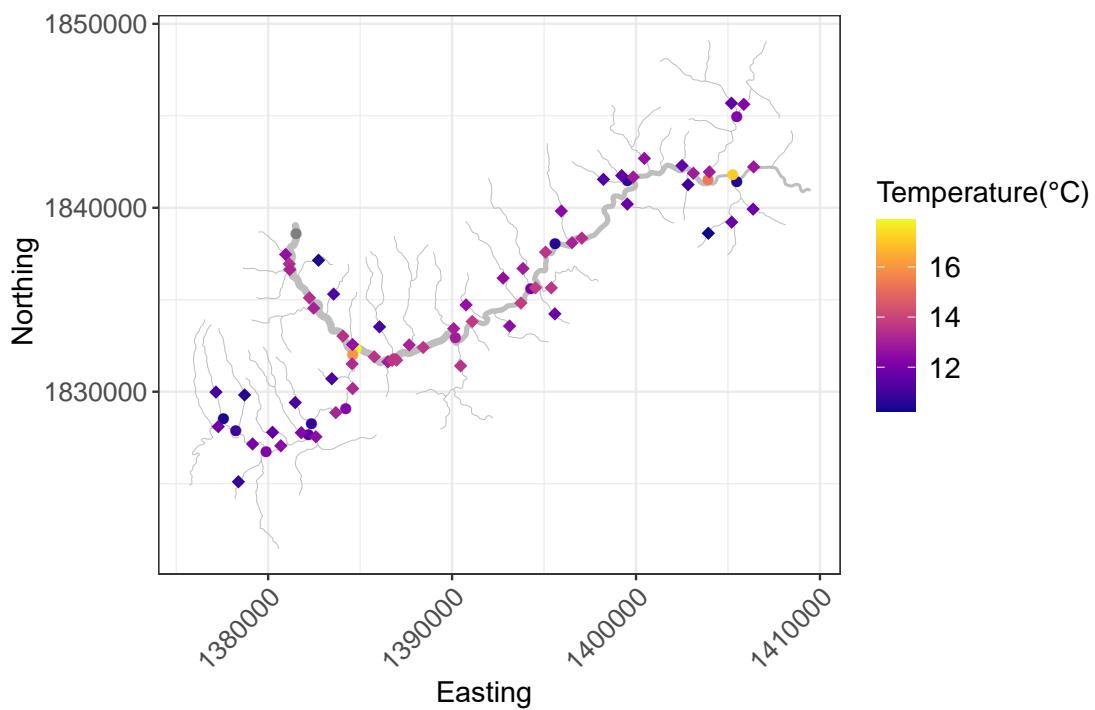


Figure 10: Mean daily stream temperature predictions (diamonds) for each of the 60 spatial locations and observations (circles) in the Clearwater network on August 1st, 2012.

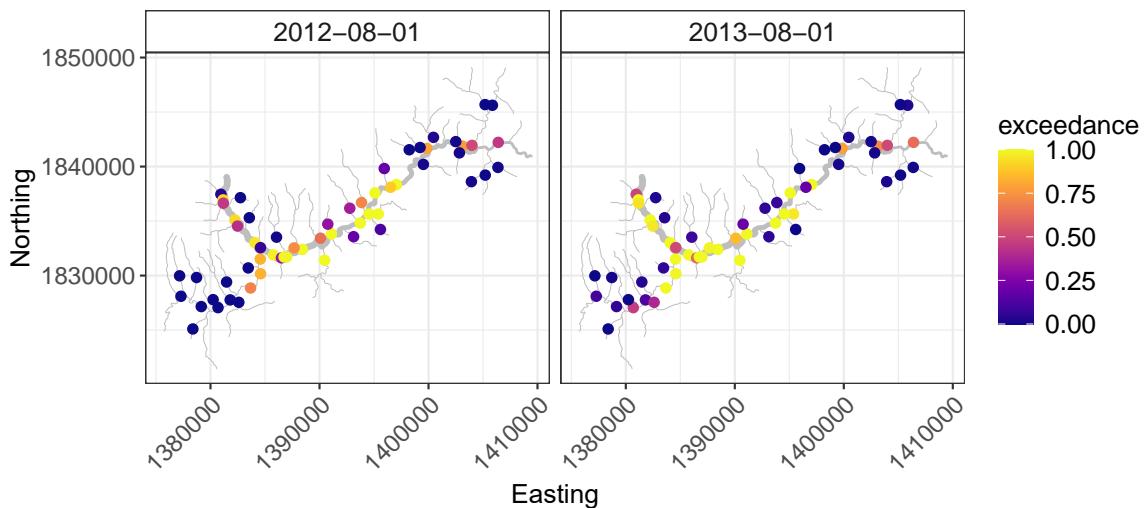


Figure 11: Probabilities that the mean stream temperature will exceed the 13 °C threshold on August 1, 2012, and August 1, 2013, respectively across the 60 prediction locations in the Clearwater network.

4 Discussion and conclusions

The growing popularity of stream sensor arrays, in which repeated observations are taken at multiple sites, requires models capable of accounting for spatial and temporal autocorrelation in stream network data. However, there are only a limited number of computational methods and software packages designed to account for the unique spatial dependence found in stream data (e.g. the R packages **SSN**, **SSN2** and **smnet**). The package described in the present paper extends the models implemented in **SSN** by accounting for temporal dependence using Bayesian inference, which offers several benefits. Enhanced features from this package and other benefits from the use of a Bayesian framework include the computation of probabilistic estimates and network exceedance probabilities, the ability to incorporate prior information, and the estimation of the proportion of degraded habitat.

We tested the performance of **SSNbayes** in multiple scenarios with simulated and real data and found that the parameters are well estimated and the predictions are accurate in terms of RMSPE. We also validated the results from a wide range of spatial model combinations to those obtained using **SSN** based on simulated data. Spatial and spatio-temporal models tend to be slow to fit and computationally intensive, which becomes more challenging within a Bayesian modelling framework. This can become computationally prohibitive when the number of spatial locations is large because the spatial covariance matrix must be iteratively inverted. We are currently exploring alternative methods (such as variational Bayes) to be implemented within **SSNbayes**.

Future implementations will incorporate other modelling variations. Two of them are: (I) expressing ϕ_s as a linear combination of covariates such as elevation and watershed (an extension of Case 2), and (II) using a 2-Nearest Neighbours (2-NN) method, where the off-diagonal elements of Φ are different from zero in the two closest, allowing temporal dependence to be established between neighbouring spatial locations connected by flow (Santos-Fernandez et al., 2022). However, other space-time covariance structures could also be implemented for stream network data, which allow more modelling flexibility. For example, this implementation is based on a vector autoregression structure, but other models such as moving averages and ARIMA could also be considered. In addition, we currently assume that the response variable is normally distributed, but other regression models could be implemented by modifying the likelihood function in **ssnbayes()**. We are also actively working on the development and implementation of models for anomaly detection in stream data (Santos-Fernandez et al., 2023). The **SSNbayes** package is under constant development and new features and model implementations are on their way.

Acknowledgement

This research was supported by the Australian Research Council (ARC) Linkage Project “Revolutionising water-quality monitoring in the information age” (ID: LP180101151) and the Centre of Excellence for Mathematical and Statistical Frontiers (ACEMS). JMM was supported by an Australian Research Council Discovery Project (DP200101263). We thank Dona Horan for the creation of the spatial stream network (.ssn) object. Data analysis and computations were undertaken using the packages `rstan` (Stan Development Team, 2018). Data visualizations were made with the packages `ggplot2` (Wickham, 2016) and `bayesplot` (Gabry and Mahr, 2018).

Appendix

In this appendix, we illustrate the fit of a spatio-temporal stream network model within the Bayesian framework with `SSNbayes` using a simulated example. These results can be reproduced using the Kaggle notebook <https://www.kaggle.com/code/edsans/ssnbayes-simulated>. Several R packages must be installed to successfully reproduce the simulation described in this section. Installing the `rstan` can be tricky because you need to configure your R installation to be able to compile C++ code. If you have not used the `rstan` before, please see <https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started> for easy-to-follow instructions about how to install the package.

Using simulated data

We generated some spatial data with the `SSN` package using a systematic design for the locations of the observations and predictions. Note that the `SSN` package has been archived on CRAN.

```
## Load the packages. Note that these packages can be installed
## using the install.packages() function
library('tidyverse')
library('Rcpp')
library('StanHeaders')
library('rstan')
library('abind')
library('SSN')
library('SSN2')
library('bayesplot')
library('lubridate')
library('viridis')
library('ggrepel')
library('devtools')
library('RColorBrewer')
if(!require('SSNbayes')) install.packages("SSNbayes", dependencies = T)
library('SSNbayes')

## Set some useful options for modelling
rstan_options(auto_write = TRUE) # avoid recompilation
options(mc.cores = parallel::detectCores())
RNGkind(sample.kind = "Rounding")

## Set the seed for reproducibility
seed <- 202008
set.seed(seed)

## Set the path for the SpatialStreamNetwork object created in the next step
path <- "./sim.ssn"

## If it does not already exist, create a SpatialStreamNetworkObject
## with 150 stream segments (edges). Use a systematic design to generate
## observed and prediction locations on the network spaced
## approximately 3 and 0.3 units apart, respectively.
if(file.exists(path)){
  ssn <- importSSN(path, "preds")
} else{ssn <- createSSN(n = c(150), # 150 edges
```

```

obsDesign = systematicDesign(spacing=3),
predDesign = systematicDesign(spacing=0.3),
importToR = TRUE,
path = path, # path where the sns object is saved
treeFunction = iterativeTreeLayout)}

```

This produces a SpatialStreamNetwork object with 50 observation locations, which is our training dataset. We also generated 499 prediction locations for testing.

```

## Plot the edges and observed locations in the SpatialStreamNetwork object
plot(ssn, lwdLineCol = "addfunccol", lwdLineEx = 8,
     lineCol = 4, col = 1, pch = 16, xlab = "x-coordinate", ylab = "y-coordinate")

## Add the prediction locations
plot(ssn, PredPointsID = "preds", add = T, pch = 16, col = "#E41A1C")

```

Consider a response variable such as stream temperature. The aim is to predict this response variable in the testing dataset borrowing information from other measurements across space and time and using covariates such as air temperature. Predictions can be made at a subset of locations or across all prediction locations on the whole network.

Distances within and between observation and prediction locations must be generated before model fitting.

```

## Create stream distance matrices
createDistMat(ssn, predpts = 'preds', o.write=TRUE, amongpreds = T)

```

We then need to simulate some data using some covariates, regression coefficients and a covariance structure.

```

## Extract the data.frames for the observed and prediction location data
rawDFobs <- getSSNdata.frame(ssn, Name = "Obs")
rawDFpred <- getSSNdata.frame(ssn, Name = "preds")

## Extract the geographic coordinates from the SpatialStreamNetwork
## object and add to data.frames
obs_data_coord <- data.frame(ssn@obspoints@SSNPoints[[1]]@point.coords)
obs_data_coord$pid<- as.numeric(rownames(obs_data_coord))
rawDFobs %>% left_join(obs_data_coord, by = c("pid"),
                           keep = FALSE)
rawDFobs$point <- "Obs" ## Create label for observed points

pred_data_coord <- data.frame(ssn@predpoints@SSNPoints[[1]]@point.coords)
pred_data_coord$pid<- as.numeric(rownames(pred_data_coord))
rawDFpred %>% left_join(pred_data_coord, by = "pid",
                           keep = FALSE)
rawDFpred$point <- "pred" ## Create label for prediction points

## Generate 3 continuous covariates at observed and prediction locations
set.seed(seed)
rawDFpred[, "X1"] <- rnorm(length(rawDFpred[, 1]))
rawDFpred[, "X2"] <- rnorm(length(rawDFpred[, 1]))
rawDFpred[, "X3"] <- rnorm(length(rawDFpred[, 1]))

rawDFobs[, "X1"] <- rnorm(length(rawDFobs[, 1]))
rawDFobs[, "X2"] <- rnorm(length(rawDFobs[, 1]))
rawDFobs[, "X3"] <- rnorm(length(rawDFobs[, 1]))

## Ensure the rownames still match the pid values used in the
## SpatialStreamNetwork object
rownames(rawDFobs)<- as.character(rawDFobs$pid)
rownames(rawDFpred)<- as.character(rawDFpred$pid)

## Put the new covariates back in the SpatialStreamNetwork object
ssn <- putSSNdata.frame(rawDFobs, ssn, Name = 'Obs')
ssn <- putSSNdata.frame(rawDFpred, ssn , Name = 'preds')

```

```

## Simulate the response variable at observed and prediction locations
set.seed(seed)
sim.out <- SimulateOnSSN(ssn.object = ssn,
                           ObsSimDF = rawDFobs, ## observed data.frame
                           PredSimDF = rawDFpred, ## prediction data.frame
                           PredID = "preds", ## name of prediction dataset
                           formula = ~ X1 + X2 + X3,
                           coefficients = c(10, 1, 0, -1), ## regression coefficients
                           CorModels = c("Exponential.taildown"), ## covariance model
                           use.nugget = TRUE, ## include nugget effect
                           CorParms = c(3, 10, .1)) ## covariance parameters

## Extract the SpatialStreamNetwork object from the list returned by
## SimulateOnSSN and extract the observed and prediction site
## data.frames. Notice the new column Sim_Values in the data.frames
sim.ssn <- sim.out$ssn.object
simDFobs <- getSSNdata.frame(sim.ssn, "Obs")
simDFpreds <- getSSNdata.frame(sim.ssn, "preds")
summary(simDFobs)

```

The SpatialStreamNetwork object we created only contains one simulated response for each observation and prediction location and we can fit a spatial statistical model to the simulated data using the `glmssn` function in the [SSN](#) package.

```

## Fit a spatial stream network model using the Exponential tail-down function
glmssn.out <- glmssn(Sim_Values ~ X1 + X2 + X3, sim.ssn,
                      CorModels = "Exponential.taildown")
summary(glmssn.out)

```

In order to fit a space-time model using the [SSNbayes](#) package, we need repeated measurements at each location. We now need to generate some time series with AR(1) error structure:

```

## Create a data.frame containing training and test data.
df_obs <- getSSNdata.frame(sim.ssn, "Obs") ## Extract observed dataset
df_obs$dataset <- 'train' ## Create new column 'dataset' and set to 'train'
df_pred <- getSSNdata.frame(sim.ssn, "preds") ## Extract prediction dataset
df_pred$dataset <- 'test' ## Create new column 'dataset' and set to 'test'

## Expand data.frames to include 10 days per location
t <- 10 # days
df_obs <- do.call("rbind", replicate(t, df_obs, simplify = FALSE))# replicating the df
df_obs$date <- rep(1:t, each = (nrow(df_obs)/t)) # Set date variable

df_pred <- do.call("rbind", replicate(t, df_pred, simplify = FALSE))# replicating the df
df_pred$date <- rep(1:t, each = (nrow(df_pred)/t)) # Set date variable

## Create a copy of the pid value used in the SpatialStreamNetwork
## object and create a new pid value for use in SSNbayes
## package. Values must be consecutively ordered from 1 to the number
## of rows in the data.frame
df_obs <- df_obs %>% mutate(pid.ssn = pid,
                               pid = rep(1:nrow(.)))
df_pred <- df_pred %>% mutate(pid.ssn = pid,
                               pid = rep(1:nrow(.)))

## Combine the training and testing datasets
df <- rbind(df_obs, df_pred)
df$dataset <- factor(df$dataset, levels = c('train', 'test'))

## Construct and initialize an autocorrelation structure of order 1
set.seed(seed)
phi <- 0.8 ## lag 1 autocorrelation value
ar1 <- corAR1(form = ~ unique(df$date), value = phi) # can also use corExp function
AR1 <- Initialize(ar1, data = data.frame(unique(df$date)))

```

```

## Create a vector of AR1 errors for each date and expand to all locations
epsilon <- t(chol(corMatrix(AR1))) %*% rnorm(length(unique(df$date)), 0, 3) #NB AR1 error
epsilon <- rep(epsilon, each = length(unique(df$locID)) ) +
  rnorm(length(epsilon)*length(unique(df$locID)), 0, 0.25) # for all the locations
epsilon_df <- data.frame(date = rep(unique(df$date), each = length(unique(df$locID))),
                         locID = rep(unique(df$locID), times = length(unique(df$date))),
                         epsilon = epsilon)
df <- df %>% left_join(epsilon_df, by = c('date' = 'date', 'locID' = 'locID'))

## Create a new simulated response variable, y, with errors added
df$y <- df$Sim_Values + df$epsilon

```

We can visualize the time series of the new simulated response at the observed and predicted locations over time:

```

## Create line plots of response over time for training and test datasets
ggplot(df) +
  geom_line(aes(x = date, y = y, group = locID, col = dataset), alpha = 0.4) +
  ylab("Simulated Temperature (\u00B0C)")+
  facet_wrap(~dataset)+
  theme_bw()

```

Figure 12 shows the stream temperature time series in the observations and predictions datasets.

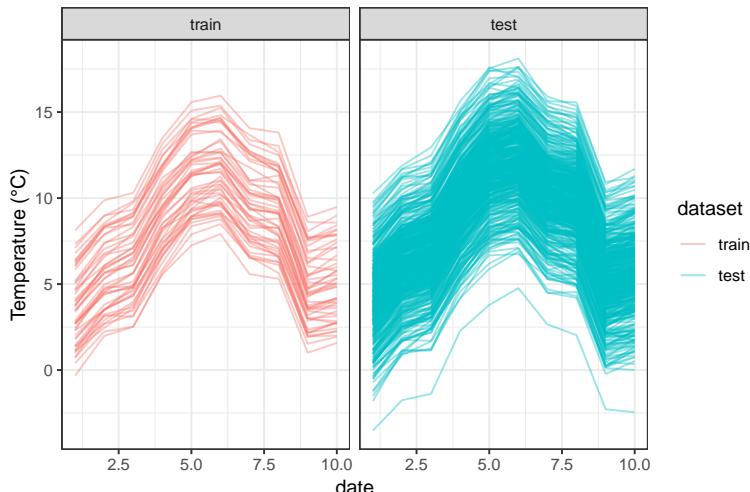


Figure 12: Evolution of the stream temperature time series in the observations and predictions datasets. Each time series represent a spatial location.

```

## Split the training and testing datasets. Ensure that date is numeric.
df <- df %>% dplyr::select(locID, pid, date, y, everything())
obs_data <- df[df$dataset == 'train',]
pred_data <- df[df$dataset == 'test',]
# NB: the order in this data.frame MUST be: spatial locations (1 to S) at time t=1,
# then locations (1 to S) at t=2 and so on.

```

The first prediction option in **SSNbayes** is to generate predictions for locations in the observed dataset with missing data. To demonstrate, let us set approximately 30% of the observations per date to missing, make predictions, and assess how well we retrieve the actual temperature values.

```

# Randomly select observations by date
set.seed(seed)
points <- length(unique(obs_data$pid))
locs <- obs_data %>% dplyr::group_by(date) %>%
  pull(pid) %>%
  sample(., round(points * 0.3), replace = F) %>% sort()

## Create a backup for the response before setting randomly selected

```

```
## measurements to NA
obs_data$y_backup <- obs_data$y
obs_data[obs_data$pid %in% locs,]$y <- NA
```

Let us visualize the network with the time series of observed temperature values. First we use the `collapse()` function to extract the network structure from the the `SpatialStreamNetwork` object. The facets (1-10) represent the date and there are a total of 150 missing observations (gray dots) which are observations that we set to missing to assess the model predictive accuracy.

```
## Extract stream (edge) network structure, including the additive function value
nets <- SSNbayes::collapse(ssn, par = 'addfunccol')

## Create additive function value categories for plotting
## Create additive function value categories for plotting
nets$afv_cat <- cut(nets$addfunccol,
                     breaks = seq(min(nets$addfunccol),
                                   max(nets$addfunccol),
                                   length.out=6),
                     labels = 1:5,
                     include.lowest = T)

## Plot simulated temperature, by date, with line width proportional to afv_cat
ggplot(nets) +
  geom_path(aes(X1, X2, group = slot, size = afv_cat), lineend = 'round',
            linejoin = 'round', col = 'lightblue')+
  geom_point(data = dplyr::filter(obs_data, date %in% 1:10),
             aes(x = coords.x1, y = coords.x2, col = y, shape = point),
             size = 1)+

  scale_size_manual(values = seq(0.2,2,length.out = 5))+

  facet_wrap(~date, nrow = 2)+

  scale_color_viridis(option = 'C')+


  scale_shape_manual(values = c(16,15))+

  xlab("x-coordinate") +
  ylab("y-coordinate")+
  theme_bw()
```

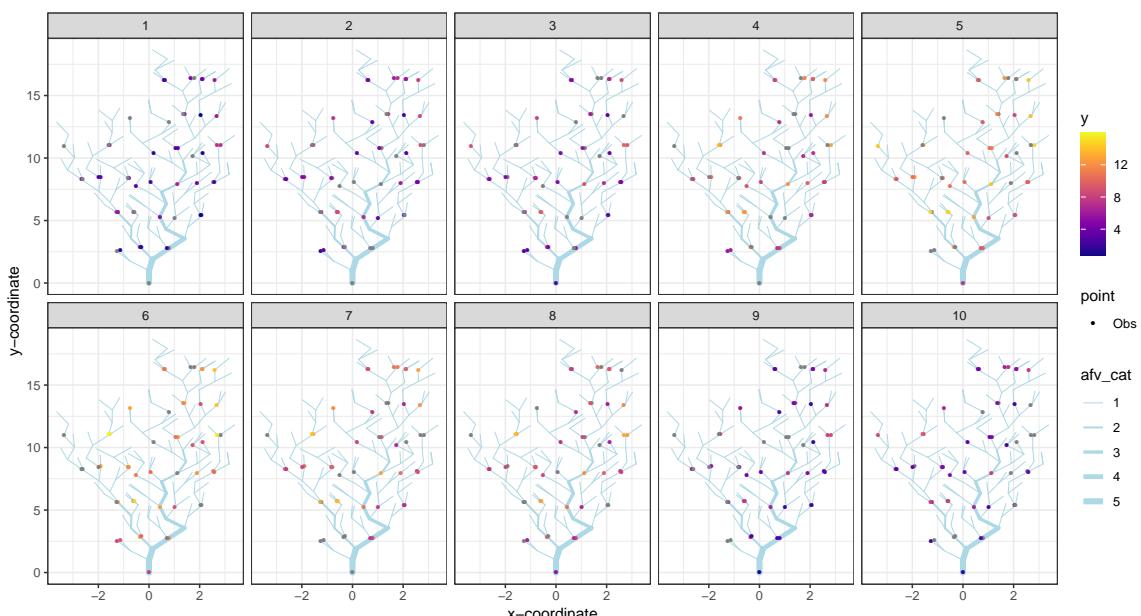


Figure 13: Evolution of the stream temperature time series in the observation dataset across ten time points.

We then fit a Bayesian space-time model, with a tail-down covariance model and an AR(1) error structure using the `ssnbayes()` function. Fitting the model took about 15 minutes on a laptop (i7 @1.80 GHz and 16GB RAM).

```
## Fit a Bayesian space-time model
fit_td <- ssnbayes(formula = y ~ X1 + X2 + X3,
                     data = obs_data,
                     path = path,
                     time_method = list("ar", "date"), # temporal model to use
                     space_method = list('use_ssn', c("Exponential.taildown")), # spatial model
                     iter = 4000,
                     warmup = 2000,
                     chains = 3,
                     addfuncoll = 'addfuncoll',
                     loglik = T)

## Create a copy of the model fit and set class so that we can take
## advantage of plotting functions for stanfit objects
fits <- fit_td
class(fits) <- c("stanfit")

## Extract summaries of the posterior distributions for the parameter
## estimates and the predictions.
stats_td <- summary(fits)
stats_td <- stats_td$summary
```

One of the main benefits of this Bayesian approach is that the model produces probabilistic estimates. Figures 14 and 15 show the posterior distributions of the four parameters in the spatio-temporal model (σ_{TU}^2 , σ_0^2 , α and ϕ) and the regression coefficients (intercept and slopes). The trace plots for of these parameters can be found in the Appendix .0.1.

```
## Create plots of the posterior distribution of the 3 regression
## coefficients
mcmc_dens_overlay(
  fits,
  pars = paste0("beta[", 1:3, "]"),
  facet_args = list(nrow = 1))

## Plot the posterior distribution of phi
mcmc_intervals(
  fits,
  pars = paste0("phi"),
  point_size = .1,
  prob_outer = 0.95
)

## Plot the posterior distribution for the nugget effect, partial sill.
## and range parameters in the tail-down model
mcmc_dens_overlay(
  fits,
  pars = c(
    "var_td",
    "alpha_td",
    "var_nug"),
  facet_args = list(nrow = 1)
)
```

We then assess how accurate the predictions of the missing temperature values are compared to the true held out values.

```
## Create a data.frame containing summaries of the posterior predictive
## distributions and the true values
ypred <- data.frame(stats_td[grep("y\\\[", row.names(stats_td)),])
ypred$ytrue <- obs_data$y_backup #
ypred$date <- rep(1:t, each = nrow(obs_data)/t)
```

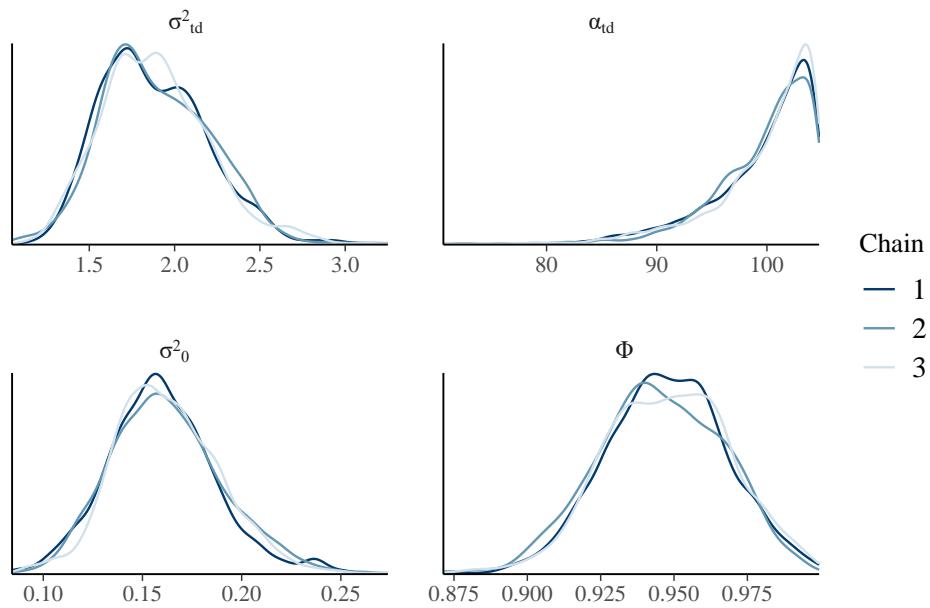


Figure 14: Posterior densities of the partial sill (σ_{TU}^2), nugget effect (σ_0^2), range (α) and temporal autocorrelation ϕ parameters. These parameters are crucial for understanding the spatial and temporal variability in the simulated stream network dataset.

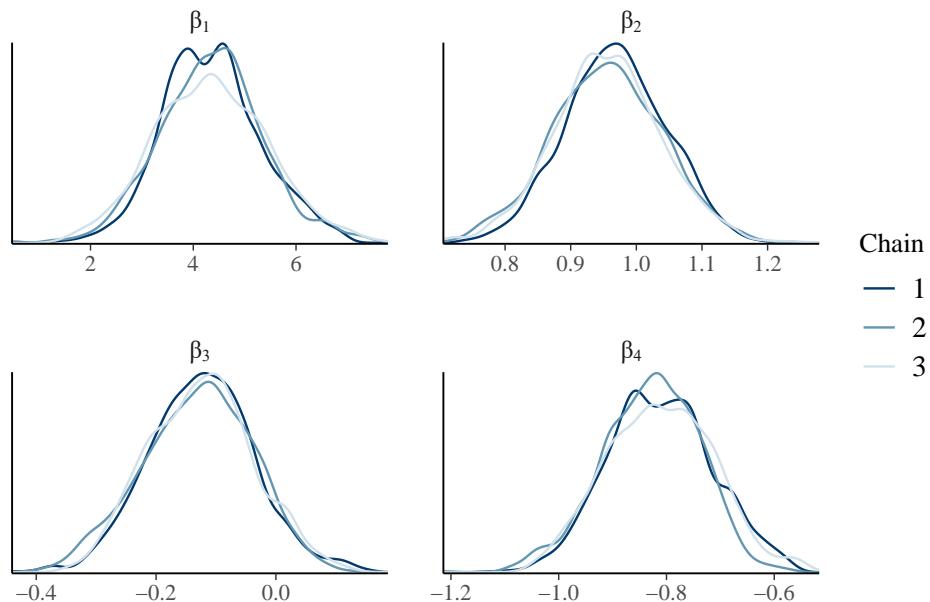


Figure 15: Posterior densities of the regression coefficients (β) for the intercept (β_1) and the three covariates X1, X2, and X3 (β_2 , β_3 , and β_4 , respectively).

```

ypred$dataset <- ifelse(ypred$sd == 0, 'obs', 'pred')
ypred$td_exp <- ypred$mean

## Create a plot of the predicted versus true values with 95% highest
## density interval
filter(ypred, dataset == 'pred') %>% ggplot() +
  geom_errorbar(data = ypred, aes(x=ytrue, ymin=X97.5., ymax=X2.5.),
                col = 2, width=0.5, size=0.5, alpha = 0.75) +
  geom_point(aes(x = ytrue , y = td_exp), col = 2)+ 
  geom_abline(intercept = 0, slope = 1)+ 
  facet_wrap(~date, nrow = 2) + coord_fixed() +
  xlab('y true') + ylab('y estimated') + theme_bw()

## Calculate the root mean square error (RMSE) between the true and
## predicted values
rmse <- sqrt(mean(((ypred$ytrue) - ypred$td_exp)^2))
rmse

# Calculate the 95% prediction coverage. Ideally, this should be close to 0.95.
ypred$cov <- ifelse(ypred$ytrue > ypred$X2.5. & ypred$ytrue<ypred$X97.5,1,0)
filter(ypred, dataset == 'pred') %>%
  group_by(dataset) %>%
  dplyr::summarize(mean(cov))

```

In Figure 17, we have plotted the means of the imputed values (with a 95% prediction interval) against the true values of the missing data. As can be seen, the means of each imputed value largely agree with the true value, with most prediction intervals (96%) containing the true value. The RMSE of the predicted temperature (y) is 0.245 which is small compared to the magnitude of y (Mean = 7.621 and standard deviation = 3.489 degrees). Figure 16 shows the posterior densities of the predicted temperature (y) for 8 measurements in the testing set.

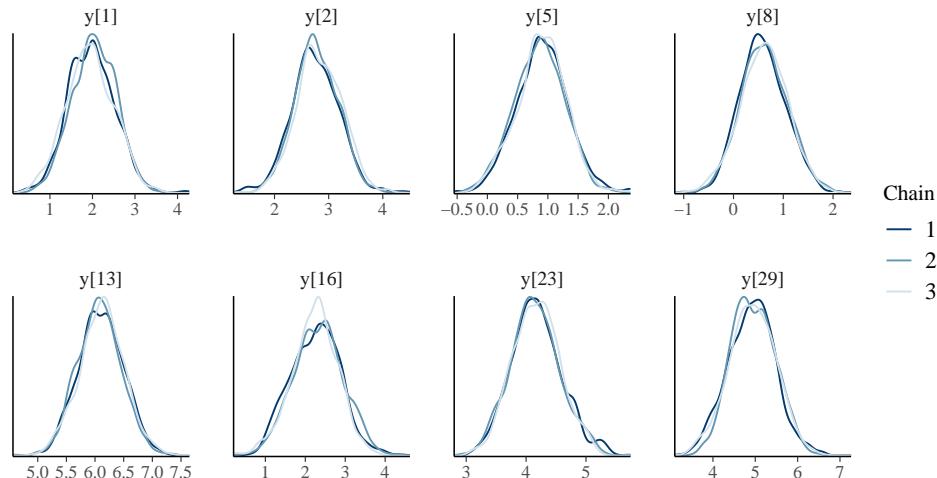


Figure 16: Posterior densities for 8 temperature predictions (y).

0.1 Predictions on a new set of prediction locations

Environmental and ecological monitoring of stream networks often requires estimation of the response variable of interest across the whole river network. In this section we illustrate how to use the fitted model to predict at a new set of locations using Kriging. We will produce temperature predictions at the 499 locations we generated previously using a systematic design.

```

## Making predictions at a new set of prediction locations
## Set the seed for reproducibility

```

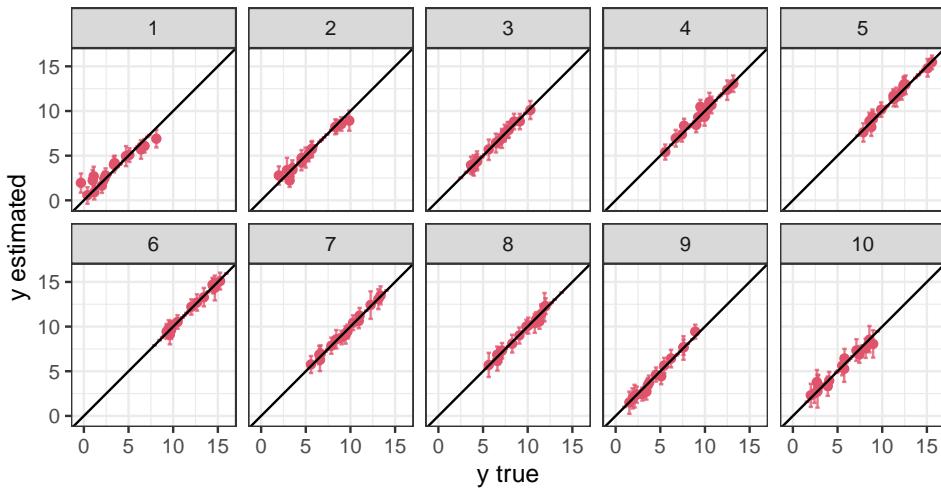


Figure 17: Scatterplot of predicted (i.e., estimated) temperature values versus the true latent values. The error bars represent the 95% highest density interval, indicating the range of the most plausible true values given the predictions.

```

set.seed(seed)

## Extract the location IDs for prediction locations
locID_pred <- sort(unique(pred_data$locID))

## Define a new column containing the response variable used in
## fit_td, including missing values
obs_data$y_traintest <- obs_data$y

## Replace y with the original response variable, containing no
## missing values
obs_data$y <- obs_data$y_backup

## Produce predictions: This takes approximately 8 minutes
pred <- predict(object = fit_td, ## fitted model
                 path = path, # path to .ssn object
                 obs_data = obs_data, # observed data.frame
                 pred_data = pred_data, # prediction data.frame
                 net = 1, # network identifier (optional)
                 nsamples = 100, # number of samples to use from the posterior
                 addfunccol = 'addfunccol', # variable used for spatial weights
                 locID_pred = locID_pred, # location identifier for predictions
                 chunk_size = 60) # split the predictions into subsets of this size

## Convert the prediction data.frame from wide to long format
ys <- reshape2::melt(pred, id.vars = c('locID0',
                                         'locID', 'date'), value.name ='y')

## Create variable representing the iteration number and set variable
## column to NULL
ys$iter <- gsub("[^0-9.-]", "", ys$variable)
ys$variable <- NULL

```

As we are using a simulated dataset, we can compare the out-of-sample predictions with the true latent values in the same way we did previously (Figure 18).

We then visualize the posterior mean of the predictions on the full network (Figure 19).

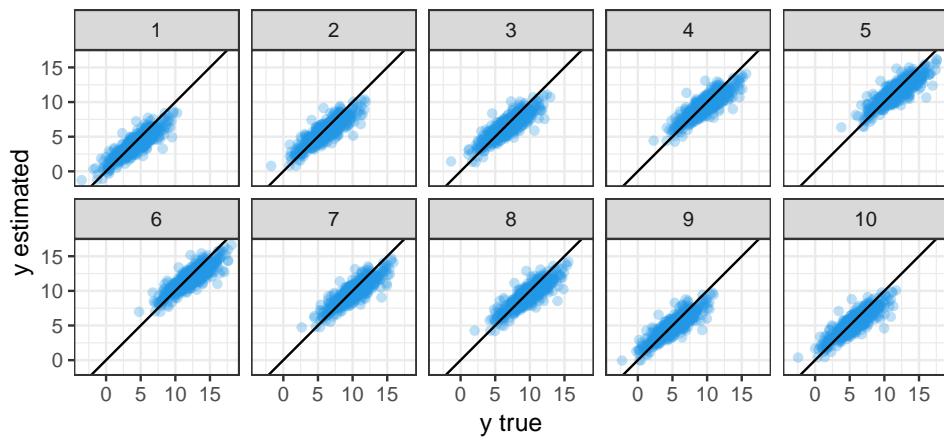


Figure 18: Scatter plot of predicted (i.e. estimated) temperature versus the true latent values. The plot shows a strong linear relationship between the two, indicating good agreement between the true and estimated values.

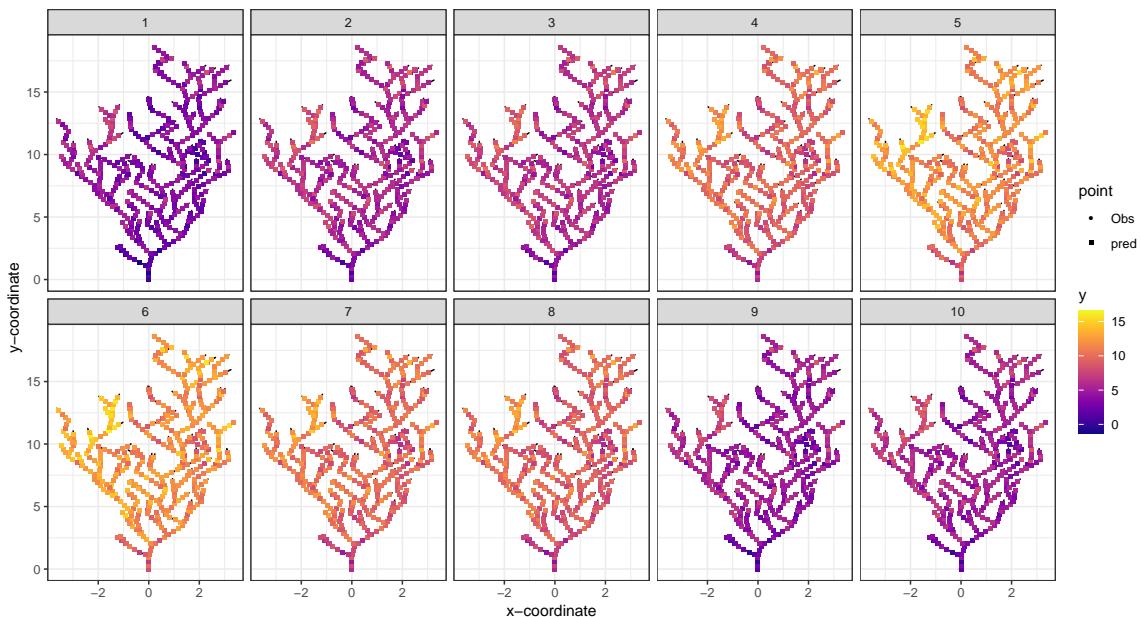


Figure 19: Evolution of observed and predicted stream temperatures over the 10 dates. The plot highlights the variability in water temperatures over time.

Exceedance probabilities throughout the network

It is straightforward to make probabilistic statements about the predictions based on samples from the posterior predictive distribution. In this example, we calculate the exceedance probabilities at each prediction location based on a biological threshold of 13 °C.

```
## Computing the exceedance probabilities

## Create an exceedance indicator based on a threshold (i.e. limit)
limit <- 13
ys$exc <- ifelse(ys$y > limit , 1, 0) ## 1== TRUE, 0== FALSE

## Calculate summary statistics for the predictions, by locID and date
## and join to prediction data.frame
ys <- data.frame(ys) %>% dplyr::group_by(date, locID) %>%
  dplyr::summarise(sd = sd(y, na.rm=T), ## prediction standard deviation
                   y_pred = mean(y, na.rm=T), ## mean temperature prediction
                   prop = mean(exc, na.rm=T)) ## exceedance probability

ys <- dplyr::arrange(ys, locID)
pred_data <- pred_data %>% left_join(ys, by = c('locID', 'date'), keep = FALSE)

## Plot the exceedance probabilities for the prediction sites on 10 dates
ggplot(nets) +
  geom_path(aes(X1, X2, group = slot, size = afv_cat), lineend = 'round',
            linejoin = 'round', col = 'gray')+
  geom_point(data = dplyr::filter(pred_data) ,
             aes(x = coords.x1, y = coords.x2, col = prop), size = 1)+
  scale_size_manual(values = seq(0.2,2,length.out = 5))+
  facet_wrap(~date, nrow = 2)+
  scale_color_viridis(option = 'C')+
  scale_shape_manual(values = c(200))+
  xlab("x-coordinate") +
  ylab("y-coordinate")+
  coord_fixed()+
  theme_bw()+
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=13),
        legend.text=element_text(size=13),
        legend.title=element_text(size=13),
        strip.text.x = element_text(size = 13),
        axis.text.x = element_text(angle = 45, hjust=1),
        strip.background =element_rect(fill='white'))+
  guides(size = 'none')+
  labs(size="", colour="exceedance")
```

Figure 20 shows the time series of the exceedance probabilities at prediction locations obtained from the posterior predictive distributions. Knowing when and where temperature or other water quality variables are likely to exceed critical thresholds provides valuable information for prioritizing management and conservation activities.

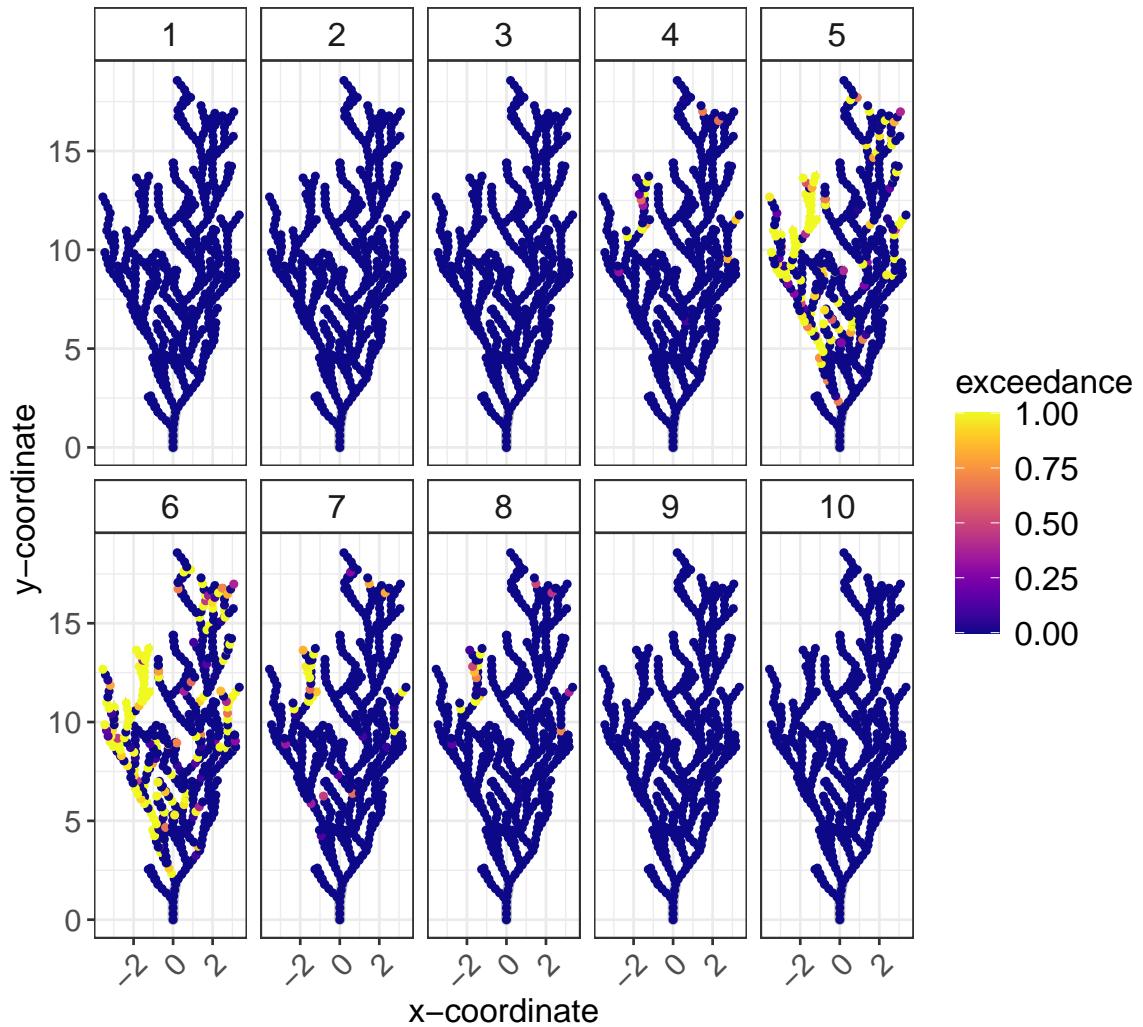


Figure 20: Network exceedance probabilities across the ten dates. The yellow regions represent the areas where the probability of exceeding the limit is high.

Other results

Trace plots are often used to visually assess convergence when parameters are estimated in a Bayesian model. These can be generated easily from the `stanfit` object we saved earlier (`fits`) and the `traceplot` function in `rstan`.

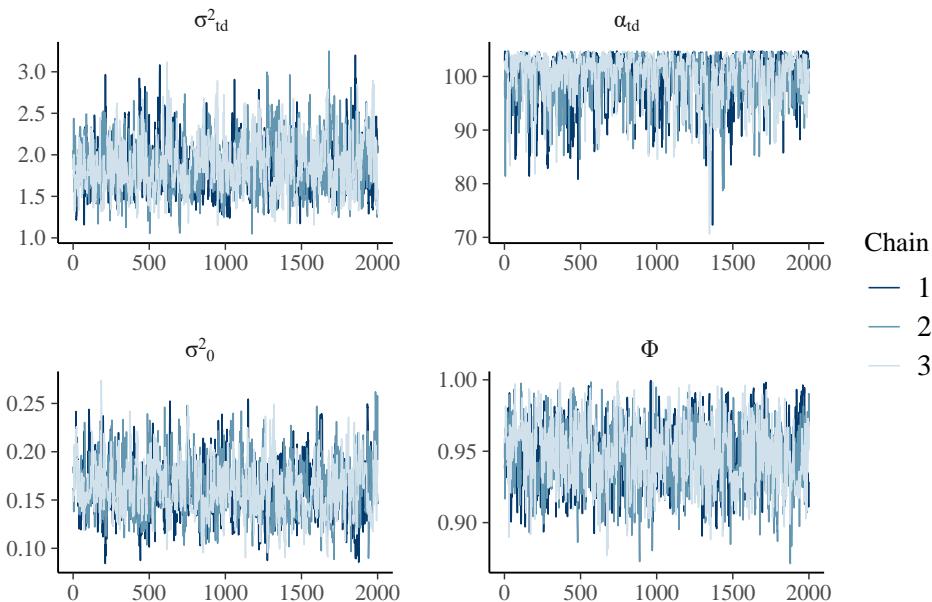


Figure 21: Trace plots of the spatial covariance parameters including the partial sill (σ_{TD}^2), nugget effect (σ_0^2), range (α), and the temporal autocorrelation parameter, ϕ , showing good mixing of the chains.

References

- G. Bal, E. Rivot, J.-L. Baglinière, J. White, and E. Prévost. A hierarchical Bayesian model to quantify uncertainty of stream water temperature forecasts. *PLoS One*, 9(12):e115659, 2014. [p34]
- S. Banerjee, B. P. Carlin, and A. E. Gelfand. *Hierarchical modeling and analysis for spatial data*. CRC press, 2014. [p28, 32]
- 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. [p27, 32]
- N. Cressie and C. K. Wikle. *Statistics for spatio-temporal data*. John Wiley & Sons, 2015. [p28]
- N. Cressie, J. Frey, B. Harch, and M. Smith. Spatial prediction on a river network. *Journal of Agricultural, Biological, and Environmental Statistics*, 11(2):127, 2006. [p29]
- C. Donegan. *geostan: Bayesian spatial analysis (R package)*, 2022. URL <https://cran.r-project.org/package=geostan>. The Comprehensive R Archive Network. [p27]
- M. Dumelle, E. Peterson, J. M. Ver Hoef, A. Pearse, and D. Isaak. *SSN2: Spatial Modeling on Stream Networks in R*, 2023. R package version 0.1.0. [p26]
- ESRI. *ArcGIS Desktop*. Environmental Systems Research Institute., Redlands, CA., 2019. [p33]
- A. O. Finley, S. Banerjee, and A. E. Gelfand. spBayes for large univariate and multivariate point-referenced spatio-temporal data models. *Journal of Statistical Software*, 63(13):1–28, 2015. URL <http://www.jstatsoft.org/v63/i13/>. [p27]
- J. C. Frieden, E. E. Peterson, J. A. Webb, and P. M. Negus. Improving the predictive power of spatial statistical models of stream macroinvertebrates using weighted autocovariance functions. *Environmental Modelling & Software*, 60:320–330, 2014. [p30]

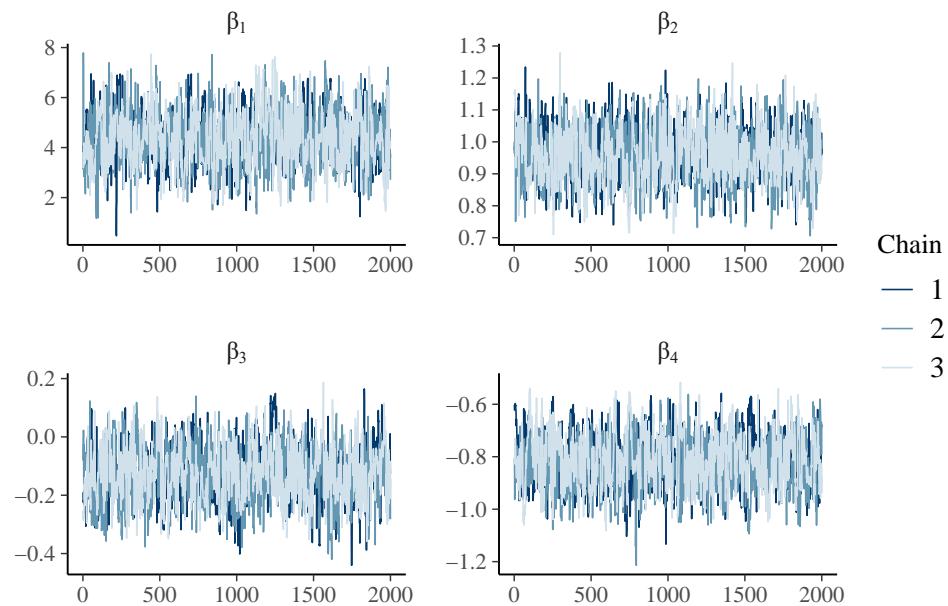


Figure 22: Trace plots of the regression coefficients (β).

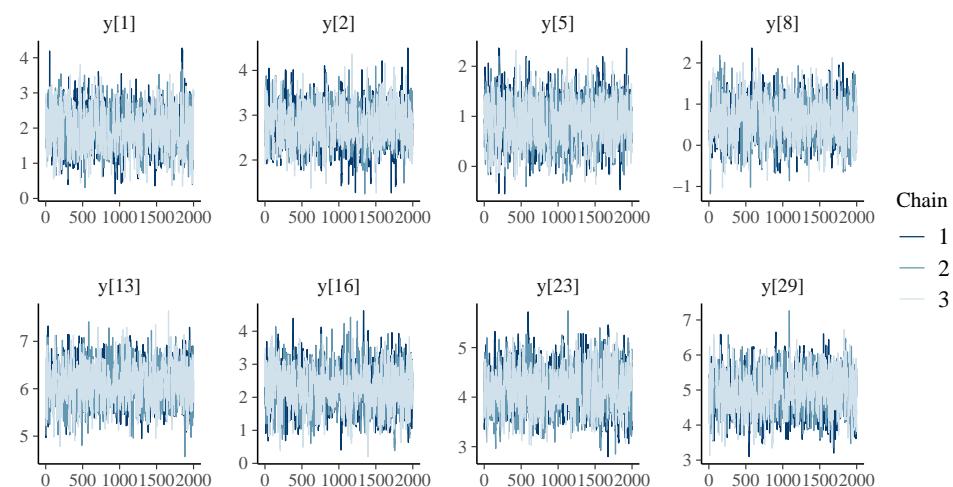


Figure 23: Trace plots of the predicted temperature (y) in eight prediction points.

- J. Gabry and T. Mahr. *bayesplot: Plotting for Bayesian Models*, 2018. URL <https://CRAN.R-project.org/package=bayesplot>. R package version 1.6.0. [p36, 42]
- V. Garreta, P. Monestiez, and J. M. Ver Hoef. Spatial modelling and prediction on river networks: up model, down model or hybrid? *Environmetrics*, 21(5):439–456, 2010. [p29]
- A. E. Gelfand, M. Fuentes, J. A. Hoeting, and R. L. Smith. *Handbook of environmental and ecological statistics*. CRC Press, 2019. [p32]
- J. D. Hamilton. *Time series analysis*. Princeton university press, 1994. [p31]
- R. J. Hyndman and Y. Khandakar. Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software*, 26(3):1–22, 2008. URL <https://www.jstatsoft.org/article/view/v027i03>. [p34]
- D. J. Isaak, E. E. Peterson, J. M. Ver Hoef, S. J. Wenger, J. A. Falke, C. E. Torgersen, C. Sowder, E. A. Steel, M.-J. Fortin, C. E. Jordan, et al. Applications of spatial statistical network models to stream data. *Wiley Interdisciplinary Reviews: Water*, 1(3):277–294, 2014. [p29]
- D. J. Isaak, M. K. Young, C. H. Luce, S. W. Hostetler, S. J. Wenger, E. E. Peterson, J. M. Ver Hoef, M. C. Groce, D. L. Horan, and D. E. Nagel. Slow climate velocities of mountain streams portend their role as refugia for cold-water biodiversity. *Proceedings of the National Academy of Sciences*, 113(16):4374–4379, 2016. [p39]
- D. J. Isaak, S. J. Wenger, E. E. Peterson, J. M. Ver Hoef, D. E. Nagel, C. H. Luce, S. W. Hostetler, J. B. Dunham, B. B. Roper, S. P. Wollrab, G. L. Chandler, D. L. Horan, and S. Parkes-Payne. The NorWeST summer stream temperature model and scenarios for the western US: A crowd-sourced database and new geospatial tools foster a user community and predict broad climate warming of rivers and streams. *Water Resources Research*, 53(11):9181–9205, 2017. doi: <https://doi.org/10.1002/2017WR020969>. URL <https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2017WR020969>. [p34, 37]
- D. J. Isaak, C. H. Luce, G. L. Chandler, D. L. Horan, and S. P. Wollrab. Principal components of thermal regimes in mountain river networks. *Hydrology and Earth System Sciences*, 22(12):6225–6240, 2018. [p26]
- F. L. Jackson, R. J. Fryer, D. M. Hannah, C. P. Millar, and I. A. Malcolm. A spatio-temporal statistical model of maximum daily river temperatures to inform the management of Scotland’s Atlantic salmon rivers under climate change. *Science of the Total Environment*, 612:1543–1558, 2018. [p29]
- M. Kattwinkel, E. Szöcs, E. Peterson, and R. B. Schäfer. Preparing gis data for analysis of stream monitoring data: The r package openstars. *PLOS ONE*, 15(9):1–10, 09 2020. doi: 10.1371/journal.pone.0239237. URL <https://doi.org/10.1371/journal.pone.0239237>. [p33]
- D. Lee. CARBayes: An R package for Bayesian spatial modeling with conditional autoregressive priors. *Journal of Statistical Software*, 55(13):1–24, 2013. URL <http://www.jstatsoft.org/v55/i13/>. [p27]
- F. Lindgren and H. Rue. Bayesian spatial modelling with R-INLA. *Journal of Statistical Software*, 63(19):1–25, 2015. URL <http://www.jstatsoft.org/v63/i19/>. [p27]
- K. J. McGuire, C. E. Torgersen, G. E. Likens, D. C. Buso, W. H. Lowe, and S. W. Bailey. Network analysis reveals multiscale controls on streamwater chemistry. *Proceedings of the National Academy of Sciences*, 111(19):7030–7035, 2014. doi: 10.1073/pnas.1404820111. URL <https://www.pnas.org/doi/abs/10.1073/pnas.1404820111>. [p39]
- M. G. McManus, E. D’Amico, E. M. Smith, R. Polinsky, J. Ackerman, and K. Tyler. Variation in stream network relationships and geospatial predictions of watershed conductivity. *Freshwater Science*, 39(4):704–721, 2020. [p29]
- E. Money, G. P. Carter, and M. L. Serre. Using river distances in the space/time estimation of dissolved oxygen along two impaired river networks in new jersey. *Water Research*, 43(7):1948–1958, 2009a. [p29]
- E. S. Money, G. P. Carter, and M. L. Serre. Modern space/time geostatistics using river distances: data integration of turbidity and E. coli measurements to assess fecal contamination along the raritan river in New Jersey. *Environmental Science & Technology*, 43(10):3736–3742, 2009b. [p29]
- E. Pebesma. CRAN Task View: Handling and Analyzing Spatio-Temporal Data. <https://cran.r-project.org/web/views/SpatioTemporal.html>, 2021. Accessed: 2021-04-13. [p27]

- E. Peterson and J. M. Ver Hoef. STARS: An arcgis toolset used to calculate the spatial information needed to fit spatial statistical models to stream network data. *Journal of Statistical Software*, 56(2):1–17, 2014. [p^{30, 33, 34}]
- E. E. Peterson and J. M. Ver Hoef. A mixed-model moving-average approach to geostatistical modeling in stream networks. *Ecology*, 91(3):644–651, 2010. [p^{28, 30, 39}]
- E. E. Peterson, A. A. Merton, D. M. Theobald, and N. S. Urquhart. Patterns of spatial autocorrelation in stream water chemistry. *Environmental Monitoring and Assessment*, 121(1):571–596, 2006. [p²⁸]
- E. E. Peterson, J. M. Ver Hoef, D. J. Isaak, J. A. Falke, M.-J. Fortin, C. E. Jordan, K. McNyset, P. Monestiez, A. S. Ruesch, A. Sengupta, et al. Modelling dendritic ecological networks in space: an integrated network perspective. *Ecology Letters*, 16(5):707–719, 2013. [p²⁸]
- J. Pinheiro, D. Bates, S. DebRoy, D. Sarkar, and R Core Team. *nlme: Linear and Nonlinear Mixed Effects Models*, 2020. URL <https://CRAN.R-project.org/package=nlme>. R package version 3.1-148. [p²⁷]
- J. C. Pinheiro and D. M. Bates. *Mixed-Effects Models in S and S-PLUS*. Springer, New York, 2000. doi: 10.1007/b98882. [p²⁷]
- P. J. Ribeiro Jr, P. J. Diggle, M. Schlather, R. Bivand, and B. Ripley. *geoR: Analysis of Geostatistical Data*, 2020. URL <https://CRAN.R-project.org/package=geoR>. R package version 1.8-1. [p²⁷]
- P. M. Rodríguez-González, C. García, A. Albuquerque, T. Monteiro-Henriques, C. Faria, J. B. Guimarães, D. Mendonça, F. Simões, M. T. Ferreira, A. Mendes, et al. A spatial stream-network approach assists in managing the remnant genetic diversity of riparian forests. *Scientific Reports*, 9(1):1–10, 2019. [p²⁹]
- A. Rushworth. *smnet: Smoothing for Stream Network Data*, 2017. URL <https://CRAN.R-project.org/package=smnet>. R package version 2.1.1. [p²⁶]
- S. K. Sahu, D. P. Lee, and K. S. Bakar. *bmstdr: Bayesian Modeling of Spatio-Temporal Data with R*, 2022. URL <https://CRAN.R-project.org/package=bmstdr>. R package version 0.3.0. [p²⁷]
- E. Santos-Fernandez. *SSNdata: Spatial stream network datasets*, 2022. URL <https://github.com/EdgarSantos-Fernandez/SSNdata>. [p³³]
- E. Santos-Fernandez, J. M. Ver Hoef, E. E. Peterson, J. McGree, D. J. Isaak, and K. Mengersen. Bayesian spatio-temporal models for stream networks. *Computational Statistics & Data Analysis*, 170:107446, 2022. [p^{30, 31, 33, 38, 39, 41}]
- E. Santos-Fernandez, J. M. Ver Hoef, E. E. Peterson, J. McGree, C. A. Villa, C. Leigh, R. Turner, C. Roberts, and K. Mengersen. Unsupervised anomaly detection in spatio-temporal stream network sensor data. *preprint*, 2023. [p⁴¹]
- M. Schlather, A. Malinowski, P. J. Menck, M. Oesting, and K. Strokorb. Analysis, simulation and prediction of multivariate random fields with package RandomFields. *Journal of Statistical Software*, 63(8):1–25, 2015. URL <http://www.jstatsoft.org/v63/i08/>. [p²⁷]
- J. O. Skoien, G. Bloschl, G. Laaha, E. Pebesma, J. Parajka, and A. Viglione. Rtop: An r package for interpolation of data with a variable spatial support, with an example from river networks. *Computers & Geosciences*, 2014. [p²⁶]
- Stan Development Team. RStan: the R interface to Stan, 2018. URL <http://mc-stan.org/>. R package version 2.18.2. [p⁴²]
- J. M. Ver Hoef and E. E. Peterson. A moving average approach for spatial statistical models of stream networks. *Journal of the American Statistical Association*, 105(489):6–18, 2010. [p^{29, 30}]
- J. M. Ver Hoef, E. Peterson, and D. Theobald. Spatial statistical models that use flow and stream distance. *Environmental and Ecological statistics*, 13(4):449–464, 2006. [p^{29, 30}]
- J. M. Ver Hoef, E. Peterson, D. Clifford, and R. Shah. SSN: An R package for spatial statistical modeling on stream networks. *Journal of Statistical Software*, 56(3):1–45, 2014. [p^{26, 30, 39}]
- C. J. Vörösmarty, P. B. McIntyre, M. O. Gessner, D. Dudgeon, A. Prusevich, P. Green, S. Glidden, S. E. Bunn, C. A. Sullivan, C. R. Liermann, et al. Global threats to human water security and river biodiversity. *Nature*, 467(7315):555–561, 2010. [p²⁶]

- H. Wickham. Reshaping data with the reshape package. *Journal of Statistical Software*, 21(12):1–20, 2007.
URL <http://www.jstatsoft.org/v21/i12/>. [p38]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p42]
- C. K. Wikle, L. M. Berliner, and N. Cressie. Hierarchical bayesian space-time models. *Environmental and Ecological Statistics*, 5(2):117–154, 1998. [p31]
- C. K. Wikle, A. Zammit-Mangion, and N. Cressie. *Spatio-temporal Statistics with R*. CRC Press, 2019. [p32]
- A. Zammit-Mangion and N. Cressie. Frk: an r package for spatial and spatio-temporal prediction with large datasets. *Journal of Statistical Software*, 98(4):1–48, 2021. [p27]

Edgar Santos-Fernandez

School of Mathematical Sciences, Queensland University of Technology

Australian Research Council Centre of Excellence for Mathematical and Statistical Frontiers (ACEMS)

Y Block, Floor 8, Gardens Point Campus. GPO Box 2434. Brisbane, QLD 4001.

Australia

ORCID: 0000-0001-5962-5417

santosfe@qut.edu.au

Jay M. Ver Hoef

Marine Mammal Laboratory, NOAA-NMFS Alaska Fisheries Science Center.

Seattle, WA and Fairbanks, AK, USA

ORCID: 0000-0003-4302-6895

jay.verhoef@noaa.gov

James McGree

School of Mathematical Sciences, Queensland University of Technology

ORCID: 0000-0003-2997-8929

james.mcgree@qut.edu.au

Daniel J. Isaak

Rocky Mountain Research Station, US Forest Service

ORCID: 0000-0002-8137-325X

daniel.isaak@usda.gov

Kerrie Mengerson

School of Mathematical Sciences, Queensland University of Technology

Australian Research Council Centre of Excellence for Mathematical and Statistical Frontiers (ACEMS)

Y Block, Floor 8, Gardens Point Campus. GPO Box 2434. Brisbane, QLD 4001.

Australia

ORCID: 0000-0001-8625-9168

k.mengerson@qut.edu.au

Erin E. Peterson

EP Consulting

School of Mathematical Sciences, Queensland University of Technology

Australian Research Council Centre of Excellence for Mathematical and Statistical Frontiers (ACEMS)

Y Block, Floor 8, Gardens Point Campus. GPO Box 2434. Brisbane, QLD 4001.

Australia

ORCID: 0000-0003-2992-0372

erin@peterson-consulting.com

C443: An R-package to see a forest for the trees

by Aniek Sies and Iven Van Mechelen

Abstract Classification trees, well-known for their ease of interpretation, are a widely used tool to solve statistical learning problems. However, researchers often end up with a forest rather than an individual classification tree, which implies a major cost due to the loss of the transparency of individual trees. Therefore, an important challenge is to enjoy the benefits of forests without paying this cost. In this paper, we propose the R-package C443. The C443 methodology simplifies a forest into one or a few condensed summary trees, to gain insight into its central tendency and heterogeneity. This is done by clustering the trees in the forest based on similarities between them, and on post-processing the clustering output. We will elaborate upon the implementation of the methodology in the package, and will illustrate its use with three examples.

1 Introduction

Classification trees are a widely used tool in the statistical learning of the relationship between a set of predictor variables and a categorical outcome (Breiman et al., 1984; Quinlan, 1986). A particularly attractive feature is that they typically lead to insightful and well interpretable results.

However, researchers often end up with a forest (i.e., a set of multiple trees) rather than with a single classification or decision tree. There are several reasons for this: First, single trees are known to be fairly unstable (e.g., Philipp et al. (2016); Philipp et al. (2018)); we may wish to investigate this instability through the creation of a forest by bringing about small changes in the learning sample (e.g., drawing bootstrap subsamples) or in the tree building process (e.g., varying tuning parameters)(Breiman, 1996a; Strobl et al., 2009; Turney, 1995). Second, we may wish to improve prediction accuracy (Bauer and Kohavi, 1999; Dietterich, 2000; Hastie et al., 2009; Skurichina and Duin, 2002) by generating a forest by means of ensemble methods such as random forests (Breiman, 2001), bagging (Breiman, 1996b), or boosting (Freund and Schapire, 1997). Third, if there are multiple outcomes, we may wish to grow one or multiple trees for each outcome variable, leading once again to the formation of a forest. Fourth, in order to deal with missing data, one preferred method is multiple imputation (Rubin, 1987), resulting in several imputed data sets, based on each of which we can construct a classification tree.

All four of these are excellent reasons to prefer forests over single trees. Nonetheless, forests have a major cost that is implied by a complete loss of the transparency of single trees. This brings about the challenge to look for ways to enjoy the benefits of a forest without having to pay this cost.

More specifically, one may wish to capture three sorts of insight from a forest:

1. Central tendency: Can a forest be summarized in one or a few central decision structures and how do these look like?
2. Heterogeneity: How much variability is there in the forest and what are its contents? For example, are all trees in the forest slight variations on one central decision structure, or are they subject to sizeable qualitative differences?
3. Linking heterogeneity to tree covariates: In case of a heterogeneous forest based on known sources of variation (e.g., different types of tuning parameters or response variables), are qualitative within-forest differences related to this variation source? For example, do different response variables result in different decision structures?

Partial answers to this challenge can be found in earlier work. As such, variable importance measures have been proposed to capture the most central predictor variables in random forests (Breiman, 2001; Breiman and Cutler, 2003). From their part, Banerjee et al. (2012); Briand et al. (2009), and Chipman et al. (1998) suggested to select one or a few summary trees within a forest based on ad hoc calculated similarities between the individual trees, whereas Philipp et al. (2016) developed a methodology and associated R-package `stablelearner` to visualize and summarize variable and split point within-forest heterogeneity. This is useful in evaluating stability of tree splits in terms of split variables and/or split-points, but does not evaluate stability of entire tree structures. A lasso regression methodology (Friedman and Popescu, 2008) along with the associated R-package `pre` (Fokkema, 2020), has been proposed to derive a sparse ensemble from an initially large prediction rule ensemble. Although this results in an easily interpretable rule, it does not give insight into the heterogeneity within the initial rule ensemble. Quite a different approach is taken in the area of interpretable machine learning, where local SHapley Additive exPlanations (SHAP) of ensemble

learners are looked for in the form of additive contributions of input features to estimated risk scores (e.g., Lundberg et al. (2020)).

A comprehensive and complementary way to recover insight from a forest could be obtained by means of a methodology proposed by Sies and Van Mechelen (2020). Unlike the SHAP-approach, the C443 methodology focuses on global rather than local explanations, remains within a tree context instead of moving to an additive framework, and targets categorical outcome variables rather than unidimensional risk scores. The methodology comprises three components:

1. Similarities calculation between all trees in the forest. A complication at this point is the vast range of similarity measures that can be invoked within a tree context, as the choice of a similarity measure has major consequences for the output of the methodology and its interpretation. Sies and Van Mechelen (2020) developed a novel, comprehensive conceptual framework to cope with this complication and assist the user in making a thoughtful choice of a suitable similarity measure.
2. A similarity-based clustering of the trees in the forest, where, given the critical concern about insightfulness, clusters that are well interpretable regarding the underlying decision structure and the relation between the predictors and the categorical response variable are obtained.
3. A post-processing of the clustering result to arrive at the essential insights into central tendency and heterogeneity looked for.

The aim of the current paper is to introduce the R-package **C443**, which implements the methodology proposed by Sies and Van Mechelen (2020). The package is available from the Comprehensive R Archive Network (CRAN) on <https://CRAN.R-project.org/package=C443> and is also accessible through Github (<https://github.com/KULeuven-PPW-OKPIV/C443>). The package accepts as input user-supplied forests with a wide range of formats (in part making use of the **partykit** package (Hothorn and Zeileis, 2015)), and a user-provided similarity matrix between the forests' trees based on a similarity measure of the user's choice (if available), and covariate information on the trees (if available). Otherwise, it also includes the option to calculate a matrix of similarities between the trees of the user-supplied forest based on one out of eight possible similarity measures within the conceptual framework proposed by Sies and Van Mechelen (2020). The package further partitions the trees in the forest under study into a few well-interpretable clusters (based on the user-provided or calculated similarities), using the PAM algorithm (Kaufman and Rousseeuw, 1990) implemented in the **cluster** package (Maechler et al., 2019). Finally, it includes a number of methods and functions that can be used for post-processing the clustering output to summarize the forest, capture within-forest heterogeneity, or link this heterogeneity to tree covariates.

2 C443 Methodology

In this section we will briefly recapitulate the main concepts of the methodology proposed by Sies and Van Mechelen (2020).

2.1 Similarities

Any set of categories or concepts has two sides: the side of the categories' meaning in terms of their defining or characteristic features, and the side of the categories' membership (with the two sides in question being referred to as the categories' intension and extension, respectively: Leibniz (1764)). In line with this, when evaluating similarities between classification trees, we could do so with respect to the trees' meaning (i.e., their predictor-related contents), with respect to the classifications of the objects or experimental units implied by them, or with respect to a hybrid combination of both facets.

When considering to evaluate tree similarities with respect to common predictor-related contents, two groups of questions are further to be addressed:

1. Does one want to calculate similarity in terms of shared predictor variables only or also in terms of shared split points on common predictors? Furthermore, in the latter case, is one willing to take into account a tolerance zone when evaluating equality of split points?
2. Does one want to calculate similarity in terms of individual predictors or rather in terms of sets of predictors as implied by the definitions of each of the leaves of the trees under study? In the latter case, is one willing to take into account: (a) the order of the predictors on the downward path from the root to the leaf under study, and (b) the relevant part of the predictor range (i.e., lower vs. upper part) involved in the definition of the leaf in question?

These two groups of questions are schematically represented in Figure 1. Orthogonally combining all 3*5 possible answers to them yields 15 possible similarity types.

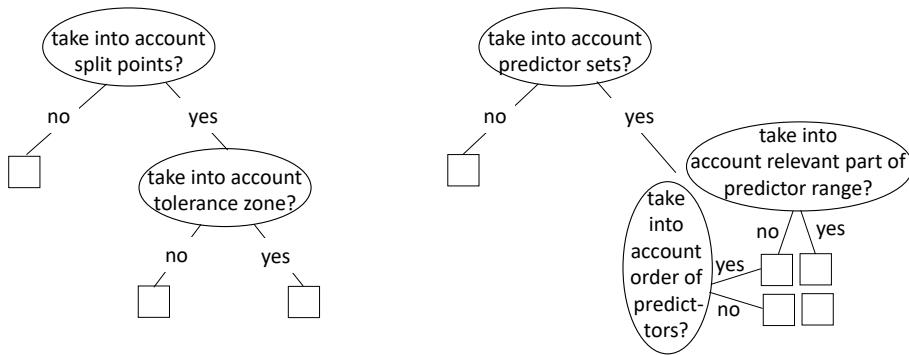


Figure 1: Schematic representation of two groups of questions within comprehensive conceptual framework with regard to tree similarities in terms of shared predictor-related contents.

When evaluating the similarity between two trees with respect to the classifications of the objects or experimental units implied by them, in general, pairs of objects that belong to the same class in each of the two trees will contribute to a higher similarity. Two questions should further be addressed for similarities with regard to implied class memberships:

1. Does one want to consider the classifications (partitions) of the experimental units by the class labels or the leaf memberships?
2. Should one factor in pairs of objects that belong to the same class in each of the two trees under study only if these classes are associated with the same class label across the two trees?

These two questions are schematically represented in Figure 2. Orthogonally combining all 2*2 possible answers to them yields 4 possible similarity types.

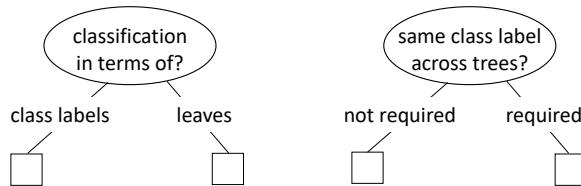


Figure 2: Schematic representation of two questions within comprehensive conceptual framework with regard to tree similarities in terms of agreement between implied classifications.

To illustrate the above, let us briefly consider three examples. As a first example, [Shannon and Banks \(1999\)](#) proposed to define a dissimilarity $d(T_1, T_2)$ between two trees T_1 and T_2 as:

$$d(T_1, T_2) = \sum_r \alpha_r |D_{T_1, T_2}^r|, \quad (1)$$

where $|D_{T_1, T_2}^r|$ equals the number of distinctive paths of length r from a root node to a leaf, with a path denoting an ordered set of relevant predictor parts (upper vs. lower) as implied by a leaf, with distinctive meaning that the path shows up in one of the two trees only, and with α_r denoting a set of prespecified weights (e.g., $\alpha_r = \frac{1}{r}$). Obviously, this is a dissimilarity measure that captures the trees' meaning or predictor-related contents only, that does not take into account split points, that does take into account predictor sets, and while doing so does take into account both the order and the range-part (lower vs. upper) of the predictors.

As a second example, while denoting the set of objects under study by $\{o_1, \dots, o_i, \dots, o_n\}$, [Chipman et al. \(1998\)](#) defined a dissimilarity $d(T_1, T_2)$ between two trees T_1 and T_2 as:

$$d(T_1, T_2) = \frac{\sum_{i>i'} |I^{T_1}(o_i, o_{i'}) - I^{T_2}(o_i, o_{i'})|}{\binom{n}{2}}, \quad (2)$$

with $I^{T_j}(o_i, o_{i'}) = 1$ if o_i and $o_{i'}$ belong to the same leaf in T_j , $j = 1, 2$, and $I^{T_j}(o_i, o_{i'}) = 0$ otherwise. Obviously, this is a dissimilarity measure that captures the trees' implied classifications only, while considering classifications implied by leaf membership without requiring equality across trees of class

labels associated with the leaves. Note that, if different observations were used to create the trees under study (e.g., in case of a forest based on bootstrap samples), the observations $o_i, i = 1, \dots, n$ are those from the original data set (e.g., the data set from which the bootstrap samples were drawn).

As a third example, we consider a hybrid tree similarity measure $sim_T(T_1, T_2)$ implemented in the package **C443**. This measure is based on a leaf similarity measure $sim_L(l, l')$ that captures predictor-related contents in terms of the (frequencies of occurrence of the) combinations of predictors and range-parts (lower vs. upper) PRP_p associated with leafs $l \in T_1$ and $l' \in T_2$ (while leaving aside split points and the order of the predictors), and that captures the trees' implied classifications by assuming a leaf similarity value of zero for leafs that are associated with different class labels (i.e., $c(l) \neq c(l')$). Specifically, if T_1 and T_2 both have height zero (i.e., consist of a root node only), by definition $sim_T(T_1, T_2) = 1$. Otherwise, $sim_T(T_1, T_2)$ is based on leaf similarity measure $sim_L(l, l')$, with:

$$sim_L(l, l') = \begin{cases} \frac{\sum_p \min(\#^l PRP_p, \#^{l'} PRP_p)}{\sum_p \max(\#^l PRP_p, \#^{l'} PRP_p)} & \text{if } c(l) = c(l') \\ 0 & \text{otherwise,} \end{cases} \quad (3)$$

with $\#^l PRP_p$ (resp. $\#^{l'} PRP_p$) denoting the frequency of occurrence of the p^{th} predictor range-part combination PRP_p in the definition of leaf l (resp. l'). The tree similarity $sim_T(T_1, T_2)$ then relies on the search for an optimal matching between (subsets of) the leafs of T_1 and T_2 , $S_{L_{T_1}} \subset L_{T_1}$ and $S_{L_{T_2}} \subset L_{T_2}$, as formalized by a bijection $g : S_{L_{T_1}} \rightarrow S_{L_{T_2}}$. Specifically:

$$sim_T(T_1, T_2) = \max_{\substack{g: S_{L_{T_1}} \xrightarrow{\text{bijection}} S_{L_{T_2}}}} \frac{\sum_{l \in S_{L_{T_1}}} sim_L(l, g(l))}{\min(\#L_{T_1}, \#L_{T_2})}. \quad (4)$$

2.2 Clustering

A second step in the proposed methodology consists of a clustering (viz., a partitioning) of the forest based on the similarity measure chosen in the first step. A major challenge at this point is the interpretability of the resulting clusters. For this purpose, Partitioning Around Medoids (PAM, [Kaufman and Rousseeuw \(1990\)](#)) was chosen as a clustering method. In this method, each cluster is centered around a medoid, which is an actual element of the cluster (unlike the centroids in other clustering methods). Given a (dis)similarity matrix and a prespecified number of clusters k , PAM looks for k medoids that are such that a loss function consisting of the sum across all objects of the dissimilarity between the object and its closest medoid is minimized. PAM starts with an initial selection of k medoids, and subsequently replaces iteratively each medoid by the object that minimizes the loss function until no further improvement in the loss function is observed.

The key advantage of using PAM to cluster a forest, is that PAM leads to a set of clusters each of which is represented by a cluster member, that is to say, a well-interpretable tree (which means that an insightfulness baby is rescued out of the bathwater of the forest!). To be sure, when interpreting the medoid trees resulting from PAM, one should never forget the (dis)similarity type on which the clustering was based. For example, if a similarity was used that exclusively reflected the objects' classifications, then an interpretation of the predictor-related contents of the medoid trees is simply not admissible. Indeed, the trees within each cluster will then be similar in terms of implied classifications, but may be very different with respect to predictors and split points.

2.3 Post-processing

To retrieve critical insights into the central tendency and heterogeneity of the forest under study, a suitable post-processing of the clustering output is called for. Below, we will list a few useful questions and steps at this point. For all of them, we must again emphasize the importance of keeping aware of the chosen similarity at the basis of the clustering, as this may strongly constrain which kinds of interpretation of the post-processed output will be admissible. (To overcome these constraints, one either has to do all of the similarity and clustering calculations over with a different type of tree similarity measure, or to revert to hybrid types of similarity measures that tap both predictor- and classification-related aspects of the trees under study.)

The most obvious place to grasp the central tendency of the forest in focus is the medoid of the one-cluster solution. Besides, one could also take a look at the medoids of solutions with a few more clusters and inspect which aspects all medoids in question have in common.

Concerning within-forest heterogeneity, to start with, one can examine this quantitatively in terms of the amount of variability at play. A straightforward question at this point is how many clusters are

needed to summarize the forest. Determining the optimal number of clusters is a common challenge in the clustering domain, for which a very broad range of measures and heuristic procedures (or "stopping rules") have been proposed. Within the methodology proposed by [Sies and Van Mechelen \(2020\)](#), three types of plots corresponding to different values of the number of clusters, k , may be singled out:

1. a plot of the average within-cluster similarities (or, alternatively, the average similarities between the cluster elements and their cluster medoids) for the similarity measure at the basis of the clustering; in general, we recommend to select the number of clusters based on one of these plots because of their direct link with the loss function that is optimized by the PAM algorithm
2. a plot of the Average Silhouette Width (ASW, [\(Rousseeuw, 1987\)](#)), that is, the average of all objects' silhouette; the silhouette of an object is a function of its mean similarity to the objects of its own cluster minus its mean similarity to the objects of its second closest cluster; ASW is recommended by the authors of PAM and the corresponding R-package ([Kaufman and Rousseeuw, 1990; Maechler et al., 2019](#))
3. suppose that the forest under study was created to arrive at a better prediction accuracy and that a similarity measure was chosen or constructed that (also) captures agreement between classifications in terms of class labels; in this scenario, we recommend determining the number of clusters also by examining: (a) a plot depicting the accuracy of the predicted class labels (for the originally observed dataset), based on combining the predictions of the first k medoid trees (weighted by their corresponding cluster cardinalities), and (b) a plot of the agreement between the class label for each object of the originally observed dataset, predicted on the basis of the random forest as a whole versus the class label predicted on the basis of the first k medoid trees (weighted by their corresponding cluster cardinalities).

Another potential quantitative approach to assess forest heterogeneity involves examining the cluster sizes in solutions where $k > 1$. For example, solutions with a single very dominant cluster (in terms of cluster size), would point at considerable within-forest homogeneity.

We can further also try to capture within-forest heterogeneity from a qualitative perspective, that is, the contents of the within-forest differences. In view of this, one may compare the cluster medoids, once more aligning with the selected similarity measure, in terms of their predictor-related contents, and/or predicted class labels. With regard to predictor-related contents, possible outcomes could be that all medoids constitute only minor variations on a common topology (e.g., in terms of a small number of distinctive secondary splits in different medoids), versus major variations that involve very different kinds of predictors and split points. With regard to predicted class labels, the predictions of the medoids can be compared in terms of marginal totals of all class labels per medoid as well as in terms of a contingency table with the numbers of experimental units assigned to each combination of class labels by each of two or more different medoid trees. A possible outcome could be that the classifications implied by one medoid are only a refinement of those implied by another medoid, with the majority of observations being assigned by both to the same class label.

Finally, one may wish to investigate the link between covariate information on the trees in the forest and within-forest heterogeneity, in case such covariate information is available. First of all, one may wish to do so in terms of the relation between covariates and cluster membership. In case of a discrete covariate (e.g., the nature of the outcome variable), this may lead to questions such as "Do trees based on a same value of the covariate mainly end up in the same cluster?", or, more in general, "Which values of the covariate lead to trees in the same cluster and which values to trees in different clusters?". In case of a continuous covariate (e.g., the value of some tuning parameter in the tree building algorithm): "Does cluster membership relate to this covariate (e.g., by comparing within- and between-cluster differences with regard to it)?". Second, if a relationship between cluster membership and a covariate would be established, then the relationship between that covariate and the predictor-related contents or implied classifications of the cluster medoids could be examined (in accordance with the chosen similarity measure). As an example, if the covariate would pertain to different outcome variables, one might wish to examine differences between these outcome variables in terms of their predictive basis (if the chosen similarity measure would allow the researcher to do so).

3 Implementation of the methodology in C443

In this Section, we will discuss how the methodology described above has been implemented in the R-package **C443**. First, we will describe user input when employing the package. In the following subsections, we will discuss how the different constituents of the methodology (i.e., the calculation of similarities, the clustering, and the post-processing of the clustering output) have been implemented in the package.

3.1 Input needed for the package

Users of the package **C443** should provide at least one mandatory piece of input, namely the forest of interest. In addition, two optional pieces of input may be supplied: a similarity matrix and covariate information on each tree in the forest.

Forest

There are many techniques and associated R-packages to grow classification trees. For example, Classification and Regression Trees (Breiman et al., 1984) can be grown using **rpart** (Therneau et al., 2015), Conditional inference trees (Hothorn et al., 2006) using **partykit** (Hothorn and Zeileis, 2015), and C4.5 trees (Quinlan, 1993) using **RWeka** (Hornik et al., 2009).

Likewise, many techniques and R-packages exist to create a forest. First, creating a forest to investigate instability of trees can be done in several ways. One could grow trees using one of the packages mentioned above on sub-samples obtained from the `sample` function with `replace = FALSE` and `size < n` (where n is the size of the original data set), or on bootstrap samples obtained from the same function with `replace = TRUE` and `size = n`. Second, to create a forest in view of improving prediction accuracy, one may employ boosting (Freund and Schapire, 1997), as implemented in R-packages such as **adabag** (Alfaro et al., 2013) and **gbm** (Ridgeway, 2007). or random forests (Breiman, 2001), as implemented in the packages **randomForest** (Liaw and Wiener, 2002) and **ranger** (Wright and Ziegler, 2017). Third, a forest can result from growing trees on a number of different outcome variables. Fourth, multiple imputation (Rubin, 1987) can be applied in case of missing data using the **mice** package (van Buuren and Groothuis-Oudshoorn, 2011), and subsequently a tree can be grown on each imputed data set, which would return a forest as well.

The **C443** package does not have a specific functionality for creating forests, given the large variety of techniques and available R-packages to do so. Rather, a multitude of types of forests are accepted as input by the package, where three pieces of information are required:

1. the entire set of observed data at the basis of the forest, stored as a data frame –in case of a forest produced by bringing about small changes in the learning data (e.g., by taking bootstrap samples from the set of experimental units), this would be the entire originally observed data set; in case of a forest based on multiple outcomes, this would be the data set with inclusion of all outcomes; and in case of a forest as a result of multiple imputation, this would be the originally observed data set without imputations–;
2. the n trees in the forest, stored as a list of party objects, or of objects inheriting from party objects (included but not limited to **glmtree**, **cmtree**), or of objects that are convertible into party objects using
 - (a) **partykit** (Hothorn and Zeileis, 2015)): for **rpart** and J48 objects
 - (b) **C443**'s internal functions: for **randomForest** and **ranger** objects, including **ranger** objects that can be delivered from packages such as **DFNET** (Pfeifer et al., 2022);
 (the class of the first tree is checked in **C443** with the `class` function);
3. if not included in the party object (e.g., when **rpart** was used to grow the trees), the n data sets on which the individual trees in the forest were based, stored as a list of data frames.

Similarity matrix

Users can provide a home-made similarity matrix as input for the clustering. This square matrix with number of rows/columns equal to the number of trees in the forest, must be symmetric. The similarity values should vary between 0 and 1 (with 0 indicating no similarity at all and 1 indicating a perfect similarity). Otherwise, quite a few tree dissimilarity measures have been proposed, which often vary between 0 and 1 (with 0 indicating no dissimilarity at all and 1 full dissimilarity). We can transform these into similarities by subtracting them from 1.

Covariate information

If the user is interested in relating the possible heterogeneity within the forest to values of the trees on one or more covariates, a vector or data frame with the values of those covariates for each tree should be provided.

Table 1: Overview of similarity measures implemented in C443.

No ¹	Similarity Respect	Basis
1	Predictor content	# common predictors
2	Predictor content	# common predictor-split point combinations
3	Predictor content	# distinctive ordered sets of predictor-range part combinations
4	Classifications	agreement of partitions implied by leaf membership
5	Classifications	agreement of partitions implied by class labels
6	Hybrid	# predictor-range part occurrences in definitions of leaves with same class label
7	Hybrid	# predictor-split point combinations in definitions of leaves with same class label
8	Hybrid	closeness to logical equivalence (applicable in case of binary predictors only)

¹ Number of section in Supplementary Materials that includes mathematical formalization and reference citation for measure in question

3.2 Similarity calculation

There is the option to have C443 calculate a similarity matrix alongside the option of a user-provided similarity matrix. For this purpose, eight similarity measures proposed by [Sies and Van Mechelen \(2020\)](#) have been implemented in the package, each of which takes into account different aspects of similarity. In Table 1, we describe each implemented similarity measure. (In the Supplementary Materials, mathematical formalizations are given.)

3.3 Clustering

Starting from a dissimilarity matrix (viz., 1 - the similarity matrix provided by the user or calculated by the package), a clustering of the trees in the forest is done based on the PAM algorithm ([Kaufman and Rousseeuw, 1990](#)), as implemented in the **pam** function of the **cluster** package ([Maechler et al., 2019](#)). The original PAM algorithm consists of two phases: a BUILD phase (an intelligent way of finding an initial set of clusters), and a SWAP phase (swapping medoids and candidate medoids until convergence).

To lower the risk of ending up in a local minimum, we implemented a multi-start approach. In addition to the BUILD phase, we use 1000 starts where the initial cluster medoids are randomly assigned. Next, for each of the 1001 initial sets of medoids, we use SWAP until convergence. Finally, the solution that yields the lowest value of the objective function across all starts is retained.

For PAM with a BUILD start, we use `medoids=NULL` and set `do.swap=TRUE`. To run PAM with random starts, we assign random initial values for the medoids using `medoids=randomstarts`. Here, `randomstarts` is a vector with k values, randomly sampled without replacement from 1 to T , where k is the number of medoids and T is the number of trees in the forest. For reproducibility purposes, the seed number is included in the final solution. Finally, we always use `pamonce=3` to speed up the SWAP phase, as suggested by [Schubert and Rousseeuw \(2019\)](#).

The procedure described above is repeated for each number of clusters between the minimum and maximum number of clusters specified by the user. For each number of clusters, the clustering result is returned as a `clusterforest` object.

3.4 Post-processing

The two main insights looked for (central tendency and forest heterogeneity) can be captured by post-processing the clustering result using several functions on the `clusterforest` object.

Central tendency

The function `medoidtrees()` returns the medoid tree(s) of a given `clusterforest` solution as `party` object(s). The function `plot()` may be used to plot the medoids, and the `predict()` function to obtain their implied classifications.

Heterogeneity

Regarding quantitative heterogeneity, first, in order to decide on the number of clusters needed to summarize the forest, one can use the function `plot()` with the `clusterforest` object as the only argument. This will return two plots that visualize for each solution: 1) the average within-cluster similarity, and 2) the ASW (Rousseeuw, 1987); optionally, a third and fourth plot can also be returned, which show for each solution the accuracy of the predicted class labels (for the originally observed dataset) based on the medoid trees, and the agreement in classifications based on the medoids only versus based on the full forest (see Section 2.3). Second, the cluster sizes can be obtained by extracting the cluster assignment of all trees for a given solution from the `clusterforest` object using the `clusters()` function, and subsequently counting the number of trees assigned to each cluster.

Regarding qualitative heterogeneity, the medoids' predictive content and classifications can again be obtained by means of the functions `medoirtrees()`, `plot()` and `predict()`. Finally, to explore the relationship between one or more tree covariates and the heterogeneity within the forest, the function `treesource()` can be employed. In case of a discrete covariate, it visualizes the number of trees that belongs to each cluster for each value of the covariate. In case of a continuous covariate, it returns the mean and standard deviation of the covariate within each cluster. One can further interpret the relation between the covariate and the clusters by utilizing the cluster medoids' predictive contents and/or implied classifications. This can be done using a combination of the functions `medoirtrees()`, `plot()` and `predict()`.

4 Practical usage of the R-package

We will illustrate the use of the R-package with three real data examples taken from the Drug Consumption data set (Fehrman et al., 2017). This data set is freely available from the UCI Machine Learning Repository (Dua and Graff, 2017) and is included in a slightly transformed format in C443. The data set contains anonymous survey data from 1885 respondents, who reported on their use of different types of drugs. Although in the original data set each of the drug-use variables had seven response categories (never used, used over a decade ago, used in the last decade, year, month, week or day), we included them as binary response variables in the R-package, with categories non-user (never used or used over a decade ago only) versus user (all other categories), similar to Fehrman et al. (2017).

In the first and second example, we will focus on one drug only, namely ecstasy. In the third example, we will broaden the scope of the application to the use of several drugs, viz., amphetamines, benzodiazepines, ecstasy, LSD and mushrooms. Finally, in all three examples we will use as predictors a number of demographic variables (age, gender, level of education), the Big Five personality traits (i.e., Extraversion, Agreeableness, Conscientiousness, Neuroticism, and Openness to experience) measured by the NEO-FFI-R (McCrae and Costa, 2004), Impulsivity measured by the BIS-11 (Patton et al., 1995), and Sensation seeking measured by the ImpSS (Zuckerman et al., 1993). Note that all predictors were initially categorical (ordinal or nominal) and were quantified by Fehrman et al. (2017).

4.1 Example 1: Assessing stability

The goal of the first analysis is to find out which types of persons are at risk of using ecstasy. Although this question can be insightfully addressed by growing a single classification tree on the full data set, the interpretation of the result may be undermined by the issue of tree instability. We will therefore investigate the stability of the obtained solution using C443. Specifically, we will draw 100 bootstrap samples and grow a classification tree on each sample. Next, we will use C443 to calculate similarities between the resulting trees, and subsequently cluster them. Finally, we will post-process the obtained cluster result to gain insight into the central tendency and heterogeneity of the forest.

We first load the package C443, and extract the relevant information for this analysis from the drugs data set:

```
R> library (C443)
R> EcstData = drugs [, c("Age", "Gender", "Edu", "Neuro", "Extr", "Open", "Agree",
+ "Consc", "Impul", "Sensat", "Ecst")]
R> EcstData$Ecst <- as.factor(EcstData$Ecst)
R> head(EcstData, 3)
```

Age	Gender	Edu	Neuro	Extr	Open	Agree
-----	--------	-----	-------	------	------	-------

```

1  0.49788  0.48246 -0.05921  0.31287 -0.57545 -0.58331 -0.91699
2 -0.07854 -0.48246  1.98437 -0.67825  1.93886  1.43533  0.76096
3  0.49788 -0.48246 -0.05921 -0.46725  0.80523 -0.84732 -1.62090
    Consc     Impul     Sensat Ecst
1 -0.00665 -0.21712 -1.18084      0
2 -0.14277 -0.71126 -0.21575      1
3 -1.01450 -1.37983  0.40148      0

```

To answer the question of which types of people may be at risk for ecstasy use, we first grow a single decision tree on the full data set using **rpart**. We prune the resulting tree using the complexity parameter associated with a cross-validated prediction error that is maximally one standard error higher than the lowest one (as suggested by [Hastie et al. \(2009\)](#)). We plot the result using **partykit**:

```

R> library(rpart)
R> library(partykit)
R> set.seed(123)
R> EcstTree <- rpart(Ecst ~ ., data = EcstData, control = rpart.control(
+   minsplit = 100, maxdepth = 3, maxsurrogate=0, maxcompete=0))
R> cp <- EcstTree$cptable[order(EcstTree$cptable[1:which.min(EcstTree$cptable[, "xerror"])]), "xerror"], ]
R> maxcp <- cp[1, "xerror"] + 1.96 * cp[1, "xstd"]
R> cp <- cp[as.numeric(which.max(1 / (maxcp - cp[, "xerror"]))), "CP"]
R> PrunedEcstTree <- prune.rpart(EcstTree, cp = cp)
R> plot(as.party(PrunedEcstTree))

```

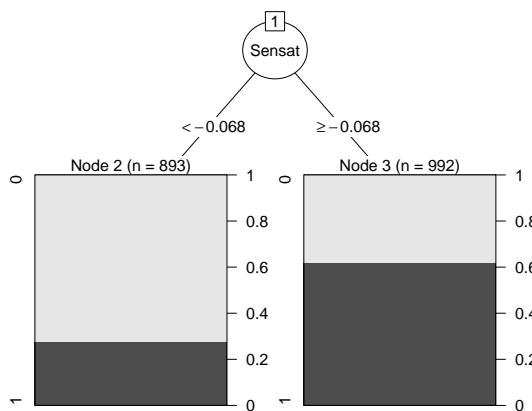


Figure 3: Decision tree estimated from the full drug data set with ecstasy as response variable.

The obtained result (as displayed in Figure 3) shows that people with a higher score on Sensation seeking are more at risk for using ecstasy, which makes sense from an intuitive point of view.

To investigate the stability of this result, we create a forest, by drawing 100 bootstrap samples from the original data set and growing a tree on each bootstrap sample:

```

R> DrawBoots <- function(dataset, i){
+   set.seed(1234 + i)
+   Boot <- dataset[sample(1:nrow(dataset), size = nrow(dataset),
+   replace = TRUE), ]
+   return(Boot)
+ }
R> GrowTree <- function(x,y,BootsSample){
+   controlrpart <- rpart.control(minsplit = 100, minbucket = 50,
+   maxdepth = 3, maxsurrogate=0, maxcompete=0)
+   tree <- rpart(as.formula(paste(noquote(paste(y, "~")),
+   noquote(paste(x,
+   collapse = "+")))), data = BootsSample, control = controlrpart)
+
+   cp <- tree$cptable[order(tree$cptable[1:which.min(tree$cptable[, "xerror"])]), "xerror"], ]
+   maxcp <- cp[1, "xerror"] + 1.96 * cp[1, "xstd"]

```

```

+   cp <- cp[as.numeric(which.max(1 / (maxcp - cp[, "xerror"]))), "CP"]
+
+   PrunedTree <- prune.rpart(tree, cp = cp)
+   return(PrunedTree)
+ }
R> set.seed(12345)
R> Boots <- lapply(1:100, function(k) DrawBoots(EcstData, k))
R> Trees <- lapply(1:100, function (i) GrowTree(x = c("Age", "Gender", "Edu",
+   "Neuro", "Extr", "Open", "Agree", "Consc", "Impul", "Sensat"),
+   y = "Ecst", Boots[[i]]))

```

To cluster the trees of this forest, we use the `clusterforest()` function. This requires as input the full originally observed data set and the trees in the forest along with the data sets on which these were based. Furthermore, we should also provide a similarity matrix or a similarity measure that is to be used to calculate similarities. In this example, we will let the R-package calculate the similarities using one of the hybrid measures based on predictor-range part occurrences in the definition of leaves with the same class label (see similarity measure No 6 in Table 1). This will allow us to interpret the clustering result in terms of the predictor-related contents of the medoids (viz., in terms of the predictors involved in the definition of each leaf along with a specification of the relevant part of each predictor's range high vs. low) as well as in terms of their class assignments. Finally, we should indicate which cluster solutions we want to explore: We will look at the cluster solutions with one through eight clusters.

```

R> set.seed(1)
R> ClusterForest<- clusterforest(observeddata=EcstData, treedata=Boots,
+   trees=Trees, m=6, fromclus=1, toclus=8)

```

The `clusterforest()` function returns a `clusterforest` object. We can use several functions to post-process this result and gain the desired insights. First, we will plot the object to estimate how many clusters are needed to summarize the forest.

```
R> plot(ClusterForest)
```

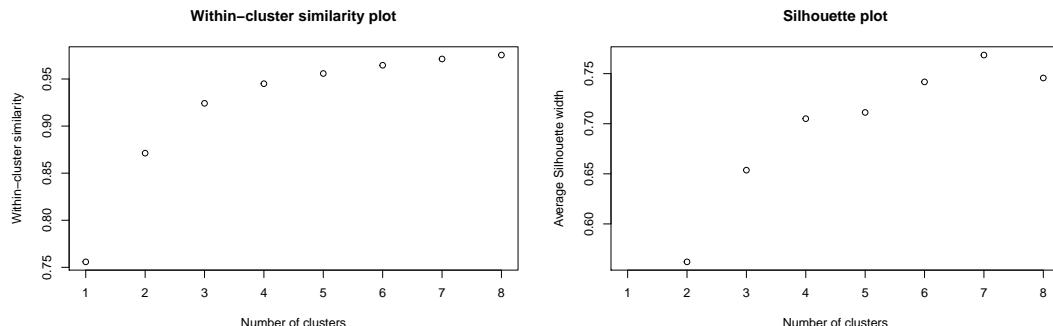


Figure 4: Average within-cluster similarity and ASW for solutions with 1 through 8 clusters for Example 1 on ecstasy use.

Figure 4 shows for each solution the average within-cluster similarity, and the ASW. The first plot shows that a solution with a relatively small number of clusters (one, two or three) could be sufficient to summarize the forest. The ASW increases until seven clusters, yet the increase flattens out after four clusters. Consequently, in the remainder of this analysis, we will focus on the solutions with one through four clusters.

To understand the central tendency of the forest, we will inspect the cluster medoid of the one-cluster solution. Using the function `plot()` on a `clusterforest` object with as second argument a solution number will plot the medoid(s) of the solution in question:

```
R> plot(ClusterForest, solution = 1)
```

The most central tree in the forest is the same as the decision tree that we obtained from the original data set (see Figure 3).

To evaluate the heterogeneity within the forest, we will first check the cluster medoids and then the cluster sizes for the selected solutions with two, three and four clusters.

```
R> plot(ClusterForest, solution = 2)
R> plot(ClusterForest, solution = 3)
R> plot(ClusterForest, solution = 4)
```

The cluster medoids for the 1-, 2-, 3-, and 4-cluster solutions are shown in Figures 3 and 5. (Note that the sample sizes and proportions shown in the leafs of each medoid in Figure 5 pertain to the bootstrap sample on which the medoid was based and not to the original full data set.) The medoid sets of the different solutions are closely related as:

1. All medoids include a root node split on Sensation seeking.
2. All medoids with additional splits under the root node imply one or two further splits of the group scoring high on Sensation seeking.
3. The medoid sets of the 1-, 3-, and 4-cluster solutions are fully nested.
4. Medoid (a) shows up in the 2-, 3-, and 4-cluster solutions.
5. The second medoid of the 2-cluster solution (i.e., medoid (b), with additional splits on Conscientiousness and Openness) is a kind of a mixture of medoid (c) (in the 3- and 4-cluster solutions, with an additional split on Conscientiousness) and medoid (d) (in the 4-cluster solution, with an additional split on Openness).

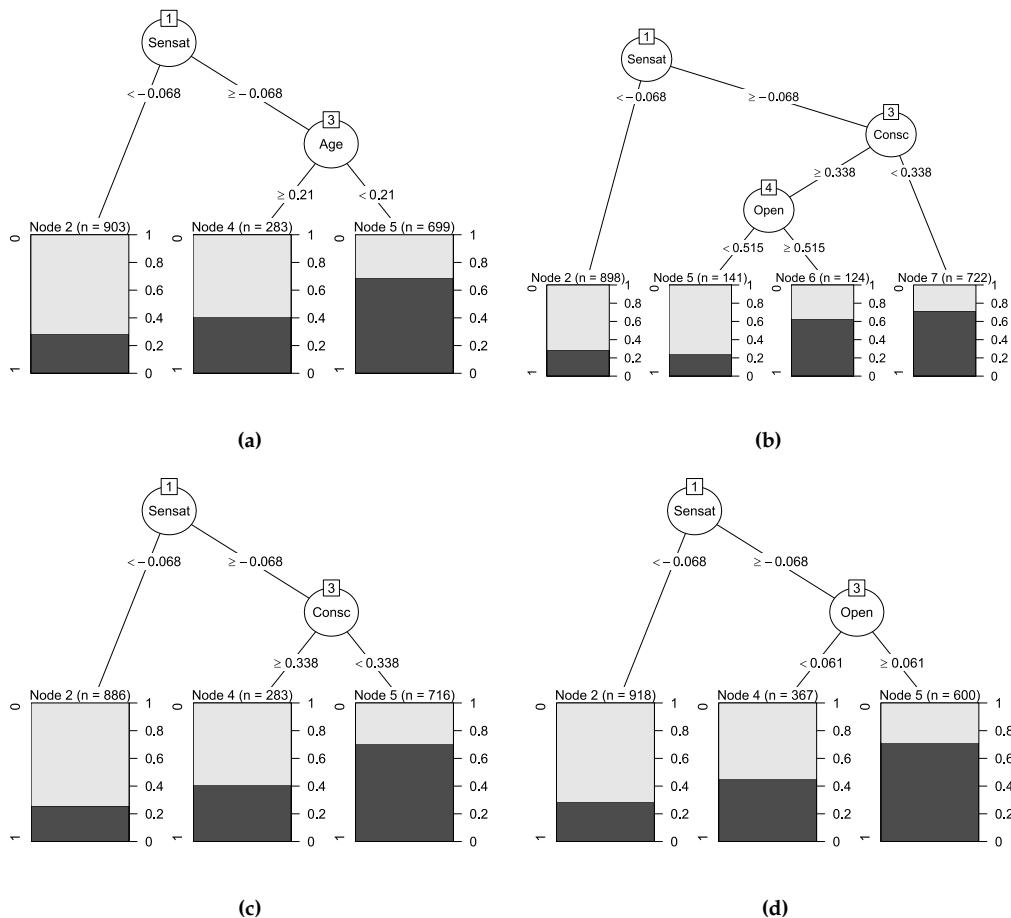


Figure 5: Elements of medoid sets of 1-, 2-, 3-, and 4-cluster solution (in addition to Figure 3) for Example 1 on ecstasy use. The ordered medoid sets for the different cluster solutions are as follows. For the 1-cluster solution: (Fig. 3); for the 2-cluster solution: (Fig. 5 (a), Fig. 5 (b)); for the 3-cluster solution: (Fig. 5 (a), Fig. 3, Fig. 5 (c)); for the 4-cluster solution: (Fig. 5 (a), Fig. 3, Fig. 5 (c), Fig. 5 (d)).

To determine the sizes of clusters in each solution, we can utilize the `cluster()` function on the `clusterforest` object, with as second argument the solution number of interest. We combine this with the `table()` function to count the number of trees assigned to each cluster.

```
R> table(clusters(ClusterForest, solution=2))
```

```

1 2
48 52

R> table(clusters(ClusterForest, solution=3))

1 2 3
44 25 31

R> table(clusters(ClusterForest, solution=4))

1 2 3 4
44 20 27 9

```

These figures show that in the 2-, 3-, and 4-cluster solutions almost half of the trees belong to the cluster with Age as secondary splitting variable. Moreover, in the three- and four-cluster solutions, the cluster with Conscientiousness as secondary splitting variable is nonnegligible, too.

Taking all these findings together, the role of Sensation seeking as primary predictor of ecstasy use is without a shadow of doubt. Sensation seeking is the first and only splitting variable in the classification tree estimated from the full data set as well in the medoid of the one-cluster solution. Moreover, it also acts as the primary splitting variable in the subsequent medoids of solutions with two or more clusters. On top of that, we may notice a relatively high proportion of people at risk for ecstasy use in all subgroups of the people scoring high on Sensation seeking. There are indications that, within the group of high sensation seekers, especially younger people (and, to a somewhat lesser extent, people scoring lower on Conscientiousness) are even more at risk for ecstasy use. One might wish to corroborate the latter results on the role of these secondary predictors in follow-up research.

4.2 Example 2: Predicting Ecstasy use

Whereas the goal of the first analysis was to find out which types of persons are at risk of using ecstasy, the goal of the second analysis is to *predict* (as accurately as possible) whether a person is at risk of using ecstasy. As the accuracy of ensemble methods is normally higher than that of single decision trees, we will revert to a random forest to address this objective. In line with the "Explainable AI" movement (Arrieta et al., 2020), we will subsequently use C443 to get insight into this random forest.

We use the same data as in the previous section, split it into a training and test set, and then fit a random forest on the training set using the **randomForest** package. Note that we could also have used the **ranger** package for this application. We limit the maximum number of nodes of each tree to six (to facilitate the interpretation of individual trees) and use default settings for the hyperparameters.

```

R> EcstData$Ecst <- as.factor(EcstData$Ecst)
R> train.id <- sample(nrow(EcstData), 3/4 * nrow(EcstData))
R> EcstData.train <- EcstData[train.id, ]
R> EcstData.test <- EcstData[-train.id, ]

R> DrugsRF <- randomForest(Ecst ~ ., data = EcstData.train, importance = TRUE,
+   proximity = TRUE, keep.inbag = T, maxnodes = 6, ntree = 500)

R> preds_test=predict(DrugsRF,EcstData.test)
R> preds_train=predict(DrugsRF,EcstData.train)

R> mean(preds_test==EcstData.test$Ecst)
[1] 0.6779661
R> mean(preds_train==EcstData.train$Ecst)
[1] 0.7296532

```

The prediction accuracy of the random forest on the training set is 0.73, and on the test set 0.68.

Next, we cluster the trees in the forest using C443 with the same similarity measure as in the previous section (as it takes into account both the trees' meaning and classifications). We evaluate solutions with up to 20 clusters.

```

R> set.seed(1)
R> ClusterForest<- clusterforest(observeddata = EcstData.train, trees = DrugsRF,
+                                     m = 6, fromclus = 1, toclus = 20)
R> plot(ClusterForest, predictive_plots=TRUE )

```

The plot function with `predictive_plots=TRUE` allows us to compare the different solutions in terms of their accuracy on the training set, as well as in terms of the agreement in the predictions with the random forest as a whole. The plot with Accuracy shows us that even with the one cluster solution, the accuracy on the training set is close to that of the random forest. The agreement in predictions is above 90% starting from the two-cluster solution.

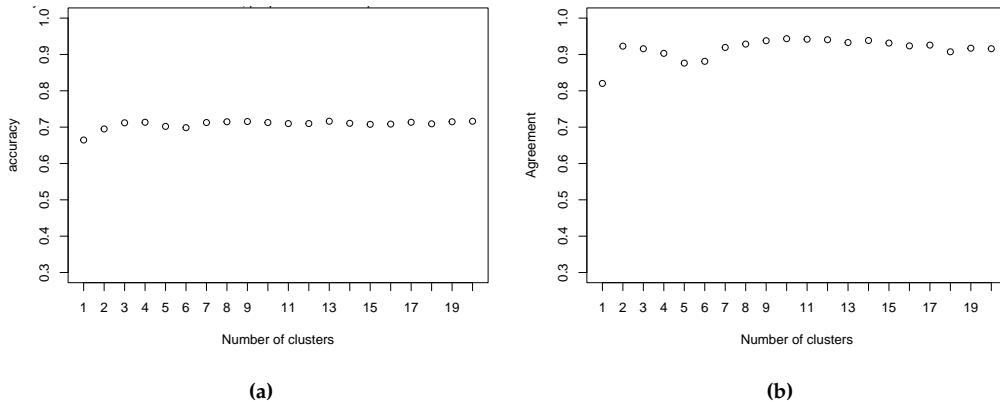


Figure 6: Plots to evaluate the different clustering solutions, with (a) the accuracy on the training dataset for each clustering solution and (b) the agreement of predictions between a given cluster solution and the full forest.

As the accuracy on the training set is improving up and until the three cluster solution, we plot the medoids of this solution.

```
R> plot(ClusterForest, solution = 3)
```

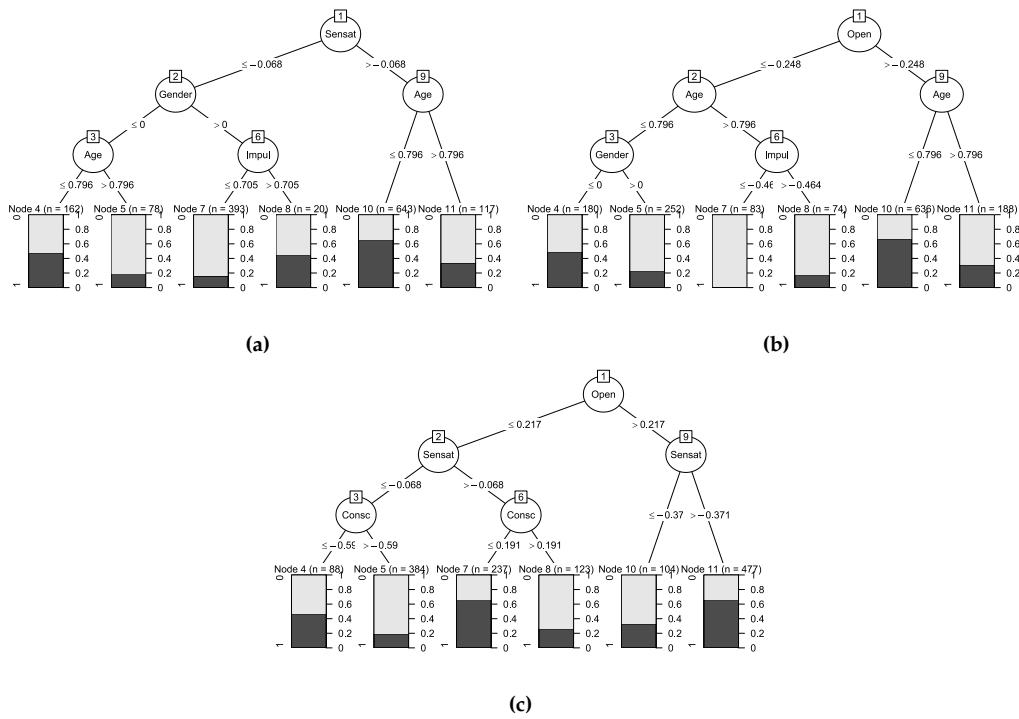


Figure 7: Medoids of the 3-cluster solution for Example 2 on ecstasy use.

In the first two medoids, Figure 7 (a) and (b), for one of the terminal nodes only, the class assignment is 1 (i.e., ecstasy use). In the first medoid this is the case for people with Sensation Seeking >-0.068 and Age ≤ 0.796 , in the second medoid this is the case for people with Openness >-0.248 and Age ≤ 0.796 . There is a correlation between Sensation Seeking and Openness, and, indeed, when looking at the class assignments of these two medoids, it turns out that they are for a large part overlapping:

```
R> table(EcstData.train$Age<=0.796 & EcstData.train$Sensat>-0.068,
EcstData.train$Age<=0.796 & EcstData.train$Open>-0.248)
      FALSE TRUE
    FALSE   556 201
    TRUE    162 494
```

In the third medoid, two terminal nodes are assigned to class 1, namely: 1) the node where Openness >0.217 and Sensation Seeking >-0.371 , and 2) the node where Openness ≤ 0.217 and Sensation seeking >-0.68 and Conscientiousness ≤ 0.191 .

Finally, we estimate the accuracy of the three-cluster solution (where the predicted class is the average of the predicted class by each medoid tree weighted by the number of trees assigned to its cluster) on the test set, and compare it to that of the random forest as a whole. We also check the agreement between the predictions of the two on the test set. To do so, we use the randomForest2party function (an internal function of the R-package) to transform the two medoid trees of the forest into a party object.

```
R> table(clusters(ClusterForest,solution=3))
  1   2   3
191 133 176
R> tree310=C443:::randomForest2party(EcstData.train, DrugsRF, ClusterForest$medoids[[3]][1])
R> tree329=C443:::randomForest2party(EcstData.train, DrugsRF, ClusterForest$medoids[[3]][2])
R> tree453=C443:::randomForest2party(EcstData.train, DrugsRF, ClusterForest$medoids[[3]][3])

preds_test310=predict(tree310,EcstData.test )
preds_test329=predict(tree329,EcstData.test )
preds_test453= predict(tree453,EcstData.test )
preds_total = round(((as.numeric(preds_test310)-1)*table(clusters(ClusterForest,solution=3))[1]+
(as.numeric(preds_test329)-1)*table(clusters(ClusterForest,solution=3))[2]+
(as.numeric(preds_test453)-1)*table(clusters(ClusterForest,solution=3))[3])/500 , 0)

R> mean(preds_total==EcstData.test$Ecst)
[1] 0.6716102
R> mean(preds_total==preds_test)
[1] 0.9088983
```

As a relatively similar number of trees is assigned to each cluster, the prediction of the three cluster solution comes down to a majority vote. It turns out that also on the test data set, the accuracy is very similar to that of the full forest (67% compared to 68% for the full forest), and the agreement of predictions with the full forest is still around 91%. Therefore, in the current example, without losing much predictive power, one could use a summarized forest with only three trees instead of a forest with 500 trees.

4.3 Example 3: Multiple outcome variables

The goal of our analysis in this third example is to find out which are the common and distinctive psychological mechanisms for being at risk for using different drugs (amphetamines, benzodiazepines, ecstasy, LSD and mushrooms). To answer this question, we will draw 100 bootstrap samples for each drug, and grow a classification tree on each sample. Subsequently, we will use **C443** to calculate similarities between these trees, and cluster them. Finally, we will post-process the obtained clustering result to answer the question outlined above.

We start by creating a separate data set for each drug:

```
R> AmphetData <- drugs[, c("Age", "Gender", "Edu", "Neuro", "Extr", "Open",
+ "Agree", "Consc", "Impul", "Sensat", "Amphet")]
> BenzoData <- drugs[, c("Age", "Gender", "Edu", "Neuro", "Extr", "Open",
+ "Agree", "Consc", "Impul", "Sensat", "Benzos")]
> EcstData <- drugs[, c("Age", "Gender", "Edu", "Neuro", "Extr", "Open",
+ "Agree", "Consc", "Impul", "Sensat", "Ecst")]
> LSDData <- drugs[, c("Age", "Gender", "Edu", "Neuro", "Extr", "Open",
+ "Agree", "Consc", "Impul", "Sensat", "LSD")]
> MushData <- drugs[, c("Age", "Gender", "Edu", "Neuro", "Extr", "Open",
+ "Agree", "Consc", "Impul", "Sensat", "Mush")]
```

Next, for each drug, we draw the bootstrap samples and grow a tree on every sample. We use the DrawBoots and GrowTree functions (already used in Example 1) to do so:

```
R> BootsAmphet <- lapply(1:100, function(k) DrawBoots(AmphetData, k))
R> set.seed(123456)
R> TreesAmphet <- lapply(1:100, function (i) GrowTree(x = c("Age", "Gender", "Edu",
+ "Neuro", "Extr", "Open", "Agree", "Consc", "Impul", "Sensat"), y="Amphet", BootsAmphet[[i]]))

> BootsBenzo <- lapply(1:100, function(k) DrawBoots(BenzoData, k))
> set.seed(123456)
R> TreesBenzo <- lapply(1:100, function (i) GrowTree(x = c("Age", "Gender", "Edu", "Neuro", "Extr",
+ "Open", "Agree", "Consc", "Impul", "Sensat"), y = "Benzos", BootsBenzo[[i]]))
>
> BootsEcst <- lapply(1:100, function(k) DrawBoots(EcstData, k))
> set.seed(123456)
R> TreesEcst <- lapply(1:100, function (i) GrowTree(x = c("Age", "Gender", "Edu", "Neuro", "Extr",
+ "Open", "Agree", "Consc", "Impul", "Sensat"), y="Ecst", BootsEcst[[i]]))
>
> BootsLSD <- lapply(1:100, function(k) DrawBoots(LSDData, k))
> set.seed(123456)
R> TreesLSD <- lapply(1:100, function (i) GrowTree(x = c("Age", "Gender", "Edu", "Neuro", "Extr",
+ "Open", "Agree", "Consc", "Impul", "Sensat"), y = "LSD", BootsLSD[[i]]))
>
> BootsMush <- lapply(1:100, function(k) DrawBoots(MushData, k))
> set.seed(123456)
R> TreesMush <- lapply(1:100, function (i) GrowTree(x = c("Age", "Gender", "Edu", "Neuro", "Extr",
+ "Open", "Agree", "Consc", "Impul", "Sensat"), y = "Mush", BootsMush[[i]]))
```

We then create one list with all data sets, and one list with all trees.

```
R> Boots <- c(BootsAmphet, BootsBenzo, BootsEcst, BootsLSD, BootsMush)
R> Trees <- c(TreesAmphet, TreesBenzo, TreesEcst, TreesLSD, TreesMush)
```

We use the resulting forest as input for the `clusterforest()` function. We also provide a vector with a covariate value of each tree, that is to say, the response variable (`drug`) on which the tree was based. We choose the same similarity measure as in Example 1.

```
R> set.seed(1)
R> ClusterForestMultiple <- clusterforest(observeddata=drugs, treedata=Boots, trees=Trees,
+ m=6, fromclus=1, toclus=8, treecov=rep(c("Amphet", "Benzo", "Ecst",
+ "LSD", "Mush"), each = 100))
```

Again, we use the `plot()` function to estimate the number of clusters needed to summarize the forest:

```
R> par(mfrow = c(2, 2))
R> plot(ClusterForestMultiple)
```

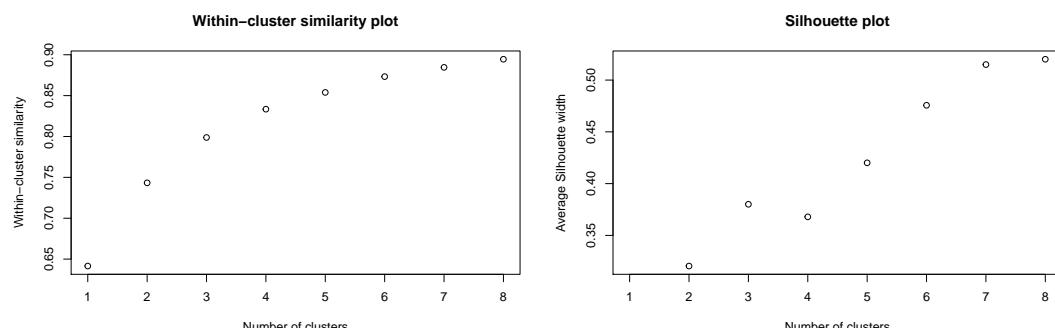


Figure 8: Average within-cluster similarity and ASW for solutions with 1 through 8 clusters for Example 3 on use of amphetamines, benzodiazepines, ecstasy, LSD and mushrooms.

The two plots (Figure 8) do not clearly point at an optimal solution. Although the ASW plot seems to favor solutions with a higher number of clusters, the three-cluster solution may be interesting to explore, because its ASW stands out compared to the 2- and 4-cluster solutions. All

things considered, this seems to be an interesting parsimonious solution for further examination. We will do so from the point of view of the relation between the clustering and the type of drug at the basis of each tree in the forest, both in extensional terms (viz., with regard to the relation between cluster membership and the type of response variable underlying each of the trees in the forest) and in intensional terms (viz., with regard to the relation between medoid contents and the substantive nature of each of the response variables).

To understand the relation between the clusters' extension and the type of response variable underlying each of the trees in the forest, we will count for each response variable the number of trees that have been assigned to each cluster. This will provide insight into whether the trees of the same versus different response variables mostly end up in the same versus in different clusters. As such, this may clarify whether there is a relationship between the nature of the drug and the clustering's extension, while factoring in tree instability (as reflected by whether trees based on the same drug consistently belong to the same cluster or not). We do this using the `treesource` function:

```
R> treesource(ClusterForestMultiple, solution= 3)
```

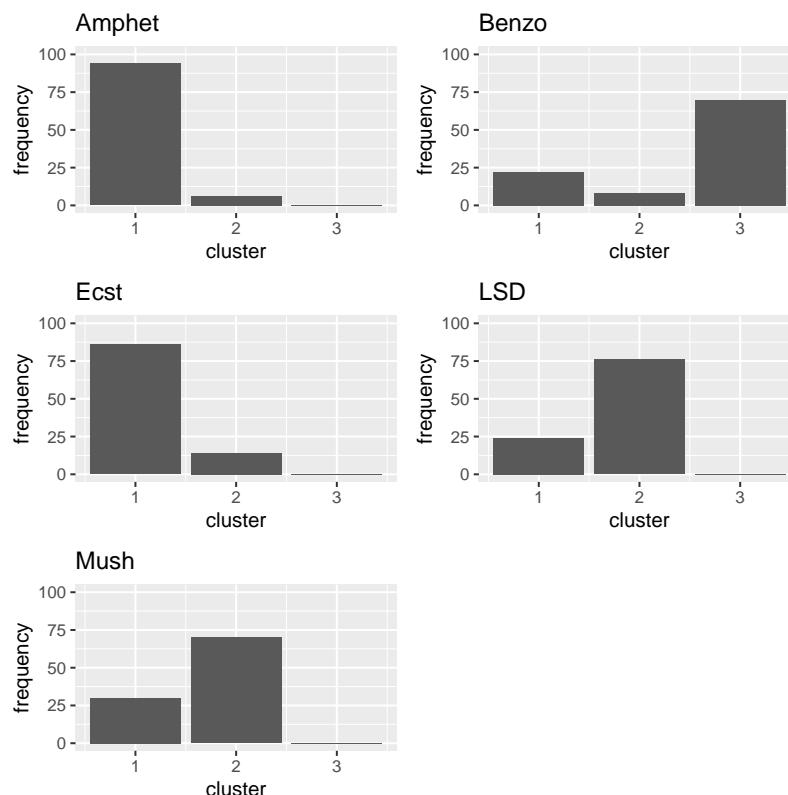


Figure 9: For each response variable frequencies of associated trees that were assigned to each of the clusters of the three-cluster solution for Example 3 on use of amphetamines, benzodiazepines, ecstasy, LSD and mushrooms.

From the result, we may conclude that the trees of most response variables are relatively stable (in the sense that for most response variables a clear majority of the trees is assigned to one specific cluster). Furthermore, it is interesting to note that the trees of the two stimulant drugs (viz., ecstasy and amphetamines) mainly end up in the same cluster and that the same holds for the trees of the two hallucinant drugs (viz., mushrooms and LSD). Conversely, the trees of the single tranquilizing drug (viz., benzodiazepines) mostly end up in a separate cluster.

To clarify the intensional relation between the clustering and the substantive nature of each of the response variables, we first plot the medoids of the three-cluster solution.

```
R> plot(ClusterForestMultiple, solution = 3)
```

The first, second and third medoid are shown in Figure 10. The medoids, along with the cluster membership frequencies of Figure 9, imply that Sensation seeking plays an important role in being at risk for using the two stimulant drugs (i.e., ecstasy and amphetamines, see Figure 10 (a)). To be at risk for using the hallucinatory drugs (LSD and mushrooms), apart from a higher score on Sensation

seeking, also a higher score on Openness to experience is needed (Figure 10 (b)). Finally, for using benzodiazepines (the only tranquilizing group of drugs), Neuroticism seems to play a key role, with people scoring higher on the Neuroticism scale being more at risk for using the drugs in question (Figure 10 (c)). All in all, from this analysis, we may derive that there is some overlap in the types of people that are at risk for using the different drugs under study, with personality characteristics differentiating in a meaningful way between people susceptible to the use of stimulant, hallucinatory, and tranquilizing substances.

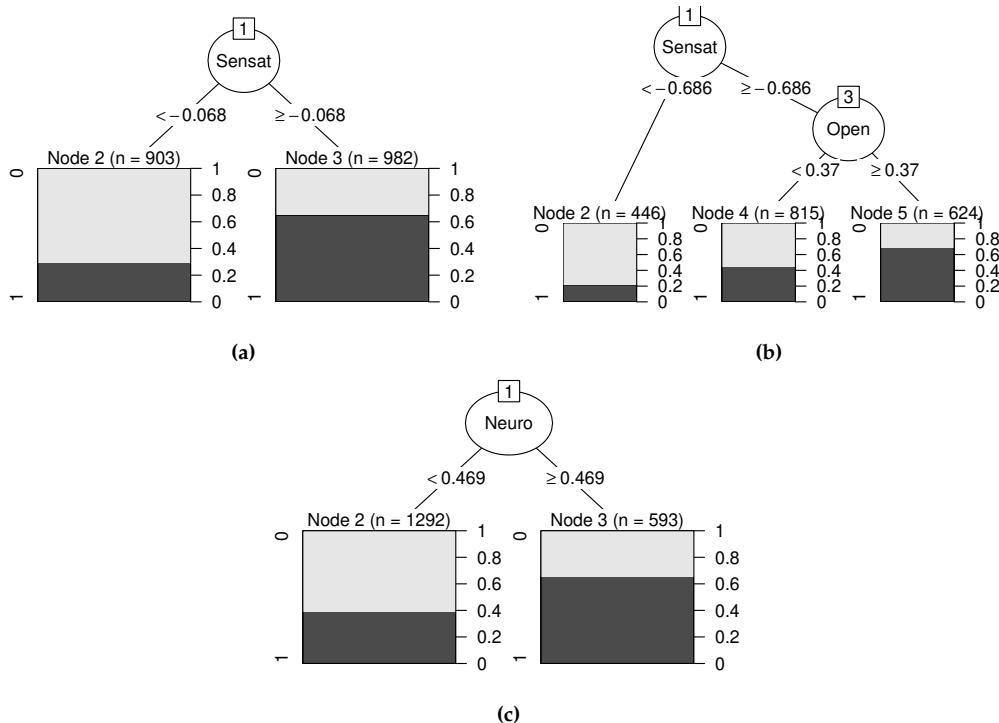


Figure 10: The first, second and third medoid of the three-cluster solution for Example 3 on use of amphetamines, benzodiazepines, ecstasy, LSD and mushrooms.

5 Concluding remark

In this paper we presented the R-package **C443**, which solves a major problem of trees (viz., their lack of stability) and a major problem of forests (viz., their black box nature). **C443** is well accessible by accepting as input trees produced by a broad range of R-packages, including all possible trees that are stored as party objects or that are convertible into such objects, including **ranger** and **randomForest** objects. Moreover, the package is also versatile by both allowing to make an appeal to in-built tools for calculating various tree similarity measures and by accepting as input all possible user-provided tree similarities. In the same vein, the package includes a comprehensive set of in-built tools to postprocess the results generated by the methodology, and allows for the integration of other user-provided postprocessing tools. **C443** operates via an optimal involvement of and synergy with neighboring R-packages such as **partykit**, **randomForest**, **ranger** and **cluster**. As a bonus, the internal functions **C443:::randomForest2party** and **C443:::ranger2party** can also be applied outside the context of **C443** to print, plot, or post-process trees obtained from the **randomForest** or **ranger** packages. As further illustrated by three applications, **C443** is user-friendly and may yield various insights into forests of classification trees and classification-related questions.

Acknowledgments

The research reported in this paper was supported in part by the Research Foundation - Flanders (G080219N and K802822N) and by the Research Fund of KU Leuven (C14/19/054). The data used in the illustrative applications were obtained from the UCI Machine Learning Repository (Dua and Graff, 2017; Fehrman et al., 2017).

References

- E. Alfaro, M. Gámez, and N. García. adabag: An R package for classification with boosting and bagging. *Journal of Statistical Software*, 54(2):1–35, 2013. URL <http://www.jstatsoft.org/v54/i02/>. [p⁶⁴]
- A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, et al. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information fusion*, 58:82–115, 2020. [p⁷⁰]
- M. Banerjee, Y. Ding, and A.-M. Noone. Identifying representative trees from ensembles. *Statistics in Medicine*, 31(15):1601–1616, 2012. doi: 10.1002/sim.4492. [p⁵⁹]
- E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139, 1999. [p⁵⁹]
- L. Breiman. Heuristics of instability and stabilization in model selection. *The Annals of Statistics*, 24(6): 2350–2383, 1996a. doi: 10.1214/aos/1032181158. [p⁵⁹]
- L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996b. [p⁵⁹]
- L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324. [p⁵⁹, 64]
- L. Breiman and A. Cutler. Setting up, using, and understanding random forests v3.1. *University of California, Department of Statistics*, 2003. [p⁵⁹]
- L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Wadsworth International Group, Belmont, 1984. [p⁵⁹, 64]
- B. Briand, G. R. Ducharme, V. Parache, and C. Mercat-Rommens. A similarity measure to assess the stability of classification trees. *Computational Statistics & Data Analysis*, 53(4):1208–1217, 2009. doi: 10.1016/j.csda.2008.10.033. [p⁵⁹]
- H. Chipman, E. George, and R. McCulloh. Making sense of a forest of trees. In S. Weisberg, editor, *Computing Science and Statistics, Proceedings of the 30th Symposium on the Interface.*, pages 84–92, Fairfax, VA, 1998. Interface Foundation of North America. [p⁵⁹, 61]
- T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning*, 40(2):139–157, 2000. [p⁵⁹]
- D. Dua and C. Graff. Uci machine learning repository, 2017. URL <http://archive.ics.uci.edu/ml>. [p⁶⁶, 75]
- E. Fehrman, A. K. Muhammad, E. M. Mirkes, V. Egan, and A. N. Gorban. The five factor model of personality and evaluation of drug consumption risk. In *Data Science*, pages 231–242. Springer, 2017. doi: 10.1037/10140-001. [p⁶⁶, 75]
- M. Fokkema. Fitting prediction rule ensembles with r package pre. *Journal of Statistical Software*, 92 (12), 2020. [p⁵⁹]
- Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997. [p⁵⁹, 64]
- J. H. Friedman and B. E. Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3):916–954, 2008. [p⁵⁹]
- T. J. Hastie, R. J. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, 2009. doi: 10.1007/978-0-387-84858-7. [p⁵⁹, 67]
- K. Hornik, C. Buchta, and A. Zeileis. Open-source machine learning: R meets Weka. *Computational Statistics*, 24(2):225–232, 2009. doi: 10.1007/s00180-008-0119-7. [p⁶⁴]
- T. Hothorn, K. Hornik, and A. Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, 15(3):651–674, 2006. doi: 10.1198/106186006X133933. [p⁶⁴]
- T. Hothorn and A. Zeileis. partykit: A modular toolkit for recursive partytioning in R. *Journal of Machine Learning Research*, 16:3905–3909, 2015. URL <http://jmlr.org/papers/v16/hothorn15a.html>. [p⁶⁰, 64]

- L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, Hoboken, 1990. doi: 10.1002/9780470316801. [p60, 62, 63, 65]
- G. W. Leibniz. Nouveaux essais sur l'entendement humain, livre iv, chap. xvii, 1764. [p60]
- A. Liaw and M. Wiener. Classification and regression by randomForest. *R News*, 2(3):18–22, 2002. URL <https://CRAN.R-project.org/doc/Rnews/>. [p64]
- S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee. From local explanations to global understanding with explainable ai for trees. *Nature machine intelligence*, 2(1):56–67, 2020. [p60]
- M. Maechler, P. Rousseeuw, A. Struyf, M. Hubert, and K. Hornik. cluster: *Cluster Analysis Basics and Extensions*, 2019. [p60, 63, 65]
- R. R. McCrae and P. T. Costa. A contemplated revision of the neo five-factor inventory. *Personality and Individual Differences*, 36(3):587–596, 2004. doi: 10.1016/s0191-8869(03)00118-1. [p66]
- J. H. Patton, M. S. Stanford, and E. S. Baratt. Factor structure of the barratt impulsiveness scale. *Journal of Clinical Psychology*, 51(6):768–774, 1995. doi: 10.1002/1097-4679(199511)51:6<768::aid-jclp2270510607>3.0.co;2-1. [p66]
- B. Pfeifer, H. Baniecki, A. Saranti, P. Biecek, and A. Holzinger. Multi-omics disease module detection with an explainable greedy decision forest. *Scientific Reports*, 12(1):1–15, 2022. [p64]
- M. Philipp, T. Rusch, K. Hornik, and C. Strobl. Measuring the stability of results from supervised statistical learning. *Journal of Computational and Graphical Statistics*, 27(4):685–700, 2018. [p59]
- M. Philipp, A. Zeileis, and C. Strobl. A toolkit for stability assessment of tree-based learners. In A. Colubi, A. Blanco, and C. Gatu, editors, *Proceedings of COMPSTAT 2016 – 22nd International Conference on Computational Statistics*, pages 315–325. The International Statistical Institute/International Association for Statistical Computing, 2016. ISBN 978-90-73592-36-0. [p59]
- J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. [p59]
- J. R. Quinlan. *C4. 5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993. [p64]
- G. Ridgeway. Generalized boosted models: A guide to the gbm package. *Update*, 1(1):2007, 2007. [p64]
- P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. doi: 10.1016/0377-0427(87)90125-7. [p63, 66]
- D. B. Rubin. *Multiple Imputation for Nonresponse in Surveys*. John Wiley & Sons, 1987. [p59, 64]
- E. Schubert and P. J. Rousseeuw. Faster k-medoids clustering: Improving the pam, clara, and clarans algorithms. In *International Conference on Similarity Search and Applications*, pages 171–187. Springer, 2019. [p65]
- W. D. Shannon and D. Banks. Combining classification trees using mle. *Statistics in Medicine*, 18(6):727–740, 1999. doi: 10.1002/(sici)1097-0258(19990330)18:6<727::aid-sim61>3.3.co;2-u. [p61]
- A. Sies and I. Van Mechelen. C443: a methodology to see a forest for the trees. *Journal of Classification*, 37:730–753, 2020. doi: 10.1007/s00357-019-09350-4. [p60, 63, 65]
- M. Skurichina and R. P. Duin. Bagging, boosting and the random subspace method for linear classifiers. *Pattern Analysis & Applications*, 5(2):121–135, 2002. [p59]
- C. Strobl, J. Malley, and G. Tutz. An introduction to recursive partitioning: Rationale, application, and characteristics of classification and regression trees, bagging, and random forests. *Psychological Methods*, 14(4):323, 2009. doi: 10.1037/a0016973. [p59]
- T. Therneau, B. Atkinson, and B. Ripley. Package rpart. Available online: <https://CRAN.R-project.org/package=rpart>, 2015. [p64]
- P. Turney. Technical note: Bias and the quantification of stability. *Machine Learning*, 20(1):23–33, 1995. doi: 10.1007/bf00993473. [p59]
- S. van Buuren and K. Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software*, 45(3):1–67, 2011. URL <https://www.jstatsoft.org/v45/i03/>. [p64]

- M. N. Wright and A. Ziegler. ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77(1):1–17, 2017. doi: 10.18637/jss.v077.i01. [p64]
- M. Zuckerman, D. M. Kuhlman, J. Joireman, P. Teta, and M. Kraft. A comparison of three structural models for personality: The big three, the big five, and the alternative five. *Journal of Paediatric Personality and Social Psychology*, 65(4):757, 1993. doi: 10.1037//0022-3514.65.4.757. [p66]

Aniek Sies

KU Leuven, Faculty of Psychology and Educational Sciences
Tiensestraat 102 - Box 3713
3000
Belgium
aniek.sies@kuleuven.be

Iven Van Mechelen

KU Leuven, Faculty of Psychology and Educational Sciences
iven.vanmechelen@kuleuven.be

TwoSampleTest.HD: An R Package for the Two-Sample Problem with High-Dimensional Data

by Marta Cousido-Rocha and Jacobo de Uña-Álvarez

Abstract The two-sample problem refers to the comparison of two probability distributions via two independent samples. With high-dimensional data, such comparison is performed along a large number p of possibly correlated variables or outcomes. In genomics, for instance, the variables may represent gene expression levels for p locations, recorded for two (usually small) groups of individuals. In this paper we introduce `TwoSampleTest.HD`, a new R package to test for the equal distribution of the p outcomes. Specifically, `TwoSampleTest.HD` implements the tests recently proposed by (Cousido-Rocha, Uña-Álvarez, and Hart 2019) for the low sample size, large dimensional setting. These tests take the possible dependence among the p variables into account, and work for sample sizes as small as two. The tests are based on the distance between the empirical characteristic functions of the two samples, when averaged along the p locations. Different options to estimate the variance of the test statistic under dependence are allowed. The package `TwoSampleTest.HD` provides the user with individual permutation p -values too, so feature discovery is possible when the null hypothesis of equal distribution is rejected. We illustrate the usage of the package through the analysis of simulated and real data, where results provided by alternative approaches are considered for comparison purposes. In particular, benefits of the implemented tests relative to ordinary multiple comparison procedures are highlighted. Practical recommendations are given.

1 Introduction

One of the most important questions in modern statistics is how to efficiently deal with the low sample size, high dimensional setting, in which a large number p of variables are measured for a relatively small number of individuals. This type of high-dimensional data arises in many different areas of science, such as genetics, medicine, pharmacy and social sciences. In microarray data, for example, the variables typically represent the expression levels of a large set of genes. In such a context, a usual goal is to compare the distributions of the gene expression levels for individuals with two different tumour types. Hence, a formal two-sample test in the low sample size and large dimension setting is required. More precisely, the aim is to test for the equality of the p marginal distributions for the two groups. In other words, one can regard the null hypothesis as the intersection of the p null hypotheses corresponding to each of the p locations (genes).

In the majority of examples with high-dimensional data the large number of variables or outcomes are not independent. In genetics, for example, dependency among expression levels of different genes on the same individual is often observed. Several two-sample tests have been developed for the high-dimensional setting under dependence; see for instance Biswas and Gosh (2014), Mondal et al. (2015), Biswas et al. (2014), Liu et al. (2015) and Wei et al. (2016). Nevertheless, all of these proposals have at least one of the following disadvantages: (1) the null hypothesis asserts that the p -variate distribution is the same for the two groups being compared instead of testing the equality of the univariate marginals; (2) the dependence structure is not considered or is too restrictive; (3) the theoretical results are only suitable for normally distributed data. Besides, to the best of our knowledge none of these methods are available in R. While gaps (2) and (3) are limiting in applications, issue (1) is more fundamental; note that usually the focus is more on the marginal outcome distribution than on the within-group correlation structure. Therefore, new ideas are needed.

Recently, Cousido-Rocha et al. (2019) overcame the aforementioned flaws by introducing a non-parametric omnibus test that, with focus on the marginal distributions, included the dependent case through mixing conditions (Doukhan, 1995). This type of dependence, being fairly general, has been frequently used in the goodness-of-fit testing literature; see for example Neumann and Paparoditis (2000) and Dehling et al. (2015). Mixing conditions imply that the dependence between the variables softens at distant locations. In genetics, for instance, this means that the correlation among expression levels of different genes lessens as the distance between the biological function of the genes increases, which is a flexible, realistic assumption for such applications.

In this paper we introduce the `TwoSampleTest.HD` R package which implements the tests proposed in Cousido-Rocha et al. (2019) for testing the (global, or intersection) null hypothesis of equality of the p univariate marginals in the two populations. The basic test statistic is the L_2 -distance between the empirical characteristic functions pertaining to the two groups, when averaged along the p locations.

Several approaches to estimate the variance of the test statistic under dependence lead then to slightly different procedures. At the same time, **TwoSampleTest.HD** provides the user with permutation p -values for each location. When the null hypothesis is rejected, these p -values can be used to rank the locations according to their contribution to the global significance, or for feature selection by performing multiple testing (Dudoit and van der Laan, 2007). Finally, a different test statistic for the global null hypothesis based on the average of the permutation p -values is implemented within **TwoSampleTest.HD**. All of these procedures are fully illustrated in this piece of work.

Alternative nonparametric approaches for the two-sample problem include Kolmogorov-Smirnov and Cramér-von Mises tests. These methods compare the empirical distribution functions, rather than the empirical characteristic functions, of the two groups. Empirical characteristic functions are related to smooth tests, which have been found preferable to distribution-based tests in many settings due to their greater power (Martínez-Camblor and de Uña-Álvarez, 2009). More importantly, whenever a test is performed locally and repeatedly, multiple comparison procedures (MCP) are needed in order to keep the type I error under control. Unfortunately, such approach may not be optimal when testing for different distributions in a global way. This relays the fact that feature discovery is more difficult than testing for the intersection null, and explains why the test based on permutation p -values implemented in **TwoSampleTest.HD** can exhibit a power lower than that of the averaged L_2 -type tests within the package. Efforts to efficiently summarize local p -values for testing an intersection null hypothesis include the so-called Higher Criticism (HC) approach, see Zhang et al. (2020) and references therein; however, the performance of HC may be dramatically affected by dependence (Hall and Jin, 2008). Further discussion of these issues is provided within this paper on the basis of empirical results.

The rest of the paper is organized as follows. In Section 2.2 the methodological background is introduced, and the tests proposed by Cousido-Rocha et al. (2019) are presented in detail. In Section 2.3 the **TwoSampleTest.HD** package is described, and its usage is illustrated through the analysis of simulated data and microarray data derived from a hereditary breast cancer study. Finally, Section 2.4 reports the main conclusions of this work.

2 Methodology

In this section we describe the four two-sample tests proposed by Cousido-Rocha et al. (2019), which are implemented in the **TwoSampleTest.HD** package. Three of the methods are based on the average of p individual L_2 -distances between the empirical characteristic functions computed from the two samples. The three versions of this test differ in the way in which the variance is estimated; in all the cases, the variance estimate takes the possible dependence among the p outcomes into account. The fourth method is based on the average of the permutation p -values derived for the individual L_2 -distances.

We consider two random matrices $X = [X_1, \dots, X_p]^T$ and $Y = [Y_1, \dots, Y_p]^T$ of respective dimensions $p \times n$ and $p \times m$, where $X_k = (X_{k1}, \dots, X_{kn})$ and $Y_k = (Y_{k1}, \dots, Y_{km})$, $k = 1, \dots, p$, are the sample values for the p target variables; the sample sizes are n and m . The variables X_k and Y_k may be discrete or continuous; normality is not assumed in the continuous case. Given sequences of characteristic functions $\{C_{X_1}, C_{X_2}, \dots, C_{X_p}\}$ and $\{C_{Y_1}, C_{Y_2}, \dots, C_{Y_p}\}$, it is assumed that X_{k1}, \dots, X_{kn} and Y_{k1}, \dots, Y_{km} are independent random samples from C_{X_k} and C_{Y_k} , respectively. Observations X_{ij} and X_{sl} for $s \neq i$ can be dependent in our framework, and similar comments apply to the components of Y .

The focus is in testing for the intersection null hypothesis

$$H_0 = \bigcap_{k=1}^p H_{0k},$$

where, for $1 \leq k \leq p$, H_{0k} states that C_{X_k} and C_{Y_k} coincide. As indicated by Cousido-Rocha et al. (2019), the L_2 -distance between the empirical characteristic functions of X_k and Y_k is given by

$$\begin{aligned} J_k &= \frac{1}{n(n-1)} \sum_{j=1}^n \sum_{l=1, l \neq j}^n \exp \left(-\frac{1}{2} \left(\frac{X_{kj} - X_{kl}}{\sqrt{2b}} \right)^2 \right) \\ &+ \frac{1}{m(m-1)} \sum_{j=1}^m \sum_{l=1, l \neq j}^m \exp \left(-\frac{1}{2} \left(\frac{Y_{kj} - Y_{kl}}{\sqrt{2b}} \right)^2 \right) \\ &- \frac{2}{nm} \sum_{j=1}^n \sum_{l=1}^m \exp \left(-\frac{1}{2} \left(\frac{X_{kj} - Y_{kl}}{\sqrt{2b}} \right)^2 \right), \end{aligned} \quad (1)$$

where $b \in \mathbb{R}^+$. The first term in J_k is an intra-sample parameter estimate for the sample X_k and the second term is an intra-sample parameter estimate for the sample Y_k , whereas the third term is an *inter-samples* parameter estimate, since X_{kj} and $Y_{k\ell}$ come from different samples, X_k and Y_k , respectively.

We consider the test statistic $T_p = \sum_{k=1}^p J_k / \sqrt{p}$ in order to test for H_0 . [Cousido-Rocha et al. \(2019\)](#) introduced two variance estimators for T_p when $(X_k, Y_k)_{k \in \{1, \dots, p\}}$ comes from a strictly stationary sequence $(X_k, Y_k)_{k \in \mathbb{N}}$. The first variance estimator $\widehat{\sigma}_S$ is based on the spectral density estimate. Indeed, the problem of estimating the variance of a sample mean based on dependent data is the same as that of estimating the spectrum of the process at frequency zero. Hence, $\text{Var}(T_p)$ can be approximated through the classical estimator of the spectrum of the process at frequency zero. It holds that $T_p / \widehat{\sigma}_S \xrightarrow{D} \mathcal{N}(0, 1)$ as p tends to ∞ . Since the expectation of T_p is strictly positive under the alternative hypothesis, the test is one-sided, and rejects H_0 at nominal level α when $T_p / \widehat{\sigma}_S$ is larger than the $1 - \alpha$ quantile of the standard normal distribution. We refer to this test as spectral test.

The second variance estimator is derived from the block bootstrap procedure proposed by [Carlstein \(1996\)](#) to estimate the variance of a general statistic computed from a strictly stationary α -mixing sequence (see [Bosq, 1998](#)). More precisely, this resampling method defines no overlapping blocks of length l on the sequence $(J_k)_{k \in \{1, \dots, p\}}$ and computes the statistic T_p in each of the blocks, with l being the maximum lag of significant autocorrelation on the $(J_k)_{k \in \{1, \dots, p\}}$ sequence according to the procedure in [Politis and White \(2004\)](#). Finally the block bootstrap variance estimator $\widehat{\sigma}_B$ is simply defined as the sample variance of the values of T_p in each of the block. As before, the standardized version of T_p based on $\widehat{\sigma}_B$, is asymptotically distributed as a $\mathcal{N}(0, 1)$ random variate. Hence, the block bootstrap test rejects the null hypothesis when $T_p / \widehat{\sigma}_B$ is larger than the $1 - \alpha$ quantile of the standard normal distribution.

[Cousido-Rocha et al. \(2019\)](#) also introduced an alternative variance estimator suitable for possibly non-stationary sequences based on U -statistics theory. More precisely, the variance of T_p can be written as the sum of two terms; the first one is the average of the variance of the statistics $(J_k)_{k \in \{1, \dots, p\}}$, whereas the second one comprises the covariance terms arising from such sequence. The first term can be estimated by replacing the unknown theoretical expectations by their corresponding sample means; on the other hand, an unbiased estimator for the covariance term $\text{Cov}(J_j, J_{j+k})$ is given by $J_j J_{j+k}, j = 1, \dots, p-l, k = 1, \dots, l$. The standardized version of T_p based on such variance estimator, $T_p / \widehat{\sigma}_U$, is asymptotically distributed as a $\mathcal{N}(0, 1)$ as $p \rightarrow \infty$. Hence, the U -statistic test rejects the null hypothesis when $T_p / \widehat{\sigma}_U$ is larger than the $1 - \alpha$ quantile of the standard normal distribution.

Interestingly, the test statistics J_k can be used locally to test for the null hypotheses H_{0k} that X_{k1} and Y_{k1} have the same distribution, $1 \leq k \leq p$. Since the common density under H_{0k} is unknown, a permutation test can be used to calibrate the null distribution of J_k . The application of the permutation test to each of the $H_{0k}, k \in \{1, \dots, p\}$ yields a set of p -values $\{P_1, \dots, P_p\}$. [Cousido-Rocha et al. \(2019\)](#) proposed a test statistic based on the average of the permutation p -values, $\bar{P} = \sum_{k=1}^p P_k / p$. The idea of combining the individual p -values to obtain a global test statistic is old, dating to [Fisher \(1934\)](#); [Stouffer et al. \(1949\)](#) and others. The statistic P is standardized taking into account that its expectation is $(N+1)/2N$, with N being the number of permutations that lead to a different value of the statistic J_k , and using a variance estimator based on the spectral analysis. The standardized version of \bar{P} , say $T_p^{pv} = \sqrt{p}(\bar{P} - (N+1)/(2N)) / \widehat{\sigma}_P$, is asymptotically distributed as a standard normal as $p \rightarrow \infty$. It is assumed that, under H_0 , $(X_k, Y_k)_{k \in \{1, \dots, p\}}$ comes from a strictly stationary and strongly mixing process $(X_k, Y_k)_{k \in \mathbb{N}}$. The null hypothesis is rejected when T_p^{pv} is smaller than the α quantile of the standard normal distribution. One advantage of the permutation p -values is that, when the intersection null is rejected, they can be used to rank the null hypotheses H_{0k} according to their contribution to the significance. In genomics, for instance, this ranking may reveal the genes which express differently between two tumors. Finally, a formal MCP can be applied to the set of permutation p -values to get rigorous conclusions on the individual nulls $H_{0k}, k \in \{1, \dots, p\}$. Note that the ranking provided by the p -values is related, but not equal, to the ranking based on the J_k 's; this is because the target outcome may be differently distributed along the p locations, and J_k is not distribution free. The situation for the Hedenfalk data example in Section 2.3.1 is depicted in Figure 1; the shift of the p -values distribution compared to uniform suggests that some genes are differently expressed in the two groups considered.

For the implementation of the aforementioned test statistics the parameter b in (1), which plays the role of a smoothing parameter or bandwidth, is set to $\widehat{b} = 1.144 s_{pool} ((n+m)/2)^{-1/5}$, where s_{pool}^2 is the average of $((n-1)s_{X_k}^2 + (m-1)s_{Y_k}^2) / (n+m-2)$, $k = 1, \dots, p$, and $s_{X_k}^2$ and $s_{Y_k}^2$ are the sample variances of X_k and Y_k , respectively, $k = 1, \dots, p$. When the permutation p -values of the statistics J_k are to be computed, a local bandwidth can be used instead; specifically, the local bandwidth for J_k is given by $\widehat{b}_k = 1.144 s_{pool} ((n+m)/2)^{-1/5}$, with $s_{pool}^2 = ((n-1)s_{X_k}^2 + (m-1)s_{Y_k}^2) / (n+m-2)$, and $s_{X_k}^2$ and $s_{Y_k}^2$ are the sample variances of X_k and Y_k .

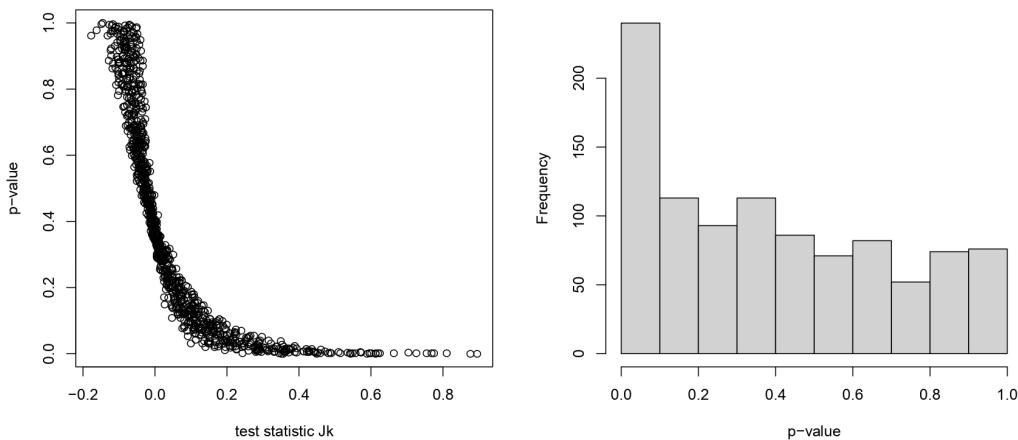


Figure 1: Left: p -values vs test statistics J_k . Right: histogram of the p -values. Hedenfalk data.

$p/n, m$	Spectral			Block bootstrap		
	2, 2	5, 5	10, 10	2, 2	5, 5	10, 10
100	0.01	0.02	0.01	0.01	0.02	0.03
500	0.03	0.03	0.06	0.03	0.05	0.10
1000	0.07	0.08	0.11	0.07	0.07	0.12
$p/n, m$	U -statistic			Permutation		
	2, 2	5, 5	10, 10	2, 2	5, 5	10, 10
100	0.01	0.11	0.58	0.03	0.89	1182.40
500	0.11	0.57	2.51	0.015	3.55	5122.53
1000	0.24	1.04	4.87	0.27	8.13	8006.70

Table 1: Execution time (in seconds) for the several versions of the proposed two-sample test. Simulated data with dimension p and sample sizes n and m .

3 Package TwoSampleTest.HD in practice

In this section the main features of **TwoSampleTest.HD** package are described. We also consider two examples with high-dimensional data in order to explain how to use **TwoSampleTest.HD** in practice. The first example refers a large number of gene expression levels measured on two groups of patients with breast cancer, classified according to BRCA mutation type. The second example is a simulation scenario in which the target outcome is differently distributed in the two groups for 10% of the p locations. This second example serves in particular to illustrate the smaller power of ordinary MCP when compared to the tests based on the averaged L_2 -distances between the empirical characteristic functions of the two groups.

3.1 Hedenfalk data

In this subsection we consider the microarray data set of hereditary breast cancer in [Hedenfalk et al. \(2001\)](#). The data set consists of $p = 3226$ logged gene expression levels measured on $n = 7$ patients with breast tumors having BRCA1 mutations and on $m = 8$ patients with breast tumors having BRCA2 mutations. The goal is to test the null hypothesis that the distribution of the p genes is the same for the two types of tumor, BRCA1 tumor and BRCA2 tumor. Since the example is merely illustrative, we only consider the first 1000 genes in order to save computational time. With 1000 locations, the execution time is reduced to < 1 second for the block bootstrap and spectral tests, to < 5 seconds for the U -statistic test and to 9 minutes for the permutation p -values test, in a laptop provided with a i5-1135G7 CPU. The waiting time of the permutation test is relatively long since $n = 7$ and $m = 8$ lead to 6435 permutations which must be carried out for each of the $p = 1000$ genes. For additional inspection, in Table 1 execution times for simulated data with several dimensions p and sample sizes n and m are reported.

The main function of the package is **TwoSampleTest.HD**. This function computes, among other things, the value of the selected test statistic and the corresponding p -value. The list of arguments of **TwoSampleTest.HD** is given in Table 2. The required arguments are X and Y , matrices where each row is one of the p -samples in the first group and second group, respectively; the other arguments have a default value. When the user forgets to include the argument X or Y in the function, the following message is returned:

```
Call:
TwoSampleTest.HD(X = X)
'us' method used by default
'global' bandwidth used by default
Error in ncol(Y) : argument "Y" is missing, with no default

Call:
TwoSampleTest.HD(Y= Y)
'us' method used by default
'global' bandwidth used by default
Error in ncol(X) : argument "X" is missing, with no default
```

With `method="spect"` the two-sample spectral test described in Section 2.2 is applied; the option `"method=spect_ind"` corresponds to a simplified version that pre-assumes the independence among the outcomes. On the other hand, the options `method="us"` and `method="us_ind"` apply the two-sample U -statistic test explained in Section 2.2 for dependent data and its simplification for independent variables, respectively. The last option based on the average of the p individual statistics, J_k , corresponding to each of the p variables is `method="boot"` which implements the two-sample block bootstrap test (Section 2.2). Finally, the function also performs the alternative test based on the average of the permutation p -values corresponding to the individual statistics J_k through the argument `method="perm"`. In our experience, the most powerful test for independent outcomes is the U -statistic test, whereas under dependence the more powerful tests are the spectral and block bootstrap tests. We also observed that the block bootstrap and spectral tests, which were developed assuming that $(X_k, Y_k)_{k \in \mathbb{N}}$ is a strictly stationary process, performed well when stationarity is violated. The "us" method has been defined as the default one.

When choosing `method="perm"`, the sequence of permutation p -values is computed and reported. On the other hand, the computation of the permutation p -values must be explicitly requested using `I.permutation.p.values=TRUE` argument for the U -statistic, spectral or block bootstrap tests. As mentioned, these individual p -values may be used to rank the outcomes according to their significance. Argument `b_I.permutation.p.values` allows the user to select the bandwidth b . The option `b_I.permutation.p.values="global"` computes a global bandwidth \hat{b} and uses it to evaluate the J_k 's, whereas option `b_I.permutation.p.values="individual"` estimates the bandwidth for each variable separately; see details in Section 2.2. The default option is `b_I.permutation.p.values="global"`. In Table 3 a summary of the results provided by the function **TwoSampleTest.HD** is given. The `I.statistics` object contains the individual statistics J_k , $k = 1, \dots, p$ described in the previous section, while the `I.permutation.p.values` object reports the permutation p -values $\{P_1, \dots, P_p\}$.

Hedenfalk data are available within **Equalden.HD** package (Cousido-Rocha and de Uña-Álvarez, 2022). In order to analyze this dataset, we load this package together with **TwoSampleTest.HD** package. For the investigation of the possible dependence among the gene expression levels, we treat the data of each patient as a time series, and we compute the sample autocorrelation function. For the first lags the autocorrelation between genes was significantly different from zero, whereas it lessened as the number of lags increased. The estimates of the autocorrelation were computed using the `acf` function of the R package **stats**. On the basis of these results, the weak dependence assumption behind the tests implemented in the **TwoSampleTest.HD** seems realistic. The four tests designed for weak dependence (spectral test, U -statistic test, block bootstrap test and permutation test) can be performed by using the following code lines:

```
> library(Equalden.HD)
> data("Hedenfalk")
> X=log(Hedenfalk[,1:7])
> Y=log(Hedenfalk[,8:15])
>
> X=X[1:1000,]
> Y=Y[1:1000,]
> library(TwoSampleTest.HD)
> res1 <- TwoSampleTest.HD(X, Y, method = "spect")
Call:
```

Usage of the function:

```
TwoSampleTest.HD(X, Y, method = c("spect",
"spect_ind", "boot", "us", "us_ind", "perm"),
I.permutation.p.values = FALSE,
b_I.permutation.p.values = c("global", "individual"))
```

X	A matrix where each row is one of the p -samples in the first group.
Y	A matrix where each row is one of the p -samples in the second group.
method	The two-sample test. By default the “us” method is computed.
I.permutation.p.values	Logical. Default is FALSE. A variable indicating whether to compute the permutation p -values or not when the selected method is not “perm”.
b_I.permutation.p.values	The bandwidth method used to compute the individual statistics on which are based the permutation p -values.

Table 2: Usage and list of the arguments of the `TwoSampleTest.HD` function.

standardized_statistic	the value of the standardized statistic.
p.value	the p -value for the test.
statistic	the value of the statistic.
variance	the value of the variance estimator.
p	number of samples or populations.
n	sample size in the first group.
m	sample size in the second group.
method	a character string indicating which two sample test is performed.
I.statistics	the p individual statistics.
I.permutation.p.values	the p individual permutation p -values.
data.name	a character string giving the name of the data.

Table 3: Summary of the results reported by `TwoSampleTest.HD` function.

```
TwoSampleTest.HD(X = X, Y = Y, method = "spect")
'global' bandwidth used by default

A two-sample test for the equality of distributions for high-dimensional data

data: c(X, Y)
standardized statistic = 11.536, p-value < 2.2e-16

> res2 <- TwoSampleTest.HD(X, Y, method = "boot")
Call:
TwoSampleTest.HD(X = X, Y = Y, method = "boot")
'global' bandwidth used by default

A two-sample test for the equality of distributions for high-dimensional data

data: c(X, Y)
standardized statistic = 11.515, p-value < 2.2e-16

> res3 <- TwoSampleTest.HD(X, Y, method = "us")
Call:
```

```

TwoSampleTest.HD(X = X, Y = Y, method = "us")
'global' bandwidth used by default

A two-sample test for the equality of distributions for high-dimensional data

data: c(X, Y)
standardized statistic = 12.104, p-value < 2.2e-16

> res4 <- TwoSampleTest.HD(X, Y, method = "perm")
Call:
TwoSampleTest.HD(X = X, Y = Y, method = "perm")
'global'bandwidth used by default

A two-sample test for the equality of distributions for high-dimensional data

data: c(X, Y)
standardized statistic = -10.955, p-value < 2.2e-16

```

The output of the function **TwoSampleTest.HD** shows that $T_p/\hat{\sigma}_S = 11.536$, $T_p/\hat{\sigma}_B = 11.515$, $T_p/\hat{\sigma}_U = 12.104$ and $T_p^{pv} = -10.955$, whereas the corresponding p -values are almost zero. The negative value of the T_p^{pv} statistic means that the average of the permutation p -values is lower than the expected mean of their (uniform) null distribution. Hence, the null hypothesis is rejected; the conclusion is that one or more genes are differently expressed depending on the tumor type. The object derived from **TwoSampleTest.HD** function is a list which saves, as usual with R functions, relevant information. Besides the standardized statistic and the p -value printed in the console when running the function (as shown above), the list of saved objects comprises the value of the statistic T_p (or $\sqrt{p}\bar{P}$ if one runs the permutation test), the variance ($\hat{\sigma}_S^2$, $\hat{\sigma}_B^2$, $\hat{\sigma}_U^2$ or $\hat{\sigma}_p^2$), the number of variables (p), the sample sizes (n and m), the method used for the data analysis, and the values of the statistics J_1, \dots, J_p . Below, we display such information for the spectral test as an illustrative example. The values of the statistics J_1, \dots, J_p are plotted in Figure 2 instead of reporting them through the console.

```

> res1$statistic
[1] 1.827471
> res1$variance
[1] 0.02509699
> res1$p
[1] 1000
> res1$n
[1] 7
> res1$m
[1] 8
> res1$method
[1] "spect"
> library(ggplot2)
>
> data=data.frame(Jk=res1$I.statistics,Genes=1:res1$p)
> ggplot(data, aes(x=Genes, y=Jk)) +
+   geom_point(shape=21, col=8) + geom_rug() + ggtitle("Individual test statistics")

```

Since the null hypothesis is rejected for Hedenfalk data, the next natural aim is to identify which genes are not equally distributed in both types of tumors. For this, we first rank the null hypotheses H_{0i} according to their contribution to the significance by using the sequence of permutation p -values. This sequence has only been computed for the permutation test, since for the remaining tests it is only computed when the argument `I.permutation.p.values` is equal to TRUE and, in our previous applications of the tests such argument has not been specified hence the default option `I.permutation.p.values=FALSE` has been used. Therefore, `res4` is the unique object which has the sequence of permutation p -values. Note that, since the argument `b_I.permutation.p.values` has not been used, the default option `b_I.permutation.p.values="global"` has been considered, and then the global \hat{b} has been employed to compute each one of the J_k , $k = 1, \dots, p$, for which the permutation p -values are calculated. Below, the code used to determine which are the 10 genes of lowest p -values is reported.

```

> pv=res4$I.permutation.p.values
> order(pv)[1:10]

```

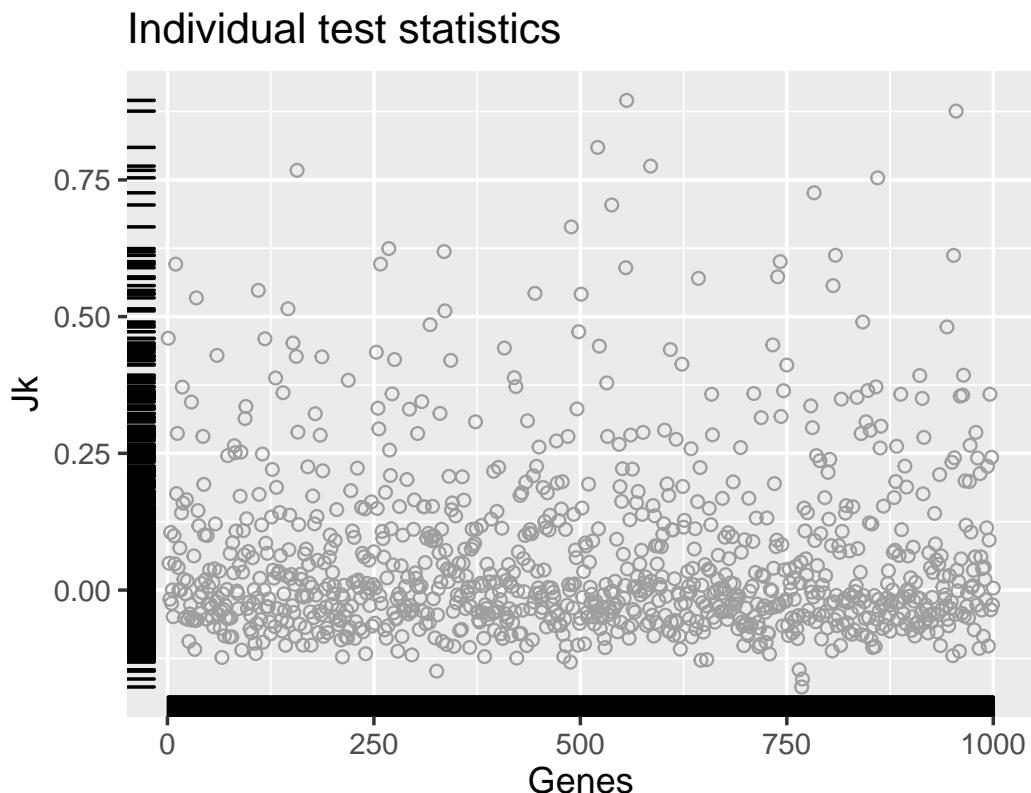


Figure 2: The values of the statistics J_1, \dots, J_p for the two-sample spectral test. Hedenfalk data.

```
[1] 556 733 952 955 445 555 914 963 118 157
```

Although the above list can be informative, any rigorous procedure should keep the type I error under control. The `p.adjust` function, available within `stats` package, implements the well-known [Benjamini and Hochberg \(1995\)](#) false discovery rate (FDR) controlling procedure. The application of this method to the sequence of 1000 permutation p -values at 5% FDR level reports 13 discoveries (see code lines below). Note that, although [Benjamini and Hochberg \(1995\)](#) has been initially studied for independent p -values, subsequent research has shown that it remains valid under weaker assumptions.

```
> alpha=0.05
> sum(p.adjust(pv,method = "BH")<=alpha)
[1] 13
```

One interesting question is whether nonparametric two-sample test statistics alternative to J_k could perform better in the multiple testing setting. As a by-product of their research, [Cousido-Rocha et al. \(2021\)](#) proved through simulations that the J_k test statistic performs similarly or even better than other well-known two-sample tests. For example, simulation results in the referred paper suggest that the Kolmogorov–Smirnov test should not be used when the sample sizes are small and the differences are other than location. For illustrative purposes, we have tested each one of the null hypothesis H_{0k} , $k \in \{1, \dots, p\}$, through Student's t test, Wilcoxon test, Levene test and Kolmogorov–Smirnov test (see [Gibbons and Chakraborti, 1992](#) and [Levene, 1960](#)). Then, [Benjamini and Hochberg \(1995\)](#) has been applied to the corresponding p -values sequence (code below).

```
> p=res1$p;n=res1$n; m=res1$m
> pv_t.test=1:p
> pv_KS=1:p
> pv_Wilcoxon=1:p
> pv_Levene=1:p
>
> library(car)
> library(exactRankTests)
> library(coin)
```

```

>
> for (i in 1:p){
+   pv_Wilcoxon[i]=wilcox.exact(X[i,],Y[i,])\$p.value
+   pv_t.test[i]=t.test(X[i,],Y[i,],var.equal = F)\$p.value
+   pv_KS[i]=ks.test(X[i,],Y[i,])\$p.value
+   pv_Levene[i]=leveneTest(c(X[i,],Y[i,])),
+   as.factor(c(rep(1,n),rep(2,m))))\$`Pr(>F)`[1]
+ }
> sum(p.adjust(pv_Wilcoxon,method = "BH")<=alpha)
[1] 13
> sum(p.adjust(pv_t.test,method = "BH")<=alpha)
[1] 1
> sum(p.adjust(pv_KS,method = "BH")<=alpha)
[1] 0
> sum(p.adjust(pv_Levene,method = "BH")<=alpha)
[1] 0

```

From results above it is seen that the Kolmogorov–Smirnov (KS) test is unable to provide any discovery at 5% of FDR. However, these results should be taken with some caution since exact KS p -values could not be computed by `ks.test` function due to the presence of ties in Hedenfalk data; note that the asymptotic distribution of the KS test may be inaccurate for small sample sizes. The lack of power of KS in the multiple testing setting has been pointed out in Cousido-Rocha et al. (2021) too. Similarly as for KS, Levene test does not declare any gene as differently expressed in the two tumor groups; this is not surprising, since differences between the two groups are mainly due to a location shift (Hedenfalk et al., 2001). The number of discoveries of the t -test is very low (only one rejection); on the contrary, Wilcoxon test provides as many discoveries as the J_k test statistic. In order to better summarize the relative power of the several testing procedures, Figure 3 depicts the number of rejections along a sequence of nominal levels for the FDR ($\alpha = 0.001, 0.002, \dots, 0.10$). Interestingly, Figure 3 supports previous comments on the poor performance of KS test.

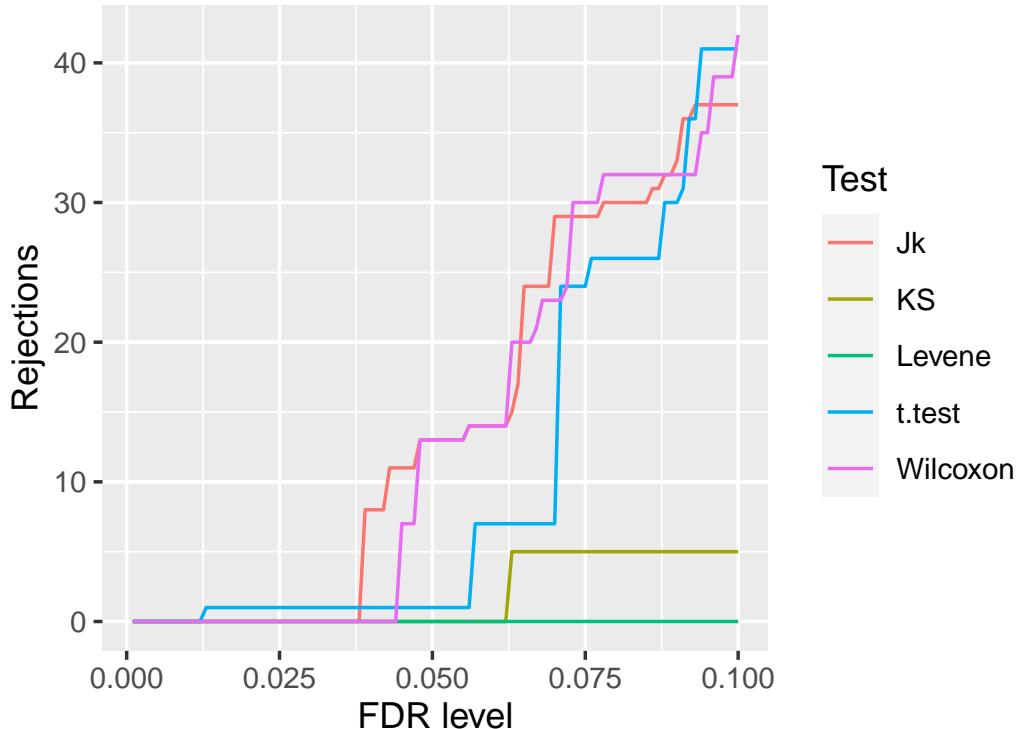


Figure 3: Number of rejections of Wilcoxon test, Kolmogorov-Smirnov test, t -test, Levene test and J_k permutation test depending on the nominal FDR level. Hedenfalk data.

3.2 Simulated data

We simulated $p = 1000$ independent variables with sample sizes $n = m = 4$ under the alternative hypothesis. More precisely, the p samples in the first group (X) were generated in 4 blocks from the following distributions, respectively: $N(0, 1)$, $N(0, 2)$, $N(1, 1)$ and $N(1, 2)$. In the second group (Y), 90% of the p samples were generated exactly as for X (true individual nulls), whereas for simulating the remaining 10% of the samples the distributions were interchanged, with a location shift as result (non-true individual nulls). To be specific, in the case $X \sim N(0, 1)$, the Y was generated from a $N(1, 1)$, and vice versa; when $X \sim N(0, 2)$, the Y was generated from a $N(1, 2)$, and vice versa. The code for the simulation is provided in Appendix 1.

Below, the two-sample tests implemented in **TwoSampleTest.HD** are applied to test the null hypothesis that the distribution of each of the samples is the same in the groups. All of the tests reject the null hypothesis. The results suggest that the simpler versions which make use of the independence assumption, "spect_ind" and "us_ind", are slightly more powerful than their counterparts for dependent data.

```
> TwoSampleTest.HD(X, Y, method = "spect")\$p.value
Call:
TwoSampleTest.HD(X = X, Y = Y, method = "spect")
'global' bandwidth used by default

A two-sample test for the equality of distributions for high-dimensional data

data: c(X, Y)
standardized statistic = 2.2275, p-value = 0.01296

[1] 0.01295652
> TwoSampleTest.HD(X, Y, method = "spect_ind")\$p.value
Call:
TwoSampleTest.HD(X = X, Y = Y, method = "spect_ind")
'global' bandwidth used by default

A two-sample test for the equality of distributions for high-dimensional data

data: c(X, Y)
standardized statistic = 2.2821, p-value = 0.01124

[1] 0.01124119
> TwoSampleTest.HD(X, Y, method = "boot")\$p.value
Call:
TwoSampleTest.HD(X = X, Y = Y, method = "boot")
'global' bandwidth used by default

A two-sample test for the equality of distributions for high-dimensional data

data: c(X, Y)
standardized statistic = 2.2643, p-value = 0.01178

[1] 0.01177765
> TwoSampleTest.HD(X, Y, method = "us")\$p.value
Call:
TwoSampleTest.HD(X = X, Y = Y, method = "us")
'global' bandwidth used by default

A two-sample test for the equality of distributions for high-dimensional data

data: c(X, Y)
standardized statistic = 2.3058, p-value = 0.01056

[1] 0.01056058
> TwoSampleTest.HD(X, Y, method = "us_ind")\$p.value
Call:
TwoSampleTest.HD(X = X, Y = Y, method = "us_ind")
'global' bandwidth used by default
```

A two-sample test for the equality of distributions for high-dimensional data

```
data: c(X, Y)
standardized statistic = 2.423, p-value = 0.007696

[1] 0.007695904
> res=TwoSampleTest.HD(X, Y, method = "perm")
Call:
TwoSampleTest.HD(X = X, Y = Y, method = "perm")
'global' bandwidth used by default
```

A two-sample test for the equality of distributions for high-dimensional data

```
data: c(X, Y)
standardized statistic = -2.2287, p-value = 0.01292
```

As done for Hedenfalk data, one can individually test for H_{0k} , $k \in \{1, \dots, p\}$, at 5% level of FDR by using the J_k statistic. In this case, the number of rejections is zero. The same occurs when using the Wilcoxon test, the Kolmogorov-Smirnov test, the t -test, or the Levene test (see code lines below). This highlights once again the need for global two-sample tests as those implemented in **TwoSampleTest.HD**.

```
> pvalues=res$I.permutation.p.values
>
> alpha=0.05
>
> sum(p.adjust(pvalues,method = "BH")<=alpha)
[1] 0
>
>
>
> pv_t.test=1:p
> pv_KS=1:p
> pv_Wilcoxon=1:p
> pv_Levene=1:p
>
> library(car)
> library(exactRankTests)
> library(coin)
>
> for (i in 1:p){
+   pv_Wilcoxon[i]=wilcox.exact(X[i,],Y[i,])\$p.value
+   pv_t.test[i]=t.test(X[i,],Y[i,],var.equal = F)\$p.value
+   pv_KS[i]=ks.test(X[i,],Y[i,])\$p.value
+   pv_Levene[i]=leveneTest(c(X[i,],Y[i,]),
+     as.factor(c(rep(1,n),rep(2,m))))\$`Pr(>F)`[1]
+ }
>
> sum(p.adjust(pv_Wilcoxon,method = "BH")<=alpha)
[1] 0
> sum(p.adjust(pv_t.test,method = "BH")<=alpha)
[1] 0
> sum(p.adjust(pv_KS,method = "BH")<=alpha)
[1] 0
> sum(p.adjust(pv_Levene,method = "BH")<=alpha)
[1] 0
```

An interesting question is the necessary computation time for **TwoSampleTest.HD**. For the simulated example, "spect" and "spect_ind" methods run in 0.16 and 0.15 seconds, respectively; "boot" method in 0.15 seconds, "us" and "us_ind" methods in 2.91 and 2.86 seconds, respectively; and, finally, the "perm" needed 4.55 seconds for running the analysis. The results shown in the current example match the general performance of the main function within the package; the spectral and block bootstrap tests are the most efficient from a computational point of view, followed by the U -statistic test and finally by the permutation test. As we increased the number of variables or (more

critically) the sample sizes, these differences in computational efficiency became more evident. On the other hand, the simplified versions for independent data did not result in a visible reduction of the run time.

4 Conclusions

Package **TwoSampleTest.HD** implements a two-sample test for the null hypothesis that all the marginal distributions of the p -variate outcome of interest coincide on the two groups. The two-sample test takes advantage of the large p , in the sense that it uses a null Gaussian distribution that holds as p goes to infinity; interestingly however, in our experience the asymptotic approximation is also correct for p as small as 20. On the other hand, the implemented test statistic is just an average of the L_2 -type deviations between the empirical characteristic functions pertaining to the two samples along the p margins. Each of these p deviations can be used to perform a local two-sample test through the preliminary computation of permutation p -values. These permutation p -values can be used to introduce an alternative testing procedure (also implemented in **TwoSampleTest.HD**), by using the asymptotic null Gaussian distribution of their average as p grows. An interesting question here is if this sequence of p -values can be used in another fashion to introduce a more powerful testing method. In principle, standard multiple comparison procedures are not competitive, since they focus (not only on the intersection null but also) on identifying the margins in which the two groups differ. However, some multiple comparison procedures have been specifically designed to test for the intersection null, and these methods could be competitive in our setting. This is an interesting open question at the time of writing.

Summarizing, **TwoSampleTest.HD** package implements for the first time omnibus two-sample tests for the high-dimensional setting under dependence. The package is user-friendly, and it is hoped that it will serve the scientific community by providing a simple and powerful tool for the analysis of high-dimensional data. Clear advice for a correct use of the package and fully illustrative examples have been given.

Acknowledgements

The authors acknowledge financial support from the Grant PID2020-118101GB-I00, Ministerio de Ciencia e Innovación.

1 Appendix: Simulated data set code

The code employed for generating the described simulated data set in Section 2.3.2 can be found below.

```
> n=m=4
> p=1000
>
> set.seed(123)
>
> p <- 1000
> n = m = 4
> inds <- sample(1:4, p, replace = TRUE)
> X <- matrix(rep(0, n * p), ncol = n)
> for (j in 1:p){
+   if (inds[j] == 1){
+     X[j, ] <- rnorm(n)
+   }
+   if (inds[j] == 2){
+     X[j, ] <- rnorm(n, sd = 2)
+   }
+   if (inds[j] == 3){
+     X[j, ] <- rnorm(n, mean = 1)
+   }
+   if (inds[j] == 4){
+     X[j, ] <- rnorm(n, mean = 1, sd = 2)
+   }
+ }
```

```

> rho <- 0.1
> ind <- sample(1:p, rho * p)
> li <- length(ind)
> inds <- inds
> for (l in 1:li){
+   if (indsy[ind[l]]==1){
+     inds[indsy[ind[l]]]=3
+   } else {
+     if (indsy[ind[l]]==2){
+       inds[indsy[ind[l]]]=4
+     } else {
+       if (indsy[ind[l]]==3){
+         inds[indsy[ind[l]]]=1
+       } else {
+         inds[indsy[ind[l]]] = 2
+       }
+     }
+   }
+ }
> Y <- matrix(rep(0, m * p), ncol = m)
> for (j in 1:p){
+   if (indsy[j] == 1){
+     Y[j, ] <- rnorm(m)}
+   if (indsy[j] == 2){
+     Y[j, ] <- rnorm(m, sd = 2)
+   }
+   if (indsy[j]==3){
+     Y[j, ] <- rnorm(m, mean = 1)
+   }
+   if (indsy[j] == 4){
+     Y[j, ] <- rnorm(m, mean = 1, sd = 2)
+   }
+ }
```

References

- Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society B*, 57:289–300, 1995. [p86]
- M. Biswas and A. K. Gosh. A nonparametric two-sample test applicable to high dimensional data. *Journal of Multivariate Analysis*, 123:160–171, 2014. [p79]
- M. Biswas, M. Mukhopadhyay, and A. K. Ghosh. A distribution-free two-sample run tests applicable to high-dimensional data. *Biometrika*, 101:913–926, 2014. [p79]
- D. Bosq. *Nonparametric Statistics for Stochastic Processes: Estimation and Prediction. Second Edition.* Springer-Verlag, New York, 1998. [p81]
- E. Carlstein. The use of subseries values for estimating the variance of a general statistic from a stationary sequence. *Annals of Statistics*, 4:1171–1179, 1996. [p81]
- M. Cousido-Rocha and J. de Uña-Álvarez. Equalden.HD: An R package for testing the equality of a high dimensional set of densities. *Computer Methods and Programs in Biomedicine*, 217:106694, 2022. doi: <https://doi.org/10.1016/j.cmpb.2022.106694>. [p83]
- M. Cousido-Rocha, J. de Uña-Álvarez, and J. Hart. A two-sample test for the equality of distributions for high-dimensional data. *Journal of Multivariate Analysis*, 174:104537, 2019. ISSN 0047-259X. doi: <https://doi.org/10.1016/j.jmva.2019.104537>. URL <https://www.sciencedirect.com/science/article/pii/S0047259X19300521>. [p79, 80, 81]
- M. Cousido-Rocha, J. de Uña-Álvarez, and S. Döhler. Multiple comparison procedures for discrete uniform and homogeneous tests. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 71: 219–243, 2021. doi: <https://doi.org/10.1111/rssc.12529>. URL <https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssc.12529>. [p86, 87]

- H. Dehling, R. Fried, I. Garcia, and M. Wendler. *Change-point Detection Under Dependence Based on Two-Sample U-Statistics*. In: Dawson D., R. Kulik, M. Ould Haye, B. Szyszkowicz, Y. Zhao (eds) *Asymptotic Laws and Methods in Stochastics*. Fields Institute Communications, vol 76. Springer, New York, NY. 2015. [p⁷⁹]
- P. Doukhan. *Mixing: Properties and Examples*. Springer-Verlag, New York, 1995. [p⁷⁹]
- S. Dudoit and M. J. van der Laan. *Multiple Testing Procedures and Applications to Genomics*. Springer, New York, 2007. [p⁸⁰]
- R. A. Fisher. *Statistical Methods for Research Workers. Fourth Edition*. Oliver and Boyd, Edinburgh, 1934. [p⁸¹]
- J. D. Gibbons and S. Chakraborti. *Nonparametric Statistical Inference. Third Edition*. Marcel Dekker, Inc, New York, 1992. [p⁸⁶]
- P. Hall and J. Jin. Properties of higher criticism under strong dependence. *The Annals of Statistics*, 36: 381–402, 2008. [p⁸⁰]
- I. Hedenfalk, D. Duggan, Y. Chen, M. Radmacher, M. Bittner, R. Simon, P. Meltzer, B. Gusterson, M. Esteller, O. Kallioniemi, B. Wilfond, A. Borg, J. Trent, M. Raffeld, Z. Yakhini, A. Ben-Dor, E. Dougherty, J. Kononen, L. Bubendorf, W. Fehrle, S. Pittaluga, G. Gruvberger, N. Loman, O. Johannsson, H. Olsesson, and G. Sauter. Gene-Expression Profiles in Hereditary Breast Cancer. *New England Journal of Medicine*, 344:539–548, 2001. [p^{82, 87}]
- H. Levene. Robust tests for equality of variances. In I. Olkin, editor, *Contributions to Probability and Statistics*, pages 278–92. Stanford University Press, Palo Alto, Calif., 1960. [p⁸⁶]
- Z. Liu, X. Xia, and W. Zhou. A test for equality of two distributions via jackknife empirical likelihood and characteristic functions. *Computational Statistics and Data Analysis*, 92:97–114, 2015. [p⁷⁹]
- P. Martínez-Camblor and J. de Uña-Álvarez. Nonparametric k-sample tests: Density functions vs distribution functions. *Computational Statistics and Data Analysis*, 53:3344–3357, 2009. [p⁸⁰]
- P. K. Mondal, M. Biswas, and A. K. Ghosh. On high dimensional two-sample tests based on nearest neighbors. *Journal of Multivariate Analysis*, 141:168–178, 2015. [p⁷⁹]
- M. H. Neumann and E. Paparoditis. On bootstrapping L_2 -type statistics in density testing. *Statistics & Probability Letters*, 50:137–147, 2000. [p⁷⁹]
- D. N. Politis and H. White. Automatic block-length selection for the dependent bootstrap. *American Economic Review*, 23:53–70, 2004. [p⁸¹]
- S. A. Stouffer, E. A. Suchman, L. C. DeVinney, S. A. Star, and R. M. Williams. *The American Soldier. Adjustment during Army Life*. Princeton University Press, England, 1949. [p⁸¹]
- S. Wei, C. Lee, L. Wichers, and J. S. Marron. Direction-projection-permutation for high-dimensional hypothesis tests. *Journal of Computational and Graphical Statistics*, 25:549–569, 2016. [p⁷⁹]
- H. Zhang, J. Jin, and Z. Wu. Distributions and power of optimal signal-detection statistics in finite case. *IEEE Transactions on Signal Processing*, 68:1021–1033, 2020. doi: <https://doi.org/10.1109/TSP.2020.2967179>. [p⁸⁰]

Marta Cousido-Rocha

Instituto Español de Oceanografía (IEO, CSIC), Centro Oceanográfico de Vigo

Subida a Radio Faro 50–52, Vigo 36390

Spain

ORCID: 0000-0002-4587-8808

marta.cousido@ieo.csic.es

Jacobo de Uña-Álvarez

CINBIO, Universidad de Vigo, SiDOR Research Group

Campus Lagoas-Marcosende, Vigo 36310

Spain

ORCID: 0000-0002-4686-8417

jacobo@uvigo.es

Statistical Models for Repeated Categorical Ratings: The R Package `rater`

by Jeffrey M. Pullin, Lyle C. Gurrin, and Damjan Vukcevic

Abstract A common problem in many disciplines is the need to assign a set of items into categories or classes with known labels. This is often done by one or more expert raters, or sometimes by an automated process. If these assignments or ‘ratings’ are difficult to make accurately, a common tactic is to repeat them by different raters, or even by the same rater multiple times on different occasions. We present an R package `rater`, available on CRAN, that implements Bayesian versions of several statistical models for analysis of repeated categorical rating data. Inference is possible for the true underlying (latent) class of each item, as well as the accuracy of each rater. The models are extensions of, and include, the Dawid–Skene model, and we implemented them using the Stan probabilistic programming language. We illustrate the use of `rater` through a few examples. We also discuss in detail the techniques of marginalisation and conditioning, which are necessary for these models but also apply more generally to other models implemented in Stan.

1 Introduction

The practice of measuring phenomena by having one or more raters assign to items one of a set of ratings is common across many fields. For example, in medicine one or more doctors may classify a diagnostic image as being evidence for one of several diagnoses, such as types of cancer. This process of rating is often done by one or more expert raters, but may also be performed by a large group of non-experts (e.g., in natural language processing (NLP), where a large number of crowd-sourced raters are used to classify pieces of text; see Passonneau and Carpenter (2014) and Ipeirotis, Provost, and Wang (2010)) or by a machine or other automated process (e.g., a laboratory diagnostic test, where ‘test’ may refer to either the raters and/or the ratings). Other fields where ratings occur include astronomy, for classifying astronomical images (Smyth et al. 1994), and bioinformatics, for inferring error rates for some bioinformatics algorithms (Jakobsdottir and Weeks 2007). Indeed, both studies apply the Dawid–Skene model that we present below and implement in our software package.

The assignment of items to categories, which are variously referred to as gradings, annotations or labelled data, we will call *ratings*. Our hope is that these are an accurate reflection of the true characteristics of the items being rated. However, this is not guaranteed; all raters and rating systems are fallible. We would expect some disagreement between the raters, especially from non-experts, and even expert raters may disagree when there are some items that are more difficult to rate than others. A typical way of dealing with this problem is to obtain multiple ratings for each item from different raters, some of whom might rate the item more than once on separate occasions. By using an aggregate index (e.g., by averaging the ratings) one can hope to reduce bias and variance when estimating both population frequencies and individual item categories.

Despite the fact that ratings data of this type are common, there are few software packages that can be used to fit useful statistical models to them. To address this, we introduce `rater`, a software package for R (R Core Team 2021) designed specifically for such data and available on CRAN. Our package provides the ability to fit the Dawid–Skene model (Dawid and Skene 1979) and several of its extensions. The goal is to estimate accurately the underlying true class of each item (to the extent that this is meaningful and well-defined) as well as to quantify and characterise the accuracy of each rater. Models of these data should account for the possibility that accuracy differs between the raters and that the errors they make might be class-specific. The package accepts data in multiple formats and provides functions for extracting and visualizing parameter estimates.

While our package implements optimisation-based inference, the default mode of inference is Bayesian, using Markov Chain Monte Carlo (MCMC). We chose a Bayesian approach for several reasons. The first is because the standard classical approach, using an EM algorithm, often gives parameter estimates on the boundary of the parameter space (i.e., probabilities of exactly 0 or 1), which would typically be implausible. Examples of this can be seen in the original paper by Dawid and Skene (1979), and also in our own results below that compare MCMC and optimisation-based inference (e.g., compare Figure 2 and Figure 4). A second reason is to take advantage of the convenient and powerful Stan software (Stan Development Team 2021), which made implementing our package considerably easier. Stan is a probabilistic programming language that implements MCMC and optimisation algorithms for fitting Bayesian models. The primary inference algorithm used by Stan is the No U-Turn Sampler (NUTS) variant of the Hamiltonian Monte Carlo (HMC) sampler, which uses gradient information to guide exploration of the posterior distribution. A third reason is that

Table 1: Formats for balanced rating data.

(a) Long		
Item	Rater	Rating
1	1	3
1	2	4
2	1	2
2	2	2
3	1	2
3	2	2

(b) Wide		
Item	Rater 1	Rater 2
1	1	3
2	2	2
3	3	2

(c) Wide (grouped)		
Rater 1	Rater 2	Tally
3	4	1
2	2	2

Table 2: Formats for unbalanced rating data. Missing values are indicated by 'NA'.

(a) Long		
Item	Rater	Rating
1	1	3
1	2	4
2	1	2
2	2	2
3	1	2
3	2	2
4	1	3
5	1	3
6	2	4

(b) Wide		
Item	Rater 1	Rater 2
1	1	3
2	2	2
3	3	2
4	4	3
5	5	3
6	6	NA

(c) Wide (grouped)		
Rater 1	Rater 2	Tally
3	4	1
2	2	2
NA	NA	2
NA	4	1

it allows users to easily incorporate any prior information when it is available. In this context, prior information might be information about the quality of the raters or specific ratings mistakes that are more common.

We illustrate our methods and software using two examples where the raters are healthcare practitioners: one with anaesthetists rating the health of patients, and another with dentists assessing the health of teeth. In both examples, we are interested in obtaining more accurate health assessments by combining all of the experts' ratings rather than using the ratings of a single expert, as well as estimating the accuracy of each expert and characterising the types of errors they make.

Our paper is organised as follows. Section 2 describes the various data formats that can be used with the function implemented in the package. Section 4 introduces the Dawid–Skene model and several of its extensions. Section 5 briefly describes the implementation and goes into more detail about some of the more interesting aspects, followed by an exposition of the technique called Rao–Blackwellization in Section 6, which is connected with many aspects of the implementation. Section 7 introduces the user interface of `rater` along with some examples. Section 8 concludes with a discussion of the potential uses of the model.

2 Data formats

There are several different formats in which categorical rating data can be expressed. We make a distinction between *balanced* and *unbalanced* data. Balanced data have each item rated at least once by each rater. Unbalanced data contain information on some items that were not rated by every rater; stated alternatively, there is at least one rater who does not rate every item.

Table 1 and Table 2 illustrate three different formats for categorical rating data. Format (a) has one row per item-rater rating episode. It requires variables (data fields) to identify the item, the rater and the rating. This presentation of data is often referred to as the 'long' format. It is possible for a rater to rate a given item multiple times; each such rating would have its own row in this format. Format (b) has one row per item, and one column per rater, with each cell being a single rating. Format (c) aggregates rows (items) that have identical rating patterns and records a frequency tally. Formats (b) and (c) are examples of 'wide' formats. They only make sense for data where each rater rates each item at most once. Note that the long format will never require any structural missing values for any of the variables, whereas the wide formats require the use of a missing value indicator unless they represent data generated by a balanced design, see Table 2. One benefit of the grouped format is that it allows a more computationally efficient implementation of the likelihood function. See Section 5 for details about how this is implemented in `rater`.

3 Existing approaches

A typical approach for modelling categorical ratings is to posit an unobserved underlying true class for each item, i.e., we represent the true class as a latent variable. One of the first such models was developed by Dawid and Skene (1979). This model is still studied and used to this day, and has served as a foundation for several extensions (e.g., Paun et al. 2018). Our R package, `rater`, is the first one specifically designed to provide inference for the Dawid–Skene model and its variants.

The Python library `pyanno` (Berkes et al. 2011) also implements a Bayesian version of the Dawid–Skene model in addition to the models described by Rzhetsky (2009). However, unlike `rater`, `pyanno` only supports parameter estimation via optimisation rather than MCMC; i.e., it will only compute posterior modes rather than provide samples from the full posterior distribution. In addition, `pyanno` does not support variants of the Dawid–Skene model nor the support for grouped data implemented in `rater`.

More broadly, many different so-called ‘latent class’ models have been developed and implemented in software, for a wide diversity of applications (e.g., Goodman 1974). A key aspect of the categorical ratings context is that we assume our categories are known ahead of time, i.e., the possible values for the latent classes are fixed. The Dawid–Skene model has this constraint built-in, whereas other latent class models are better tailored to other scenarios. Nevertheless, many of these models are closely related and are implemented as R packages.

Of most interest for categorical ratings analysis are the packages `poLCA` (Linzer and Lewis 2011), `BayesLCA` (White and Murphy 2014) and `randomLCA` (Beath 2017), all of which are capable of fitting limited versions of the Dawid–Skene model. We explore the relationship between different latent class models in more detail in Section 4.7. Briefly: `randomLCA` can fit the Dawid–Skene model only when the data uses binary categories, `BayesLCA` can only fit the homogeneous Dawid–Skene model, and `poLCA` can fit the Dawid–Skene model with an arbitrary number of categories but only supports wide data (where raters do not make repeated ratings on items). Neither `poLCA` nor `randomLCA` support fully Bayesian inference with MCMC, and none of the packages support fitting variants of the Dawid–Skene model (which are available in `rater`).

4 Models

One of the first statistical models proposed for categorical rating data was that of Dawid and Skene (1979). We describe this model below, extended to include prior distributions to allow for Bayesian inference. Recently, a number of direct extensions to the Dawid–Skene model have been proposed. We describe several of these below, most of which are implemented in `rater`. For ease of exposition, unless otherwise stated, all notation in this section will assume we are working with balanced data and where each item is rated exactly once by each rater, one exception is that we present the Dawid–Skene model for arbitrary data at the end of Section 4.1.

Assume we have I items (for example, images, people, etc.) and J raters, with each item presumed to belong to one of the K categories: we refer to this as its ‘true’ category and also as its latent class (since it is unobserved). Let $y_{i,j} \in \{1, \dots, K\}$ be the rating for item i given by rater j .

4.1 Dawid–Skene model

The model has two sets of parameters:

- π_k : the prevalence of category k in the population from which the items are sampled. $\pi_k \in (0, 1)$ for $k \in \{1, \dots, K\}$, with $\sum_{k=1}^K \pi_k = 1$. All classes have some non-zero probability of occurring. We collect all the π_k parameters together into a vector, $\pi = (\pi_1, \dots, \pi_K)$.
- $\theta_{j,k,k'}$: the probability that rater j responds with class k' when rating an item of true class k . Here, $j \in \{1, \dots, J\}$ and $k, k' \in \{1, 2, \dots, K\}$. We will refer to the $K \times K$ matrix $\theta_{j,k,k'}$ for a given j as the *error matrix* for rater j . We represent the k th row of this matrix as the vector $\theta_{j,k} = (\theta_{j,k,1}, \theta_{j,k,2}, \dots, \theta_{j,k,K})$: this shows how rater j responds to an item with true class k , by rating it as being in class $1, 2, \dots, K$, according to the respective probabilities.

The model also has the following set of latent variables:

- z_i : the true class of item i . $z_i \in \{1, \dots, K\}$ for $i \in \{1, \dots, I\}$.

Under the Bayesian perspective, the model parameters and latent variables are both unobserved random variables that define the joint probability distribution of the model. As such, inference on

them is conducted in the same way. However, we make the distinction here to help clarify aspects of our implementation later on.

The model is defined by the following distributional assumptions:

$$\begin{aligned} z_i &\sim \text{Categorical}(\pi), \quad \forall i \in \{1, \dots, I\}, \\ y_{i,j} \mid z_i &\sim \text{Categorical}(\theta_{j,z_i}), \quad \forall i \in \{1, \dots, I\}, j \in \{1, \dots, J\}. \end{aligned}$$

In words, the rating for item i given by rater j will follow the distribution specified by: the error matrix for rater j , but taking only the row of that matrix that corresponds to the **value** of the latent variable for item i (the z_i th row).

We now have a fully specified model, which allows use of likelihood-based methods. The likelihood function for the observed data (the $y_{i,j}$ s) is

$$\Pr(y \mid \theta, \pi) = \prod_{i=1}^I \left(\sum_{k=1}^K \left(\pi_k \cdot \prod_{j=1}^J \theta_{j,k,y_{i,j}} \right) \right).$$

Note that the unobserved latent variables, $z = (z_1, z_2, \dots, z_I)$, do not appear because they are integrated out (via the sum over the categories k). Often it is useful to work with the complete data likelihood where the z have specific values, for example if implementing an EM algorithm (such as described by Dawid and Skene (1979)). The somewhat simpler likelihood function in that case is

$$\Pr(y, z \mid \theta, \pi) = \prod_{i=1}^I \left(\pi_{z_i} \cdot \prod_{j=1}^J \theta_{j,z_i,y_{i,j}} \right).$$

In our implementation we use the first version of the likelihood, which marginalises over z (see [Section 5.1](#)). We do this to avoid needing to sample from the posterior distribution of z using Stan, as the HMC sampling algorithm used by Stan requires gradients with respect to all parameters and gradients cannot be calculated for discrete parameters such as z . Alternative Bayesian implementations, perhaps using Gibbs sampling approaches, could work with the second version of the likelihood and sample values of z directly.

To allow for Bayesian inference, we place weakly informative prior probability distributions on the parameters:

$$\begin{aligned} \pi &\sim \text{Dirichlet}(\alpha), \\ \theta_{j,k} &\sim \text{Dirichlet}(\beta_k). \end{aligned}$$

The hyper-parameters defining these prior probability distributions are:

- α : a vector of length K with all elements greater than 0
- β : a $K \times K$ matrix with all elements greater than 0, with β_k referring to the k th row of this matrix.

We use the following default values for these hyper-parameters in `rater`: $\alpha_k = 3$ for $k \in \{1, \dots, K\}$, and

$$\beta_{k,k'} = \begin{cases} Np & \text{if } k = k' \\ \frac{N(1-p)}{K-1} & \text{otherwise} \end{cases} \quad \forall k, k' \in \{1, \dots, K\}.$$

Here, N corresponds to an approximate pseudocount of hypothetical prior observations and p an approximate probability of a correct rating (applied uniformly across all raters and categories). This affords us the flexibility to centre the prior on a particular assumed accuracy (via p) as well as tune how informative the prior will be (via N). In `rater` we set the default values to be $N = 8$ and $p = 0.6$, reflecting a weakly held belief that the raters are have a better than coin-flip chance of choosing the correct class. This should be suitable for many datasets, where the number of categories is small, for example, less than ten. This default would, however, be optimistic if the number of categories is very large, for example, one hundred. In that case it would make sense for the user to specify a more realistic prior (see [Section 7.5](#)). A derivation of the default prior specification is shown in [Section 4.2](#).

We can also write the Dawid–Skene model in notation that does not assume the data are balanced. Let I, J, K be as above. Let N be the total number of ratings in the dataset (previously we had the special case $N = I \cdot J$). Define the following quantities relating to the n th rating: it was performed by rater j_n , who rated item i_n , and the rating itself is y_n . We can now write the Dawid–Skene model as:

$$\begin{aligned} z_i &\sim \text{Categorical}(\pi), \quad \forall i \in \{1, \dots, I\}, \\ y_n \mid z_{i_n} &\sim \text{Categorical}(\theta_{j_n, z_{i_n}}), \quad \forall n \in \{1, \dots, N\}. \end{aligned}$$

4.2 Hyper-parameters for the Dawid–Skene model

Hyper-parameters for the error matrices

The values we proposed for the β hyper-parameters are designed to be flexible and have an approximate intuitive interpretation. The derivation is as follows.

First, consider the variant of the model where the true latent class (z) of each item is known; this model is described under ‘Case 1. True responses are available’ by Dawid and Skene (1979). Under this model, we can ignore π because the distribution of y depends only on θ . It is a categorical distribution with parameters given by specific rows of the error matrices (as determined by z , which is known) rather than being a sum over all possible rows (according to the latent class distribution).

Under this model, the Dirichlet prior is conjugate; we obtain a Dirichlet posterior for each row of each rater’s error matrix. Consider an arbitrary rater j and an arbitrary latent class k' . Let $c = (c_1, c_2, \dots, c_K)$ be the number of times this rater rates an item as being in each of the K categories when it is of true class k' . Let $n_{jk'} = \sum_{k=1}^K c_k$ be the total number of such ratings. Also, recall that $\theta_{j,k'}$ refers to the vector of probabilities from the k' th row of the error matrix for rater j . We have that

$$c \sim \text{Multinomial}\left(n_{jk'}, \theta_{j,k'}\right),$$

giving the posterior

$$\theta_{j,k'} \mid c \sim \text{Dirichlet}(\beta_{k'} + c).$$

Under this model, the hyper-parameter vector $\beta_{k'}$ has the same influence on the posterior as the vector of counts c . We can therefore interpret the hyper-parameters as ‘pseudocounts’. This gives a convenient way to understand and set the values of the hyper-parameters. Our choices were as follows:

- We constrain their sum to be a given value $N = \sum_{k=1}^K \beta_{k',k}$, which we interpret as a pseudo-sample size (of hypothetical ratings from rater j of items with true class k').
- We then set $\beta_{k',k}$ to a desired value so that it reflects a specific assumed accuracy. In particular, the prior mean of $\theta_{k',k}$, the probability of a correct rating, will be $\beta_{k',k}/N$. Let this be equal to p , an assumed average prior accuracy.
- Finally, in the absence of any information about the different categories, it is reasonable to treat them as exchangeable and thus assume that errors in the ratings are on average uniform across categories. This implies that the values of all of the other hyper-parameters ($\beta_{k',k}$ where $k \neq k'$) are equal and need to be set to a value to meet the constraint that the sum of the hyper-parameters in the vector $\beta_{k'}$ is N .
- These choices imply the hyper-parameter values described in Section 4.1.

In the Dawid–Skene model, the latent classes (z) are of course not known. Therefore, we do not have a direct interpretation of the β hyper-parameters as pseudocounts. However, we can treat them approximately as such.

We chose $N = 8$ and $p = 0.6$ as default values based on obtaining reasonably good performance in simulations across a diverse range of scenarios (data not shown).

Relationship to previous work

The *Stan User’s Guide* (Section 7.4) (Stan Development Team 2021) suggests the following choice of values for β :

$$\beta_{k',k} = \begin{cases} 2.5 \times K & \text{if } k' = k \\ 1 & \text{otherwise} \end{cases}, \quad \forall k', k \in \{1, \dots, K\}.$$

It is interesting to compare this to our suggested values, above. By equating the two, we can re-write the Stan choice in terms of p and N :

$$p = 2.5 \times K/N$$

$$N = 3.5 \times K - 1.$$

We see that the Stan default is to use an increasingly informative prior (N gets larger) as the number of categories (K) increases. The assumed average prior accuracy also varies based on K . For example:

- For $K = 2$ categories, $p = 0.83$ and $N = 6$.
- For $K = 5$ categories, $p = 0.76$ and $N = 16.5$.
- As K increases, p slowly reduces, with a limiting value of $2.5/3.5 = 0.71$.

Our default values of $p = 0.6$ and $N = 8$ give a prior that is less informative and less optimistic about the accuracy of the raters.

In practice we have experienced issues when fitting models via optimisation when the non-diagonal entries of β are less than 1 (i.e., when $\beta_{k',k} < 1$ for some $k' \neq k$). The default hyper-parameters selected in `rater` ensure that when there are only a few possible categories—specifically, when $K \leq 4$ —then this does not occur. When K is larger than this and the default hyper-parameters give rise to non-diagonal entries smaller than 1, `rater` will report a useful warning message. This only arises when using optimisation; it does not arise when using MCMC. Users who wish to use optimisation with a large number of categories can specify different hyper-parameters to avoid the warning message.

Hyper-parameters for the prevalence

The α hyper-parameters define the prior for the prevalence parameters π . All of these are essentially nuisance parameters and not of direct interest, and would typically have little influence on the inference for θ or z . The inclusion in the model of a large amount of prior information on the values of the prevalence parameters would clearly influence the posterior distributions of the corresponding population class frequencies. We would, however, expect inferences about the accuracy of raters (whether this is assumed to be class-specific or not) to depend largely on the number of ratings captured by the dataset, not the prevalence parameters themselves. We have, therefore, not explored varying α and have simply set them to the same values as suggested in the *Stan User’s Guide* (Stan Development Team 2021): $\alpha_k = 3$ for all k .

4.3 Hierarchical Dawid–Skene model

Paun et al. (2018) introduce a ‘Hierarchical Dawid–Skene’ model that extends the original one by replacing the Dirichlet prior on $\theta_{j,k}$ (and the associated hyper-parameter β) with a hierarchical specification that allows for partial pooling, which allows the sharing of information about raters’ performance across rater-specific parameters. It requires two $K \times K$ matrices of parameters, μ and σ , with

$$\begin{aligned}\mu_{k,k'} &\sim \begin{cases} \text{Normal}(2, 1), & k = k' \\ \text{Normal}(0, 1), & k \neq k' \end{cases} \quad \forall k, k' \in \{1, \dots, K\} \\ \sigma_{k,k'} &\sim \text{Half-Normal}(0, 1) \quad \forall k, k' \in \{1, \dots, K\}.\end{aligned}$$

We then have that:

$$\gamma_{j,k,k'} \sim \text{Normal}(\mu_{k,k'}, \sigma_{k,k'}) \quad \forall j \in \{1, \dots, J\}, k, k' \in \{1, \dots, K\}$$

which are then normalised into probabilities via the multinomial logit link function (also known as the *softmax* function) in order to define the elements of the error matrices, which are:

$$\theta_{j,k,k'} = \frac{e^{\gamma_{j,k,k'}}}{\sum_{k'=1}^K e^{\gamma_{j,k,k'}}}.$$

Other details, such as the distribution of z and y , are defined in the same way as per the Dawid–Skene model. The implementation in `rater` differs from the implementation from Paun et al. (2018) by modifying the prior parameters to encode the assumption that the raters are generally accurate. Specifically, the higher means of the diagonal elements of μ encode that the raters are accurate, as after transformation by the softmax function larger values of μ will produce higher probabilities in θ . The assumption that the raters are accurate is also encoded in the prior distribution for θ in the Dawid–Skene model described above.

One interpretation of the hierarchical model is as a ‘partial pooling’ compromise between the full Dawid–Skene model and the model with only one rater (see Section 4.5). This is depicted in Figure 1. Another interpretation of the model is that it treats the accuracy of raters given a specific latent class as a random effect, not a fixed effect as in the Dawid–Skene model. Paun et al. (2018) show, via a series of simulations and examples, that this hierarchical model generally improves the quality of predictions over the original Dawid–Skene model.

4.4 Class-conditional Dawid–Skene

Another variant of the Dawid–Skene model imposes constraints on the entries in the error matrices of the raters. Given the presumption of accurate raters, one natural way to do this is to force all non-diagonal entries in a given row of the error matrix to take a constant value that is smaller than the

corresponding diagonal entry. Formally the model is the same as the Dawid–Skene model except that we have:

$$\theta_{j,k,k'} = \begin{cases} p_{j,k} & \text{if } k = k' \\ \frac{1-p_{j,k}}{(K-1)} & \text{otherwise.} \end{cases}$$

To make the model fully Bayesian we use the following prior probability distributions:

$$p_{j,k} \sim \text{Beta}(\beta_{1,k}, \beta_{2,k}), \quad \forall j \in 1, \dots, J.$$

In [rater](#) we set $\beta_{1,k} = Np$ and $\beta_{2,k} = N(1-p)$ for all k , where $N = 8$ and $p = 0.6$ as in [Section 4.1](#). While the constraints in this model may lead to imprecise or biased estimation of parameters for any raters that are substantially inaccurate or even antagonistic, the reduced number of parameters of this model relative to the full Dawid–Skene model make it much easier to fit. This model was referred to as the ‘class-conditional Dawid–Skene model’ by Li and Yu (2014).

4.5 Homogeneous Dawid–Skene model

Another related model is the ‘multinomial model’ described by Paun et al. (2018), where the full Dawid–Skene model is constrained so that the accuracy of all raters is assumed to be identical. The constraint can be equivalently formulated as the assumption that all of the ratings were done by a single rater, thus the raters and the sets of rating they produce are exchangeable. This model can be fitted using [rater](#) by simply modifying the data so that only one rater rates every item present and then fitting the usual Dawid–Skene model.

4.6 Relationships between the models

All of the models presented in this paper can be directly related to the original Dawid–Skene model. [Figure 1](#) shows the relationships between the models implemented in the package, which are coloured blue. The hierarchical model can be seen as a ‘partial pooling’ model where information about the performance of the raters is shared. The Dawid–Skene model can then be seen as the ‘no pooling’ extreme where all raters’ error matrices are estimated independently, while the homogeneous model is the other extreme of complete pooling where one error matrix is fitted for all of the raters. In addition, the class-conditional model is a direct restriction of the Dawid–Skene model where extra constraints are placed on the error matrices.

4.7 Relationships to existing models

The Dawid–Skene model is also closely related to existing latent class models. Usually, in latent class models, the number of latent classes can be chosen to maximise a suitable model selection criterion, such as the BIC, or to make the interpretation of the latent classes easier. In the Dawid–Skene model, however, the number of latent classes must be fixed to the number of categories to ensure that the error matrices can be interpreted in terms of the ability of the raters to rate the categories that appear in the data. Some specific models to which the Dawid–Skene model is related are depicted in [Figure 1](#) and are coloured white.

The latent class model implemented in [BayesLCA](#) is equivalent to the Dawid–Skene model fitted to binary data if the number of classes (G in the notation of White and Murphy (2014)) is set to 2. In that case, each of the M dimensions of the binary response variable can be interpreted as a rater, so that $J = M$.

The Dawid–Skene model is also a special case of the model implemented in [poLCA](#) (Linzer and Lewis 2011). The models are equivalent if they have the same number of latent classes ($R = K$, where R is the number of latent classes in the [poLCA](#) model) and if all of the J categorical variables in the [poLCA](#) model take values in the set $\{1, \dots, K\}$ (that is, $K_j = K$ for all $j \in J$), so that each of the J variables can be interpreted as the ratings of a particular rater.

In addition, the Dawid–Skene model is a special case of the model implemented in [randomLCA](#) (Beath 2017). When no random effects are specified and $K = 2$, the model is equivalent to the Dawid–Skene model with two classes. While [randomLCA](#) allows selecting an arbitrary number of latent classes, it only supports binary data.

Finally, when $K = 2$ the Dawid–Skene model reduces to the so-called ‘traditional’ or ‘standard’ latent class model as first described by Hui and Walter (1980). See Asselineau et al. (2018) for a more modern description.

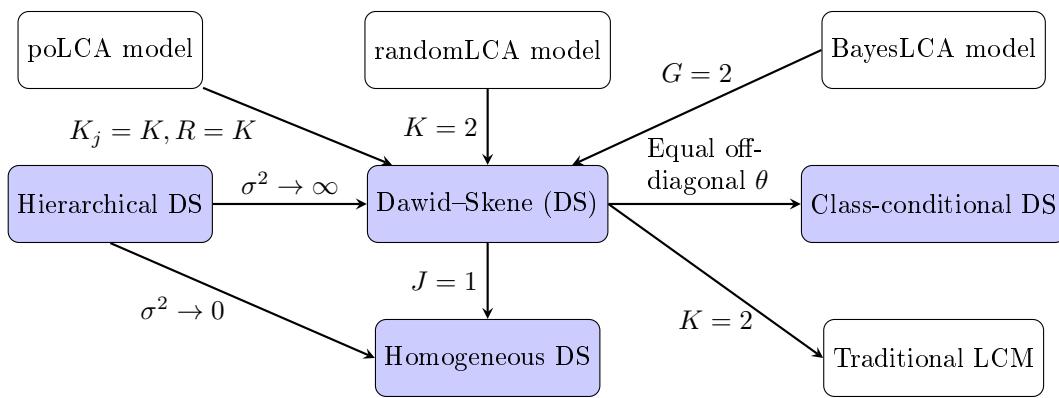


Figure 1: Relationships between models. Models coloured blue are implemented in [rater](https://CRAN.R-project.org/package=rater). DS: Dawid–Skene. LCM: latent class model.

Table 3: Features of **rater** and existing R packages for fitting latent class models. DS: Dawid–Skene.

Package	DS model	Fitting method	Response type	Repeated ratings	Data formats
rater	Yes	Bayesian	Polytomous	Yes	Wide, grouped, long
BayesLCA	When $K = 2$	Bayesian	Binary	No	Wide, grouped
randomLCA	When $K = 2$	Frequentist	Binary	No	Wide, grouped
poLCA	Yes	Frequentist	Polytomous	No	Wide

4.8 Relationships to existing packages

These relationships to existing latent class models means the functionality implemented in **rater** is similar to that of existing R packages for fitting these models. The features of **rater** and existing packages are summarised in Table 3.

Some further details about these relationships:

- The two Bayesian packages implement different methods of inference. **rater** uses Hamiltonian Monte Carlo and optimisation of the log-posterior. **BayesLCA** implements Gibbs sampling, the expectation–maximization (EM) algorithm and a Variational Bayesian approach.
- Unlike the other packages, the implementation of the wide data format in **randomLCA** supports missing data. This feature is not implemented in **rater** because the long data format allows both missing data and repeated ratings.

Overall, the main features that distinguish **rater** from the other packages are its full support for the Dawid–Skene model and extensions, and its support for repeated ratings.

5 Implementation details

rater uses Stan (Carpenter et al. 2017) to fit the above models to data. It therefore supports both optimisation and Markov chain Monte Carlo (MCMC) sampling, in particular using the No U-Turn Sampler (NUTS) algorithm (Hoffman and Gelman 2014). For most datasets we recommend using NUTS, due to the fact that it will generate realisations of the full posterior distribution. Optimisation may be useful, however, for particularly large datasets, especially if K is large, or if there is a need to fit a large number of models.

5.1 Marginalisation

The NUTS algorithm relies on computing derivatives of the (log) posterior distribution with respect to all parameters. It cannot, therefore, be used when the posterior contains discrete parameters, such as the latent class in the Dawid–Skene model. To overcome this difficulty the Stan models implemented in **rater** use marginalised forms of the posterior to allow NUTS to be used. In other words, for the Dawid–Skene model we implement the likelihood $\Pr(y | \theta, \pi)$, as described earlier in Section 4.1, together with priors for θ and π .

To avoid any confusion, we stress the fact that our choice to marginalise over the vector z is purely because it is discrete and we want to use the NUTS algorithm. It is not due to the fact that the components of the vector z are inherently latent variables. Alternative Bayesian implementations could avoid marginalisation and sample the z_i 's directly (although this may be less efficient, see [Section 6.3](#) for discussion). Similarly, if the components of z were all continuous, we would be able to sample them using NUTS. If one or more of the other non-latent parameters were discrete, then we would need to marginalise over them too.

5.2 Inference for the true class via conditioning

Marginalising over z allows us to use NUTS and sample efficiently from the posterior probability distribution of θ and π . We are still able to obtain the posterior distribution of the components z_i of z as follows. For each posterior draw of θ and π , we calculate the conditional posterior for z_i , $\Pr(z_i | \theta^*, \pi^*, y)$, where the conditioning is on the drawn values, θ^* and π^* . This can be done using Bayes' theorem, for example in the Dawid–Skene model,

$$\begin{aligned} \Pr(z_i = k | \theta^*, \pi^*, y) &= \frac{\Pr(y | z_i = k, \theta^*, \pi^*) \Pr(z_i = k | \theta^*, \pi^*)}{\Pr(y | \theta^*, \pi^*)} \\ &= \frac{\pi_k^* \prod_{j=1}^J \theta_{j,k,y_{ij}}^*}{\sum_{m=1}^K \pi_m^* \prod_{j=1}^J \theta_{j,m,y_{ij}}^*}. \end{aligned}$$

To get a Monte Carlo estimate of the marginal posterior, $\Pr(z_i | y)$, we take the mean of the above conditional probabilities across the posteriors draws of θ and π . To formally justify this step we can use the same argument that justifies the general MCMC technique of Rao–Blackwellization, see [Section 6](#). In practice, this calculation is straightforward with Stan because we can write the log posterior in terms of the (log scale) unnormalised versions of the conditional probabilities and output them alongside the posterior draws. Then it is simply a matter of renormalising them, followed by taking the mean across draws.

5.3 Data summarisation

[rater](#) implements the ability to use different forms of the likelihood depending on the format of the data that is available. For example, for a balanced design the full likelihood for the Dawid–Skene model is

$$\Pr(y | \theta, \pi) = \prod_{i=1}^I \left(\sum_{k=1}^K \left(\pi_k \cdot \prod_{j=1}^J \theta_{j,k,y_{ij}} \right) \right).$$

Looking closely, we see that the contribution of any two items with the same pattern of ratings, that is, the same raters giving the same ratings, is identical. We can therefore rewrite it as a product over patterns rather than items,

$$\Pr(y | \theta, \pi) = \prod_{l=1}^L \left(\sum_{k=1}^K \left(\pi_k \cdot \prod_{j=1}^J \theta_{j,k,y_{lj}} \right) \right)^{n_l},$$

where L is the number of distinct rating ‘patterns’, n_l is the number of times pattern l occurs and y_{lj} denotes the rating given by rater j within pattern l . This corresponds directly to having the data in the ‘grouped’ format, see [Section 2](#).

This rewriting is useful for balanced designs where there are many ratings but few distinct patterns, as it removes the need to iterate over every rating during computation of the posterior, reducing the time for model fitting. An example of this type of data are the caries data presented in [Section 7](#).

This technique is not always helpful. For example, if the data has many missing entries, the number of distinct patterns L may be similar or equal to than the total number of ratings. Consequently, rewriting to use the grouped format may not save much computation. For this reason, the implementation of [rater](#) does not allow missing values in grouped data. Note that in the long data format we simply drop rows that correspond to missing data, a result of which is that missing data is never explicitly represented in [rater](#).

Both the [randomLCA](#) and [BayesLCA](#) packages support grouped format data input. Of these, CRANpkg{randomLCA} supports missing values in grouped format data.

6 Marginalisation and Rao–Blackwellization

The technique of marginalisation for sampling discrete random variables discussed in Section 5 is well known in the Stan community. We feel, however, that the interpretation and use of the relevant conditional probabilities and expectations has not been clearly described before, so we attempt to do so here. We begin with a brief review of the theory of marginalisation, which is a special case of a more general technique often referred to as ‘Rao–Blackwellization’. See Owen (2013) for an introduction and Robert and Roberts (2021) for a recent survey of the use of Rao–Blackwellization in MCMC.

6.1 Connection with the Rao–Blackwell theorem

Suppose we are interested in estimating $\mu = \mathbb{E}(f(X, Y))$ for some function f of random variables X and Y . An example of such an expectation is $\mu = \mathbb{E}(Y)$ where we take $f(x, y) = y$. If we can sample from the joint distribution of X and Y , $p_{X,Y}(x, y)$, the obvious Monte Carlo estimator of this expectation is

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n f(x_i, y_i)$$

where (x_i, y_i) are samples from the joint distribution. Now let $h(x) = \mathbb{E}(f(X, Y) \mid X = x)$. An alternate conditional estimator—the so-called *Rao–Blackwellized estimator*—of μ is

$$\hat{\mu}_{\text{cond}} = \frac{1}{n} \sum_{i=1}^n h(x_i)$$

where the x_i are sampled from the distribution of X . This estimator is justified by the fact that,

$$\mathbb{E}(h(X)) = \mathbb{E}(\mathbb{E}(f(X, Y) \mid X)) = \mathbb{E}(f(X, Y)) = \mu$$

For this to be effective we need to be able to efficiently compute (ideally, analytically) the conditional expectation $h(X)$. The original motivation for using this technique was to reduce the variance of Monte Carlo estimators. This follows from noting that

$$\begin{aligned} \text{var}(f(X, Y)) &= \mathbb{E}(\text{var}(f(X, Y) \mid X)) + \text{var}(\mathbb{E}(f(X, Y) \mid X)) \\ &= \mathbb{E}(\text{var}(f(X, Y) \mid X)) + \text{var}(h(X)), \end{aligned}$$

and since the first term on the right-hand side is non-negative, we have $(f(X, Y)) \geq (h(X))$.

From this result it follows that, if the draws x_i are independent samples, then

$$\text{var}(\hat{\mu}_{\text{cond}}) < \text{var}(\hat{\mu})$$

for all functions $f(X, Y)$. In the applications we consider here, however, x_i will be drawn using MCMC and therefore draws will not be independent. In this case it is possible to construct models and functions $f(\cdot)$ for which conditioning can increase the variance of the estimator (Geyer 1995).

The name ‘Rao–Blackwellization’ arose due to the similarity of the above argument to that used in the proof of the Rao–Blackwell theorem (Blackwell 1947), which states that given an estimator $\hat{\mu}$ of a parameter μ , the conditional expectation of $\hat{\mu}$ given a sufficient statistic for μ is potentially more efficient and certainly no less efficient than $\hat{\mu}$ as an estimator of μ . The Rao–Blackwellized estimators presented above do not, however, make any use of sufficiency and do not have the same optimality guarantees that the Rao–Blackwell theorem provides, making the name less than apt. Following Geyer (1995), we prefer to think of the estimators presented above as simply being formed from averaging conditional expectations.

6.2 Marginalisation in Stan

The motivation for marginalisation in Stan is to enable estimation of $\mu = \mathbb{E}(f(X, Y))$ without having to sample from (X, Y) if either X or Y is discrete. Suppose that Y is discrete and X continuous. To compute a Monte Carlo estimate of $\mathbb{E}(f(X, Y))$ using Stan we carry out four steps.

First, marginalise out Y from $p_{X,Y}(x, y)$ to give $p_X(x)$. (See Equation (2.4.1) in Section 4.1 for how this is done in the Dawid–Skene model.) This marginal distribution, which only contains continuous parameters, should then be implemented as a Stan model. Second, using Stan as usual, sample from the distribution of the continuous parameters X to give Monte Carlo samples $\{x_i\}$. Given only the samples from the distribution of X , we can estimate $\mathbb{E}(f(X, Y))$ using the Rao–Blackwellized estimator described in the previous section. Doing so requires us to evaluate $h(X) = \mathbb{E}(f(X, Y) \mid X)$ for all

functions $f(\cdot)$ in which we may be interested; this is the third step of the process. Finally, as the fourth step, we can evaluate $h(x)$ for each of the Monte Carlo draws x_i and estimate $\mathbb{E}(f(X, Y))$ by

$$\frac{1}{n} \sum_{i=1}^n h(x_i).$$

Evaluating the conditional expectation

The third step outlined above may appear to require substantial mathematical manipulation. In practice, however, we can use the discrete nature of the latent class to simplify the calculation. Specifically, for any function $f(x, y)$ we have

$$h(X) = \mathbb{E}(f(X, Y) | X) = \sum_k f(X, k) \Pr(Y = k | X),$$

where we sum over the values in the latent discrete parameters. An essential part of this formula is the probability of the discrete parameters conditional on the continuous parameters, $\Pr(Y = k | X)$. This quantity can be derived easily through Bayes' theorem or can be encoded as part of the marginalised Stan model; see [Section 5.2](#) or the next section for how this is done in the case of the Dawid–Skene model.

In the Dawid–Skene model, and many other models with discrete variables, the discrete variables only take values in a finite set. Some models however may have discrete parameters which take countably infinite values, such as Poisson distributions. In this case, marginalisation is still possible but will generally involve approximating infinite sums, both when marginalising out the discrete random variables and calculating the expectations (as in the equation above), which may be computationally expensive.

Furthermore, for the cases where $f(x, y)$ is a function of only x or y , the general formula simplifies further. Firstly, when $f(x, y) = f(x)$ we have

$$h(X) = \sum_k f(X) \Pr(Y = k | X) = f(X) \sum_k \Pr(Y = k | X) = f(X).$$

This means that we can estimate $\mathbb{E}(f(X))$ with the standard, seemingly unconditional, estimator:

$$\sum_{i=1}^n f(x_i).$$

Even after marginalisation, computing expectations of functions of the continuous parameters can be performed as if no marginalisation had taken place.

Secondly, when $f(x, y) = f(y)$ we have that

$$h(X) = \sum_k f(k) \Pr(Y = k | X).$$

An important special case of this result is when $f(x, y) = \mathbf{1}(y = k)$, where $\mathbf{1}$ is the indicator function. This is important because it allows us to recover the probability mass function of the discrete random variable Y , since $\mathbb{E}(f(X, Y)) = \mathbb{E}(\mathbf{1}(Y = k)) = \Pr(Y = k)$. In this case we have

$$h(X) = \sum_k \mathbf{1}(y = k) \Pr(Y = k | X) = \Pr(Y = k | X).$$

We therefore estimate $\Pr(Y = k)$ with:

$$\frac{1}{n} \sum_{i=1}^n \Pr(Y = k | X = x_i).$$

Again, we stress that our ability to do these calculations relies upon being able to easily compute $\Pr(Y = k | X = x_i)$ for each of the Monte Carlo draws x_i .

Estimating the conditional probability of the true class

For the categorical rating problem (using the Dawid–Skene model), the discrete random variable of interest for each item i is the true class of the item, z_i . We use the technique from the previous section to calculate the posterior probability of the true class, as described in [Section 5.2](#). In this case, the discrete variable is z_i (taking on the role that Y played in the previous section), the continuous variables are

θ and π (which together take on the role that X played in the previous section), and all probability calculations are always conditional on the data (the ratings, y).

6.3 Efficiency of marginalisation

It is not immediately obvious whether marginalisation is more or less efficient than techniques that actually realise a value for the discrete random variable at each iteration, such as Gibbs sampling. Marginalising can be viewed as a form of Rao–Blackwellization, a general MCMC technique designed to reduce the variability of Monte Carlo estimators. This link strongly suggests that marginalisation is more efficient than using discrete random variables. Unfortunately, limitations of the theory of Rao–Blackwellization (see Section 6.1) and the difficulty of theoretically comparing different sampling algorithms, such as Gibbs sampling and NUTS, means that it is unclear whether marginalisation will always be computationally superior for a given problem.

However, in practice marginalisation does seem to improve convergence at least for non-conjugate models. For example, Yackulic et al. (2020) show that for the Cormack–Jolly–Seber model marginalisation greatly speeds up inference in JAGS, BUGS and WinBUGS. They also demonstrate that the marginalised model implemented in Stan is orders of magnitude more efficient than the marginalised models in the other languages. These results show that marginalisation has the potential to speed up classic algorithms and also allows the use of more efficient gradient-based algorithms, such as NUTS, for problems involving discrete random variables.

A recent similar exploration of marginalisation for some simple mixture models and the Dawid–Skene model, using JAGS and Stan, suggests that the software implementation has a greater impact on efficiency than the choice of whether or not marginalisation is used (Zhang et al. 2022). In their comparisons, Stan usually achieved the best performance (and note that Stan requires marginalisation of discrete parameters).

For these reasons, we recommend that practitioners using models with discrete parameters consider marginalisation if more efficiency is desired, and implement their models using Stan rather than JAGS.

7 Example usage

To demonstrate `rater` we use two example datasets.

The first dataset is taken from the original paper introducing the Dawid–Skene model (Dawid and Skene 1979). The data consist of ratings, on a 4-point scale, made by five anaesthetists of patients' pre-operative health. The ratings were based on the anaesthetists assessments of a standard form completed for all of the patients. There are 45 patients (items) and four anaesthetists (raters) in total. The first anaesthetist assessed the forms a total of three times, spaced several weeks apart. The other anaesthetists each assessed the forms once. As in the Dawid–Skene paper, we will not seek to model the effect of time on the ratings of the first anaesthetist.

First we load the `rater` package:

```
library(rater)

#> * The rater package uses `Stan` to fit bayesian models.
#> * If you are working on a local, multicore CPU with excess RAM please call:
#> * options(mc.cores = parallel::detectCores())
#> * This will allow Stan to run inference on multiple cores in parallel.
```

This will display information about altering options used by Stan to make best use of computational resources on your machine. We can then load and look at a snippet of the anaesthesia data, which is included in the package.

```
data("anesthesia", package = "rater")
head(anesthesia)

#>   item rater rating
#> 1    1     1      1
#> 2    1     1      1
#> 3    1     1      1
#> 4    1     2      1
#> 5    1     3      1
#> 6    1     4      1
```

These data are arranged in ‘long’ format where each row corresponds to a single rating. The first column gives the index of the item that was rated, the second the index of the rater that rated the item and the third the actual rating that was given. This layout of the data supports raters rating the same item multiple times, as happens in the dataset. It is also the most convenient from the perspective of fitting models but may not always be the optimal way to store or represent categorical rating data; see [Using grouped data](#) for an example using the ‘grouped’ data format.

7.1 Fitting the model

We can fit the Dawid–Skene model using MCMC by running the command:

```
fit_1 <- rater(anesthesia, dawid_skene())
```

This command will print the running output from Stan, providing an indication of the progress of the sampler (for brevity, we have not shown this output here). To fit the model via optimisation, we set the `method` argument to “`optim`”:

```
fit_2 <- rater(anesthesia, dawid_skene(), method = "optim")
```

The second argument of the `rater()` function specifies the model to use. We have implemented this similarly to the `family` argument in `glm()`, which can be passed as either a function or as a character string. The above examples pass the model as a function. We could have instead passed it as a string, for example:

```
fit_2 <- rater(anesthesia, "dawid_skene", method = "optim")
```

Either version will fit the Dawid–Skene model using the default choice of prior. The benefit of passing the model as a function is that it allows you to change the prior, see [Different models and priors](#).

7.2 Inspecting the fitted model

`rater` includes several ways to inspect the output of fitted models. These are summarised in [Table 4](#), and we illustrate many of them here. Firstly, we can generate a text summary:

```
summary(fit_1)

#> Model:
#> Bayesian Dawid and Skene Model
#>
#> Prior parameters:
#>
#> alpha: default
#> beta: default
#>
#> Fitting method: MCMC
#>
#> pi/theta samples:
#>               mean   5%  95% Rhat ess_bulk
#> pi[1]      0.37 0.27 0.48    1  9094.91
#> pi[2]      0.41 0.30 0.52    1  8050.47
#> pi[3]      0.14 0.07 0.23    1  6864.65
#> pi[4]      0.07 0.03 0.14    1  6665.16
#> theta[1, 1, 1] 0.86 0.79 0.93    1  7728.01
#> theta[1, 1, 2] 0.10 0.05 0.17    1  7857.38
#> theta[1, 1, 3] 0.02 0.00 0.05    1  5619.81
#> theta[1, 1, 4] 0.02 0.00 0.05    1  6283.33
#> # ... with 76 more rows
#>
#> z:
#>     MAP Pr(z = 1) Pr(z = 2) Pr(z = 3) Pr(z = 4)
#> z[1] 1       1.00      0.00      0.00      0.00
```

Table 4: Methods to inspect fitted models and what values they return.

	MCMC mode	Optimisation mode
'summary()'	Basic information about the fitted model	Basic information about the fitted model
'point_estimate()'	Posterior means for π and θ , posterior modes for z	Posterior modes for z
'posterior_interval()'	Credible intervals for π and θ	N/A
'posterior_samples()'	MCMC draws for π and θ	N/A
'mcmc_diagnostics()'	MCMC convergence diagnostics for π and θ	N/A
'class_probabilities()'	Posterior distribution for z	Posterior distribution for z

```
#> z[2] 3 0.00 0.00 0.98 0.02
#> z[3] 2 0.38 0.62 0.00 0.00
#> z[4] 2 0.01 0.99 0.00 0.00
#> z[5] 2 0.00 1.00 0.00 0.00
#> z[6] 2 0.00 1.00 0.00 0.00
#> z[7] 1 1.00 0.00 0.00 0.00
#> z[8] 3 0.00 0.00 1.00 0.00
#> # ... with 37 more items
```

This function will show information about which model has been fitted and the values of the prior parameters that were used. The displayed text also contains information about the parameter estimates and posterior distributions, and the convergence of the sampler.

We can extract point estimates for any of the parameters or the latent classes via the `point_estimate()` function. For example, the following will return the latent class with the highest posterior probability (i.e., the posterior modes) for each item:

```
point_estimate(fit_1, "z")

#> $z
#> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
#> 1 3 2 2 2 1 3 2 2 4 2 1 2 1 1 1 1 2 2 2 2 2 2 1 1
#> 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
#> 2 1 1 1 1 3 1 2 2 3 2 2 3 1 1 1 2 1 2
```

The following will return the posterior means of the prevalence probabilities:

```
point_estimate(fit_1, "pi")

#> $pi
#> [1] 0.37432122 0.40674116 0.14406295 0.07487466
```

From the outputs above, we can see that the model has inferred that most patients have health that should be classified as category 1 or 2 (roughly 40% each), and categories 3 and 4 being rarer (about 14% and 7% respectively). Based on the point estimates, there are examples of patients from each category in the dataset.

The function call `point_estimate(fit1, "theta")` will return posterior means for the parameters in the error matrices (not shown here, for brevity). When used with optimisation fits, `point_estimate()` will return posterior modes rather than posterior means for the parameters.

7.3 Inspecting posterior distributions

To represent uncertainty, we need to look beyond point estimates. The function `posterior_interval()` will return credible intervals for the parameters. For example, 80% credible intervals for the elements of the error matrices:

```
head(posterior_interval(fit_1, 0.8, "theta"))

#> 10% 90%
#> theta[1, 1, 1] 0.805074415 0.91666962
#> theta[1, 1, 2] 0.055802572 0.15309851
```

```
#> theta[1, 1, 3] 0.001813371 0.03941595
#> theta[1, 1, 4] 0.001760845 0.03832660
#> theta[1, 2, 1] 0.026218408 0.10590753
#> theta[1, 2, 2] 0.791396057 0.90637035
```

The function `posterior_samples()` will return the actual MCMC draws. For example (here just illustrating the draws of the π parameter):

```
head(posterior_samples(fit_1, "pi")$pi)

#>
#> iterations      [,1]      [,2]      [,3]      [,4]
#> [1,] 0.4830595 0.3579169 0.1254949 0.03352871
#> [2,] 0.2939839 0.3834238 0.2516929 0.07089945
#> [3,] 0.3636097 0.4148082 0.1309119 0.09067012
#> [4,] 0.3798456 0.3133947 0.2353036 0.07145615
#> [5,] 0.3639428 0.4291340 0.1760525 0.03087066
#> [6,] 0.3845465 0.3831443 0.1721656 0.06014354
```

Neither of the above will work for optimisation fits, which are limited to point estimates only. For the latent classes, we can produce the probability distribution as follows:

```
head(class_probabilities(fit_1))

#>
#>      [,1]      [,2]      [,3]      [,4]
#> 1 9.999994e-01 1.340790e-07 2.916232e-08 4.521956e-07
#> 2 8.469190e-08 2.570243e-05 9.773942e-01 2.257999e-02
#> 3 3.823085e-01 6.171346e-01 1.075246e-04 4.493620e-04
#> 4 5.252130e-03 9.942742e-01 3.418545e-04 1.318565e-04
#> 5 2.473083e-07 9.999633e-01 3.338995e-05 3.062785e-06
#> 6 1.674456e-06 9.993819e-01 5.813534e-04 3.502596e-05
```

This works for both MCMC and optimisation fits. For the former, the output is the posterior distribution on z , while for the latter it is the distribution conditional on the point estimates of the parameters (π and θ).

7.4 Plots

It is often easier to interpret model outputs visually. Using `rater`, we can plot the parameter estimates and distribution of latent classes.

The following command will visualise the posterior means of the error rate parameters, with the output shown in Figure 2:

```
plot(fit_1, "raters")
```

We can see high values along the diagonals of these matrices, which indicates that each of the 5 anaesthetists is inferred as being fairly accurate at rating pre-operative health. Looking at the non-diagonal entries, we can see that typically the most common errors are 1-point differences on the rating scale.

The following command visualises the latent class probabilities, with the output shown in Figure 3:

```
plot(fit_1, "latent_class", item_index = c(2, 3, 12, 36, 38))
```

For the purpose of illustration, for this plot we have selected the 5 patients with the greatest uncertainty in their pre-operative health (latent class). The other 40 patients all have almost no posterior uncertainty for their pre-operative health. Thus, suppose we wished to use the model to infer the pre-operative health by combining all of the anaesthetists' ratings, we can do so confidently for all but a handful of patients.

The same output for the models fitted via optimisation rather than MCMC (using `fit_2` instead of `fit_1`) are shown in Figure 4 and Figure 5. We can see that this estimation method leads to the same

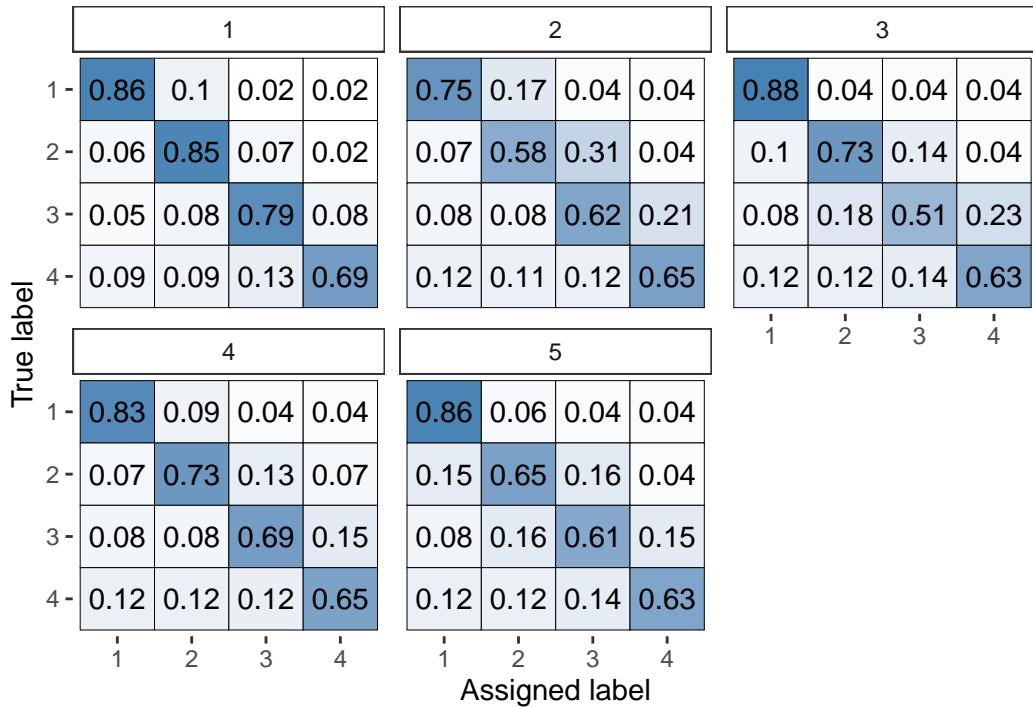


Figure 2: Visual representation of the inferred parameters in the error matrices (θ) for the Dawid–Skene model fitted via MCMC to the anaesthesia dataset. The values shown are posterior means, with each cell shaded on a gradient from white (value close to 0) to dark blue (value close to 1).

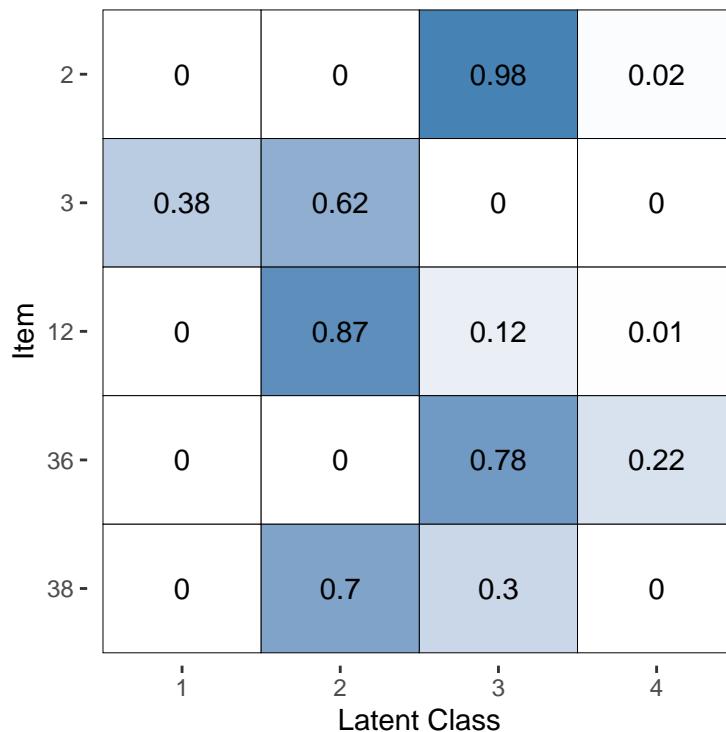


Figure 3: Visualisation of the inferred probability of each latent class, for a selected subset of items, for the Dawid–Skene model fitted via MCMC to the anaesthesia dataset.

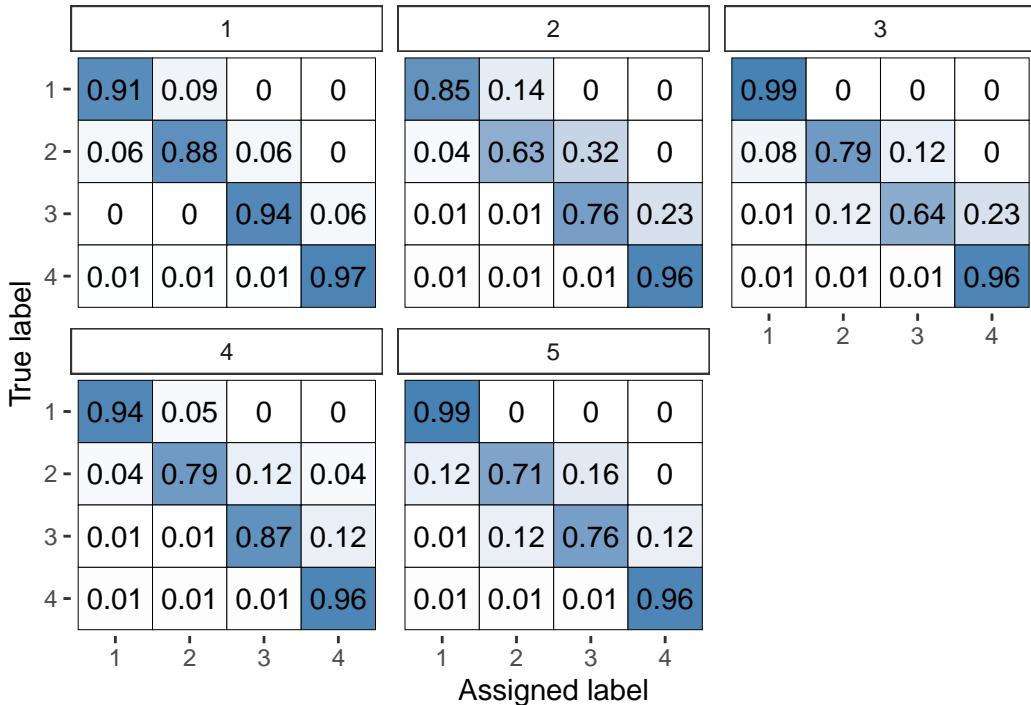


Figure 4: Visual representation of the inferred parameters in the error matrices (θ) for the Dawid–Skene model fitted via optimisation to the anaesthesia dataset. Compare with [Figure 2](#), which used MCMC instead of optimisation.

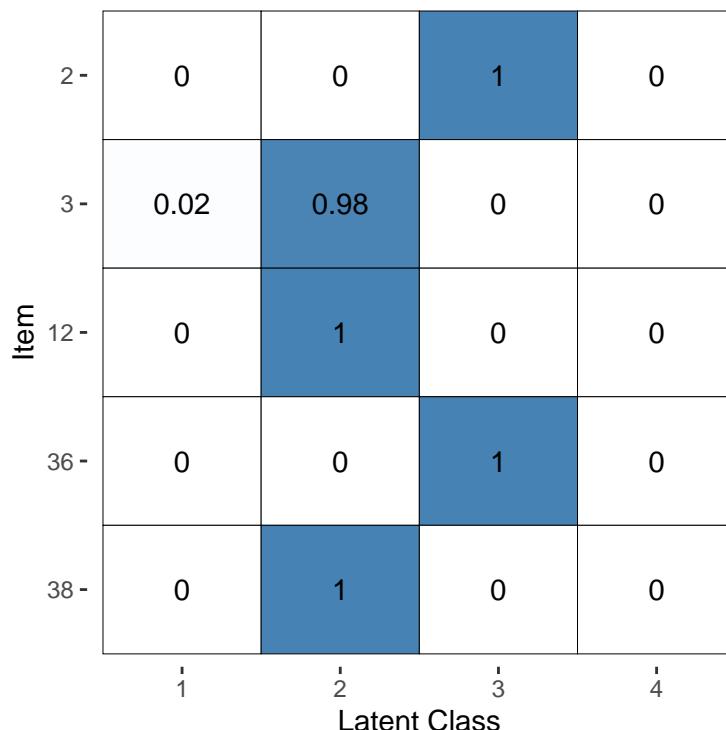


Figure 5: Visualisation of the inferred probability of each latent class, for a selected subset of items, for the Dawid–Skene model fitted via optimisation to the anaesthesia dataset. Compare with [Figure 3](#), which used MCMC instead of optimisation.

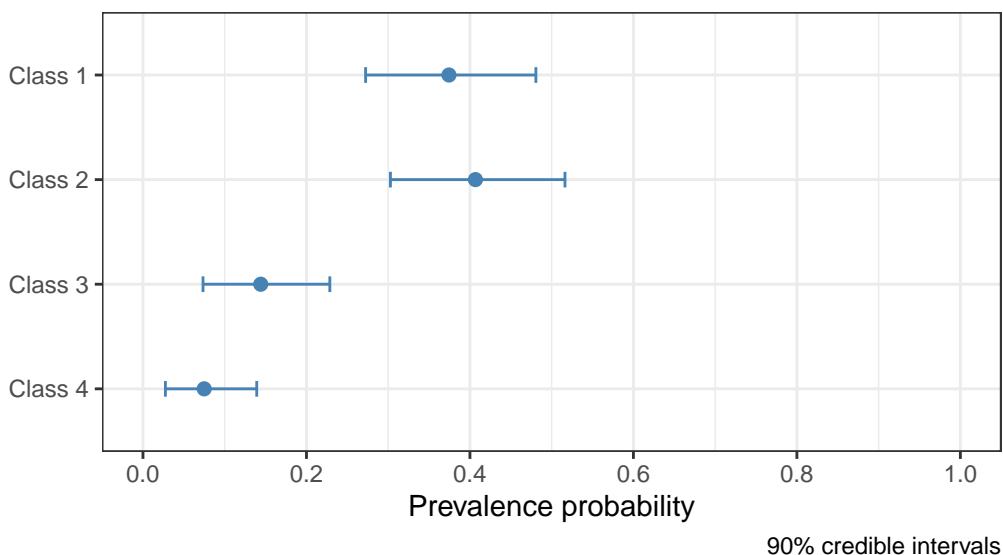


Figure 6: Visualisation of the inferred population prevalence parameters, with 90% credible intervals, for the Dawid–Skene model fitted to the anaesthesia dataset.

broad conclusions, however the optimisation-based estimates have considerably less uncertainty: they are typically closer to 0 or 1. This behaviour reflects the fact that optimisation-based inference will usually not capture the full uncertainty and will lead to overconfident estimates. Thus, we recommend using MCMC (which we have set as the default).

While it is typically of less interest, it is also possible to visualise the prevalence estimates, together with credible intervals, using the following command (see output in Figure 6):

```
plot(fit_1, "prevalence")
```

7.5 Different models and priors

The second argument to the `rater()` function specifies what model to use, including details of the prior if not using the default one. This gives a unified place to specify both the model and prior.

For example, this is how to set a different prior using the Dawid–Skene model:

```
diff_alpha_fit <- rater(anesthesia, dawid_skene(alpha = rep(10, 4)))
```

When specifying the β hyper-parameters for the Dawid–Skene model, the user can either specify a single matrix or a 3-dimensional array. If only a matrix specified, it will be interpreted as the hyper-parameter values for all raters. This is useful in the common situation where the overall quality of the raters is known but there is no information on the quality of any specific rater. When a 3-dimensional array is passed, it is taken to specify the hyper-parameters for each of the raters (i.e., a separate matrix for each rater). This is useful when prior information about the quality of specific raters is available, for example when some ‘raters’ are diagnostic tests with known performance characteristics.

This is how to use the class-conditional Dawid–Skene model (with default prior):

```
diff_model_fit <- rater(anesthesia, class_conditional_dawid_skene())
```

Compared with the Dawid–Skene model, the latter uses error matrices with the constraint that all off-diagonal entries must be the same. We can visualise the θ parameter using the following command (output in Figure 7) and see that all off-diagonal elements are indeed equal:

```
plot(diff_model_fit, "theta")
```

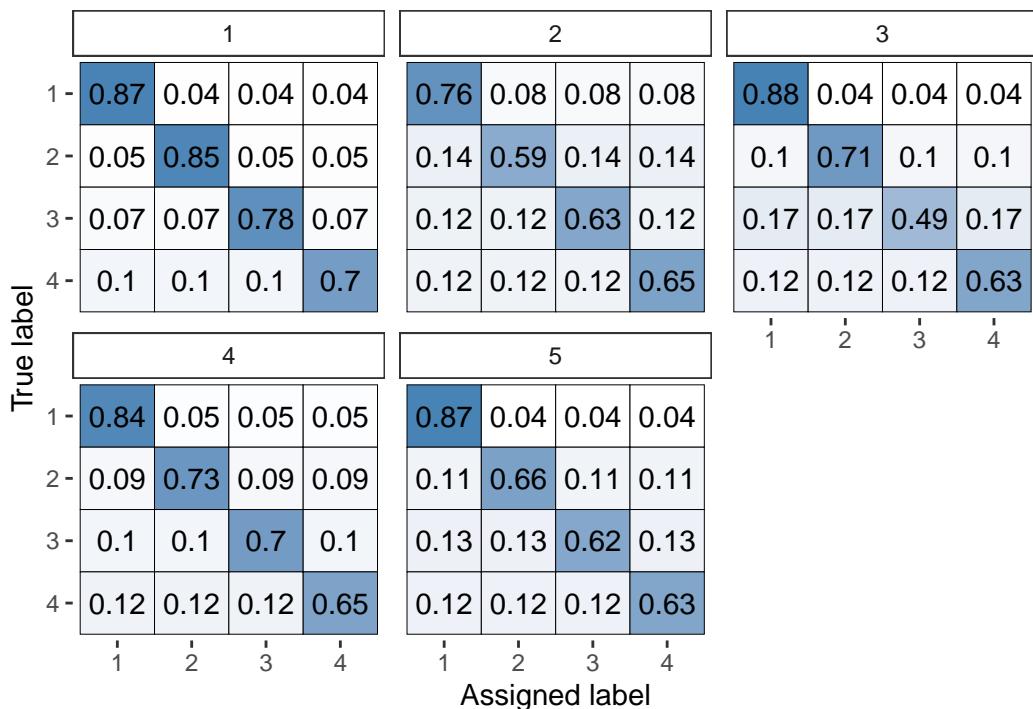


Figure 7: Visual representation of the inferred parameters in the error matrices (θ) for the class-conditional Dawid–Skene model fitted via MCMC to the anaesthesia dataset. Compare with Figure 2, which used the standard Dawid–Skene model.

7.6 Using grouped data

The second example dataset is taken from Espeland and Handelman (1989). It consists of 3,689 binary ratings, made by 5 dentists, of whether a given tooth was healthy or had caries/cavities. The ratings were performed using X-ray only, which was thought to be more error-prone than visual/tactile assessment of each tooth (see Handelman et al. (1986) for more information and a description of the wider dataset from which these binary ratings were taken).

```
data("caries", package = "rater")
head(caries)

#>   rater_1 rater_2 rater_3 rater_4 rater_5   n
#> 1      1      1      1      1      1 1880
#> 2      1      1      1      1      2   789
#> 3      1      1      1      2      1    43
#> 4      1      1      1      2      2    75
#> 5      1      1      2      1      1    23
#> 6      1      1      2      1      2    63
```

This is an example of ‘grouped’ data. Each row represents a particular ratings ‘pattern’, with the final column being a tally that records how many instances of that pattern appear in the data. `rater` accepts data in either ‘long’, ‘wide’ or ‘grouped’ format. The ‘long’ format is the default because it can represent data with repeated ratings. When available the ‘grouped’ format can greatly speed up the computation of certain models and is convenient for datasets that are already recorded in that format. The ‘wide’ format doesn’t provide any computational advantages but is a common format for data without repeated ratings and so is provided for convenience.

Here’s how we fit the model to grouped data (note the `data_format` argument):

```
fit3 <- rater(caries, dawid_skene(), data_format = "grouped")
```

From there, we can inspect the model fit in the same way as before. For example (output shown in Figure 8):

```
plot(fit3, "raters")
```

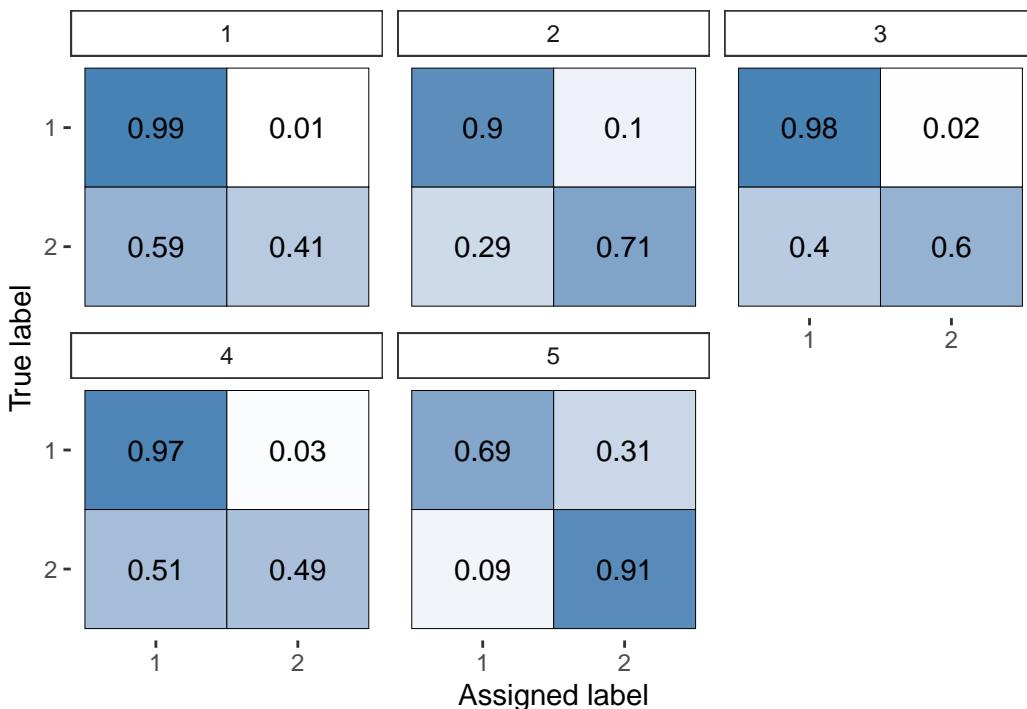


Figure 8: Visual representation of the inferred parameters in the error matrices (θ) for the Dawid–Skene model fitted via MCMC to the caries dataset.

The first 4 dentists are highly accurate at diagnosing healthy teeth (rating = 1), but less so at diagnosing caries (rating = 2). In contrast, the 5th dentist was much better at diagnosing caries than healthy teeth.

7.7 Convergence diagnostics

A key part of applied Bayesian statistics is assessing whether the MCMC sampler has converged on the posterior distribution. To summarise the convergence of a model fit using MCMC, `rater` provides the `mcmc_diagnostics()` function.

```
head(mcmc_diagnostics(fit_1))

#>                               Rhat ess_bulk
#> pi[1]           0.9997012 9094.906
#> pi[2]           1.0009095 8050.465
#> pi[3]           1.0000406 6864.648
#> pi[4]           1.0021824 6665.159
#> theta[1, 1, 1] 1.0013982 7728.007
#> theta[1, 1, 2] 1.0011678 7857.378
```

This function calculates and displays the \hat{R} statistic (Rhat) and bulk effective sample size (ess_bulk) for all of the π and θ parameters. Users can then check for the convergence of specific parameters by applying standard rules, such as considering that convergence has been reached for a specific parameter if its $\hat{R} < 1.01$ (Vehtari et al. 2021).

Users wishing to calculate other metrics, or produce convergence visualisations such as trace plots, can either extract the underlying Stan model from the `rater` fit using the `get_stanmodel()` function, or convert the fit into a `mcmc.list` from the `coda` package using the `as_mcmc.list()` function. Functions in the `rstan` and `coda` packages will then allow visualisation based assessment of convergence and the calculation of other, more advanced, diagnostics.

For example, the following code uses `rstan` to draw the trace plot (shown in Figure 9) for one of the parameters from the Dawid–Skene model fitted to the anaesthesia data. We can see that the four chains show good mixing. The trace plots for the other parameters (not shown here) are similar, indicating that the MCMC sampling procedure has converged.

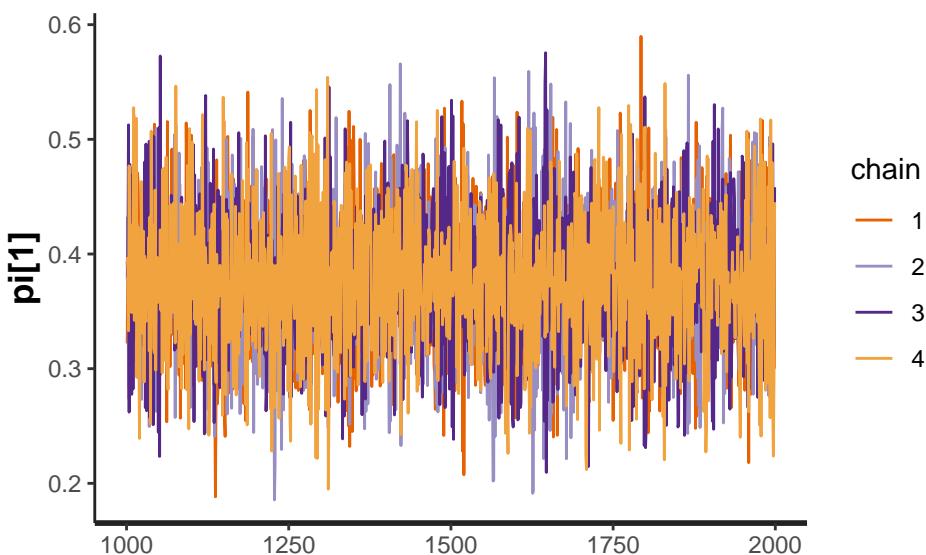


Figure 9: A trace plot for the π_1 parameter from the Dawid–Skene model fitted via MCMC to the anaesthesia dataset.

```
stan_fit <- get_stanfit(fit_1)
rstan::traceplot(stan_fit, pars = "pi[1]")
```

7.8 Model assessment and comparison

Finally, `rater` provides facilities to assess and compare fitted models. First, it provides the `posterior_predict()` function to simulate from the posterior predictive distributions of all the models implemented in `rater`. The simulated data can then be used, for example, to perform posterior predictive checks. These checks compare the data simulated from the fitted model to the observed data. A number of datasets are simulated from the posterior predictive distribution and a summary statistic is calculated for each dataset. The same summary statistic is calculated on the observed dataset and is compared to the distribution of simulated statistics. A large discrepancy between the simulated statistics and the observed statistic indicates possible model misspecification.

To illustrate this functionality, we consider assessing the fit of the Dawid–Skene model to the anaesthesia dataset. In this example, we simulate 1,000 datasets from the posterior predictive distribution, and use the proportion of the ratings that are class 2 as a statistic to summarise each dataset.

```
class2prop <- function() {
  simdata <- posterior_predict(fit_1, anesthesia[, 1:2])
  sum(simdata$rating == 2) / nrow(simdata)
}
ppc_statistics <- replicate(1000, class2prop())
head(ppc_statistics)

#> [1] 0.5587302 0.4507937 0.4126984 0.4190476 0.4666667 0.2539683
```

We can then graphically compare the distribution of these statistics to the value of the same statistic applied to the anaesthesia dataset. The following commands will create such a plot (shown in Figure 10):

```
ggplot(data.frame(prop_class_two = ppc_statistics), aes(prop_class_two)) +
  geom_histogram(binwidth = 0.01, fill = "steelblue", colour = "black") +
  geom_vline(xintercept = sum(anesthesia$rating == 2) / nrow(anesthesia),
             colour = "black", linewidth = 2) +
  theme_bw() +
  coord_cartesian(xlim = c(0.1, 0.6)) +
  labs(
    x = "Proportion of class 2 ratings",
    y = "Count"
  )
```

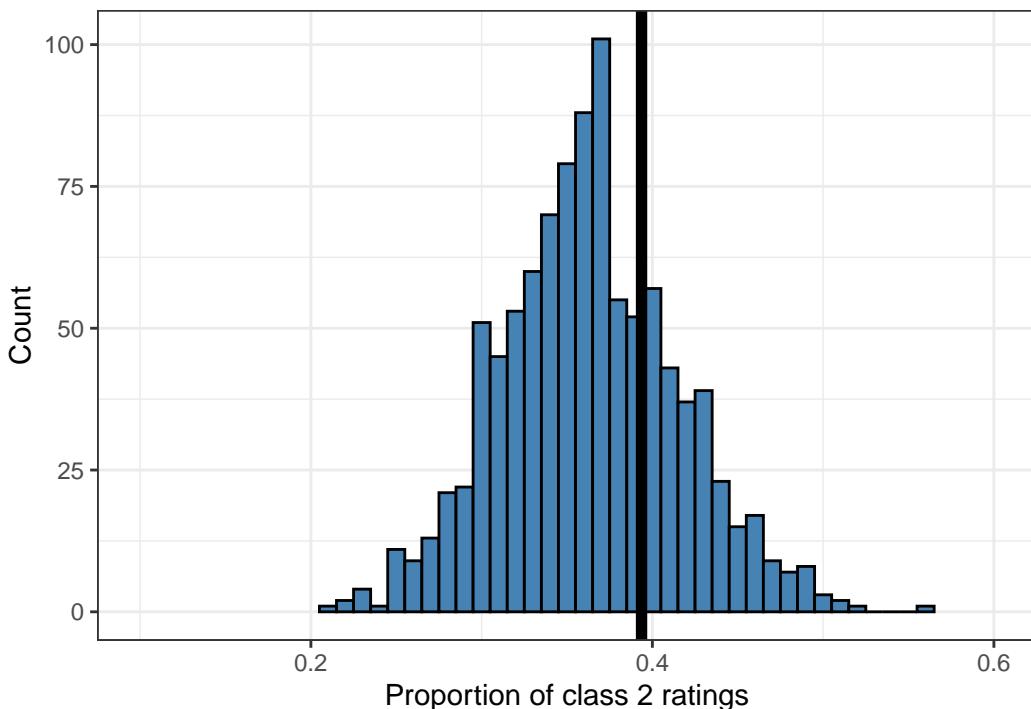


Figure 10: An example of a posterior predictive check for the Dawid–Skene model fitted via MCMC to the anaesthesia dataset. Shown is a histogram of 1,000 simulated datasets from the posterior predictive distribution, each summarised by the proportion of ratings that are class 2. The vertical black line shows the proportion of ratings that are class 2 in the original dataset, for comparison against the histogram.

The statistic calculated on the anaesthesia dataset lies towards the centre of the distribution of statistics calculated from datasets drawn from the posterior predictive distribution. This is consistent with the model fitting the data well. (This is an illustrative example only. A more comprehensive set of such comparisons should be conducted before concluding that the model adequately describes the data.)

Second, `rater` implements the functions `loo()` and `waic()` to provide model comparison metrics using the `loo` package. The function `loo()` calculates an approximation to leave-out-one cross-validation, a measure of model performance using Pareto Smoothed Importance Sampling (Vehtari, Gelman, and Gabry 2017). The function `waic()` calculates measures related to the Widely Applicable Information Criterion (Watanabe 2013). Because `rater` fits models in a Bayesian framework, information criteria such as the AIC and BIC are not implemented.

In the context of `rater`, model comparison is only possible between the different implemented models, not between the same model with different parameter values or included covariates, as is more commonly the case. For this reason, considerations of data size and what is known about the characteristics of the raters should also be taken into account when making such choices, in addition to the output of `loo()` and `waic()`.

To briefly illustrate some of this functionality, we compare the standard Dawid–Skene model to the class-conditional one.

```
loo(fit_1)

#>
#> Computed from 4000 by 45 log-likelihood matrix
#>
#>       Estimate   SE
#> elpd_loo -233.7 16.9
#> p_loo      19.8  2.6
#> looic     467.5 33.9
#> -----
#> Monte Carlo SE of elpd_loo is 0.1.
#>
#> Pareto k diagnostic values:
```

```

#>                               Count Pct.   Min. n_eff
#> (-Inf, 0.5] (good)        43  95.6%  734
#> (0.5, 0.7] (ok)           2   4.4%  497
#> (0.7, 1] (bad)           0   0.0% <NA>
#> (1, Inf) (very bad)      0   0.0% <NA>
#>
#> All Pareto k estimates are ok (k < 0.7).
#> See help('pareto-k-diagnostic') for details.

loo(diff_model_fit)

#>
#> Computed from 4000 by 45 log-likelihood matrix
#>
#>           Estimate    SE
#> elpd_loo -245.7 18.1
#> p_loo     10.3  1.2
#> looic     491.3 36.2
#> -----
#> Monte Carlo SE of elpd_loo is 0.1.
#>
#> All Pareto k estimates are good (k < 0.5).
#> See help('pareto-k-diagnostic') for details.

loo_compare(loo(fit_1), loo(diff_model_fit))

#>           elpd_diff se_diff
#> model1     0.0      0.0
#> model2 -11.9      3.2

```

From these results, we see that both models fit the data well, with the class-conditional model being slightly preferred.

8 Summary and discussion

Rating procedures, in which items are sorted into categories, are subject to both classification error and uncertainty when the categories themselves are defined subjectively. Data that results from these types of tasks require a proper statistical treatment if: (1) the population frequencies of the categories are to be estimated with low bias, (2) items are to be assigned to these categories reliably, and (3) the agreement of raters is to be assessed. To date there have been few options for practitioners seeking software that implements the required range of statistical models. The R package described in this paper, [rater](#), provides the ability to fit Bayesian versions of a variety of statistical models to categorical rating data, with a range of tools to extract and visualise parameter estimates.

The statistical models that we have presented are based on the Dawid–Skene model and recent modifications of it including extensions, such as the hierarchical Dawid–Skene model where the rater-specific parameters are assumed to be drawn from a population distribution, and simplifications that, for example, assume exchangeable raters with identical performance characteristics (homogeneous Dawid–Skene), or homogeneity criteria where classification probabilities are identical regardless of the true, underlying category of an item (class-conditional Dawid–Skene).

We provided: (1) an explanation of the type of data formats that categorical ratings are recorded in, (2) a description of the construction and implementation of the package, (3) a comparison of our package to other existing packages for fitting latent class models in R, (4) an introduction to the user interface, and (5) worked examples of real-world data analysis using [rater](#).

We devoted an entire section to motivating, deriving and explaining the use of a marginalised version of the joint posterior probability distribution to remove dependence on the unknown value of the true, underlying rating category of an item. This is necessary because the No-U-Turn Sampler, the main MCMC algorithm used in Stan, relies on computing derivatives with respect to all parameters that must, therefore, be continuously-valued. The technique involves the use of conditional expectation and is a special case of a more general technique of conditioning or ‘Rao–Blackwellization’, the process of transforming an estimator using the Rao–Blackwell theorem to improve its efficiency.

Our package was developed with the classification of medical images in mind, where there may be a large number of images but typically only a small number of possible categories and a limited

number of expert raters. The techniques proposed based on extensions of the Dawid–Skene model readily extend to scenarios where the datasets are much larger, or where the raters behave erratically (cannot be relied on to rate correctly more frequently than they rate incorrectly) or are antagonistic (deliberately attempt to allocate items to a category not favoured by the majority of raters).

One possible extension of the Dawid–Skene model is to add item- and rater-specific covariates. These covariates would encode the difficulty of items and the expertise (or lack thereof) of the raters. This extension is particularly attractive as it would partially alleviate the strong assumption of independence conditional on the latent class, replacing it with the weaker assumption of independence conditional on the latent class and the covariates. Unfortunately, these types of models would contain many more parameters than the original Dawid–Skene model making them difficult to fit, especially when used with relatively small datasets common in the context of rating data in medical research. Therefore, future methodological research is needed before these models can be included in the `rater` package. Latent class models with covariates can be fitted, although only in a frequentist framework, using the R package `poLCA`.

9 Computational details

The results in this paper were obtained using R 4.2.2 (R Core Team 2021) and the following packages: `coda` 0.19.4 (Plummer et al. 2006), `ggplot2` 3.4.3 (Wickham 2016), `knitr` 1.45 (Xie 2014, 2015, 2023), `loo` 2.6.0 (Vehtari et al. 2023), `rater` 1.3.1 (Pullin and Vukcevic 2023), `rjtools` 1.0.12 (O’Hara-Wild et al. 2023), `rmarkdown` 1.0.12 (Xie, Allaire, and Grolemund 2018; Xie, Dervieux, and Riederer 2020; Allaire et al. 2023), `rstan` 2.26.23 (Stan Development Team 2023). R itself and all packages used are available from the Comprehensive R Archive Network (CRAN).

10 Acknowledgments

We would like to thank Bob Carpenter for many helpful suggestions, and David Whitelaw for his flexibility in allowing the first author to work on this paper while employed at the Australian Institute of Heath and Welfare. Thanks also to Lars Mølgaard Saxhaug for his code contributions to `rater`.

References

- Allaire, JJ, Yihui Xie, Christophe Dervieux, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, et al. 2023. *rmarkdown: Dynamic Documents for r*. <https://github.com/rstudio/rmarkdown>.
- Asselineau, J., A. Paye, E. Bessède, P. Perez, and C. Proust-Lima. 2018. “Different Latent Class Models Were Used and Evaluated for Assessing the Accuracy of Campylobacter Diagnostic Tests: Overcoming Imperfect Reference Standards?” *Epidemiology and Infection* 146 (12): 1556–64. <https://doi.org/10.1017/S0950268818001723>.
- Beath, Ken J. 2017. “randomLCA: An R Package for Latent Class with Random Effects Analysis.” *Journal of Statistical Software* 81 (13): 1–25. <https://doi.org/10.18637/jss.v081.i13>.
- Berkes, Pietro, Bob Carpenter, Andrey Rzhetsky, and James Evans. 2011. <http://docs.enthought.com/uchicago-pyanno/>.
- Blackwell, David. 1947. “Conditional Expectation and Unbiased Sequential Estimation.” *Ann. Math. Statist.* 18 (1): 105–10. <https://doi.org/10.1214/aoms/1177730497>.
- Carpenter, Bob, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. “Stan: A Probabilistic Programming Language.” *Journal of Statistical Software* 76 (1). <https://doi.org/10.18637/jss.v076.i01>.
- Dawid, A. P., and A. M. Skene. 1979. “Maximum Likelihood Estimation of Observer Error-Rates Using the EM Algorithm.” *Applied Statistics* 28 (1): 20. <https://doi.org/10.2307/2346806>.
- Espeland, Mark A., and Stanley L. Handelman. 1989. “Using Latent Class Models to Characterize and Assess Relative Error in Discrete Measurements.” *Biometrics* 45 (2): 587–99. <https://doi.org/10.2307/2531499>.
- Geyer, Charles J. 1995. “Conditioning in Markov Chain Monte Carlo.” *Journal of Computational and Graphical Statistics* 4 (2): 148–54. <https://doi.org/10.1080/10618600.1995.10474672>.
- Goodman, Leo A. 1974. “Exploratory Latent Structure Analysis Using Both Identifiable and Unidentifiable Models.” *Biometrika* 61 (2): 215–31. <https://doi.org/10.1093/biomet/61.2.215>.
- Handelman, Stanley L., D. H. Leverett, Mark A. Espeland, and Jennifer A. Curzon. 1986. “Clinical Radiographic Evaluation of Sealed Carious and Sound Tooth Surfaces.” *The Journal of the American Dental Association* 113 (5): 751–54. <https://doi.org/10.14219/jada.archive.1986.0269>.

- Hoffman, Matthew D., and Andrew Gelman. 2014. "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo." *Journal of Machine Learning Research* 15: 1593–623. <http://jmlr.org/papers/v15/hoffman14a.html>.
- Hui, S. L., and S. D. Walter. 1980. "Estimating the Error Rates of Diagnostic Tests." *Biometrics* 36 (1): 167–71. <https://doi.org/10.2307/2530508>.
- Ipeirotis, Panagiotis G., Foster Provost, and Jing Wang. 2010. "Quality Management on Amazon Mechanical Turk." In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, 64–67.
- Jakobsdottir, Johanna, and Daniel E Weeks. 2007. "Estimating Prevalence, False-Positive Rate, and False-Negative Rate with Use of Repeated Testing When True Responses Are Unknown." *The American Journal of Human Genetics* 81 (5): 1111–13. <https://doi.org/10.1086/521582>.
- Li, Hongwei, and Bin Yu. 2014. "Error Rate Bounds and Iterative Weighted Majority Voting for Crowdsourcing." *arXiv Preprint arXiv:1411.4086*. <https://arxiv.org/abs/1411.4086>.
- Linzer, Drew A., and Jeffrey B. Lewis. 2011. "poLCA: An R Package for Polytomous Variable Latent Class Analysis." *Journal of Statistical Software* 42 (1): 1–29. <https://doi.org/10.18637/jss.v042.i10>.
- O'Hara-Wild, Mitchell, Stephanie Kobakian, H. Sherry Zhang, Di Cook, Simon Urbanek, and Christophe Dervieux. 2023. *rjtools: Preparing, Checking, and Submitting Articles to the "R Journal"*. <https://CRAN.R-project.org/package=rjtools>.
- Owen, Art B. 2013. *Monte Carlo Theory, Methods and Examples*. <https://statweb.stanford.edu/~owen/mc/>.
- Passonneau, Rebecca J., and Bob Carpenter. 2014. "The Benefits of a Model of Annotation." *Transactions of the Association for Computational Linguistics* 2 (December): 311–26. https://doi.org/10.1162/tacl_a_00185.
- Paun, Silviu, Bob Carpenter, Jon Chamberlain, Dirk Hovy, Udo Kruschwitz, and Massimo Poesio. 2018. "Comparing Bayesian Models of Annotation." *Transactions of the Association for Computational Linguistics* 6 (December): 571–85. https://doi.org/10.1162/tacl_a_00040.
- Plummer, Martyn, Nicky Best, Kate Cowles, and Karen Vines. 2006. "CODA: Convergence Diagnosis and Output Analysis for MCMC." *R News* 6 (1): 7–11. <https://journal.r-project.org/archive/>.
- Pullin, Jeffrey, and Damjan Vukcevic. 2023. *rater: Statistical Models of Repeated Categorical Rating Data*. <https://CRAN.R-project.org/package=rater>.
- R Core Team. 2021. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. <https://www.R-project.org/>.
- Robert, Christian P., and Gareth O. Roberts. 2021. "Rao-Blackwellization in the MCMC Era." *arXiv:2101.01011 [Stat]*. <https://arxiv.org/abs/2101.01011>.
- Rzhetsky, Hagit AND Wilbur, Andrey AND Shatkay. 2009. "How to Get the Most Out of Your Curation Effort." *PLOS Computational Biology* 5 (5): 1–13. <https://doi.org/10.1371/journal.pcbi.1000391>.
- Smyth, Padhraic, Usama Fayyad, Michael Burl, Pietro Perona, and Pierre Baldi. 1994. "Inferring Ground Truth from Subjective Labelling of Venus Images." In *Advances in Neural Information Processing Systems*, edited by G. Tesauro, D. Touretzky, and T. Leen. Vol. 7. MIT Press. https://proceedings.neurips.cc/paper_files/paper/1994/file/3cef96dcc9b8035d23f69e30bb19218a-Paper.pdf.
- Stan Development Team. 2021. *Stan User's Guide*. https://mc-stan.org/docs/2_27/stan-users-guide/.
- . 2023. "RStan: The R Interface to Stan." <http://mc-stan.org/>.
- Vehtari, Aki, Jonah Gabry, Mans Magnusson, Yuling Yao, Paul-Christian Bürkner, Topi Paananen, and Andrew Gelman. 2023. "Loo: Efficient Leave-One-Out Cross-Validation and WAIC for Bayesian Models." <https://mc-stan.org/loo/>.
- Vehtari, Aki, Andrew Gelman, and Jonah Gabry. 2017. "Practical Bayesian Model Evaluation Using Leave-One-Out Cross-Validation and WAIC." *Statistics and Computing* 27 (5): 1413–32. <https://doi.org/10.1007/s11222-016-9696-4>.
- Vehtari, Aki, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner. 2021. "Rank-Normalization, Folding, and Localization: An Improved R⁺ for Assessing Convergence of MCMC (with Discussion)." *Bayesian Analysis* 16 (2): 667–718. <https://doi.org/10.1214/20-BA1221>.
- Watanabe, Sumio. 2013. "A Widely Applicable Bayesian Information Criterion." *Journal of Machine Learning Research* 14 (March): 867–97.
- White, Arthur, and Thomas Brendan Murphy. 2014. "BayesLCA: An R Package for Bayesian Latent Class Analysis." *Journal of Statistical Software* 61 (13). <https://doi.org/10.18637/jss.v061.i13>.
- Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. <https://ggplot2.tidyverse.org>.
- Xie, Yihui. 2014. "knitr: A Comprehensive Tool for Reproducible Research in R." In *Implementing Reproducible Computational Research*, edited by Victoria Stodden, Friedrich Leisch, and Roger D. Peng. Chapman; Hall/CRC.

- . 2015. *Dynamic Documents with R and Knitr*. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. <https://yihui.org/knitr/>.
- . 2023. *knitr: A General-Purpose Package for Dynamic Report Generation in r*. <https://yihui.org/knitr/>.
- 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>.
- Yackulic, Charles B., Michael Dodrill, Maria Dzul, Jamie S. Sanderlin, and Janice A. Reid. 2020. “A Need for Speed in Bayesian Population Models: A Practical Guide to Marginalizing and Recovering Discrete Latent States.” *Ecological Applications* 30 (5): e02112. <https://doi.org/10.1002/eap.2112>.
- Zhang, Wen, Jeffrey Pullin, Lyle Gurrin, and Damjan Vukcevic. 2022. “Investigating the Efficiency of Marginalising over Discrete Parameters in Bayesian Computations.” *arXiv:2204.06313 [Stat]*. <https://doi.org/10.48550/ARXIV.2204.06313>.

Jeffrey M. Pullin
University of Melbourne
School of Mathematics and Statistics
Melbourne, Australia
ORCID: [0000-0003-3651-5471](https://orcid.org/0000-0003-3651-5471)

Lyle C. Gurrin
University of Melbourne
School of Population and Global Health
Melbourne, Australia
ORCID: [0000-0001-7052-1969](https://orcid.org/0000-0001-7052-1969)

Damjan Vukcevic
Monash University
Department of Econometrics and Business Statistics
Melbourne, Australia
ORCID: [0000-0001-7780-9586](https://orcid.org/0000-0001-7780-9586)
damjan.vukcevic@monash.edu

GREENeR: An R Package to Estimate and Visualize Nutrients Pressures on Surface Waters

by Angel Udías, Bruna Grizzetti, Olga Vigiak, Alberto Aloe, Cesar Alfaro, and Javier Gomez

Abstract Nutrient pollution affects fresh and coastal waters around the globe. Planning mitigating actions requires tools to assess fluxes of nutrient emissions to waters and expected restoration impacts. Conceptual river basin models take advantage of data on nutrient emissions and concentrations at monitoring stations, providing a physical interpretation of monitored conditions, and enabling scenario analysis. The GREENeR package streamlines water quality model in a region of interest, considering nutrient pathways and the hydrological structure of the river network. The package merges data sources, analyzes local conditions, calibrate the model, and assesses yearly nutrient levels along the river network, determining contributions of load in freshwaters from diffuse and point sources. The package is enriched with functions to perform thorough parameter sensitivity analysis and for mapping nutrient sources and fluxes. The functionalities of the package are demonstrated using datasets from the Vistula river basin.

1 Introduction

Nitrogen and phosphorus are key nutrients that heavily impact aquatic ecosystems. Detecting primary sources of nutrient pollution and their downstream spread, alongside assessing achievable reductions through restoration policies, are crucial for effective natural resource management. They aid in identifying priority intervention areas and planning actions to restore the ecological balance of receiving waters.

Modelling tools can be useful to assess the impacts of future scenarios, policy measures, and climate changes at the regional and continental scale (Arheimer, Dahné, and Donnelly 2012; Bartosova et al. 2019; Beusen et al. 2022; Bouraoui et al. 2014; Ludwig et al. 2010; Seitzinger et al. 2005), and to check the coherence of different policy targets, for instance between water and agricultural policies. The assessment of policy scenarios requires a flexible and spatially detailed analysis to account for climatic, hydrological, and socio-economic gradients (Bruna Grizzetti et al. 2021).

Several types of models can be applied to predict the transport of nutrients in river basins Fu et al. (2019). Among them, statistical or conceptual models have the advantage of being readily applied in large watersheds. These model rely on calibration of few parameters for establishing links between emissions at sources and fate in the stream network. They take full advantage of nutrient emissions and concentrations data at monitoring stations, which are now accessible with increasing spatial and temporal resolution.

A classic example of river basin conceptual model is SPARROW (Smith, Schwarz, and Alexander 1997; Schwarz et al. 2006), which is widely applied in the U.S. to assess water quality over large regions. SPARROW has inspired the Geospatial Regression Equation for European Nutrient (GREEN) losses model (B. Grizzetti et al. 2005; Bruna Grizzetti, Bouraoui, and Aloe 2012; Bruna Grizzetti et al. 2021) which is adapted to European conditions. GREEN has been used for assessing the nutrient loads to the European seas (Bruna Grizzetti, Bouraoui, and Aloe 2012; Bruna Grizzetti et al. 2021), nitrogen retention in European freshwaters (Bruna Grizzetti, Bouraoui, and Aloe 2012; La Notte et al. 2017), and for policy scenario analysis (La Notte et al. 2017; Bouraoui et al. 2014; Leip et al. 2015; Malagó et al. 2019; Bruna Grizzetti et al. 2021). Despite their usefulness, this type of river basin models are seldom compiled as dedicated R packages, for example SPARROW has some scripts available to process information for and analysis of models results in the R environment.

The GREEN model application comprises several key steps, including data extraction and organization, data harmonization and integration, examination and validation of input data sets, model calibration and parameter selection, model run, and result visualization. All of these features are now integrated into an R package (**GREENeR**) that comprises functions to streamline the process, evaluate and visualize all the steps, thus strengthening the robustness of model application.

1.1 About water surface nutrients estimation with GREEN

GREEN (B. Grizzetti et al. 2005; Bruna Grizzetti, Bouraoui, and Aloe 2012; B. Grizzetti, Bouraoui, and De Marsily 2008; Bruna Grizzetti et al. 2021) is a conceptual model to assess total nitrogen TN and

total phosphorous TP from a region of interest (usually a river basin), accounting for both diffuse and point sources. The package allows the analysis of different scenarios of nutrient input in the region of interest, where “scenario” indicates a combination of annual time-series of inputs, such as nitrogen or phosphorus and the topological structure of the region. The model comprises several nutrient sources and pathways:

1. agricultural diffuse sources include nutrients from mineral fertilisers and manure application, nitrogen crop and soil fixation. These sources undergo retention in the land phase (basin retention that account for e.g. crop uptake and volatilization losses) before reaching the stream;
2. other diffuse emissions, such as from scattered dwellings (i.e., isolated houses and small agglomerations that are not connected to sewerage systems), and atmospheric nitrogen deposition (for nitrogen module) or background losses (for phosphorus module), are also reduced, e.g. due to soil processes, before reaching the stream network;
3. point sources consist of urban and industrial wastewater discharges that are discharged into surface waters directly.

Once in the river network, all nutrient loads are reduced by in-stream retention in rivers and lakes.

Basin retention of agriculture sources is a decay function proportional to the inverse of the total annual precipitation in the catchment. Conversely, river retention is a decay function proportional to the river length, considered as a proxy for water residence time. Finally, lake retention is simulated as a function of the lakes residence time and average depth.

The basin is divided into spatial subunits (called catchments), with a given area, a river reach, an inlet node, and an outlet node. The catchments are topologically connected from the headwaters to the outlet in a cascading sequence. The sequence of nutrient load accumulation through the stream network is defined by Shreve's order (Shreve 1966). Nutrient input from the different sources, basin and river retention are simulated in each catchment and routed through the river network. For each catchment i in the basin, the GREEN nutrient load L_i is estimated by the general equation:

$$L_{i,y} = (1 - Lret_i) \cdot (DSA_{i,y} \cdot (1 - Bret_{i,y}) + DSB_{i,y} + PS_{i,y} + U_{i,y}) \cdot (1 - Rret_i) \quad (1)$$

where $L_{i,y}$ is the nutrient load at the catchment outlet-node (ton/yr) in y year; and the other variables represent different sources and sinks of nutrients.

Sources of nutrients are:

- $DSA_{i,y}$. Annual nutrient diffuse sources on agricultural land in the catchment (ton/yr): mineral and manure fertilization, atmospheric nitrogen deposition, plant and soil fixation for TN; mineral and manure fertilization, and background losses for TP.
- $DSB_{i,y}$. Annual nutrient diffuse sources in the catchment related with scatter dwellings and atmospheric nitrogen deposition on non-agricultural land for TN ($DSB_{i,y} = 0.38 \cdot FF_{i,y} \cdot AtmN_{i,y} + sd_{coeff} \cdot SdN_{i,y}$, where $FF_{i,y}$ is the fraction of non-agricultural land cover in the catchment, and $AtmN_{i,y}$ is the annual atmospheric nitrogen deposition on the catchment (ton/yr)); nutrient diffuse sources in the catchment related with scatter dwellings and background losses on non-agricultural areas for TP ($DSB_{i,y} = sd_{coeff} \cdot SdP_{i,y}$).
- $PS_{i,y}$. Nutrient point sources in the catchment (ton/yr).
- $U_{i,y}$. Nutrient load from upstream catchments (ton/yr).

Sinks of nutrients are:

- $Lret_i$ denotes the lake retention (fraction) of the i catchment. $Lret$ is currently defined according to Kronvang et al. (2004), but limited to a 10% maximum reduction:

$$Lret = \max \left(0.1, 1 - \frac{1}{1 + (dref/z) \cdot RT} \right) \quad (2)$$

where z represents the average lake depth (m); RT is the hydraulic residence time (yr); and $dref$ denotes a nutrient-related coefficient ($dref = 7.3$ for TN and $dref = 26$ for TP).

- $Bret_{i,y}$ is the fraction of basin retention of the i catchment in y year:

$$Bret_{i,y} = 1 - \exp(-alphap \cdot NrmInvRain_{i,y}) \quad (3)$$

where $NrmInvRain_{i,y}$ is the inverse of annual precipitation (mm) of the i catchment in y year, normalized by its maximum (Frank and Todeschini 1994). To keep basin retention coefficients comparable

across regions, the minimum precipitation $minPrec$ is set to 50 mm/y, and thus $NrmInvRain_{i,y}$ is defined as

$$NrmInvRain_{i,y} = \frac{1 / \max(50, precipitation_{i,y})}{(1/minPrec)}$$

- $Rret_i$ is the fraction of river retention of the i catchment:

$$Rret_i = 1 - \exp(-alpha_L \cdot NrmLengthKm_i) \quad (4)$$

where $NrmLengthKm_i$ is the length (km) of the catchment reach, normalized by the maximum in the dataset:

$$NrmLengthKm_i = \frac{\text{catchment length } i \text{ reach, in km}}{\max(\text{Reach length in the region, in km})}$$

Since the maximum reach length depends on the region and its subdivision of network reaches, the calibrated coefficient $alpha_L$ cannot be compared across regions that adopt different discretization. Note that basin retention varies from year to year according to annual precipitation, whereas the fraction of river and lake retention for a given catchment is constant in time.

Equation (1) is applied sequentially from the most upstream nodes to the basin outlet. The model parameters are:

1. the basin retention coefficient ($alphap$), which together with annual precipitation regulates nutrient retention of diffuse agricultural sources (Equation (3));
2. the river retention coefficient ($alpha_L$: Equation (4)), which regulates the river retention per river km.
3. the fraction of domestic diffuse sources that reaches the stream network (sd_coeff).

2 About the GREENeR package

2.1 Package organization

This work presents a new efficient and enhanced implementation of the model GREEN developed as an R package named **GREENeR**. **GREENeR** is developed in the R statistical software and provides tools and methods for applying GREEN to an area of interest and assessing annual time series of nutrient loads in a river network and at the basin outlet, plus contributions of nutrient sources to these loads. Some of the key features of the package comprise: a) functions for the creation of scenarios from data sources; b) computational efficiency; c) parallel-capable ; d) an extended suite of fine tuning options to control model calibration; e) built-in parameter sensitivity analysis; and f) functions to perform customised post-process analyses.

The functions of **GREENeR** package are arranged in three groups: i) functions to perform graphical summaries of model inputs; ii) functions to perform model parameters calibration and sensitivity analysis; iii) functions to compute, analyze and visualize through graphs and maps the model outputs, i.e. total loads and contributions by source. The package supports parallel processing, which helps reduce the computational load in handling large basins.

The time-series can represent either historic (current or past) conditions that can be associated to observations for model calibration, or hypothetical conditions foreseen under theoretical changes (e.g. to forecast results of nutrient management plans, or under climatic change).

A scenario contains:

1. The geospatial geometry of the catchments of the region, which currently is defined according to the Catchment Characterisation and Modelling Database for European Rivers v2 (CCM2) (De Jager and Vogt 2007).
2. The annual time series (1990-2018) of nutrient inputs per catchment.
3. Additional catchment information (annual precipitation, annual forest fraction, lake retention, reach length).
4. Observed total load (TN or TP) from monitoring stations.

Two historical scenarios, one for TN and one for TP, of the Lay basin (France) are provided with the **GREENeR** package. The Lay basin has an area of 1971 km², and is divided into 189 CCM2

catchments. The mean catchment area is 10.4 km². Nutrient observations comprised 22 TN entries from six monitoring stations and 58 TP data entries from eight monitoring stations (data from WISE Waterbase (EEA 2021)).

Further, in this article we present examples drawn for the TN historic scenario of the transboundary Vistula (Wisla) river basin, one of the largest river basins of Europe. The Vistula scenario comprises 15465 CCM2 catchments that compose the 193.894 km² river basin, TN inputs for 1990-2018, as well as 1364 observed TN loads from 412 monitoring stations. The TN input dataset is part of the larger dataset created to assess nutrient concentration in the European freshwaters (Vigiak et al. 2023).

2.2 Key functions in the GREENeR package

A brief overview of the package and its main functions is given in Figure 1.

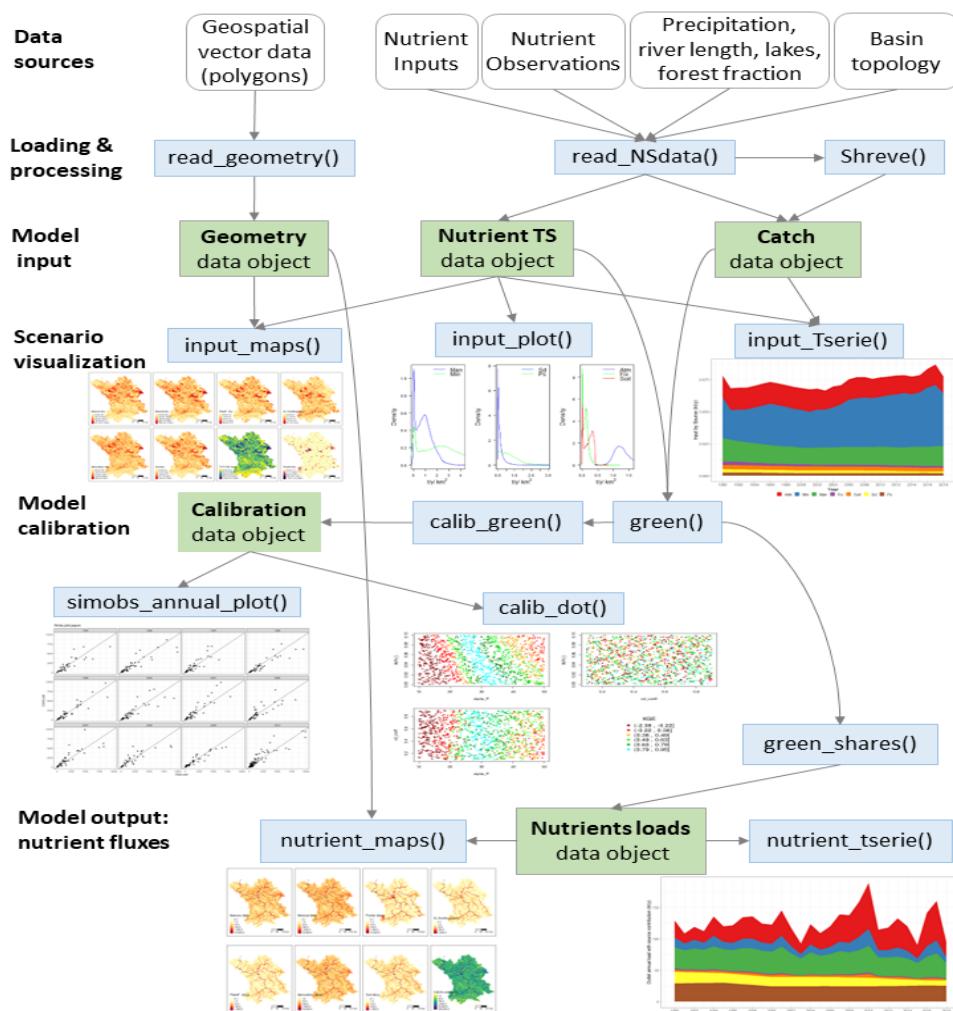


Figure 1: Schematic diagram of the procedure of GREENeR, including functions, data inputs and outputs, to estimate the nutrient loads. The green boxes represent data objects and the blue boxes represent the functions.

The key functions included in the package are:

- `read_geometry()`: imports the geospatial vector format (shapefile or ESRI shapefile) with the spatial information of the catchments in R.
- `read_NSdata()`: imports the annual time series of nutrient inputs per catchment and type of nutrient source (manure, mineral fertilization, point sources, scatter dwellings), plus the forest

fraction, and the observed annual nutrient loads from monitoring station data. The original data are stored in several comma-separated tables (CSV files).

- `input_maps()`: creates a map showing the mean nutrient load input by source.
- `input_plot()`: creates either a grouped barplot representing the average input load by source for the whole basin, or three density plots showing the distribution of nutrient sources.
- `input_Tserie()`: creates a time series plot showing basin inputs by source.
- `shreve()`: returns the Shreve order of the catchments, a useful indicator of stream size, discharge, and drainage area (Strahler 1957) calculated based on the sum of the orders of up of upstream tributaries (Shreve 1966). Commercial GIS software usually provides a Shreve calculation function, but in other software, it is harder to find. In GREEN, the Shreve's order defines the cascade of upstream-downstream catchments.
- `calib_green()`: conducts sensitivity analysis of model calibration utilizing a Latin Hypercube Sampling (LHS) scheme (Manache and Melching 2004) for three model parameters. It evaluates model performance by calculating several “goodness-of-fit” (GoF) metrics (Refsgaard and Henriksen 2004) against available observations during the specified simulation period (years).
- `select_params()`: extracts the best parameter set according to a selected GoF metric from the object generated by the calibration function, `calib_green()`.
- `calib_boxplot()`: creates a figure of boxplots that show the relationships between the best parameter sets determined by the specific GoF metric in each boxplot title, and six other commonly used hydrological metrics (Althoff and Rodrigues 2021; Mauricio Zambrano-Bigiarini 2014). In the lower panel, the figure also highlights the distribution of model parameters; the value of the best parameter set is marked as a red dot in each boxplot.
- `region_nut_balance()`: runs the GREEN model with the selected parameter set, and returns the mean annual mass balance of nutrient fluxes for the whole simulation period of the region. The results of this function can be visualized using a Sankey diagram via the `N4_sankey()` function.
- `green()`: runs the GREEN model with the selected parameter set and returns nutrient load time series for the simulation period. It generates less information than the `green_shares()` function, but its execution is faster, so it is used as a function for calibration iterations and is embedded in the `calib_green()` function.
- `green_shares()`: runs the GREEN model with the selected parameter set and returns time series of nutrient loads and the contributions of each nutrient source in the simulation period. The results of the model can be examined by `nutrient_tserie()` and `nutrient_maps()` functions.
- `scatter_plot()`: generates dot plots correlating parameters realizations in the calibration data with GoF metric, visualizing the impact of each parameter on model outcomes. The plot vary based on selected GoF metric from `green_calib()` function.
- `simobs_annual_plot()`: generates scatter plots comparing model load predictions (`PredictLoad`) with observations (`ObsLoad`) for each year of the stimulation period.

2.3 Input data requirements

GREENeR requires information on the nutrient inputs, the topology, and the geospatial geometry of the region of interest. The spatio-temporal input data must include all nutrient source fields, and differs in the two nutrient scenarios (TN or TP). In the case of TN (e.g. Figure 2), fields are (Equation 5) in Appendix 1):

- *Atmospheric*: Annual amount of atmospheric nitrogen deposition (ton/yr).
- *Mineral*: Annual amount of nitrogen from mineral fertilisers (ton/yr).
- *Manure*: Annual amount of nitrogen in manure fertilisers (ton/yr).
- *Fix*: Annual amount of nitrogen fixation by leguminous crops and fodder (ton/yr).
- *Soil*: Annual amount of nitrogen fixation in soils (ton/yr).
- *Sc.Dwellings*: Nitrogen input from scattered dwellings (ton/yr).
- *PointS*: Nitrogen input from point sources (ton/yr).

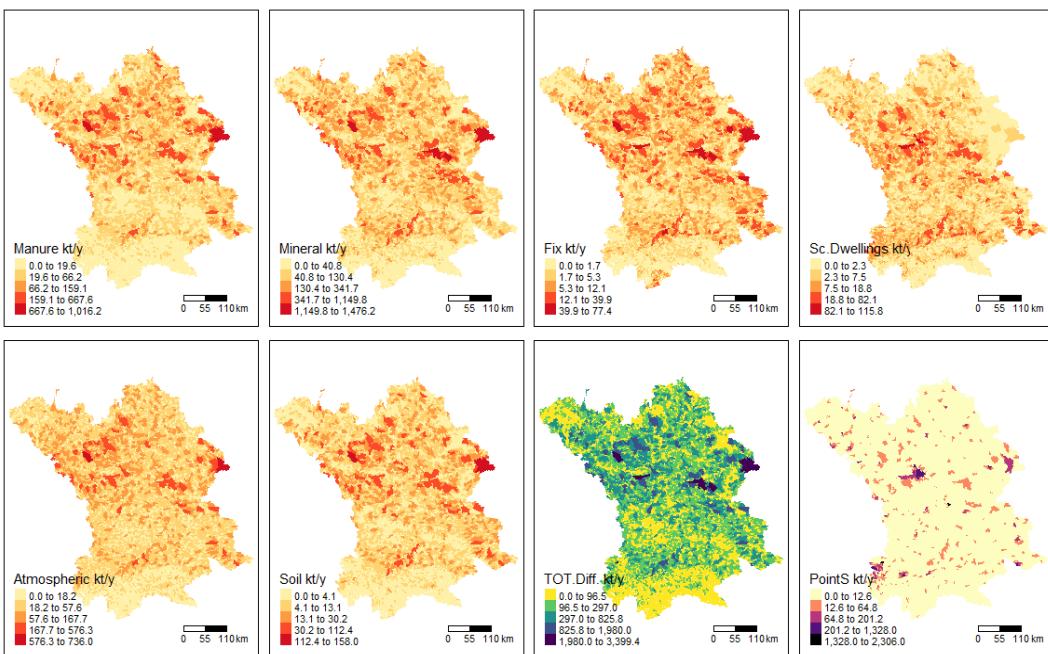


Figure 2: Maps showing the mean annual TN inputs in the Vistula river basin in 1990–2018. The figure was generated with GREENeR `input_maps()` function. TOT.Diff = sum of diffuse inputs.

In the case of TP, fields are (Equation (6) in Appendix 1):

- *Background*: annual amount of phosphorus from background losses (ton/yr).
- *Mineral*: annual amount of phosphorus from mineral fertilisers (ton/yr).
- *Manure*: annual amount of phosphorus in manure fertilisers (ton/yr).
- *Sc.Dwellings*: phosphorus input from scattered dwellings (ton/yr).
- *PointS*: phosphorus input from point sources (ton/yr).

European datasets for 1990–2018 generated to assess historic and current nutrient fluxes (Vigiak et al. 2023) are available upon request. The model and the package are compatible with an external dataset and can be customized with local information as long as the data structure is respected.

In both nutrient cases, **GREENeR** needs additional annual catchment information:

- *ForestFraction*: annual fraction (0–1) of non-agricultural land cover area in the catchment (*FF* in Equations (5) and (6), in the Appendix 1).
- *Rain*: annual precipitation (mm, Equation (3)).
- *LakeFrRet*: lake retention fraction (0–1) (*Lret* in Equation (1), in the Appendix 1). Note, this differs for TN or TP scenarios (Equation (2)). At European scale, average lake depth and hydraulic residence time can be obtained from HydroLAKES database (<https://www.hydrosheids.org/pages/hydrolakes>, Messager et al. (2016)).
- *Length*: is the length (km) of the catchment reach (Equation (4)).

The catchment topology outlines the hydrological network configuration within the region. Each catchment must have a unique numerical identifier (HydroID)), to establish the network structure via a source HydroID and destination HydroID table. It is important to note that any outlet of the basin will be given as destination HydroID identifier “-1”. Complementing the topology of the hydrologic network, the length of catchment reach should also be included (Equation (4)).

Finally, in order to calibrate the model, it is necessary to have some observed nutrient loads from monitoring stations associated to any catchment of the region. Ultimately, the quality, size, and spatial distribution of observed loads determine the robustness of the calibration process. In Europe, a large dataset of annual concentrations is publicly available (WISE Waterbase, EEA 2021), but annual flow must be derived from other sources (Bruna Grizzetti et al. 2021; Vigiak et al. 2023).

2.4 Performance and memory use

The **GREENeR** package performance and memory usage depends on the size of the region (number of catchments) and the number of years of the simulation. As a reference, the Danube basin (i.e. the largest European basin, with 138013 catchments) TN dataset for 1990-2018 requires approximately 389 Mb of memory to store the annual data. Table 1 shows the memory occupied by the scenarios for 6 European basins. In addition, it shows the computation times required in the execution of some key functions of the package. All executions were conducted on two computer configurations:

- CPU1: Desktop running window 10 with one Intel(R) Core i5-8259U CPU (4 cores, two threads per core) CPU at 2.30 GHz and 16 GB of RAM memory
- CPU2: Workstation running Linux with one AMD EPYC 7352 (24 cores, two threads per core) CPU at 2.3 GHz and 128 GB of RAM memory

The time required for the execution of the different functions increases linearly with the number of catchments (or Shreve order) of the region (Table 1). The `calib_green()` function is the most computationally demanding. However, the parallel implementation of this function drastically reduces the time required for this process. Calibration can be carried out in about 6 hours for practical implementation in large basins.

Table 1: Computational requirements to run some **GREENeR** functions in six European basins scenarios of 29 years. The columns Shreve order, Area and Number of catchments characterize the size of each basin. Mem is the amount of memory occupied by the GREEN model scenario generated by **GREENeR** package. The `green()`, `green_shares()` and `calib_green()` columns show the computation time required to execute the corresponding **GREENeR** functions for each scenarios under CPU1 and CPU2 computer configurations. All runs of `calib_green()` have been performed with 200 iterations. The reported times are the average of 5 runs of each process.

Basin name	Shreve order	Area (Km ²)	Number of catchments	Mem (MB)	green() (sec)		green_shares() (sec)		calib_green() (sec)	
					CPU1	CPU2	CPU1	CPU2	CPU1	CPU2
Lay	95	1971	189	0.5	6	4	15	12	316	31
Miño	2803	16985	5572	15.7	57	45	147	90	4110	394
Seine	2902	75989	5793	16.3	77	55	177	106	6320	452
Ebro	9351	85611	18568	52.3	160	123	425	224	10876	995
Vistula	7757	193894	15465	43.6	132	100	314	187	9004	814
Danube	69505	802032	138013	388.6	1120	766	2855	1108	94682	5638

3 Estimating nutrient loads using the **GREENeR** package

The entire procedure is summarized on Figure 1. Once the input data have been uploaded, the region scenario (Nutrient data and Catch data) is generated. The `calib_green()` function explores the parameter set ranges with LHC scheme, calculating GoF of parameter sets. Sensitivity analysis of its results is conducted to determine the best parameter set. Finally, the `green_shares()` function is used to estimate the nutrient loads and source apportionment (i.e. contribution to loads by source), per year and per catchment.

3.1 Scenario preparation

Assembling input data for running the GREEN model is time consuming. To facilitate the process, the `read_NSdata()` function assembles and organises annual information to generate a list of two objects: Nutrient Time Series Data Object and Catch Data Object of GREEN scenario. It needs four CSV files with specific data, namely:

- Time-series of annual nutrient inputs per catchment.
- Time-series of annual observed loads at monitoring stations.
- Basin topology and lake properties.
- Other: precipitation, forest fraction, reach length.

The spatial identifiers (HydroID) and temporal (year) units must be coherent in all the files. Besides organizing the input data, the `read_NSdata()` function also calculates the Shreve order of each catchment, normalises the precipitation (calculating $NrmInvRain_{i,y}$) and the reach length ($NrmLengthKm_i$).

```

csv_path <- "data/csv/"
scen <- read_NSdata(csv_path,"nutrients.csv","monitoring.csv", "forestFr.csv",
                     "precipitation.csv","topology.csv", "lakeProperties.csv",
                     "length.csv")
nutrient <- scen[[1]]
catch   <- scen[[2]]

```

The shreve() function can calculate the Shreve order independently based on the topology of the basin:

```
shreve_order <- shreve(catch)
```

Finally, the geospatial geometry of the catchments should be uploaded to enable the visualisation functionalities in map form. The read_geometry() function loads the geometry information file, i.e. a geospatial vector with the geometry of the catchment region:

```
geometry <- read_geometry(file = "data/shapes/Wisla.shp")
```

The geo-reference information, geometry and attributes of the spatial entities must be in a shapefile format (.shp), editable with ArcGIS or similar software. The identifiers of each catchment (HydroID) must be consistent with those in the CSV files of the scenario dataset.

Once the scenario has been generated, the library includes functions to examine the nutrient sources in a basin, and explore their distribution over time or in space. The input_plot() function provides annual average nutrient loads for the whole period and density plots of nutrient loads. The input_Tserie() function allows to examine the temporal evolution of the inputs, whereas the input_maps() function generates maps of nutrient inputs distribution in the basin (Figure 2).

```
input_maps(nutrient, catch, geometry, "Wisla", "gr1")
```

3.2 Calibration procedure and sensitivity analysis

The calibration process is essential to find a suitable set of parameters for the model. In **GREENeR** package, the core function calib_green() enables to run a parameter sensitivity analysis and concurrently assessing the model performance according to several GoF metrics.

Sensitivity Analysis (SA) investigates how the variation in the output of a numerical model can be attributed to variations of its input factors. SA aims at identifying the most influential inputs or parameters, and quantifying how much they contribute to the variability/uncertainty of the model outputs. SA provides information on how much of the output variance is controlled by each parameter, or combination of parameters. SA is increasingly being used in environmental modelling for a variety of purposes, including uncertainty assessment, model calibration and diagnostic evaluation, dominant control analysis, and robust decision-making (Pianosi et al. 2016; Saltelli, Annoni, et al. 2011; Butler et al. 2014; Norton 2015; Mannina et al. 2018). This is achieved by running the model for many samples of the parameter space to determine their impact on the model outputs. SA allows identification of the parameters and input variables that strongly influence the model response (model output). Conversely, it may be of interest to the modeller to verify that although some model parameters may not be very well established they do not significantly contribute to output uncertainty. Saltelli, Tarantola, and Campolongo (2000) single out three main classes of SA methods: screening, local, and global methods. Local sensitivity analysis methods focus on assessing the impact of small variations in the input values of a model on the output results. Global sensitivity methods seek to understand how variations in the full range of input values affect the results. Puy et al. (2022) provide a comprehensive overview of several R packages for performing global sensitivity analysis.

Screening methods are economical and qualitative methods. Only screening methods are included in this implementation of the library as they provide a quick assessment of the relative importance of variables. This is particularly useful in exploratory studies and in cases where an initial subset of relevant variables needs to be identified.

Adequate sampling of parameter space is very important in model calibration. Several studies of uncertainty analysis in water resources problems (Melching and Bauwens 2001) concluded that Monte Carlo simulation and Latin Hypercube Sampling (LHS) (McKay, Beckman, and Conover 1979; Carnell 2012) methods are very powerful, robust, and flexible. Other approaches are possible (Mauricio Zambrano-Bigiarini 2014), but the LHS has the advantage that it is easily parallelisable, it explores the full range of parameter sets, it does not direct the search depending on the values of previous iterations like other optimisation methods (gradients, genetic algorithms, etc.), and is therefore independent of

the GoF metric. This allows performing a posteriori sensitivity analysis for several metrics without having to repeat the process.

The `calib_green()` function computes 15 GoF metrics (Althoff and Rodrigues 2021; Mauricio Zambrano-Bigiarini 2014) (listed in [Appendix 2](#)), and returns an object with all parameter sets generated by the LHS and the corresponding GoF scores. Running `calib_green()` requires the following settings:

1. The expected ranges for each parameter. The ranges are defined by two vectors of three values, one for the lower limits and one for the upper limits of the three parameters. The values correspond to each model parameter in sequence: α_{P} (Equation (3)), α_{L} (Equation (4)), and sd_{coeff} (Equations (5) and (6)).
2. The number of iterations to be performed during the calibration process. The higher the number of iterations, the more likely it is to achieve good results, but the longer the computation time. Depending on the basin and parameter range width, it is recommended to run at least 200 iterations to have enough information to continue the calibration process.
3. The years to be included in the calibration process.

```
n_iter <- 2000
low <- c(10, 0.000, 0.1)
upp <- c(50, 0.1, 0.9)
years <- c(1990:2018)

calibration <- calib_green(nutrient, catch, n_iter, low, upp, years)
```

The calibration process is automatically parallelised by the package and uses all computer cores except one. The computation time depends on the computer, the number of catchments in the basin, and the number of iterations ([Table 1](#)). The `calib_green()` function returns a data frame with a row for each iteration with parameter values and the resulting 15 GoF metric scores. An example of four parameter sets is shown below (cut to the first five GoF metrics).

	α_P	α_L	sd_{coeff}	NSE	mNSE	rNSE	KGE	PBIAS %
#> s1	39.31203	0.07628558	0.6325056	0.5623	0.6452	0.9595	0.2512	-44.1
#> s2	35.06049	0.05510985	0.1577470	0.6733	0.6948	0.9563	0.3763	-36.3
#> s3	28.34947	0.04418381	0.7209566	0.8690	0.7737	0.8998	0.7926	-1.7
#> s4	40.73533	0.01351723	0.6195734	0.8370	0.7734	0.9607	0.6292	-22.6

The selection of the appropriate GoF metric(s) for calibration and evaluation of hydrological models can be challenging even for very experienced hydrologists. Choosing the right GoF metric for model calibration largely depends on the overall study scope, which defines the main interest (e.g. high or low load, upper or lower catchment area), and on the available observation dataset (size and quality) (Kumarasamy and Belmont 2018). Automated calibration typically relies, often exclusively, on a single GoF metric, with the Nash-Sutcliffe efficiency (NSE) been the most frequently used metric in hydrological models (Hoshin V. Gupta et al. 2009; Westerberg et al. 2011; Wöhling, Samaniego, and Kumar 2013). As described in Singh and Frevert (2002), a single criterion is inherently predisposed to bias towards certain components of the data time series. Automated procedures may be improved by using more than one criteria as discussed by Hoshin Vijai Gupta, Sorooshian, and Yapo (1998). Although automation can help the calibration process become more objective, efficient and practical, it is not a substitute for expert hydrologic intuition and understanding. Whether automated or manual calibration is used, a common approach is to adjust the parameters that display the highest sensitivity (Madser 2003; Doherty and Skahill 2006).

GREENeR includes several functions to assist in selecting the best parameter set. The `calib_boxplot()` function shows ([Figure 3](#)) relationships between best parameter sets chosen according to one GoF parameter (title of each boxplot) in relation to six most frequently used metrics. Additionally, in the lower panel, the figure shows the distribution of model parameters in the most performing subset of parameter sets. The best parameter set according to each GoF metrics is marked as a red dot in each boxplot ([Figure 3](#)).

```
calib_boxplot(calibration, rate_bs = 5)
```

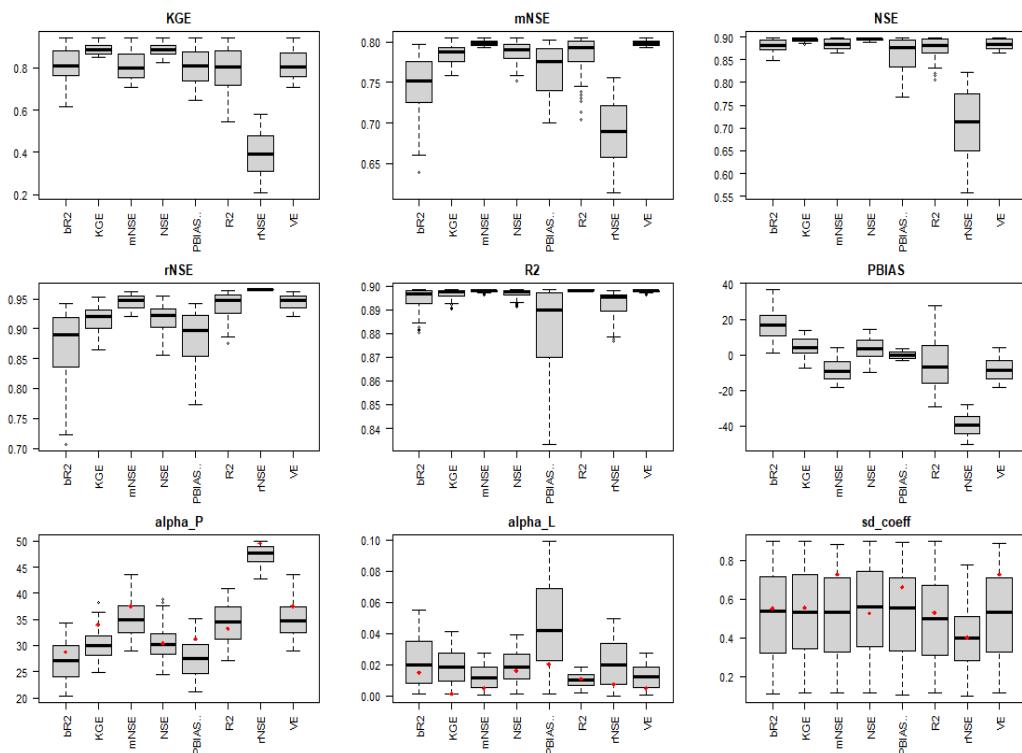


Figure 3: Calibration results for the Vistula river basin (TN Scenario). The figure displays the output of the `calib_boxplot()` function, and compare several GoF metrics (bR2, KGE, mNSE, NSE, PBIAS, R2, rNSE, VE) with each other and with the model parameters distributions. The top two rows show boxplots of each metric (specified in the title) with others (in the x labels). The lowest row shows the parameter value distributions for the top 5% (or another threshold indicated in `rate_bs` parameter of the function) parameter sets ranked according to the GoF metrics in the x label. The red dots represent the best parameter values for each boxplot.

The `select_params()` function extracts the parameter set that scored the best GoF metric of choice from the calibration result object.

```
best_params <- select_params(calibration, param = "NSE")

alpha_P <- best_params$alpha_P
alpha_L <- best_params$alpha_L
sd_coeff <- best_params$sd_coeff
```

It is not recommended to use the parameters extracted by the `select_params()` function without having performed an analysis of all the calibration results. Instead, alternative parameter sets (according to different GoF) should be compared before making the final selection.

Screening and SA of model parameters can be done via the `scatter_plot()` and `calib_dot()` functions of **GREENeR** package. The `scatter_plot()` function shows all parameter realizations in the LHS dataset against a selected GoF metric to visualize the influence of each parameter on the metric scores. The result depends on the GoF metric of choice (any metric calculated with `calib_green()` can be selected) (Figure 4).

```
scatter_plot(calibration, param = "R2")
```

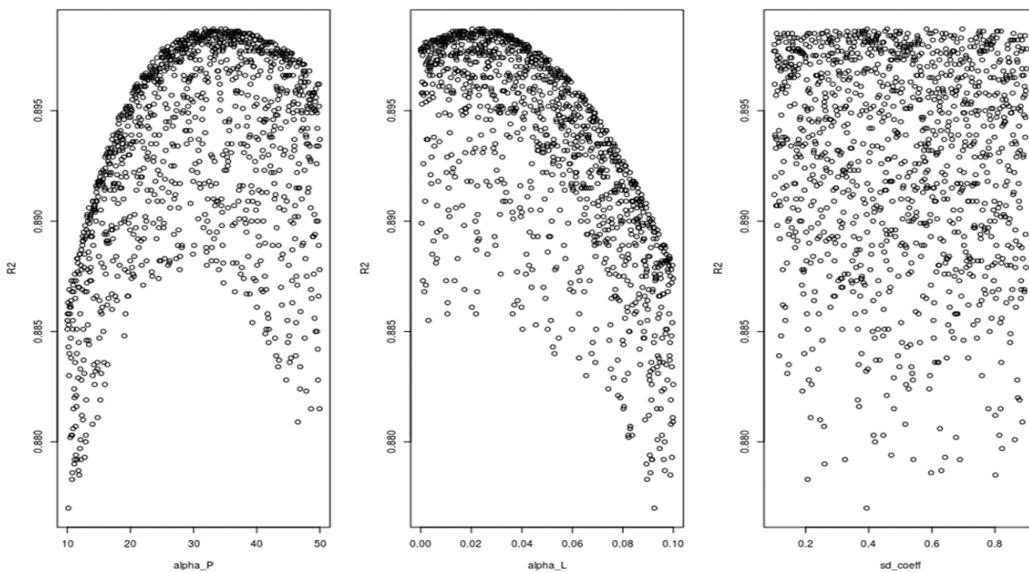


Figure 4: Scatter plots of model parameters against R2 metric generated by the scatter_plot() function for the Vistula river basin (TN).

Figure 4 example shows that the highest R2 are achieved for values of the parameter α_{P} around 33.3 and for the parameter α_{L} around 0.025, whereas the model is insensitive to the sd_{coeff} parameter. Further, scatter_plot() function helps check if the parameter ranges defined in the search were suitable, or if the number of iterations was sufficient. In Figure 4, the best values of the parameters α_{P} and α_{L} are in the central part of the plots, thus the search parameter ranges were appropriate, and the distribution of dots is sufficient to visualize the effect of each parameter, indicating that the number of iterations was adequate.

The calib_dot() function shows the distribution of parameters in relation to each other for a chosen GoF metric, and highlights potential correlations between parameters (Figure 5).

```
calib_dot(calibration, param = "KGE")
```

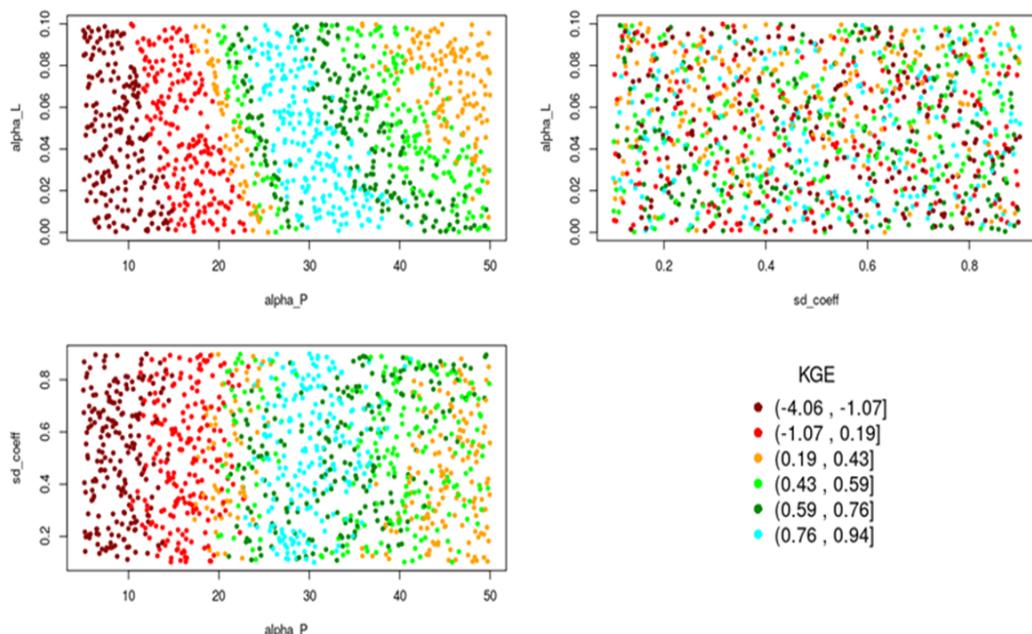


Figure 5: Dot plots of parameter pairs for the Kling-Gupta efficiency (KGE) generated by the calib_dot() function for the Vistula river basin (TN).

In Figure 5, the Kling-Gupta Efficiency metric (KGE, Althoff and Rodrigues 2021) shows interactions between α_{ph} and α_{L} , i.e. a higher α_{ph} should be associated to a lower α_{L} for achieving similar KGE. The best α_{ph} values identified for both R2 and KGE metrics are similar, whereas α_{L} values vary considerably (see also Figure 3).

The GREENeR package includes two useful model calibration functions: `compare_calib()` and `simobs_annual_plot()`. `compare_calib()` shows a scatter plot that compares observed data points with corresponding values predicted by two parameter sets. Figure 6 displays a plot generated by the `compare_calib()` for the Vistula (TN) basin scenario when using the best parameter sets based on NSE and rNSE metrics. The plot reveals that when rNSE parameter set result in an underestimation of loads in the upper range.

```
metrics <- c("NSE", "rNSE")
compare_calib(nutrient, catch, alpha_p1 = 31, alpha_l1 = 0.02, sd_coef1 = 0.6,
              alpha_p2 = 50, alpha_l2 = 0.01, sd_coef2 = 0.6, years = c(1990:2018),
              name_basin = "Wisla", metrics)
```

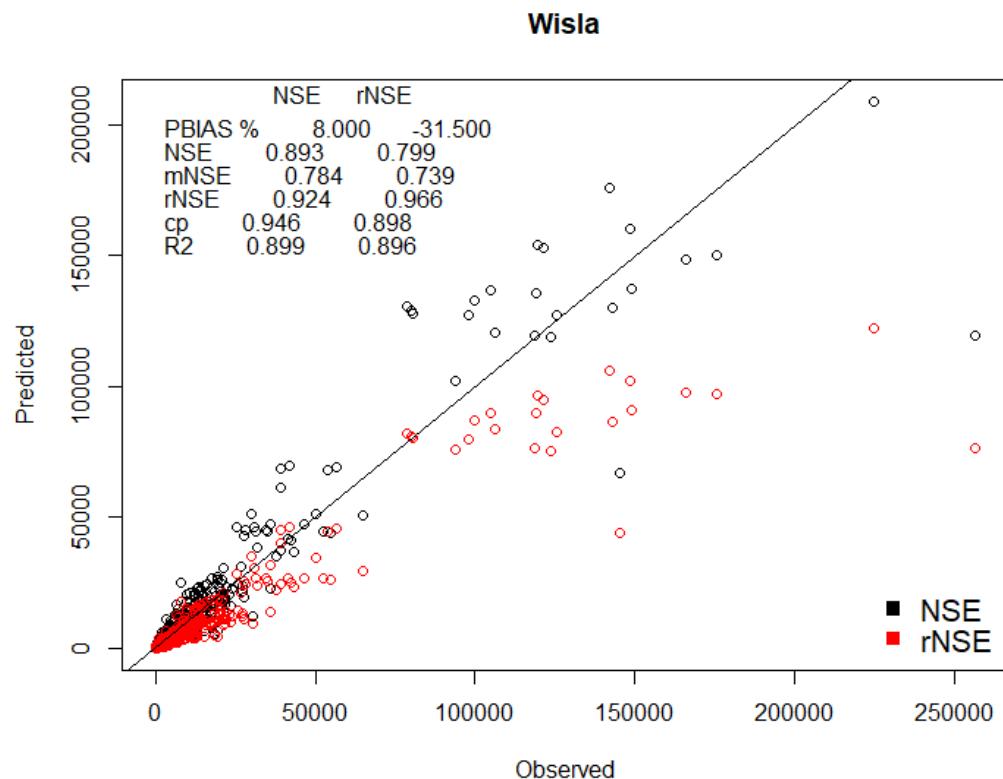


Figure 6: Result of `compare_calib()` for the Vistula river basin (TN), comparing the best set of parameters selected according to the NSE and rNSE metrics.

The `simobs_annual_plot()` function compares observed and predicted values over the years (Figure 7), which identifies erroneous model performance or data distribution problems over time.

```
year_range <- c(1994, 1996:2001, 2006:2009, 2012)
simobs_annual_plot(nutrient, catch, alpha_P, alpha_L, sd_coeff, year_range,
                    name_basin = "Wisla", max_value = 10000)
```

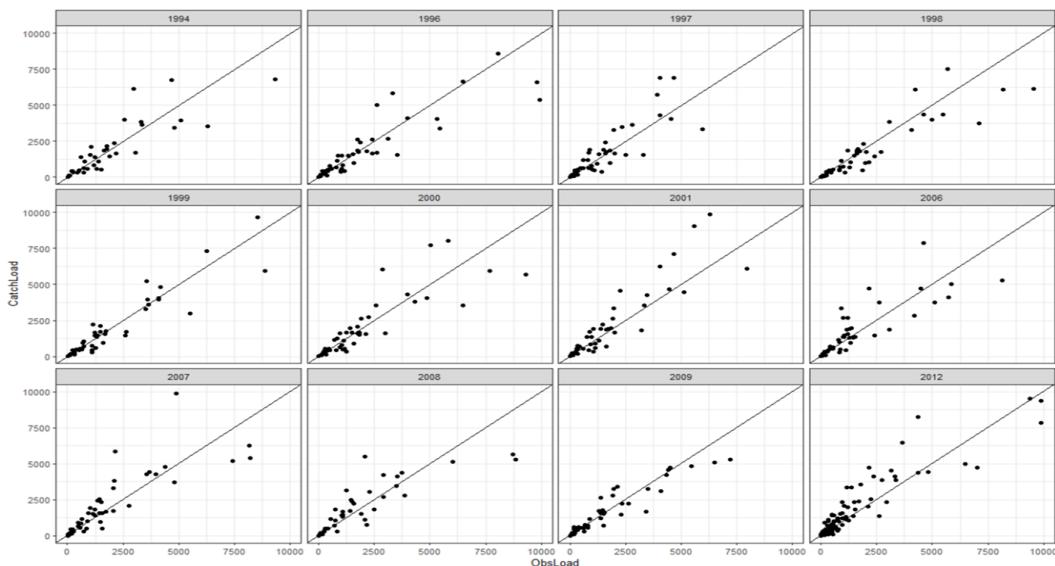


Figure 7: Result of the simobs_annual_plot() for the Vistula river basin (TN), with the parameter set selected for highest NSE and limiting the plots to the interval between 0 and 10000

3.3 Estimation of catchment nutrient loads and contribution in the basin

Once the most appropriate parameter set has been selected, it is possible to run the model to estimate nutrient loads in the region. The green_shares() function runs GREEN with the selected parameter set and returns catchment nutrient loads and the contributions by source in the simulation period.

The nutrient_tserie() function shows the temporal evolution of the total load at the basin outlet in the simulation period with contributions by sources (Figure 8). Other options of the function show nutrient loads in different zones of the basin (upper, central and lower part).

```
nutrient_load <- green_shares(nutrient, catch, alpha_P, alpha_L, sd_coeff, years)
nutrient_tserie(nutrient_load, geometry, "Wisla basin", "gr3")
```

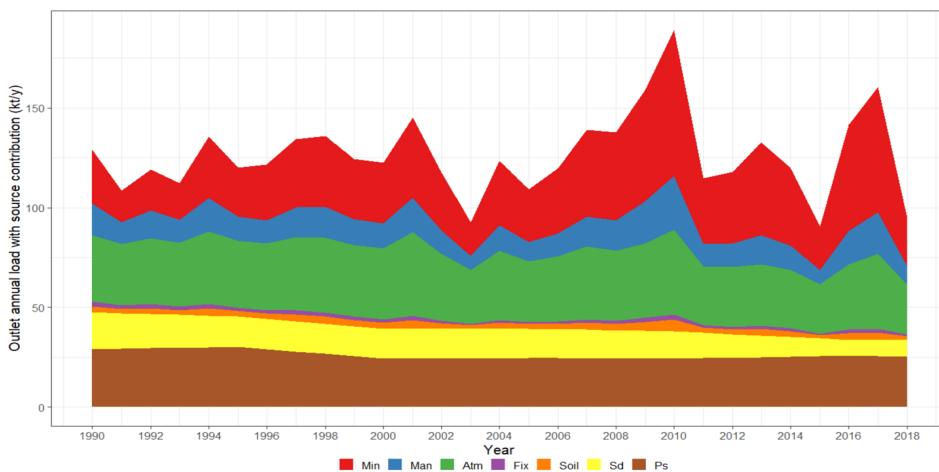


Figure 8: TN load at the Vistula river basin outlet from 1990 to 2018. Colors indicate the contribution of different sources (Min = mineral, Man = manure, Atm = atmospheric deposition, Fix = crop fixation, Soil = soil fixation, Sd = scattered dwellings and Ps = point sources).

The nutrient_maps() function uses the object returned by green_shares() to generate maps of the distribution of nutrient loads by source for a given year or as the mean for several years (Figure 9). The results are shown in logarithmic scale to improve the visualization as nutrient loads in a region may vary over several orders of magnitude. Alternative options of the function show the total load at the outlets of the catchment or the specific loads (i.e. load per catchment area; in $\text{kt}/\text{km}^2/\text{y}$).

```
map_title <- "Output Loads for the Vistula basin"
nutrient_maps(nutrient_load, geometry, map_title, "gr1", legend_position = 1)
```

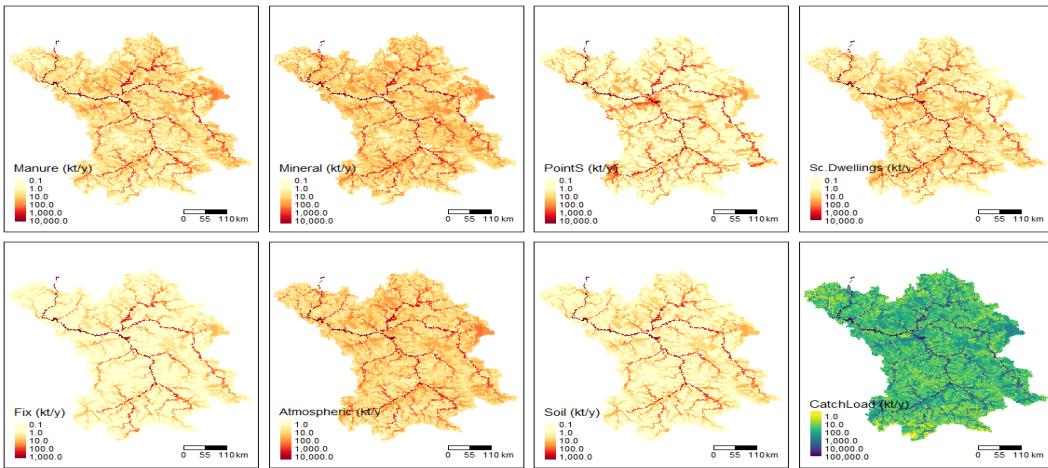


Figure 9: Maps of TN loads in the Vistula river basin by different sources and in total (the sum of all other maps, bottom right), in logarithm scale as generated with `nutrient_maps()` function.

3.4 Estimation of nutrient fluxes and sources contribution in the basin

The `region_nut_balance()` function runs GREEN with the selected parameter set, and returns the nutrient mass balance and fluxes of the region (mean value for the simulation period). The results of this function can be visualized in a Sankey diagram with the function `N4_sankey()` (Figure 10).

```
nut_bal <- region_nut_balance(nutrient, catch, alpha_P, alpha_L, sd_coeff,
                                years)
sank <- N4_sankey(nut_bal)
```

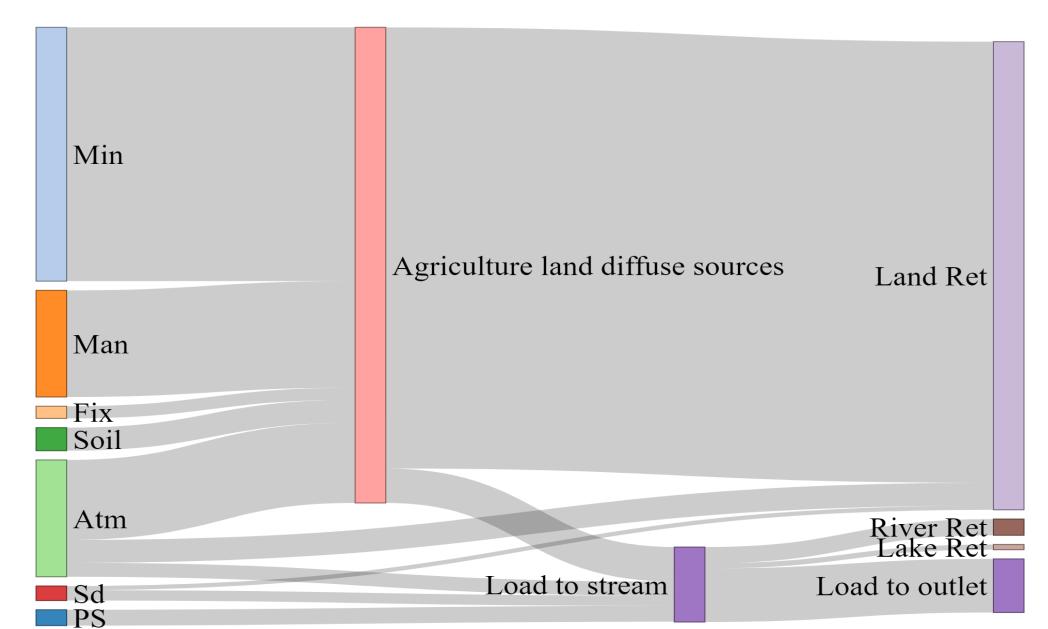


Figure 10: Sankey plot for the Vistula river basin for TN scenario, average for period 1990-2018. The plot represents nitrogen input sources on the left (Min=mineral, Man=manure, Atm=atmospheric deposition, Fix=crop fixation, Soil=soil fixation, Sd=scattered dwellings and Ps=point sources), and nitrogen sinks (land, river and lake retention) and outlet discharge (load to outlet) on the right. The bars in the middle visualize nitrogen agricultural diffuse sources and loads to the stream network.

4 Conclusion

The GREENeR package provides several functions to assess nutrient pressures in fresh and coastal waters based on the GREEN model (Bruna Grizzetti, Bouraoui, and Aloe 2012; Bruna Grizzetti et al. 2021). It allows assessing annual nutrient loads and concentrations throughout the river network and at river outlets, as well as the contributions of diffuse and point sources to the total load. GREENeR includes functions to route nutrient sources in a region, considering different pathways and the hydrological structure of the river network. The package is enriched by functions that aid selecting the set of parameters that best suits the study scope. Several functions assist in the preparation of scenarios by assembling input data from the appropriate sources, and in visualising inputs and nutrient fluxes in space and time. The version of the GREEN model implemented in the package is computationally efficient and includes parallel running capabilities for the calibration process, greatly reducing the time required for large basins or regional applications, e.g. at continental scale as in (Vigiak et al. 2023).

5 Acknowledgements

This work has been partially supported by grant XMIDAS, ref. PID2021-122640OB-I00, funded by the Spanish Ministry of Science and Innovation.

6 Appendix

6.1 Appendix 1

Diffuse and point sources are defined differently for each nutrient module, i.e. nitrogen or phosphorus. Equation (5) formulates the general GREEN function stated in equation (1) in the case of nitrogen. In GREEN nitrogen model, the total nitrogen load L_i of catchment i is estimated as:

$$L_i = (1 - Lret_i) \cdot ((MinN_i + ManN_i + FixN_i + SoilN_i + (1 - FF_i) \cdot AtmN_i) \cdot (1 - Bret_i) + 0.38 \cdot FF_i \cdot AtmN_i + sd_{coeff} \cdot SdN_i + PsN_i + U_i) \cdot (1 - Rret_i) \quad (5)$$

where $MinN_i$ is the annual amount of nitrogen from mineral fertilisers (ton/yr); $ManN_i$ is the annual amount of nitrogen in manure fertilisers (ton/yr); $FixN_i$ is the annual amount of nitrogen fixation by leguminous crops and fodder (ton/yr); $SoilN_i$ is the annual amount of nitrogen fixation by bacteria in soils (ton/yr); $AtmN_i$ is the annual nitrogen deposition from atmosphere (ton/yr); FF_i is the non-agricultural land cover in the catchment (fraction); SdN_i is the nitrogen input from scattered dwellings (ton/yr); PsN_i is the nitrogen input from point sources (ton/yr); and, finally, U_i is the nitrogen load from upstream catchments (ton/yr).

Note that nitrogen atmospheric deposition losses are split into two parts, i.e. inputs to agricultural land, which undergo the basin retention (such as crop uptake) that depends on annual precipitation, while in forest areas they are reduced by a fixed rate, before entering into the stream. The contribution of atmospheric nitrogen deposition in non-agricultural land is thus estimated as $0.38 \cdot FF \cdot AtmN$. For an atmospheric deposition of 10 kgN/ha this corresponds to a background of 3.8 kgN/ha (in line with the values reported by HELCOM (2003)).

Phosphorous background losses are split into two parts, with the inputs to agricultural land undergoing basin retention, while in forest areas they are considered entering into the stream. Background losses for phosphorus are estimated at 0.15 kgP/ha (in line with the values reported by (HELCOM 2003)).

In GREEN phosphorus model, the total annual phosphorus load L_i of catchment i the equation estimates:

$$L_i = (1 - Lret_i) \cdot ((MinP_i + ManP_i + (1 - FF_i) \cdot BgP_i) \cdot (1 - Bret_i) + FF_i \cdot BgP_i + sd_{coeff} \cdot SdP_i + PsP_i + U_i) \cdot (1 - Rret_i) \quad (6)$$

where $MinP_i$ is the annual amount of phosphorus mineral fertilisers (ton/yr); $ManP_i$ is the annual amount of phosphorus in manure fertilisers (ton/yr); FF_i is the non-agricultural land cover in the catchment (fraction); BgP_i is the annual amount of phosphorus background losses (ton/yr); SdP_i is the phosphorus input from scattered dwellings (ton/yr); PsP_i is the phosphorus input from point sources (ton/yr); and U_i is the phosphorus load from upstream catchments (ton/yr).

6.2 Appendix 2

The `calib_green()` function applies the following GoF metrics (Althoff and Rodrigues 2021; Mauricio Zambrano-Bigiarini 2014) (NSE, rNSE, mNSE, R2, PBIAS), where:

- NSE: Nash-Sutcliffe efficiency.
- rNSE: Relative Nash-Sutcliffe efficiency.
- mNSE: Modified Nash-Sutcliffe efficiency.
- cp: Coefficient of Persistence.
- VE: Volumetric Efficiency.
- KGE : Kling-Gupta efficiency.
- d : Index of Agreement.
- md: Modified index of agreement.
- rd: Relative Index of Agreement.
- r: Linear correlation coefficient.
- R2: Coefficient of determination.
- PBIAS: Percent bias.
- MAE: mean absolute error.
- RMSE: Root mean square error.
- ME: Mean error.
- MSE: Mean square error.
- NRMSE: Normalized Root Mean Square Error.

See (Papacharalampous, Tyralis, and Koutsoyiannis 2019; Nash and Sutcliffe 1970; Kitanidis and Bras 1980; Yapo, Gupta, and Sorooshian 1996; Krause, Boyle, and Bäse 2005; Criss and Winston 2008; Hoshin Vijai Gupta, Sorooshian, and Yapo 1998; Mauricio Zambrano-Bigiarini 2014) for further details.

References

- Althoff, Daniel, and Lineu Neiva Rodrigues. 2021. "Goodness-of-Fit Criteria for Hydrological Models: Model Calibration and Performance Assessment." *Journal of Hydrology* 600: 126674. <https://doi.org/10.1016/j.jhydrol.2021.126674>.
- Arheimer, Berit, Joel Dahné, and Chantal Donnelly. 2012. "Climate Change Impact on Riverine Nutrient Load and Land-Based Remedial Measures of the Baltic Sea Action Plan." *Ambio* 41 (6): 600–612. <https://doi.org/10.1007/s13280-012-0323-0>.
- Bartosova, Alena, René Capell, Jørgen E Olesen, Mohamed Jabloun, Jens Christian Refsgaard, Chantal Donnelly, Kari Hyttiäinen, Sampo Pihlainen, Marianne Zandersen, and Berit Arheimer. 2019. "Future Socioeconomic Conditions May Have a Larger Impact Than Climate Change on Nutrient Loads to the Baltic Sea." *Ambio* 48 (11): 1325–36. <https://doi.org/10.1007/s13280-019-01243-5>.
- Beusen, A. H. W., J. C. Doelman, L. P. H. Van Beek, P. J. T. M. Van Puijenbroek, J. M. Mogollón, H. J. M. Van Grinsven, E. Stehfest, D. P. Van Vuuren, and A. F. Bouwman. 2022. "Exploring River Nitrogen and Phosphorus Loading and Export to Global Coastal Waters in the Shared Socio-Economic Pathways." *Global Environmental Change* 72: 102426. <https://doi.org/https://doi.org/10.1016/j.gloenvcha.2021.102426>.
- Bouraoui, F, V Thieu, B Grizzetti, W Britz, and G Bidoglio. 2014. "Scenario Analysis for Nutrient Emission Reduction in the European Inland Waters." *Environmental Research Letters* 9 (12): 125007. <https://doi.org/10.1088/1748-9326/9/12/125007>.
- Butler, Martha P, Patrick M Reed, Karen Fisher-Vanden, Klaus Keller, and Thorsten Wagener. 2014. "Identifying Parametric Controls and Dependencies in Integrated Assessment Models Using Global Sensitivity Analysis." *Environmental Modelling & Software* 59: 10–29. <https://doi.org/10.1016/j.envsoft.2014.05.001>.
- Carnell, Rob. 2012. *Lhs: Latin Hypercube Samples*. <https://CRAN.R-project.org/package=lhs>.
- Criss, Robert E, and William E Winston. 2008. "Do Nash Values Have Value? Discussion and Alternate Proposals." *Hydrological Processes: An International Journal* 22 (14): 2723–25. <https://doi.org/10.1002/hyp.7072>.
- De Jager, A, and J Vogt. 2007. "Rivers and Catchments of Europe-Catchment Characterisation Model (CCM)." *European Commission, Joint Research Centre (JRC), Ispra, Italy*.

- Doherty, John, and Brian E Skahill. 2006. "An Advanced Regularization Methodology for Use in Watershed Model Calibration." *Journal of Hydrology* 327 (3-4): 564–77. <https://doi.org/10.1016/j.jhydrol.2005.11.058>.
- EEA. 2021. "European Environment Agency (EEA)." *WISE Water Framework Directive Database*. <https://www.eea.europa.eu/data-and-maps/data/wise-wfd-4>.
- Frank, Ildiko E, and Roberto Todeschini. 1994. *The Data Analysis Handbook*. Elsevier.
- Fu, Baihua, Wendy S Merritt, Barry FW Croke, Tony R Weber, and Anthony J Jakeman. 2019. "A Review of Catchment-Scale Water Quality and Erosion Models and a Synthesis of Future Prospects." *Environmental Modelling & Software* 114: 75–97. <https://doi.org/10.1016/j.envsoft.2018.12.008>.
- Grizzetti, B, F Bouraoui, and G De Marsily. 2008. "Assessing Nitrogen Pressures on European Surface Water." *Global Biogeochemical Cycles* 22 (4). <https://doi.org/10.1029/2007GB003085>.
- Grizzetti, B, F Bouraoui, G De Marsily, and G Bidoglio. 2005. "A Statistical Method for Source Apportionment of Riverine Nitrogen Loads." *Journal of Hydrology* 304 (1-4): 302–15. <https://doi.org/10.1016/j.jhydrol.2004.07.036>.
- Grizzetti, Bruna, Fayçal Bouraoui, and Alberto Aloe. 2012. "Changes of Nitrogen and Phosphorus Loads to European Seas." *Global Change Biology* 18 (2): 769–82. <https://doi.org/10.1111/j.1365-2486.2011.02576.x>.
- Grizzetti, Bruna, Olga Vigiak, Angel Udias, Alberto Aloe, Michela Zanni, Faycal Bouraoui, Alberto Pistocchi, et al. 2021. "How EU Policies Could Reduce Nutrient Pollution in European Inland and Coastal Waters." *Global Environmental Change* 69: 102281. <https://doi.org/10.1016/j.gloenvcha.2021.102281>.
- Gupta, Hoshin Vijai, Soroosh Sorooshian, and Patrice Ogou Yapo. 1998. "Toward Improved Calibration of Hydrologic Models: Multiple and Noncommensurable Measures of Information." *Water Resources Research* 34 (4): 751–63. <https://doi.org/10.1029/97WR03495>.
- Gupta, Hoshin V, Harald Kling, Koray K Yilmaz, and Guillermo F Martinez. 2009. "Decomposition of the Mean Squared Error and NSE Performance Criteria: Implications for Improving Hydrological Modelling." *Journal of Hydrology* 377 (1-2): 80–91. <https://doi.org/10.1016/j.jhydrol.2009.08.003>.
- HELCOM, Helsinki Commission. 2003. "Fourth Baltic Sea Load Compilation (PLC-4)." 93. *Fourth Baltic Sea Load Compilation*.
- Kitanidis, Peter K, and Rafael L Bras. 1980. "Real-Time Forecasting with a Conceptual Hydrologic Model: 2. Applications and Results." *Water Resources Research* 16 (6): 1034–44. <https://doi.org/10.1029/WR016i006p01034>.
- Krause, Peter, DP Boyle, and Frank Bäse. 2005. "Comparison of Different Efficiency Criteria for Hydrological Model Assessment." *Advances in Geosciences* 5: 89–97. <https://doi.org/10.5194/adgeo-5-89-2005>.
- Kronvang, Brian, Josef Hezlar, Paul Boers, Jens P Jensen, Horst Behrendt, Tom Anderson, Berit Arheimer, Marcus Venohr, Carl Christian Hoffmann, and CB Nielsen. 2004. "Nutrient Retention Handbook." *Software Manual for EUROHARP-NUTRET and Scientific Review on Nutrient Retention. EUROHARP Report*, 9–2004.
- Kumarasamy, Karthik, and Patrick Belmont. 2018. "Calibration Parameter Selection and Watershed Hydrology Model Evaluation in Time and Frequency Domains." *Water* 10 (6): 710. <https://doi.org/10.3390/w10060710>.
- La Notte, Alessandra, Joachim Maes, Silvana Dalmazzone, Neville D Crossman, Bruna Grizzetti, and Giovanni Bidoglio. 2017. "Physical and Monetary Ecosystem Service Accounts for Europe: A Case Study for in-Stream Nitrogen Retention." *Ecosystem Services* 23: 18–29. <https://doi.org/10.1016/j.ecoser.2016.11.002>.
- Leip, Adrian, Gilles Billen, Josette Garnier, Bruna Grizzetti, Luis Lassaletta, Stefan Reis, David Simpson, et al. 2015. "Impacts of European Livestock Production: Nitrogen, Sulphur, Phosphorus and Greenhouse Gas Emissions, Land-Use, Water Eutrophication and Biodiversity." *Environmental Research Letters* 10 (11): 115004. <https://doi.org/10.1088/1748-9326/10/11/115004>.
- Ludwig, W, AF Bouwman, E Dumont, and F Lespinas. 2010. "Water and Nutrient Fluxes from Major Mediterranean and Black Sea Rivers: Past and Future Trends and Their Implications for the Basin-Scale Budgets." *Global Biogeochemical Cycles* 24 (4). <https://doi.org/10.1029/2009GB003594>.
- Madsen, Henrik. 2003. "Parameter Estimation in Distributed Hydrological Catchment Modelling Using Automatic Calibration with Multiple Objectives." *Advances in Water Resources* 26 (2): 205–16. [https://doi.org/10.1016/S0309-1708\(02\)00092-1](https://doi.org/10.1016/S0309-1708(02)00092-1).
- Malagó, Anna, Fayçal Bouraoui, Bruna Grizzetti, and Ad De Roo. 2019. "Modelling Nutrient Fluxes into the Mediterranean Sea." *Journal of Hydrology: Regional Studies* 22: 100592. <https://doi.org/10.1016/j.ejrh.2019.01.004>.
- Manache, Gemma, and Charles S Melching. 2004. "Sensitivity Analysis of a Water-Quality Model Using Latin Hypercube Sampling." *Journal of Water Resources Planning and Management* 130 (3): 232–42. [https://doi.org/10.1061/\(ASCE\)0733-9496\(2004\)130:3\(232\)](https://doi.org/10.1061/(ASCE)0733-9496(2004)130:3(232)).

- Mannina, G, A Cosenza, MB Neumann, and PA Vanrolleghem. 2018. "Global Sensitivity Analysis in Wastewater Treatment Modelling." *Advances in Wastewater Treatment* 1: 32.
- Mauricio Zambrano-Bigiarini. 2014. *hydroGOF: Goodness-of-Fit Functions for Comparison of Simulated and Observed Hydrological Time Series*. <https://doi.org/10.5281/zenodo.839854>.
- McKay, MD, RJ Beckman, and WJ Conover. 1979. "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code." *Technometrics* 21 (2): 239–45. <https://doi.org/10.2307/1268522>.
- Melching, Charles S, and Willy Bauwens. 2001. "Uncertainty in Coupled Nonpoint Source and Stream Water-Quality Models." *Journal of Water Resources Planning and Management* 127 (6): 403–13. [https://doi.org/10.1061/\(ASCE\)0733-9496\(2001\)127:6\(403\)](https://doi.org/10.1061/(ASCE)0733-9496(2001)127:6(403)).
- Messenger, Mathis Loïc, Bernhard Lehner, Günther Grill, Irena Nedeva, and Oliver Schmitt. 2016. "Estimating the Volume and Age of Water Stored in Global Lakes Using a Geo-Statistical Approach." *Nature Communications* 7 (1): 1–11. <https://doi.org/10.1038/ncomms13603>.
- Nash, J Eamonn, and Jonh V Sutcliffe. 1970. "River Flow Forecasting Through Conceptual Models Part i-a Discussion of Principles." *Journal of Hydrology* 10 (3): 282–90. [https://doi.org/10.1016/0022-1694\(70\)90255-6](https://doi.org/10.1016/0022-1694(70)90255-6).
- Norton, John. 2015. "An Introduction to Sensitivity Assessment of Simulation Models." *Environmental Modelling & Software* 69: 166–74. <https://doi.org/10.1016/j.envsoft.2015.03.020>.
- Papacharalampous, Georgia, Hristos Tyralis, and Demetris Koutsoyiannis. 2019. "Comparison of Stochastic and Machine Learning Methods for Multi-Step Ahead Forecasting of Hydrological Processes." *Stochastic Environmental Research and Risk Assessment* 33 (2): 481–514. <https://doi.org/10.1007/s00477-018-1638-6>.
- Pianosi, Francesca, Keith Beven, Jim Freer, Jim W Hall, Jonathan Rougier, David B Stephenson, and Thorsten Wagener. 2016. "Sensitivity Analysis of Environmental Models: A Systematic Review with Practical Workflow." *Environmental Modelling & Software* 79: 214–32. <https://doi.org/10.1016/j.envsoft.2016.02.008>.
- Puy, Arnald, Samuele Lo Piano, Andrea Saltelli, and Simon A. Levin. 2022. "Sensobol: An r Package to Compute Variance-Based Sensitivity Indices." *Journal of Statistical Software* 102 (5): 1–37. <https://doi.org/10.18637/jss.v102.i05>.
- Refsgaard, Jens Christian, and Hans Jørgen Henriksen. 2004. "Modelling Guidelines—Terminology and Guiding Principles." *Advances in Water Resources* 27 (1): 71–82. <https://doi.org/10.1016/j.advwatres.2003.08.006>.
- Saltelli, Andrea, Paola Annoni, et al. 2011. "Sensitivity Analysis."
- Saltelli, Andrea, Stefano Tarantola, and Francesca Campolongo. 2000. "Sensitivity Analysis as an Ingredient of Modeling." *Statistical Science*, 377–95. <http://www.jstor.org/stable/2676831>.
- Schwarz, Gregory E, Anne B Hoos, RB Alexander, and RA Smith. 2006. "The SPARROW Surface Water-Quality Model: Theory, Application and User Documentation."
- Seitzinger, SP, JA Harrison, Egon Dumont, Arthur HW Beusen, and AF Bouwman. 2005. "Sources and Delivery of Carbon, Nitrogen, and Phosphorus to the Coastal Zone: An Overview of Global Nutrient Export from Watersheds (NEWS) Models and Their Application." *Global Biogeochemical Cycles* 19 (4). <https://doi.org/10.1029/2005GB002606>.
- Shreve, Ronald L. 1966. "Statistical Law of Stream Numbers." *The Journal of Geology* 74 (1): 17–37. <http://www.jstor.org/stable/30075174>.
- Singh, Vijay P, and Donald K Frevert. 2002. *Mathematical Models of Large Watershed Hydrology*. Water Resources Publication.
- Smith, Richard A, Gregory E Schwarz, and Richard B Alexander. 1997. "Regional Interpretation of Water-Quality Monitoring Data." *Water Resources Research* 33 (12): 2781–98. <https://doi.org/10.1029/97WR02171>.
- Strahler, Arthur N. 1957. "Quantitative Analysis of Watershed Geomorphology." *Eos, Transactions American Geophysical Union* 38 (6): 913–20. <https://doi.org/10.1029/TR038i006p00913>.
- Vigiak, Olga, Angel Udiá, Bruna Grizzetti, Michela Zanni, Alberto Aloe, Franz Weiss, Jordan Hristov, Berry Bisselink, Ad de Roo, and Alberto Pistocchi. 2023. "Recent Regional Changes in Nutrient Fluxes of European Surface Waters." *Science of The Total Environment* 858: 160063. <https://data.jrc.ec.europa.eu/collection/wpi>.
- Westerberg, IK, J-L Guerrero, PM Younger, KJ Beven, Jan Seibert, Sven Halldin, JE Freer, and C-Y Xu. 2011. "Calibration of Hydrological Models Using Flow-Duration Curves." *Hydrology and Earth System Sciences* 15 (7): 2205–27. <https://doi.org/10.5194/hess-15-2205-2011>.
- Wöhling, Thomas, Luis Samaniego, and Rohini Kumar. 2013. "Evaluating Multiple Performance Criteria to Calibrate the Distributed Hydrological Model of the Upper Neckar Catchment." *Environmental Earth Sciences* 69 (2): 453–68. <https://doi.org/10.1007/s12665-013-2306-2>.
- Yapo, Patrice O, Hoshin Vijai Gupta, and Soroosh Sorooshian. 1996. "Automatic Calibration of Conceptual Rainfall-Runoff Models: Sensitivity to Calibration Data." *Journal of Hydrology* 181 (1-4): 23–48. [https://doi.org/10.1016/0022-1694\(95\)02918-4](https://doi.org/10.1016/0022-1694(95)02918-4).

Angel Udías
European Commission, Joint Research Centre (JRC)
Via Enrico Fermi 2749, I-21027, Ispra
Italy
Departament of Computer Science and Statistics, Rey Juan Carlos University
Calle Tulipán S/N, 28933, Móstoles, Madrid
Spain
ORCID: 0000-0003-1219-0465
angel.luis.udias@urjc.es

Bruna Grizzetti
European Commission, Joint Research Centre (JRC)
Via Enrico Fermi 2749, I-21027, Ispra
Italy
bruna.grizzetti@jrc.it

Olga Vigiak
European Commission, Joint Research Centre (JRC)
Via Enrico Fermi 2749, I-21027, Ispra
Italy
Olga.VIGIAK@ec.europa.eu

Alberto Aloe
European Commission, Joint Research Centre (JRC)
Via Enrico Fermi 2749, I-21027, Ispra
Italy
alberto.aloe@jrc.it

Cesar Alfaro
Departament of Computer Science and Statistics, Rey Juan Carlos University
Calle Tulipán S/N, 28933, Móstoles, Madrid
Spain
ORCID: 0000-0002-9146-4067
cesar.alfaro@urjc.es

Javier Gomez
Departament of Computer Science and Statistics, Rey Juan Carlos University
Calle Tulipán S/N, 28933, Móstoles, Madrid
Spain
ORCID: 0000-0001-6434-7263
javier.gomez@urjc.es

bayesassurance: An R Package for Calculating Sample Size and Bayesian Assurance

by Jane Pan and Sudipto Banerjee

Abstract In this paper, we present **bayesassurance**, an R package designed for computing Bayesian assurance criteria which can be used to determine sample size in Bayesian inference setting. The functions included in the R package offer a two-stage framework using design priors to specify the population from which the data will be collected and analysis priors to fit a Bayesian model. We also demonstrate that frequentist sample size calculations are exactly reproduced as special cases of evaluating Bayesian assurance functions using appropriately specified priors.

1 Introduction

Power is an important feature of statistical tests that refers to the probability of correctly identifying the occurrence of an event given the event is actually present. Specifically, in the frequentist setting, the power of a test describes the probability that the test will correctly reject the null hypothesis, H_0 , given that the alternative hypothesis, H_1 , is true. Conversely, $\beta = 1 - \text{power}$ represents the probability that a test will fail to identify a true effect. For any given test, power is a function of the underlying effect size, the significance level, α , and the sample size, n . While the underlying effect size is rarely under the experimenter's control, the significance level and the sample size are. Hence, it is important to conduct power analysis to determine appropriate assignments of α and n for an experiment to have a good chance of detecting an effect if one exists. Power curves serve as a useful tool in quantifying the degree of assurance held towards meeting a study's analysis objective across a range of sample sizes. Formulating sample size determination as a decision problem so that power is an increasing function of sample size, offers the investigator a visual aid in helping deduce the minimum sample size needed to achieve a desired power.

The analogue of power in the Bayesian setting is *assurance*, which is based upon the probability of meeting a desired analysis objective. As an example, our analysis objective could be to ascertain if the difference between two population quantities, θ_1 and θ_2 , exceeds a certain threshold θ_0 , i.e., $\theta_1 - \theta_2 > \theta_0$. We decide that our analysis objective is met if the posterior probability of the above event given data we have observed exceeds ω , i.e., $P(\theta_1 - \theta_2 > \theta_0 | y) > \omega$ with y being the realized data. Assurance is defined as the probability of the analysis objective being met under an assumed distribution of the data, i.e., $\delta = P_y\{y : P(\theta > \theta_0 | y) > \omega\}$, where y denotes the observed data. Several publications adopt a similar approach, where they determine sample size based upon some criteria of analysis or model performance (Rahme et al., 2000; Gelfand and Wang, 2002; O'Hagan and Stevens, 2001). Other proposed solutions of the sample size problem introduce frameworks that prioritize conditions specific to the problem at hand, e.g. the use of Bayesian average errors to simultaneously control for Type I and Type II errors (Reyes and Ghosh, 2013), the use of posterior credible interval lengths to evaluate sample size estimates (Joseph et al., 1997), and the use of survival regression models to target Bayesian meta-experimental designs (Reyes and Ghosh, 2013).

There are several R packages for Bayesian sample size determination using specified analytic criteria. The **SampleSizeMeans** package contains a series of functions used for determining appropriate sample sizes based on various Bayesian criteria for estimating means or differences between means of normal variables (Joseph and Belisle, 2012). Criteria considered include the Average Length Criterion, the Average Coverage Criterion, and the Modified Worst Outcome Criterion (Joseph et al., 1995; Joseph and Belisle, 1997). A supplementary package, **SampleSizeProportions**, addresses study designs for estimation of binomial proportions using the same set of criteria (Joseph and Belisle, 2009). Our package, **bayesassurance** calculates Bayesian assurance and sample sizes for analysis objectives using normal and binomial models. We devise a two stage framework using possibly different sets of prior distributions in the design and analysis stages. The prior in the design stage represents a model that generates the data. The prior in the analysis stage represents the analyst's beliefs about the data. These two prior distributions are possibly different because the analyst does not usually have enough prior information on the processes generating the data. We primarily demonstrate sample size determination using conjugate Bayesian linear regression models as a prototype for this article, although the R package offers functions for calculating Bayesian assurance for binary or binomial models as well. In the current article, we focus on the flexibility of Bayesian linear regression models and demonstrate determination of unequal sample sizes for two samples and longitudinal study

Function	Type	Description
pwr_freq	closed-form solution	Computes the frequentist power of the specified hypothesis test (either one or two-sided z -tests).
assurance_nd_na	closed-form solution	Computes the exact Bayesian assurance of attaining a specified analysis objective.
bayes_sim	simulation	Approximates the Bayesian assurance of attaining a specified condition for a balanced study design through Monte Carlo sampling.
bayes_sim_unbalanced	simulation	Approximates the Bayesian assurance of attaining a specified condition for an unbalanced study design through Monte Carlo sampling.
bayes_sim_unknownvar	simulation	Similar to <code>bayes_sim</code> but approximates the assurance assuming unknown variance.
bayes_adcock	simulation	Approximates the probability (assurance) that the absolute difference between the true population parameter and the sample estimate falls within a margin of error no greater than a pre-specified precision level, d (Adcock, 1997).
bayes_sim_betabin	simulation	Approximates the probability (assurance) that there exists a difference between two independent proportions (Pham-Gia, 1997).
bayes_goal_func	simulation	Approximates the rate of correct classification using a utility-based approach within a linear hypothesis testing setting (Inoue et al., 2005).
pwr_curve	visual tool	Constructs a plot with the power and assurance curves overlayed on top of each other for comparison.
gen_Xn	design tool	Constructs design matrix using given sample size(s). Used for power and sample size analysis in the Bayesian setting.
gen_Xn_longitudinal	design tool	Constructs design matrix using inputs that correspond to a balanced longitudinal study design.

Table 1: Overview of the functions available for use within the package.

designs.

The **bayesassurance** package is available on CRAN (Pan and Banerjee, 2022) and contains a collection of functions that can be divided into three categories based on design and usage. These include closed-form solutions, simulation-based solutions, and visualization and/or design purposes. All available functions are presented in Table 1. Fully worked-out examples and tutorials can be found on our Github page at https://github.com/jpan928/bayesassurance_rpackage. This article describes the basic underlying framework leading to analytically tractable expressions for Bayesian assurance. We will also illustrate the relationship between Bayesian and frequentist sample size determination. In addition, we briefly explore simulation-based assurance methods, outlining the statistical distribution theory associated with each method, followed by examples worked out in R that users will be able to replicate. Finally, we offer some useful graphical features and design matrix generators offered by the package. In the following sections, we provide a detailed overview for each of the available functions grouped by category followed with worked out examples in R.

2 Closed-form Solution of Assurance

Bayesian assurance evaluates the tenability of attaining a specified outcome through the implementation of prior and posterior distributions. The `assurance_nd_na` function computes the exact assurance using a closed-form solution. Classical frequentist (Neyman-Pearson) inference proceeds as follows. Prior to an experiment, the analyst determines the sample size, as well as significance level (α), which defines the maximum frequency of false positives (type I errors) they are willing to tolerate, if the null hypothesis is true. Then, once the data has been collected, a statistical test is conducted which yields the p-value or the probability of seeing the observed data under the null hypothesis. If the p-value is lower than α , the null hypothesis is rejected. In contrast, in Bayesian inference refrains from "rejecting" or "failing to reject the null hypothesis". Instead, we look at the tenability of a hypothesis based upon realized data. As an example, suppose we seek to evaluate the tenability of $H : \theta > \theta_0$ given data from a Gaussian population with mean θ_0 and known variance σ^2 . We assign two sets of priors for θ , one at the *design stage* and the other at the *analysis stage*. These two stages are the primary components that make up the skeleton of our generalized solution in the Bayesian setting and will be revisited in later sections. The analysis objective specifies the condition that needs to be satisfied. It defines a positive outcome, which serves as an overarching criteria that characterizes the study. Our analysis objective is to ascertain if $P(\theta > \theta_0 | \bar{y}) > 1 - \alpha$, where \bar{y} is the data average and α is a specified threshold. Assuming the prior $\theta \sim N\left(\mu, \frac{\sigma^2}{n_a}\right)$, the posterior distribution of θ is

$$N\left(\theta \mid \mu, \frac{\sigma^2}{n_a}\right) \times N\left(\bar{y} \mid \theta, \frac{\sigma^2}{n}\right) \propto N\left(\theta \mid \frac{n_a}{n+n_a}\mu + \frac{n}{n+n_a}\bar{y}, \frac{\sigma^2}{n+n_a}\right), \quad (1)$$

where n denotes the sample size of the data, \bar{y} denotes the mean of the data, and n_a is specified by the data analyst to quantify the prior degree of belief on θ .

The design objective is to find the sample size needed to ensure that the analysis objective is met $100\delta\%$ of the time, where δ denotes the assurance. At the design stage, we specify a model for the underlying population from which the data is generated. Our belief about this population is quantified using a *design prior* (we borrow this terminology from O'Hagan and Stevens, 2001), say $\theta \sim N\left(\mu, \frac{\sigma^2}{n_d}\right)$, where n_d is specified by the user to quantify the degree of belief (or amount of confidence) on the population parameters. Bayesian assurance is given by

$$\delta = P_{\bar{y}} \{ \bar{y} : P(\theta > \theta_0 | \bar{y}) > 1 - \alpha \}, \quad (2)$$

where $P_{\bar{y}}(\cdot)$ denotes the marginal distribution of \bar{y} obtained from the design priors

$$\int N\left(\theta \mid \mu, \frac{\sigma^2}{n_d}\right) \times N\left(\bar{y} \mid \theta, \frac{\sigma^2}{n}\right) d\theta = N\left(\bar{y} \mid \mu, \left(\frac{1}{n} + \frac{1}{n_d}\right)\sigma^2\right).$$

Applying this to (2) we obtain

$$\delta(\Delta, n) = \Phi\left(\sqrt{\frac{nn_d}{n+n_d}} \left[\frac{n+n_a}{n} \frac{\Delta}{\sigma} + Z_\alpha \frac{n+n_a}{n} \right]\right), \quad (3)$$

where $\Phi(\cdot)$ is the standard normal CDF with Z_α being its α -th quantile and $\Delta = \mu - \theta_0$.

We remark that the above expression assumes that σ^2 is known. This is a customary assumption made about the population in sample size calculations and is usually based upon pilot studies or historic data. Nevertheless, in theory we can relax this assumption and assign a prior distribution to σ^2 . Conjugate priors include the inverse-Gamma family of distributions and analogous formulas to (3) may be derived in terms of distribution functions of the (non-central) t-distribution. Nevertheless, we do not pursue this development here but discuss the case of unknown σ^2 later in the linear regression setting. We now describe the function to compute (3). Table 2 lists the set of parameters used in `assurance_nd_na`, with `alpha` taking a default value of 0.05.

The following code loads the `bayesassurance` package and assigns arbitrary parameters to `assurance_nd_na` prior to executing the function.

```
R> library(bayesassurance)

R> n <- seq(100, 250, 10)
R> n_a <- 10
R> n_d <- 10
R> theta_0 <- 0.15
R> theta_1 <- 0.25
```

assurance_nd_na: Parameters	
Variable	Description
n	sample size (either scalar or vector)
n_a	precision parameter within the analysis stage that quantifies the degree of belief carried towards parameter θ
n_d	precision parameter within the design stage that quantifies the degree of belief of the population from which we are generating samples from
theta_0	initial parameter value provided by the client
theta_1	prior mean of θ assigned in the analysis and design stage
sigsq	known variance
alt	specifies alternative test case, where alt = "greater" tests if $\theta_1 > \theta_0$, alt = "less" tests if $\theta_1 < \theta_0$, and alt = "two.sided" performs a two-sided test for $\theta_1 \neq \theta_0$. By default, alt = "greater"
alpha	significance level

Table 2: Parameter specifications needed to run assurance_nd_na.

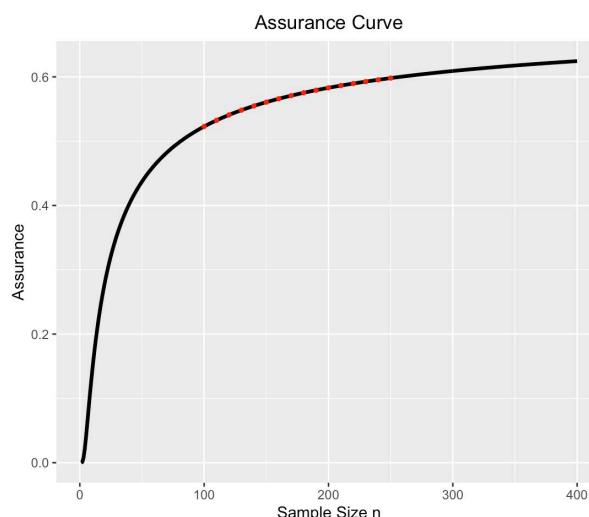


Figure 1: Resulting assurance plot with specific points passed in marked in red.

```
R> sigsq <- 0.30
R> out <- assurance_nd_na(n = n, n_a = n_a, n_d = n_d,
  theta_0 = theta_0, theta_1 = theta_1, sigsq = sigsq,
  alt = "greater", alpha = 0.05)
R> head(out$assurance_table)
R> out$assurance_plot
```

n	Assurance
1 100	0.5228078
2 110	0.5324414
3 120	0.5408288
4 130	0.5482139
5 140	0.5547789
6 150	0.5606632

Running this block of code will return a table of assurance values and a graphical display of the assurance curve, shown in Figure 1. The first six rows of the table are reported in the outputs.

We make a few remarks pertaining to the functions in **bayesassurance**. First, we are passing a

single value of sample size or a vector of sample sizes for n . We save the results as a variable `out`. n can be either a scalar or vector. If n is a scalar, this tells the function that we only want to determine the assurance for one particular sample size. When this is the case, `out` will return a single assurance value with no plot. Alternatively, if a vector of sample sizes is passed in as n , as is the case of the example above, assurance is computed for a list of sample sizes, and the function will produce both a table and an assurance curve showing the results. As long as n is of length two or greater, `assurance_nd_na` will plot the assurance curve with red points denoting user-specified assurance values and black points denoting all other points outside those specified points. Figure 1 presents the assurance curve resulting from the example. The graph is created using `ggplot2` Wickham (2016) which is imported into `bayesassurance`. Simply typing `out$assurance_table` and `out$assurance_plot` will display the table and plot respectively in this particular set of examples.

2.1 Special case: convergence with the frequentist setting

Depending on how we define the parameters in `assurance_nd_na`, we can demonstrate a relationship between the Bayesian and classical power analysis. In particular, the classical, or frequentist, power function is a special case of Bayesian solution when letting $n_d \rightarrow \infty$ and $n_a = 0$ in (3). Therefore, assigning a weak analysis prior and a strong design prior yields

$$\Phi\left(\sqrt{n} \frac{\Delta}{\sigma} + Z_\alpha\right), \quad (4)$$

which is equivalent to the frequentist power expression that takes the form

$$1 - \beta = P\left(\bar{y} > \theta_0 + \frac{\sigma}{\sqrt{n}} Z_{1-\alpha}\right) = \Phi\left(\sqrt{n} \frac{\Delta}{\sigma} + Z_\alpha\right).$$

The following code chunk demonstrates this special case in R using the `assurance_nd_na` function:

```
R> library(bayesassurance)

R> n <- seq(10, 250, 5)
R> n_a <- 1e-8
R> n_d <- 1e+8
R> theta_0 <- 0.15
R> theta_1 <- 0.25
R> sigsq <- 0.104

R> out <- assurance_nd_na(n = n, n_a = n_a, n_d = n_d,
  theta_0 = theta_0, theta_1 = theta_1, sigsq = sigsq,
  alt = "greater", alpha = 0.05)

R> head(out$assurance_table)
R> out$assurance_plot

      n Assurance
1 10 0.2532578
2 15 0.3285602
3 20 0.3981637
4 25 0.4623880
5 30 0.5213579
6 35 0.5752063
```

The `bayesassurance` package includes a `pwr_freq` function that determines the statistical power of testing the difference between two means given a set of fixed parameter values that yield a closed-form solution of power and sample size. Continuing with the one-sided case, the solution is

$$1 - \beta = P\left(\bar{y} > \theta_0 + \frac{\sigma}{\sqrt{n}} Z_{1-\alpha}\right) = \Phi\left(\sqrt{n} \frac{\Delta}{\sigma} + Z_\alpha\right), \quad (5)$$

where $\Delta = \theta_1 - \theta_0$ is the critical difference and Φ denotes the cumulative distribution function of the standard normal. Note this formula is equivalent to the special case of the assurance definition expressed in Equation (4). Table 3 includes the set of parameters needed to run this function.

pwr_freq: Parameters	
Variable	Description
n	sample size (either scalar or vector)
theta_0	value specified in the null hypothesis; to be provided by the user
theta_1	alternative value to test against the null value; this is the assumed effect size in the population
alt	specifies alternative test case, where alt = "greater" tests if $\theta_1 > \theta_0$, alt = "less" tests if $\theta_1 < \theta_0$, and alt = "two.sided" performs a two-sided test for $\theta_1 \neq \theta_0$. By default, alt = "greater"
sigsq	known variance
alpha	significance level

Table 3: Parameter specifications needed to run pwr_freq.

As a simple example, consider the following code segment that directly runs pwr_freq through specifying the above parameters and loading in **bayesassurance**:

```
R> library(bayesassurance)
R> pwr_freq(n = 20, theta_0 = 0.15, theta_1 = 0.35, sigsq = 0.30,
            alt = "greater", alpha = 0.05)

"Power: 0.495"
```

Running this returns the associated power printed as a statement rather than a table as we are only passing in one sample size, n = 20, to undergo evaluation. Now consider the next code example.

```
R> library(bayesassurance)
R> n <- seq(10, 250, 5)
R> out <- pwr_freq(n = n, theta_0 = 0.15, theta_1 = 0.25, sigsq = 0.104,
                    alt = "greater", alpha = 0.05)

R> head(out$pwr_table)
R> out$pwr_plot
```

	n	Power
1	10	0.2532578
2	15	0.3285602
3	20	0.3981637
4	25	0.4623880
5	30	0.5213579
6	35	0.5752063

This code produces identical results as the assurance values obtained in the example using the assurance_nd_na function, where we assigned a weak analysis prior and a strong design prior. This demonstrates that under these conditions, Bayesian assurance converges to frequentist power. Figure 2 provides a side-by-side comparison of the resulting power and assurance curves, portraying identical plots in this particular setting.

3 Simulation-based functions using conjugate linear models

Henceforth, we focus on computing assurance through simulation-based means and highlight the common scenario of performing sample size analysis where closed-form solutions are unavailable. In the following sections, we extend upon the two-stage design structure and discuss how the design and analysis objectives are constructed based on the sample size criteria.

We seek to evaluate the tenability of a well-defined analysis objective using our simulation-based functions. The functions take an iterative approach alternating between generating a dataset in the

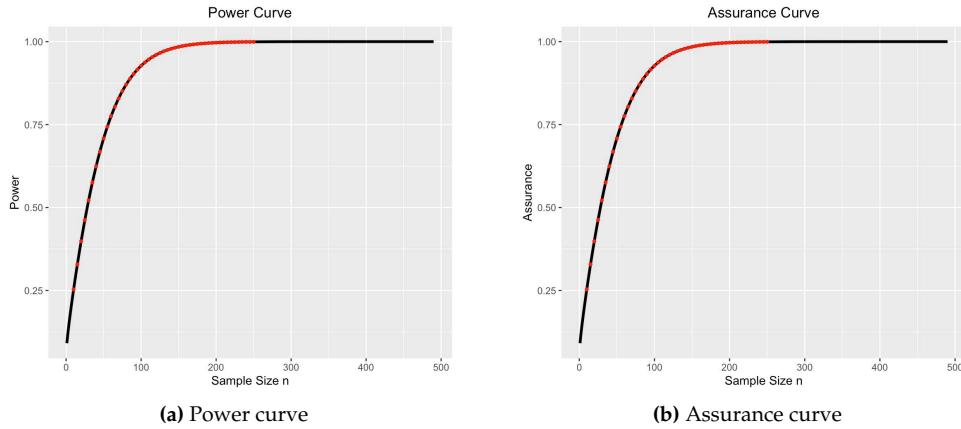


Figure 2: Power and assurance curves are identical when the design priors are strong and the analysis priors are weak.

design stage and evaluating whether or not the dataset satisfies the analysis objective. The assurance equates to the proportion of datasets that meet the objective.

We start in the analysis stage. Consider a set of n observations denoted by $y = (y_1, y_2, \dots, y_n)^\top$ that are to be collected together with p controlled explanatory variables, x_1, x_2, \dots, x_p . Specifically,

$$y = X_n\beta + \epsilon_n,$$

where X_n is an $n \times p$ design matrix whose i th row x_i^\top , and $\epsilon_n \sim N(0, \sigma^2 V_n)$, where V_n is a known $n \times n$ correlation matrix. We assume X_n has linearly independent columns. A conjugate Bayesian linear regression model specifies the joint distribution of the parameters $\{\beta, \sigma^2\}$ and y as

$$IG(\sigma^2 | a_\sigma, b_\sigma) \times N(\beta | \mu_\beta^{(a)}, \sigma^2 V_\beta^{(a)}) \times N(y | X_n\beta, \sigma^2 V_n),$$

where superscripts (a) indicate parameters in the analysis stage. The objective is to find the assurance of the realized data favoring $H : u^\top \beta > C$, where u is a $p \times 1$ vector of fixed contrasts and C is a known constant. Inference proceeds from the posterior distribution given by

$$p(\beta, \sigma^2 | y) = IG(\sigma^2 | a_\sigma^*, b_\sigma^*) \times N(\beta | M_n m_n, \sigma^2 M_n), \quad (6)$$

where

$$\begin{aligned} M_n^{-1} &= V_\beta^{-1(a)} + X_n^\top V_n^{-1} X_n; & m_n &= V_\beta^{-1(a)} \mu_\beta^{(a)} + X_n^\top V_n^{-1} y \\ a_\sigma^* &= a_\sigma + \frac{n}{2}; & b_\sigma^* &= b_\sigma + \frac{1}{2} \left\{ \mu_\beta^\top V_\beta^{-1} \mu_\beta + y_n^\top V_n^{-1} y - m_n^\top M_n m_n \right\}. \end{aligned}$$

The posterior distribution helps shape our analysis objective.

If σ^2 is known and fixed, then the posterior distribution of β is $p(\beta | \sigma^2, y) = N(\beta | M_n m_n, \sigma^2 M_n)$ shown in the Equation (6). We use the posterior components of β to evaluate $H : u^\top \beta > C$, where standardization leads to

$$\frac{u^\top \beta - u^\top M_n m_n}{\sigma \sqrt{u^\top M_n u}} \Big| \sigma^2, y \sim N(0, 1). \quad (7)$$

Hence, to assess the tenability of $H : u^\top \beta > C$, we decide in favor of H if the observed data belongs in the set

$$A_\alpha(u, \beta, C) = \left\{ y : P(u^\top \beta \leq C | y) < \alpha \right\} = \left\{ y : \Phi \left(\frac{C - u^\top M_n m_n}{\sigma \sqrt{u^\top M_n u}} \right) < \alpha \right\}.$$

This defines our analysis objective, which we will monitor within each sample iteration. Sample generation is taken to account for in the design stage discussed in the next section.

In the design stage, the goal is to seek a sample size n such that the analysis objective is met at least $100\delta\%$ of the time, where δ is the assurance. This step requires determining the marginal distribution of y , which is assigned a separate set of priors to quantify our belief about the population from which

the sample will be taken. Hence, the marginal of y under the design priors will be derived from

$$y = X_n\beta + \epsilon_n; \quad \epsilon_n \sim N(0, \sigma^2 V_n); \quad \beta = \mu_\beta^{(d)} + \omega; \quad \omega \sim N(0, \sigma^2 V_\beta^{(d)}), \quad (8)$$

where $\beta \sim N(\mu_\beta^{(d)}, \sigma^2 V_\beta^{(d)})$ is the design prior on β and (d) denotes parameters in the design stage. Substituting the equation for β into the equation for y gives $y = X\mu_\beta^{(d)} + (X\omega + \epsilon_n)$ and, hence,

$$y \sim N\left(X\mu_\beta^{(d)}, \sigma^2 V_n^*\right) \quad \text{and} \quad V_n^* = \left(XV_\beta^{(d)}X^\top + V_n\right).$$

To summarize our simulation strategy for estimating the Bayesian assurance, we fix sample size n and generate a sequence of J datasets $y^{(1)}, y^{(2)}, \dots, y^{(J)}$. A Monte Carlo estimate of the Bayesian assurance is computed as

$$\hat{\delta}(n) = \frac{1}{J} \sum_{j=1}^J \mathbb{I}\left(\left\{y^{(j)} : \Phi\left(\frac{C - u^\top M_n^{(j)} m_n^{(j)}}{\sigma \sqrt{u^\top M_n^{(j)} u}}\right) < \alpha\right\}\right),$$

where $\mathbb{I}(\cdot)$ is the indicator function of the event in its argument, $M_n^{(j)}$ and $m_n^{(j)}$ are the values of M_n and m_n computed from $y^{(j)}$.

3.1 Assurance computation with known variance

The simulation-based function, `bayes_sim`, determines the assurance within the context of conjugate Bayesian linear regression models assuming known variance, σ^2 . The execution of `bayes_sim` is straightforward. An important feature is that users are not required to provide their own design matrix, X_n , when executing `bayes_sim`. When $X_n = \text{NULL}$, the function automatically constructs an appropriate design matrix based on the specified sample size(s), using the built-in `gen_Xn` function.

Later sections discuss design matrix generators in greater detail. Setting $X_n = \text{NULL}$ facilitates calculation of assurances across a vector of sample sizes, where the function sequentially updates the design matrix for each unique sample size undergoing evaluation. Table 4 lists the required set of parameters.

Example 1: scalar parameter

This example computes the tenability of $H : u^\top \beta > C$ in the case when β is a scalar. In the following code block, we assign a set of values for the parameters of `bayes_sim` and saves the outputs as `assur_vals`. The first ten rows of the table is shown.

```
R> library(bayesassurance)
R> n <- seq(100, 300, 10)
R> assur_vals <- bayes_sim(n, p = 1, u = 1,
  C = 0.15, Xn = NULL, Vbeta_d = 0, Vbeta_a_inv = 0,
  Vn = NULL, sigsq = 0.265, mu_beta_d = 0.25, mu_beta_a = 0,
  alt = "greater", alpha = 0.05, mc_iter = 5000)

R> head(assur_vals$assurance_table)
R> assur_vals$assurance_plot
```

	Observations per Group (n)	Assurance
1	100	0.6162
2	110	0.6612
3	120	0.6886
4	130	0.7148
5	140	0.7390
6	150	0.7746

Each unique value passed into `n` corresponds to a separate balanced study design containing that particular sample size for each of the p groups undergoing assessment. In this example, setting $p = 1$, $u = 1$ and $C = 0.15$ implies that we are evaluating the tenability of $H : \beta > 0.15$, where β is a scalar. Furthermore, `Vbeta_d` and `Vbeta_a_inv` are scalars to align with the dimension of β . A weak analysis prior (`Vbeta_a_inv = 0`) and a strong design prior (`Vbeta_d = 0`) produces the classical power analysis.

bayes_sim: Parameters	
Variable	Description
n	Sample size (either vector or scalar). If vector, each value corresponds to a separate study design.
p	Number of explanatory variables being considered. Also denotes the column dimension of design matrix Xn. If Xn = NULL, p must be specified for the function to assign a default design matrix for Xn.
u	a scalar or vector included in the expression to be evaluated, e.g. $u^\top \beta > C$, where β is an unknown parameter that is to be estimated.
C	constant to be compared to
Xn	design matrix characterizing the observations given by the normal linear regression model $y_n = X_n \beta + \epsilon_n$, where $\epsilon_n \sim N(0, \sigma^2 V_n)$. See above description for details. Default Xn is an $np \times p$ matrix comprised of $n \times 1$ ones vectors that run across the diagonal of the matrix.
Vbeta_d	correlation matrix that characterizes prior information on β in the design stage, i.e. $\beta \sim N(\mu_\beta^{(d)}, \sigma^2 V_\beta^{(d)})$.
Vbeta_a_inv	inverse-correlation matrix that characterizes prior information on β in the analysis stage, i.e. $\beta \sim N(\mu_\beta^{(a)}, \sigma^2 V_\beta^{(a)})$. The inverse is passed in for computation efficiency, i.e. $V_\beta^{-1(a)}$.
Vn	an $n \times n$ correlation matrix for the marginal distribution of the sample data y_n . Takes on an identity matrix when set to NULL.
sigsq	a known and fixed constant preceding all correlation matrices Vn, Vbeta_d and Vbeta_a_inv.
mu_beta_d	design stage mean, $\mu_\beta^{(d)}$
mu_beta_a	analysis stage mean, $\mu_\beta^{(a)}$
alt	specifies alternative test case, where alt = "greater" tests if $u^\top \beta > C$, alt = "less" tests if $u^\top \beta < C$, and alt = "two.sided" performs a two-sided test for $u^\top \beta \neq C$. By default, alt = "greater".
alpha	significance level
mc_iter	number of MC samples evaluated under the analysis objective

Table 4: Parameter specifications needed to run bayes_sim.

We revisit this example in a later section when reviewing features that allow users to simultaneously visualize the Bayesian and frequentist power analyses. Finally, Xn and Vn are set to NULL, which means they will take on the default settings described above.

Example 2: linear contrasts

In this example, we assume β is a vector of unknown components rather than a scalar. We use the real-world example discussed in O'Hagan and Stevens (2001). Specifically, we consider a randomized clinical trial that compares the cost-effectiveness of two treatments. Cost-effectiveness is evaluated using a net monetary benefit measure expressed as

$$\xi = K(\mu_2 - \mu_1) - (\gamma_2 - \gamma_1),$$

where μ_1 and μ_2 denote the efficacy of treatments 1 and 2, and γ_1 and γ_2 denote the costs, respectively. Hence, $\mu_2 - \mu_1$ and $\gamma_2 - \gamma_1$ correspond to the true differences in treatment efficacy and costs. The threshold unit cost, K , represents the maximum price that a health care provider is willing to pay for a

unit increase in efficacy.

In this setting, we seek the tenability of $H : \xi > 0$, which, if true, indicates that treatment 2 is more cost-effective than treatment 1. To comply with the conjugate linear model framework outlined in (6), we set $u = (-K, 1, K, -1)^\top$, $\beta = (\mu_1, \gamma_1, \mu_2, \gamma_2)^\top$, and $C = 0$, giving us an equivalent form of $\xi > 0$ expressed as $u^\top \beta > 0$. All other inputs of this application were directly pulled from O'Hagan and Stevens (2001). The following code sets up the inputs to be passed into bayes_sim.

```
R> n <- 285
R> p <- 4
R> K <- 20000 # threshold unit cost
R> C <- 0
R> u <- as.matrix(c(-K, 1, K, -1))
R> sigsq <- 4.04^2

## Assign mean parameters to analysis and design stage priors
R> mu_beta_d <- as.matrix(c(5, 6000, 6.5, 7200))
R> mu_beta_a <- as.matrix(rep(0, p))

## Assign correlation matrices (specified in paper)
## to analysis and design stage priors
R> Vbeta_a_inv <- matrix(rep(0, p^2), nrow = p, ncol = p)
R> Vbeta_d <- (1 / sigsq) * matrix(c(4, 0, 3, 0, 0, 10^7, 0,
  0, 3, 0, 4, 0, 0, 0, 10^7), nrow = 4, ncol = 4)

R> tau1 <- tau2 <- 8700
R> sig <- sqrt(sigsq)
R> Vn <- matrix(0, nrow = n*p, ncol = n*p)
R> Vn[1:n, 1:n] <- diag(n)
R> Vn[(2*n - (n-1)):(2*n), (2*n - (n-1)):(2*n)] <- (tau1 / sig)^2 * diag(n)
R> Vn[(3*n - (n-1)):(3*n), (3*n - (n-1)):(3*n)] <- diag(n)
R> Vn[(4*n - (n-1)):(4*n), (4*n - (n-1)):(4*n)] <- (tau2 / sig)^2 * diag(n)
```

The inputs specified above should result in an assurance of approximately 0.70 according to O'Hagan and Stevens (2001). The bayes_sim returns a similar value, demonstrating that sampling from the posterior yields results similar to those reported in the paper.

```
R> library(bayesassurance)

R> assur_vals <- bayes_sim(n = 285, p = 4, u = as.matrix(c(-K, 1, K, -1)),
  C = 0, Xn = NULL, Vbeta_d = Vbeta_d, Vbeta_a_inv = Vbeta_a_inv,
  Vn = Vn, sigsq = 4.04^2, mu_beta_d = as.matrix(c(5, 6000, 6.5, 7200)),
  mu_beta_a = as.matrix(rep(0, p)), alt = "greater", alpha = 0.05, mc_iter = 10000)

R> assur_vals

## [1] "Assurance: 0.722"
```

3.2 Assurance computation in the longitudinal setting

We demonstrate an additional feature embedded in the function tailored to longitudinal data. In this setting, n no longer refers to the number of subjects but rather the number of repeated measures reported for each subject, assuming a balanced study design. Referring back to the linear regression model discussed in the general framework, we can construct a longitudinal model that utilizes this same linear regression form, where $y_n = X_n\beta + \epsilon_n$.

Consider a group of subjects in a balanced longitudinal study with the same number of repeated measures at equally-spaced time points. In the base case, where time is treated as a linear term, subjects can be characterized as

$$y_{ij} = \alpha_i + \beta_i t_{ij} + \epsilon_i,$$

where y_{ij} denotes the j^{th} observation on subject i at time t_{ij} , α_i and β_i denote the intercept and slope terms for subject i , respectively, and $\epsilon_i \sim N(0, \sigma_i^2)$ is an error term.

In a simple case with two subjects, we can individually express the observations as

$$y_{11} = \alpha_1 + \beta_1 t_{11} + \epsilon_1$$

⋮

$$y_{1n} = \alpha_1 + \beta_1 t_{1n} + \epsilon_1$$

$$y_{21} = \alpha_2 + \beta_2 t_{21} + \epsilon_2$$

⋮

$$y_{2n} = \alpha_2 + \beta_2 t_{2n} + \epsilon_2,$$

assuming that each subject contains n observations. The model can also be expressed using matrices:

$$\begin{pmatrix} y_{11} \\ \vdots \\ y_{1n} \\ y_{21} \\ \vdots \\ y_{2n} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & t_{11} & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & t_{1n} & 0 \\ 0 & 1 & 0 & t_{21} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & 0 & t_{2n} \end{pmatrix}}_{X_n} \underbrace{\begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \beta_1 \\ \beta_2 \end{pmatrix}}_{\beta} + \underbrace{\begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_2 \end{pmatrix}}_{\epsilon_n} \quad (9)$$

bringing us back to the linear model structure. If higher degrees are to be considered for the time variable, such as the inclusion of a quadratic term, the model would be altered to include additional covariate terms that can accommodate these changes. In the two-subject case, incorporating a quadratic term for the time variable in (9) will result in the model being modified as follows:

$$\begin{pmatrix} y_{11} \\ \vdots \\ y_{1n} \\ y_{21} \\ \vdots \\ y_{2n} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & t_{11} & 0 & t_{11}^2 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & t_{1n} & 0 & t_{1n}^2 & 0 \\ 0 & 1 & 0 & t_{21} & 0 & t_{21}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & 0 & t_{2n} & 0 & t_{2n}^2 \end{pmatrix}}_{X_n} \underbrace{\begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \beta_1 \\ \beta_2 \\ \phi_1 \\ \phi_2 \end{pmatrix}}_{\beta} + \underbrace{\begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_2 \end{pmatrix}}_{\epsilon_n}$$

In general, for m subjects who each have n repeated measures, a one-unit increase in the degree of the time-based covariate will result in m additional columns being added to the design matrix X_n and m additional rows added to the β vector.

When working in the longitudinal setting, additional parameters need to be specified in the `bayes_sim` function, which can be found in Table 5. By default, `longitudinal = FALSE` and `ids, from, and to` are set to `NULL` when working within the standard conjugate linear model. When `longitudinal = TRUE`, `n` takes on a different meaning as its value(s) correspond to the number of repeated measures for each subject rather than the total number of subjects in each group. When `longitudinal = TRUE` and `Xn = NULL`, `bayes_sim` implicitly relies on a design matrix generator, `gen_Xn_longitudinal`, that is specific to the longitudinal setting to construct appropriate design matrices.

Example 3: longitudinal setting

The following example uses similar parameter settings as the cost-effectiveness example we had previously discussed in Example 2, now with longitudinal specifications. We assume two subjects and want to test whether the growth rate of subject 1 is different from that of subject 2. Figure 3 displays the estimated assurance points given the specifications.

Assigning an appropriate linear contrast lets us evaluate the tenability of an outcome. Let us consider the tenability of $u^\top \beta \neq C$ in this next example that uses simulated data, where $u = (1, -1, 1, -1)^\top$ and $C = 0$. There are 120 arbitrary timepoints. The number of repeated measurements per subject to be tested includes values 10 through 100 in increments of 5. This indicates that we are evaluating the assurance for 19 study designs in total. $n = 10$ divides the specified time interval into 10 evenly-spaced timepoints between 0 and 120.

For a more complicated study design comprised of more than two subjects that are divided into two treatment groups, consider testing if the mean growth rate is higher in the first treatment group

bayes_sim: Additional Parameters used in the Longitudinal Setting	
Variable	Description
longitudinal	logical that indicates the simulation will be based in a longitudinal setting. If $Xn = \text{NULL}$, the function will construct a design matrix using inputs that correspond to a balanced longitudinal study design.
ids	vector of unique subject ids.
from	start time of repeated measures for each subject
to	end time of repeated measures for each subject
num_repeated_measures	desired length of the repeated measures sequence. This should be a non-negative number, will be rounded up otherwise if fractional.
poly_degree	degree of polynomial in longitudinal model, set to 1 by default.

Table 5: Additional parameter specifications needed to run bayes_sim in the longitudinal setting.

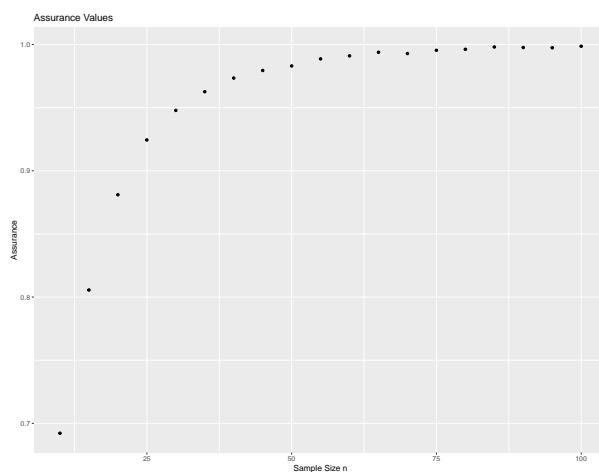


Figure 3: Estimated assurance points for longitudinal example.

than that of the second, e.g. if we have three subjects per treatment group, the linear contrast would be set as $u = (0, 0, 0, 0, 0, 1/3, 1/3, 1/3, -1/3, -1/3, -1/3)^T$.

```
R> n <- seq(10, 100, 5)
R> ids <- c(1,2)
R> Vbeta_a_inv <- matrix(rep(0, 16), nrow = 4, ncol = 4)
R> sigsq <- 100
R> Vbeta_d <- (1 / sigsq) * matrix(c(4, 0, 3, 0, 0, 6, 0, 0, 3, 0, 4, 0, 0, 0, 0, 6),
  nrow = 4, ncol = 4)

R> assur_out <- bayes_sim(n = n, p = NULL, u = c(1, -1, 1, -1), C = 0, Xn = NULL,
  Vbeta_d = Vbeta_d, Vbeta_a_inv = Vbeta_a_inv,
  Vn = NULL, sigsq = 100,
  mu_beta_d = as.matrix(c(5, 6.5, 62, 84)),
  mu_beta_a = as.matrix(rep(0, 4)), mc_iter = 5000,
  alt = "two.sided", alpha = 0.05, longitudinal = TRUE, ids = ids,
  from = 10, to = 120)

R> head(assur_out$assurance_table)
R> assur_out$assurance_plot

Observations per Group (n) Assurance
1                      10      0.6922
```

bayes_sim_unbalanced: Parameters for Unbalanced Study Designs		
Variable	Description	
n1	first sample size (either vector or scalar).	
n2	second sample size (either vector or scalar).	
repeats	an integer denoting the number of $c(n1, n2)$ pairs we are considering. For example, if repeats = 2, this means we will compute the assurance corresponding to the sample size set of $c(n1, n2, n1, n2)$. By default, repeats = 1. See Example 5.	
surface_plot	logical parameter that indicates whether a contour plot is to be constructed. When set to TRUE, and n1 and n2 are vectors, a contour plot (i.e. heat map) showcasing assurances obtained for all unique combinations of n1 and n2 is produced.	

Table 6: Parameter specifications needed to run bayes_sim_unbalanced.

2	15	0.8056
3	20	0.8810
4	25	0.9244
5	30	0.9478
6	35	0.9626

3.3 Assurance computation for unbalanced study designs

The `bayes_sim_unbalanced` function operates similarly to `bayes_sim` but estimates the assurance of attaining $u^\top \beta > C$ specifically in unbalanced design settings. Users provide two sets of sample sizes of equal length, whose corresponding pairs are considered for each study design case. The `bayes_sim_unbalanced` function provides a higher degree of flexibility for designing unbalanced studies and offers a more advanced visualization feature. Users have the option of viewing assurance as a 3-D contour plot and assess how the assurance behaves across varying combinations of the two sets of sample sizes that run along the x and y axes.

The `bayes_sim_unbalanced` function is similar to `bayes_sim` in terms of parameter specifications with a few exceptions. Parameters unique to `bayes_sim_unbalanced` are summarized in Table 6. Here, $Xn = \text{NULL}$, $Vn = \text{NULL}$, $\text{repeats} = 1$ and `surface_plot = TRUE` by default.

As in `bayes_sim`, it is recommended that users set $Xn = \text{NULL}$ to facilitate the automatic construction of appropriate design matrices that best aligns with the conjugate linear model. Recall that every unique sample size (or sample size pair) passed in corresponds to a separate study that requires a separate design matrix. Should users choose to provide their own design matrix, it is advised that they evaluate the assurance for one study design at a time, in which a single design matrix is passed into Xn along with scalar values assigned for the sample size parameter(s).

Saved outputs from executing the function include

1. `assurance_table`: table of sample size and corresponding assurance values
2. `contourplot`: contour map of assurance values if `surface.plot = TRUE`
3. `mc_samples`: number of Monte Carlo samples that were generated for evaluation

Example 4: unbalanced assurance computation with surface plot

The following code provides a basic example of how `bayes_sim_unbalanced` is executed. It is important to check that the parameters passed in are appropriate in dimensions, e.g. `mu_beta_a` and `mu_beta_d` should each contain the same length as that of u , and the length of u should be equal to the row and column dimensions of `Vbeta_d` and `Vbeta_a_inv`.

A table of assurance values is printed simply by calling `assur_out$assurance_table`, which contains the exact assurance values corresponding to each sample size pair. The contour plot, shown in Figure 4, is displayed using `assur_out$contourplot`, and offers a visual depiction of how the assurance varies across unique combinations of $n1$ and $n2$. Areas with lighter shades denote higher assurance levels. The next example implements the function in a real-world setting that offers more sensible results.

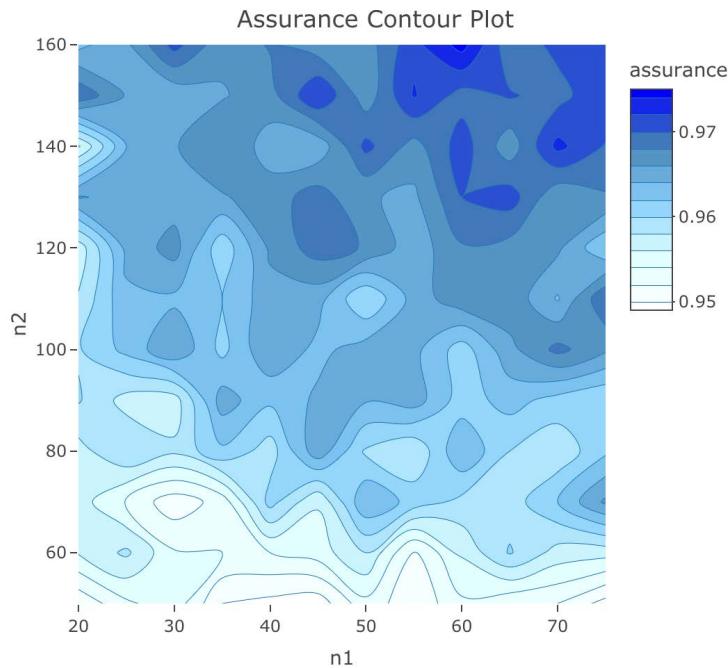


Figure 4: Contour map of assurance values with varying sample sizes n_1 and n_2 .

```
R> library(bayesassurance)

R> n1 <- seq(20, 75, 5)
R> n2 <- seq(50, 160, 10)

R> assur_out <- bayes_sim_unbalanced(n1 = n1, n2 = n2, repeats = 1, u = c(1, -1),
  C = 0, Xn = NULL, Vbeta_d = matrix(c(50, 0, 0, 10), nrow = 2, ncol = 2),
  Vbeta_a_inv = matrix(rep(0, 4), nrow = 2, ncol = 2),
  Vn = NULL, sigsq = 100, mu_beta_d = c(1.17, 1.25),
  mu_beta_a = c(0, 0), alt = "two.sided", alpha = 0.05, mc_iter = 5000,
  surface_plot = TRUE)

R> head(assur_out$assurance_table)
R> assur_out$contourplot

  n1  n2 Assurance
1 20  50    0.9504
2 25  60    0.9584
3 30  70    0.9508
4 35  80    0.9616
5 40  90    0.9624
6 45 100    0.9634
```

Example 5: cost-effectiveness application

We revisit the cost-effectiveness problem discussed in Example 2. In addition to providing a 3-D graphical display of the assurance, this example also demonstrates how the `repeats` parameter is applied.

Recall from Example 2 that two distinct sets of efficacy and cost measures are used to compare the cost-effectiveness of treatments 1 and 2. The efficacy and costs are denoted by μ_i and γ_i for $i = 1, 2$ treatments. Hence, the parameter we want to estimate contains four elements tied to the unknown efficacy and costs of treatments 1 and 2, i.e. $\beta = (\mu_1, \gamma_1, \mu_2, \gamma_2)^\top$. It was previously assumed that the treatments contain an equal number of observations, suggesting that the sample sizes across each of the four explanatory variables are also equal. Using `bayes_sim_unbalanced` offers the added flexibility of constructing an unbalanced study design between treatments 1 and 2. Since the two treatments each

contain two components to be measured, we use the repeats parameter to indicate that we want two sets of sample sizes, $c(n_1, n_2)$, passed in, i.e. $c(n_1, n_2, n_1, n_2)$. It then becomes clear that our study design consists of n_1 observations for the efficacy and cost of treatment 1, and n_2 observations for those of treatment 2. Figure 5 displays a contour plot with a noticeable increasing trend of assurance values across larger sets of sample sizes.

```
R> library(bayesassurance)
R> n1 <- c(4, 5, 15, 25, 30, 100, 200)
R> n2 <- c(8, 10, 20, 40, 50, 200, 250)

R> mu_beta_d <- as.matrix(c(5, 6000, 6.5, 7200))
R> mu_beta_a <- as.matrix(rep(0, 4))
R> K = 20000 # threshold unit cost
R> C <- 0
R> u <- as.matrix(c(-K, 1, K, -1))
R> sigsq <- 4.04^2
R> Vbeta_a_inv <- matrix(rep(0, 16), nrow = 4, ncol = 4)
R> Vbeta_d <- (1 / sigsq) * matrix(c(4, 0, 3, 0, 0, 10^7, 0, 0,
3, 0, 4, 0, 0, 0, 0, 10^7), nrow = 4, ncol = 4)

R> assur_out <- bayes_sim_unbalanced(n1 = n1, n2 = n2, repeats = 2,
u = as.matrix(c(-K, 1, K, -1)), C = 0, Xn = NULL,
Vbeta_d = Vbeta_d, Vbeta_a_inv = Vbeta_a_inv,
Vn = NULL, sigsq = 4.04^2,
mu_beta_d = as.matrix(c(5, 6000, 6.5, 7200)),
mu_beta_a = as.matrix(rep(0, 4)),
alt = "greater", alpha = 0.05, mc_iter = 5000,
surface_plot = TRUE)

R> assur_out$assurance_table
R> assur_out$contourplot

n1  n2 Assurance
1   4   8   0.1614
2   5  10   0.1724
3  15  20   0.3162
4  25  40   0.3942
5  30  50   0.4440
6 100 200   0.6184
7 200 250   0.7022
```

4 Bayesian assurance using other conditions

The **bayesassurance** R package contains several other assurance functions characterized by analysis stage objectives that are dependent on fixed precision levels (Adcock, 1997) and posterior credible intervals (Pham-Gia, 1997). These functions are denoted respectively as `bayes_adcock` and `bayes_sim_betabin`. The package also includes a `bayes_goal_func` function framed under a utility-based setting (see, e.g., Raiffa and Schlaifer, 1961; Berger, 1985; Lindley, 1997; Müller and Parmigiani, 1995; Parmigiani, 2002; Inoue et al., 2005) that determines sample size in relation to the rate of correct classification (Inoue et al., 2005). Since the simulation-based assurance functions all follow a similar format, for the sake of brevity, we will not include detailed descriptions of them in this article. Vignettes outlining detailed descriptions and walkthrough tutorials can be found on our Github page (https://github.com/jpan928/bayesassurance_rpackage), which contains examples that users can easily follow along and reproduce on their own machines.

5 Visualization Features and Useful Tools

5.1 Overlapping power and assurance curves

To facilitate an understanding of the relationship held between Bayesian and frequentist power analysis, the `pwr_curves` function produces a single plot displaying both power and assurance points.

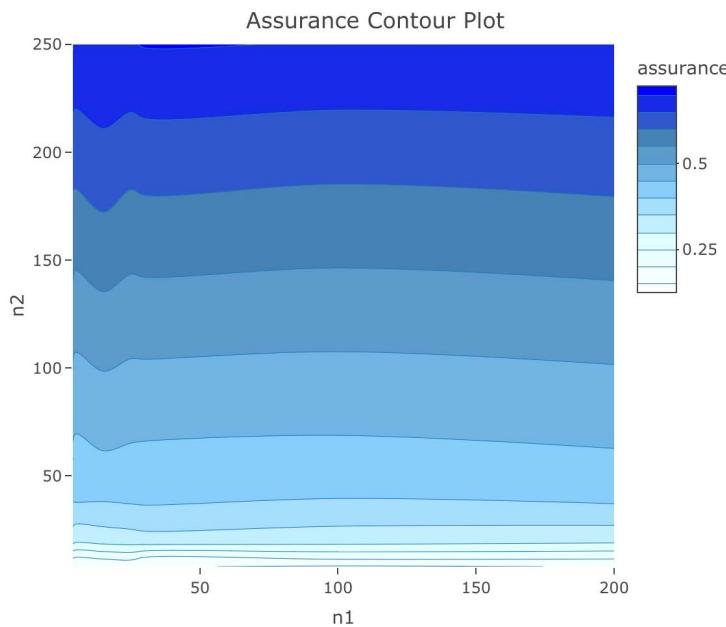


Figure 5: Contour map of assurance values in cost-effectiveness application.

Recall that the primary difference held between `pwr_freq` and `assurance_nd_na` is the need to specify additional precision parameters, n_a and n_d , in `assurance_nd_na`. Knowing that power and sample size analysis in the frequentist setting is essentially a special case of the Bayesian assurance with precision parameters tailored to weak analysis priors and strong design priors, the `pwr_curves` function serves as a visualization tool in seeing how varying precision levels affect assurance values and how these assurance values compare to those we would expect under classical/frequentist power analysis (strong design priors, weak analysis priors).

The `pwr_curves` function takes the combined set of parameters presented in `pwr_freq` and `assurance_nd_na`, which includes n , n_a , n_d , θ_0 , θ_1 , sig^2 , and α . For further customization, users have the option to include a third set of points in their plot along with the power and assurance curves. These additional points would correspond to the simulated assurance results obtained using `bayes_sim`. Optional parameters to implement this include

1. `bayes_sim`: logical that indicates whether the user wishes to include simulated assurance results obtained from `bayes_sim`. Default setting is FALSE.
2. `mc_iter`: specifies the number of MC samples to evaluate given `bayes_sim = TRUE`.

The following code segment runs the `pwr_curves` function using a weak analysis stage prior (n_a is set to be small) and a strong design stage prior (n_d is set to be large). Implementing this produces a plot where the assurance points lay perfectly on top of the power curve as shown in Figure 6. The simulated assurance points obtained from `bayes_sim` are also plotted as we set `bayes_sim = TRUE`. These points are highlighted in blue, which lie very close in proximity to those of the exact assurance points highlighted in red. We can also view individual tables of the three sets of points by directly calling them from the saved outputs, e.g. `out$power_table` shows the individual frequentist power values for each sample size. The output we provide shows the first ten rows.

```
R> library(bayesassurance)

R> out <- pwr_curve(n = seq(10, 200, 10), n_a = 1e-8, n_d = 1e+8,
  sigsq = 0.104, theta_0 = 0.15, theta_1 = 0.25, alt = "greater", alpha = 0.05,
  bayes_sim = TRUE, mc_iter = 5000)

R> head(out$power_table)
R> head(out$assurance_table)
R> out$plot
```

	n	Power
1	10	0.2532578
2	20	0.3981637

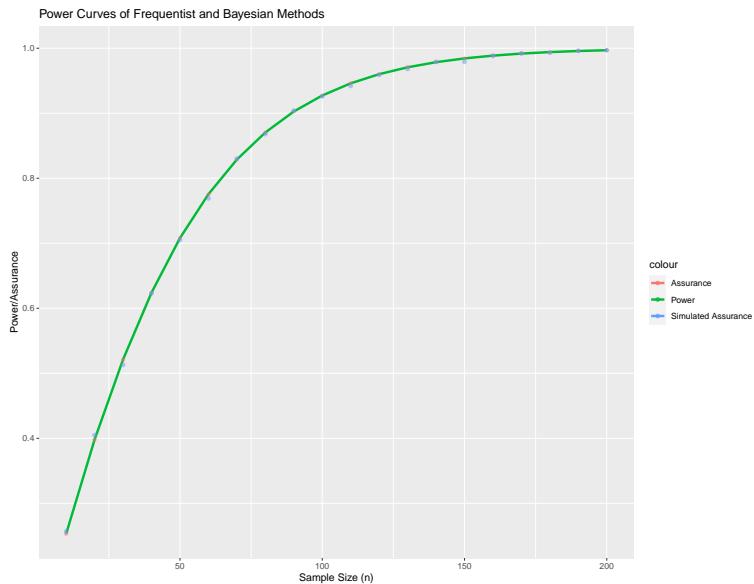


Figure 6: Power curve with exact and simulated assurance points for weak analysis prior and strong design prior.

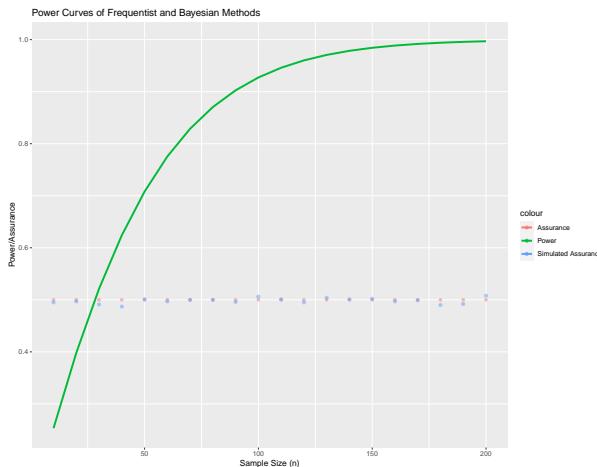


Figure 7: Power curve with exact and simulated assurance points for weak analysis and design priors.

```

3 30 0.5213579
4 40 0.6241155
5 50 0.7080824
6 60 0.7754956

```

	n	Assurance
1	10	0.2532578
2	20	0.3981637
3	30	0.5213579
4	40	0.6241155
5	50	0.7080824
6	60	0.7754956

The next code segment considers the scenario in which both analysis and design stage priors are weak (n_a and n_d are set to be small). This special case shows how the assurance behaves when vague priors are assigned. Substituting 0 in for both n_a and n_d in Equation (3) results in a constant assurance of $\Phi(0) = 0.5$ regardless of the sample size and critical difference. Figure 7 illustrates these results, where we have the regular power curve and the flat set of assurance points at 0.5 for both exact and simulated cases.

```
R> library(bayesassurance)
```

```
R> pwr_curve(n = seq(10, 200, 10), n_a = 1e-8, n_d = 1e-8,
  sigsq = 0.104, theta_0 = 0.15, theta_1 = 0.25, alt = "greater", alpha = 0.05,
  bayes_sim = TRUE, mc_iter = 5000)
```

5.2 Design matrix generators

In the last few sections, we go over design matrix generators that are used inside functions within the **bayesassurance** package when the X_n parameter is set to `NULL`. We include these functions in case users wish to see how design matrices are constructed under this particular setting.

Standard design matrix generator

The standard design matrix generator, `gen_Xn`, is relevant to a majority of the simulation-based assurance functions discussed throughout the paper. It should be noted that all simulation-based functions available in this package do not require users to specify their own design matrix X_n . Users have the option of leaving $X_n = \text{NULL}$, which prompts the function to construct a default design matrix using `gen_Xn` that complies with the general linear model $y_n = X_n\beta + \epsilon$, $\epsilon \sim N(0, \sigma^2 V_n)$. The function runs in the background while `bayes_sim` is used.

When called directly, the `gen_Xn` function takes in a single parameter, n , which can either be a scalar or vector. The length of n corresponds to the number of groups being assessed in the study design as well as the column dimension of the design matrix, denoted as p . Therefore, in general, the resulting design matrix is of dimension $n \times p$. If a scalar value is specified for n , the resulting design matrix carries a dimension of $n \times 1$.

In the following example, we pass in a vector of length $p = 4$, which outputs a design matrix of column dimension 4. Each column is comprised of ones vectors with lengths that align with the sample sizes passed in for n . The row dimension is therefore the sum of all the entries in n . In this case, since the values 1, 3, 5, and 8 are being passed in to n , the design matrix to be constructed carries a row dimension of $1 + 3 + 5 + 8 = 17$ and a column dimension of 4.

```
R> n <- c(1,3,5,8)
R> gen_Xn(n = n)

 [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    1    0    0
[4,]    0    1    0    0
[5,]    0    0    1    0
[6,]    0    0    1    0
[7,]    0    0    1    0
[8,]    0    0    1    0
[9,]    0    0    1    0
[10,]   0    0    0    1
[11,]   0    0    0    1
[12,]   0    0    0    1
[13,]   0    0    0    1
[14,]   0    0    0    1
[15,]   0    0    0    1
[16,]   0    0    0    1
[17,]   0    0    0    1
```

The `bayes_sim` function and its related family of functions generate design matrices using `gen_Xn` in the following way. Each unique value contained in n that is passed into `bayes_sim` corresponds to a distinct study design and thus requires a distinct design matrix. The `gen_Xn` function interprets each i^{th} component of n as a separate balanced study design comprised of n_i participants within each of the p groups, where p is a parameter specified in `bayes_sim`. For example, if we let $X_n = \text{NULL}$ and pass in $n <- 2, p <- 4$ for `bayes_sim`, `gen_Xn` will process the vector $n <- c(2, 2, 2, 2)$ in the background. Hence,

we'd obtain an 8×4 matrix of the form

$$X_n = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Design matrix generator in longitudinal setting

Example 3 demonstrates how the linear model is extended to incorporate time-based covariates within the context of a longitudinal setting. For this special case, a separate function is used to generate design matrices that are appropriate for this setting. The `genXn_longitudinal` constructs its design matrices differently than `gen_Xn` and therefore requires a different set of parameter specifications. When the `longitudinal` parameter is set to `TRUE` in `bayes_sim`, the user is required to specify the following set of parameters, which are directly passed into `genXn_longitudinal`:

1. `ids`: vector of unique subject ids, usually of length 2 for study design purposes
2. `from`: start time of repeated measures for each subject
3. `to`: end time of repeated measures for each subject
4. `num_repeated_measures`: desired length of the repeated measures sequence. Should be a non-negative number, will be rounded up if fractional.
5. `poly_degree`: degree of polynomial in longitudinal model, set to 1 by default.

Referring back to the model that was constructed for the case involving two subjects, we observe in Equation (9) that the design matrix contains vectors of ones within the first half of its column dimension and lists the timepoints for each subject in the second half. Constructing this design matrix requires several components. The user needs to specify subject IDs that are capable of uniquely identifying each individual in the study. Next, the user needs to specify the start and end time as well as the number of repeated measures reported for each subject. The number of repeated measures denotes the number of evenly-spaced timepoints that take place in between the start and end time. Since we are assuming a balanced longitudinal study design, each subject considers the same set of timepoints. Finally, if the user wishes to consider time covariates of higher degrees, such as a quadratic or cubic function, this can be altered using the `poly_degree` parameter, which takes on a default assignment of 1.

In the following code, we pass in a vector of subject IDs and specify the start and end timepoints along with the desired length of the sequence. The resulting design matrix contains vectors of ones with lengths that correspond to the number of repeated measures for each unique subject.

```
R> ids <- c(1,2,3,4)
R> gen_Xn_longitudinal(ids, from = 1, to = 10, num_repeated_measures = 4)

 [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,] 1 0 0 0 1 0 0 0
[2,] 1 0 0 0 4 0 0 0
[3,] 1 0 0 0 7 0 0 0
[4,] 1 0 0 0 10 0 0 0
[5,] 0 1 0 0 0 1 0 0
[6,] 0 1 0 0 0 4 0 0
[7,] 0 1 0 0 0 7 0 0
[8,] 0 1 0 0 0 10 0 0
[9,] 0 0 1 0 0 0 1 0
[10,] 0 0 1 0 0 0 4 0
[11,] 0 0 1 0 0 0 7 0
[12,] 0 0 1 0 0 0 0 10
[13,] 0 0 0 1 0 0 0 1
[14,] 0 0 0 1 0 0 0 4
[15,] 0 0 0 1 0 0 0 7
[16,] 0 0 0 1 0 0 0 10
```

The next code block modifies the previous example to incorporate a quadratic term. Notice there are four additional columns being aggregated to the design matrix. These four columns are obtained from squaring the four columns of the linear term.

```
R> ids <- c(1,2,3,4)
R> gen_Xn_longitudinal(ids, from = 1, to = 10, num_repeated_measures = 4,
poly_degree = 2)

  1 2 3 4  1 2 3 4  1 2 3 4
[1,] 1 0 0 0  1 0 0 0  1 0 0 0
[2,] 1 0 0 0  4 0 0 0  16 0 0 0
[3,] 1 0 0 0  7 0 0 0  49 0 0 0
[4,] 1 0 0 0 10 0 0 0 100 0 0 0
[5,] 0 1 0 0  0 1 0 0  0 1 0 0
[6,] 0 1 0 0  0 4 0 0  0 16 0 0
[7,] 0 1 0 0  0 7 0 0  0 49 0 0
[8,] 0 1 0 0  0 10 0 0  0 100 0 0
[9,] 0 0 1 0  0 0 1 0  0 0 1 0
[10,] 0 0 1 0  0 0 4 0  0 0 16 0
[11,] 0 0 1 0  0 0 7 0  0 0 49 0
[12,] 0 0 1 0  0 0 10 0  0 0 100 0
[13,] 0 0 0 1  0 0 0 1  0 0 0 1
[14,] 0 0 0 1  0 0 0 4  0 0 0 16
[15,] 0 0 0 1  0 0 0 7  0 0 0 49
[16,] 0 0 0 1  0 0 0 10 0 0 0 100
```

6 Discussion

This article introduced **bayesassurance**, a new R package for computing Bayesian assurance under various conditions using a two-stage framework. The goal of this package is to provide a convenient and user-friendly interface to statisticians and data scientists who seek sample size calculations using the assurance function in Bayesian data analysis. We have attempted to provide an organized, well-documented open-source code that can be used to address a wide range of study design problems, such as in the case of clinical trials, and demonstrate the feasibility of applying Bayesian methods to such problems.

References

- C. Adcock. Sample size determination: A review. *The Statistician*, 46(2), 1997. [p139, 152]
- J. O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer New York, New York, NY, 1985. [p152]
- A. E. Gelfand and F. Wang. A simulation based approach to bayesian sample size determination for performance under a given model and for separating models. *Statistical Science*, 17(2), 2002. [p138]
- L. Y. Inoue, D. A. Berry, and G. Parmigiani. Relationship between bayesian and frequentist sample size determination. *The American Statistician*, 59(1), 2005. [p139, 152]
- L. Joseph and P. Belisle. Bayesian sample size determination for normal means and differences between normal means. *The Statistician*, 46(2), 1997. [p138]
- L. Joseph and P. Belisle. *Package SampleSizeProportions*, 2009. URL <https://cran.r-project.org/web/packages/SampleSizeProportions/SampleSizeProportions.pdf>. R package version 1.0. [p138]
- L. Joseph and P. Belisle. *Package SampleSizeMeans*, 2012. URL <https://cran.r-project.org/web/packages/SampleSizeMeans/SampleSizeMeans.pdf>. R package version 1.1. [p138]
- L. Joseph, D. B. Wolfson, and R. du Berger. Sample size calculations for binomial proportions via highest posterior density intervals. *The Statistician*, 44(2), 1995. [p138]
- L. Joseph, R. D. Berger, and P. Belisle. Bayesian and mixed bayesian/likelihood criteria for sample size determination. *Statistics in Medicine*, 16(7), 1997. [p138]
- D. V. Lindley. The choice of sample size. *The Statistician*, 46(2), 1997. [p152]

- P. Müller and G. Parmigiani. Optimal design via curve fitting of monte carlo experiments. *Journal of the American Statistical Association*, 90(432), 1995. [p152]
- A. O'Hagan and J. W. Stevens. Bayesian assessment of sample size for clinical trials of cost-effectiveness. *Medical Decision Making*, 21(3), 2001. [p138, 140, 146, 147]
- J. Pan and S. Banerjee. *Package bayesassurance*, 2022. URL <https://cran.r-project.org/web/packages/bayesassurance/bayesassurance.pdf>. R package version 0.1.0. [p139]
- G. Parmigiani. *Modeling in Medical Decision Making: A Bayesian Approach*. Wiley, Hoboken, NJ, 2002. [p152]
- T. Pham-Gia. On bayesian analysis, bayesian decision theory and the sample size problem. *The Statistician*, 46(2), 1997. [p139, 152]
- E. Rahme, Lawrence, and T. W. Gyorkos. Bayesian sample size determination for estimating binomial parameters from data subject to misclassification. *Journal of Royal Statistical Society*, 49(1), 2000. [p138]
- H. Raiffa and R. Schlaifer. *Applied Statistical Decision Theory*. Harvard University Graduate School of Business Administration (Division of Research), Massachusetts, 1961. [p152]
- E. M. Reyes and S. K. Ghosh. Bayesian average error based approach to sample size calculations for hypothesis testing. *Biopharm*, 23(3), 2013. [p138]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p142]

Jane Pan
University of California, Los Angeles
650 Charles E Young Dr S, Los Angeles, CA 90095
USA
jpan1@ucla.edu

Sudipto Banerjee
University of California, Los Angeles
650 Charles E Young Dr S, Los Angeles, CA 90095
USA
sudipto@ucla.edu

fasano.franceschini.test: An Implementation of a Multivariate KS Test in R

by Connor Puritz, Elan Ness-Cohn, and Rosemary Braun

Abstract The Kolmogorov–Smirnov (KS) test is a nonparametric statistical test used to test for differences between univariate probability distributions. The versatility of the KS test has made it a cornerstone of statistical analysis across many scientific disciplines. However, the test proposed by Kolmogorov and Smirnov does not easily extend to multivariate distributions. Here we present the `fasano.franceschini.test` package, an R implementation of a multivariate two-sample KS test described by Fasano and Franceschini (1987). The `fasano.franceschini.test` package provides a test that is computationally efficient, applicable to data of any dimension and type (continuous, discrete, or mixed), and that performs competitively with similar R packages.

1 Introduction

The Kolmogorov–Smirnov (KS) test is a nonparametric, univariate statistical test designed to assess whether a sample of data is consistent with a given probability distribution (or, in the two-sample case, whether the two samples came from the same underlying distribution). First described by Kolmogorov and Smirnov in a series of papers (Kolmogorov, 1933a,b; Smirnov, 1936, 1937, 1939, 1944, 1948), the KS test is a popular goodness-of-fit test that has found use across a wide variety of scientific disciplines, including neuroscience (Atasoy et al., 2017), climatology (Chiang et al., 2018), robotics (Hahne et al., 2018), epidemiology (Wong and Collins, 2020), and cell biology (Kaczanowska et al., 2021).

Due to its popularity, several multivariate extensions of the KS test have been described in literature. Justel et al. (1997) proposed a multivariate test based on Rosenblatt’s transformation, which reduces to the KS test in the univariate case. While the test statistic is distribution-free, it is difficult to compute in more than two dimensions, and an approximate test with reduced power must be used instead. Furthermore, the test is only applicable in the one-sample case. Heuchenne and Mordant (2022) proposed to use the Hilbert space-filling curve to define an ordering in \mathbb{R}^2 . The preimage of both samples is computed under the space-filling curve map, and the two-sample KS test is performed on the preimages. While it is theoretically possible to extend this approach to higher dimensions, the authors note that this would be computationally challenging and leave it as an open problem. Naaman (2021) derived a multivariate extension of the DKW inequality and used it to provide estimates of the tail properties of the asymptotic distribution of the KS test statistic in multiple dimensions. While an important theoretical result, it is of limited practical use absent a method for computing exact *p*-values.

Peacock (1983) proposed a test which addresses the fact that there are multiple ways to order points in higher dimensions, and thus multiple ways of defining a cumulative distribution function. In one dimension, probability density can be integrated from left to right, resulting in the canonical CDF $P(X < x)$; or from right to left, resulting in the survival function $P(X > x)$. However, since $P(X < x) = 1 - P(X > x)$ (for continuous random variables), the KS test statistic is independent of this choice. In two dimensions, there are four ways of ordering points, and thus four possible cumulative distribution functions: $P(X < x, Y < y)$, $P(X > x, Y < y)$, $P(X < x, Y > y)$, and $P(X > x, Y > y)$. Since any three of these are independent of one another, the KS test statistic will not be independent of which ordering is chosen. To address this, Peacock (1983) proposed to compute a KS statistic using each possible cumulative distribution function, and to take the test statistic to be the maximum of those.

Peacock (1983) suggested that for a sample $(X_1, Y_1), \dots, (X_n, Y_n)$, each of the four KS statistics should be maximized over the set of all coordinate-wise combinations $\{(X_i, Y_j) : 1 \leq i, j \leq n\}$. The complexity of computing Peacock’s test statistic thus scales cubically with sample size, which is expensive and can become intractable for large sample sizes. Fasano and Franceschini (1987) proposed a simple change to Peacock’s test: instead of maximizing each KS statistic over all coordinate-wise combinations of points in the sample, the statistics should be maximized over just the points in the sample itself. This slight change greatly reduces the computational complexity of the test while maintaining a similar power across a variety of alternatives (Fasano and Franceschini, 1987; Lopes et al., 2007). Fasano and Franceschini (1987) proposed both a one-sample and two-sample version of their test, although we focus on the two-sample test here.

In this article we present the `fasano.franceschini.test` package, an R implementation of the two-sample Fasano–Franceschini test. Our implementation can be applied to continuous, discrete, or

mixed datasets of any size and of any dimension. We first introduce the test by detailing how the test statistic is computed, how it can be computed efficiently, and how p -values can be computed. We then describe the package structure and provide several basic examples illustrating its usage. We conclude by comparing the package to three other CRAN packages implementing multivariate two-sample goodness-of-fit tests.

2 Fasano–Franceschini test

2.1 Two-sample test statistic

Let $\mathbf{X} = (\mathbf{X}_1, \dots, \mathbf{X}_{n_1})$ and $\mathbf{Y} = (\mathbf{Y}_1, \dots, \mathbf{Y}_{n_2})$ be samples of i.i.d. d -dimensional random vectors drawn from unknown distributions F_1 and F_2 , respectively. The two-sample Fasano–Franceschini test evaluates the null hypothesis

$$H_0 : F_1 = F_2$$

against the alternative

$$H_1 : F_1 \neq F_2.$$

In their original paper, [Fasano and Franceschini \(1987\)](#) only considered two- and three-dimensional random vectors, although their test naturally extends to arbitrary dimensions as follows.

For $\mathbf{x} \in \mathbb{R}^d$, we define the i th open orthant with origin \mathbf{x} as

$$\mathcal{O}_i(\mathbf{x}) = \left\{ \mathbf{y} \in \mathbb{R}^d \mid \mathbf{e}_{ij}(\mathbf{y}_j - \mathbf{x}_j) > 0, j = 1, \dots, d \right\}$$

where $\mathbf{e}_i \in \{-1, 1\}^d$ is a length d combination of ± 1 . For example, in two dimensions, the four combinations $\mathbf{e}_1 = (1, 1)$, $\mathbf{e}_2 = (-1, 1)$, $\mathbf{e}_3 = (-1, -1)$, and $\mathbf{e}_4 = (1, -1)$ correspond to quadrants one through four in the plane, respectively. In general there are 2^d such combinations, corresponding to the 2^d orthants that divide \mathbb{R}^d . Using the indicator function

$$I_j(\mathbf{x} \mid \mathbf{y}) = \begin{cases} 1, & \mathbf{x} \in \mathcal{O}_j(\mathbf{y}) \\ 0, & \mathbf{x} \notin \mathcal{O}_j(\mathbf{y}) \end{cases}$$

we define

$$D(\mathbf{p} \mid \mathbf{X}, \mathbf{Y}) = \max_{1 \leq j \leq 2^d} \left| \frac{1}{n_1} \sum_{k=1}^{n_1} I_j(\mathbf{X}_k \mid \mathbf{p}) - \frac{1}{n_2} \sum_{k=1}^{n_2} I_j(\mathbf{Y}_k \mid \mathbf{p}) \right|. \quad (1)$$

This is similar to the distance used in the two-sample KS test, but takes into account all possible ways of ordering points in \mathbb{R}^d . Note that this function does not depend on the enumeration of the orthants. Maximizing D over each sample separately leads to the difference statistics

$$D_1(\mathbf{X}, \mathbf{Y}) = \max_{1 \leq i \leq n_1} D(\mathbf{X}_i \mid \mathbf{X}, \mathbf{Y})$$

and

$$D_2(\mathbf{X}, \mathbf{Y}) = \max_{1 \leq i \leq n_2} D(\mathbf{Y}_i \mid \mathbf{X}, \mathbf{Y}).$$

The two-sample Fasano–Franceschini test statistic, as originally defined by [Fasano and Franceschini \(1987\)](#), is the average of the difference statistics scaled by the sample sizes:

$$\mathcal{D}_0(\mathbf{X}, \mathbf{Y}) = \sqrt{\frac{n_1 n_2}{n_1 + n_2}} \left(\frac{D_1(\mathbf{X}, \mathbf{Y}) + D_2(\mathbf{X}, \mathbf{Y})}{2} \right). \quad (2)$$

This test statistic is discrete, but in general is not integer-valued. Note that

$$n_1 n_2 D(\mathbf{p} \mid \mathbf{X}, \mathbf{Y}) = \max_{1 \leq j \leq 2^d} \left| n_2 \sum_{k=1}^{n_1} I_j(\mathbf{X}_k \mid \mathbf{p}) - n_1 \sum_{k=1}^{n_2} I_j(\mathbf{Y}_k \mid \mathbf{p}) \right| \in \mathbb{Z},$$

and thus

$$n_1 n_2 D_i(\mathbf{X}, \mathbf{Y}) \in \mathbb{Z}, i \in \{1, 2\}.$$

Let

$$C_{n_1, n_2} = 2 \sqrt{n_1 n_2 (n_1 + n_2)}.$$

Then

$$\begin{aligned} C_{n_1, n_2} \mathcal{D}_0(\mathbf{X}, \mathbf{Y}) &= 2\sqrt{n_1 n_2 (n_1 + n_2)} \sqrt{\frac{n_1 n_2}{n_1 + n_2}} \left(\frac{D_1(\mathbf{X}, \mathbf{Y}) + D_2(\mathbf{X}, \mathbf{Y})}{2} \right) \\ &= n_1 n_2 (D_1(\mathbf{X}, \mathbf{Y}) + D_2(\mathbf{X}, \mathbf{Y})) \in \mathbb{Z}. \end{aligned}$$

To avoid comparing floating point numbers, it is preferable for the test statistic to be integer-valued, and thus we use

$$\mathcal{D}(\mathbf{X}, \mathbf{Y}) = C_{n_1, n_2} \mathcal{D}_0(\mathbf{X}, \mathbf{Y}) \quad (3)$$

as our test statistic. As will be shown, the p -value of the test is independent of scalar rescaling of the test statistic.

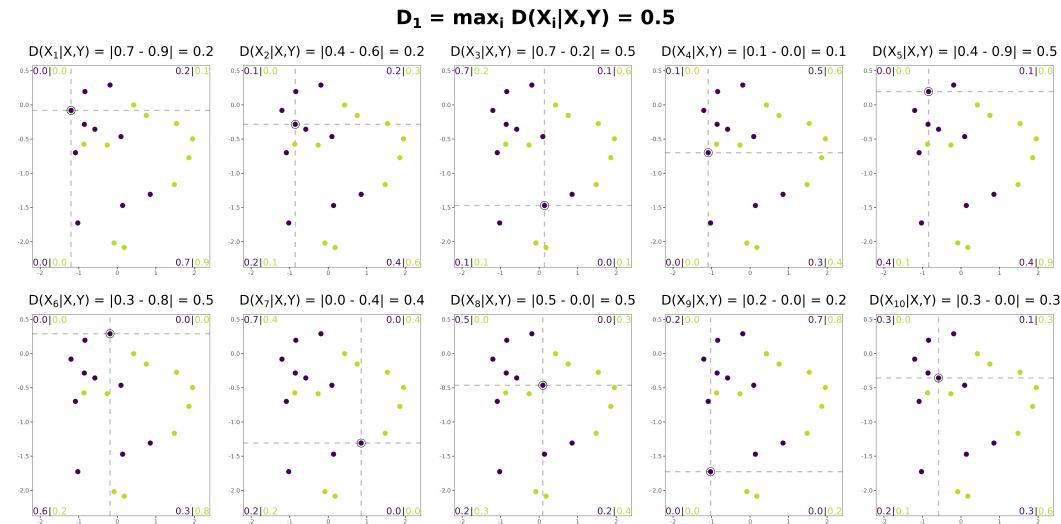


Figure 1: Illustration of the computation of the difference statistic D_1 in two dimensions. Each point in the first sample is used to divide the plane into four quadrants, and both samples are cumulated in each of the four quadrants. The fraction of each sample in each quadrant is shown in the corresponding plot corner, and the maximum difference over all four quadrants is shown above each plot. D_1 is taken as the maximum of these differences. To compute the test statistic, we would next compute D_2 by repeating the same procedure, but using points in the second sample to divide the plane instead.

2.2 Computational complexity

The bulk of the time required to compute the test statistic in (3) is spent evaluating sums of the form

$$\sum_{\mathbf{x} \in S} I_f(\mathbf{x} | \mathbf{y}),$$

which count the number of points in a set S that lie in a given d -dimensional region. The simplest approach to computing such sums is brute force, where every point $\mathbf{x} \in S$ is checked independently. The orthant a point lies in can be determined using d binary checks, resulting in a time complexity of $O(N^2)$, where $N = \max(n_1, n_2)$, to evaluate (3) for fixed d .

Alternatively, we can consider each sum as a single query rather than a sequence of independent ones. Specifically, both sums in (1) are orthogonal range counting queries, which ask how many points in a set $S \subset \mathbb{R}^d$ lie in an axis-aligned box $(x_1, x'_1) \times \dots \times (x_d, x'_d)$. Range counting is an important problem in the field of computational geometry, and as such a variety of data structures have been described to provide efficient solutions (de Berg et al., 2008). One solution, first introduced by Bentley (1979), is a multi-layer binary search tree termed a range tree. Other slightly more efficient data structures have been proposed for range counting, but range trees are well suited for our purposes, particularly because their construction scales easily to arbitrary dimensions (Bentley, 1979; de Berg et al., 2008).

A range tree can be constructed on a set of n points in d -dimensional space using $O(n \log^{d-1} n)$ space in $O(n \log^{d-1} n)$ time. The number of points that lie in an axis-aligned box can be reported in $O(\log^d n)$ time, and this time can be further reduced to $O(\log^{d-1} n)$ when $d > 1$ using fractional cascading (de Berg et al., 2008). To compute (3), we construct one range tree for each of the two

samples, and then query each tree 2^d times. Thus the total time complexity to compute the test statistic using range trees for fixed d is $O(N \log^{d-1} N)$, where $N = \max(n_1, n_2)$.

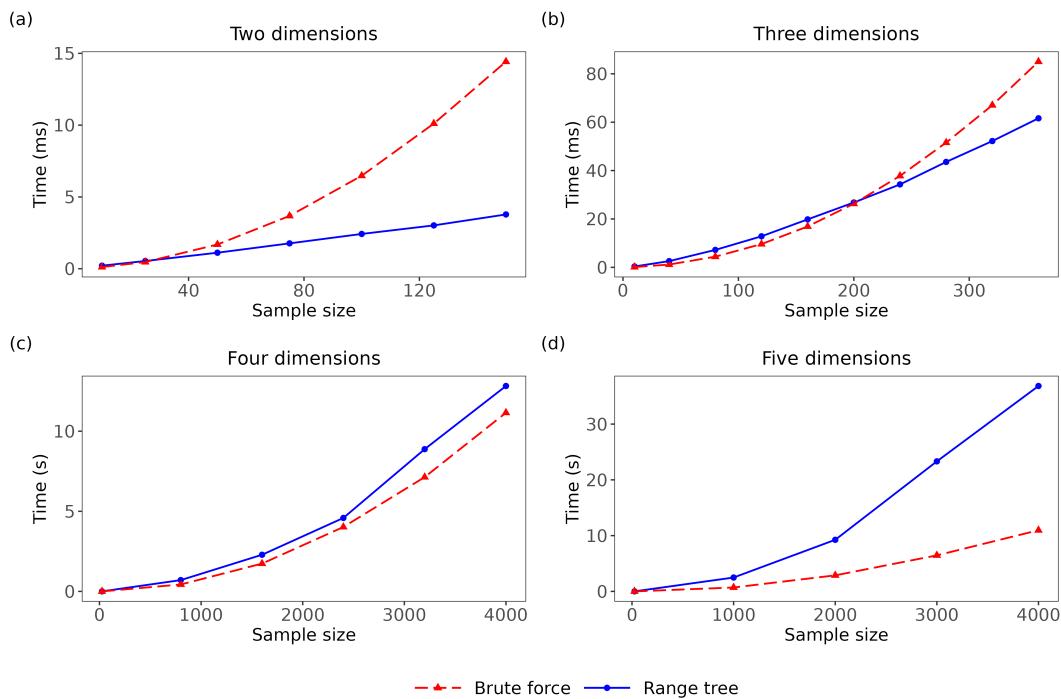


Figure 2: Time to compute the Fasano–Franceschini test statistic as a function of sample size, comparing the brute force and range tree methods for data of dimensions two through five. Points represent the mean time of 200 evaluations. Samples are taken to be of the same size and are drawn from multivariate standard normal distributions.

As the range tree method has a better asymptotic time complexity, we expect it to outperform the brute force method for larger sample sizes. However, for smaller sample sizes, the cost of building the range trees can outweigh the benefit gained by more efficient querying. As exact computation times can vary depending on the geometry of the samples, it is not possible to determine in general when one method will outperform the other. Despite this, we sought to establish rough benchmarks. Drawing equal sized samples from multivariate standard normal distributions, we sought to determine the sample size N^* at which the range tree method becomes more efficient than the brute force method (Figure 2). For $d = 2$, $N^* \approx 25$; for $d = 3$, $N^* \approx 200$; for $d = 4, 5$, and presumably all higher dimensions, $N^* > 4000$. Based on these benchmarking results, our package automatically selects which of the two methods is likely faster based on the dimension and samples sizes of the supplied data. If users are interested in performing more precise benchmarking for their specific dataset, the argument `nPermute` can be set equal to 0, which bypasses the permutation test and only computes the test statistic.

2.3 Significance testing

To the best of our knowledge, no results have been published concerning the distribution of the Fasano–Franceschini test statistic. Any analysis would likely be complicated by the fact that, unlike the KS test statistic, the Fasano–Franceschini test statistic is not distribution free (Fasano and Franceschini, 1987). In their original paper, Fasano and Franceschini (1987) did not attempt any analytical analysis and instead performed simulations to estimate critical values of their test statistic for various two- and three-dimensional distributions. By fitting a curve to their results, Press et al. (2007) proposed an explicit formula for p -values in the two-dimensional case. However, this formula is only approximate, and its accuracy degrades as sample sizes decrease or the true p -value becomes large (greater than 0.2). While this would still allow a simple rejection decision at any common significance level, it is sometimes useful to quantify large p -values more exactly (such as if one was to do a cross-study concordance analysis comparing p -values between studies as in Ness-Cohn et al. 2020). Effort could be made to improve this approximation, however it is still only valid in two dimensions, and thus an alternative method would be needed in higher dimensions.

To ensure the broadest applicability of the test, we assess significance using a permutation test. Let

$\mathbf{Z} = (\mathbf{Z}_1, \dots, \mathbf{Z}_N)$ be defined by

$$\mathbf{Z}_i = \begin{cases} \mathbf{X}_i, & 1 \leq i \leq n_1 \\ \mathbf{Y}_{i-n_1}, & n_1 + 1 \leq i \leq N \end{cases}$$

where $N = n_1 + n_2$. The test statistic in (3) can then be written as

$$\mathcal{D}(\mathbf{Z}) = \mathcal{D}(\mathbf{X}, \mathbf{Y}).$$

Denote the symmetric group on $\{1, \dots, n\} \subset \mathbb{N}$ by S_n . For $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ and $\sigma \in S_n$, define

$$\mathbf{x}_\sigma = (\mathbf{x}_{\sigma(1)}, \dots, \mathbf{x}_{\sigma(n)}).$$

Under the null hypothesis, \mathbf{X} and \mathbf{Y} were drawn from the same distribution, and thus the elements of \mathbf{Z} are exchangeable. We can therefore compute the permutation test p-value

$$p = \frac{\sum_{\sigma \in S_N} I(\mathcal{D}(\mathbf{Z}_\sigma) \geq \mathcal{D}(\mathbf{Z}))}{N!} \quad (4)$$

where I denotes the indicator function (Hemerik and Goeman, 2018; Ramdas et al., 2022). As it is generally infeasible to iterate over all $N!$ permutations, we can instead consider for $M \in \mathbb{N}$

$$p_M = \frac{1 + \sum_{m=1}^M I(\mathcal{D}(\mathbf{Z}_{\sigma_m}) \geq \mathcal{D}(\mathbf{Z}))}{1 + M} \quad (5)$$

where $\sigma_1, \dots, \sigma_M$ are independent permutations drawn uniformly from S_N . This p -value is valid, as under the null hypothesis

$$\mathbb{P}(p_m \leq \alpha) \leq \alpha \forall \alpha \in [0, 1].$$

Moreover, $p_M \rightarrow p$ almost surely. These results hold for any sampling distributions (continuous, discrete, or mixed) and any valid test statistic (Hemerik and Goeman, 2018; Ramdas et al., 2022).

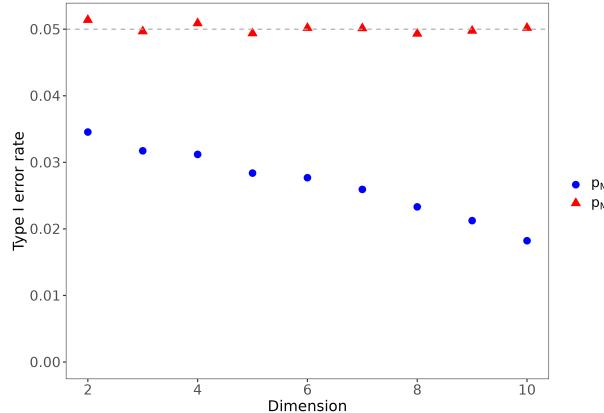


Figure 3: Type I error rate of the test using p_M and p'_M as dimension increases. Samples are both of size 10 and are drawn from standard multivariate normal distributions of the specified dimension. The number of permutations used is 100, and the error rate is estimated using 10^5 replications.

Under certain conditions (see Proposition 2 of Hemerik and Goeman 2018), the permutation test using the p -value p_M is exact, which is to say that under the null hypothesis

$$\mathbb{P}(p_m \leq \alpha) = \alpha \forall \alpha \in [0, 1].$$

However, when there is a nonzero probability of ties in the test statistic, this test can be quite conservative (Hemerik and Goeman, 2018). In such cases, the test can be made exact by instead using

$$p'_M = \frac{\sum_{m=1}^M I(\mathcal{D}(\mathbf{Z}_{\sigma_m}) > \mathcal{D}(\mathbf{Z}))}{1 + M} + U \frac{1 + \sum_{m=1}^M I(\mathcal{D}(\mathbf{Z}_{\sigma_m}) = \mathcal{D}(\mathbf{Z}))}{1 + M}, \quad (6)$$

where $U \sim \text{Unif}(0, 1)$ (Hoeffding, 1952; Hemerik and Goeman, 2018). The fact that p'_M is randomized is not inherently problematic, since it is already randomized due to the selection of random permutations (Hemerik and Goeman, 2018).

As the test statistic in (3) is discrete, ties are possible, and thus the test using p_M is generally conservative. In high dimensions, ties can become quite prevalent, leading the type I error rate to

decrease dramatically (Figure 3). We thus use p'_M as our p -value instead of p_M . As a final remark, we note that if the test statistic is scaled by a constant scalar, p'_M remains unchanged since both indicator functions are invariant under scalar rescaling of \mathcal{D} . Therefore, the outcome of the test does not depend on our choice to use the integer-valued test statistic \mathcal{D} in (3) over Fasano and Franceschini's original test statistic \mathcal{D}_0 in (2).

3 Package overview

The **fasano.franceschini.test** package is written primarily in C++, and interfaces with R using **Rcpp** (Eddelbuettel et al., 2022). The C++ range tree class (Weihs, 2020) was based on the description in de Berg et al. (2008), including an implementation of fractional cascading. The permutation test is parallelized using **RcppParallel** (Allaire et al., 2022). The package consists of one function, **fasano.franceschini.test**, for performing the two-sample Fasano–Franceschini test. The arguments of this function are described below.

- S1 and S2: the two samples to compare. Both should be either numeric `matrix` or `data.frame` objects with the same number of columns.
- nPermute: the number of permutations to use for performing the permutation test. The default is 100. If set equal to 0, the permutation test is bypassed and only the test statistic is computed.
- threads: the number of threads to use when performing the permutation test. The default is one thread. This parameter can also be set to "auto", which uses the value returned by `RcppParallel::defaultNumThreads()`.
- seed: an optional seed for the pseudorandom number generator (PRNG) used during the permutation test.
- verbose: whether to display a progress bar while performing the permutation test. The default is TRUE. This functionality is only available when `threads = 1`.
- method: an optional character indicating which method to use to compute the test statistic. The two methods are 'r' (range tree) and 'b' (brute force). Both methods return the same results but may vary in computation speed. If this argument is not passed, the sample sizes and dimension of the data are used to infer which method is likely faster.

The output is an object of the class `htest`, and consists of the following components:

- `statistic`: the value of the test statistic.
- `p.value`: the permutation test p-value.
- `method`: the name of the test (i.e. 'Fasano–Franceschini Test').
- `data.name`: the names of the original data objects.

4 Examples

Here we demonstrate the basic usage and features of the **fasano.franceschini.test** package. We begin by loading the necessary libraries and setting a seed for reproducibility.

```
> library(fasano.franceschini.test)
> library(MASS)
> set.seed(0)
```

Note that to produce reproducible results, we need to set two seeds: the `set.seed` function sets the seed in R, ensuring we draw reproducible samples; and the seed passed as an argument to the `fasano.franceschini.test` function sets the seed for the C++ PRNG, ensuring we compute reproducible p -values.

As a first example, we draw two samples from the bivariate standard normal distribution. The Fasano–Franceschini test fails to reject the null hypothesis — that the samples were drawn from the same distribution — at an $\alpha = 0.05$ significance level.

```
> S1 <- mvrnorm(n = 50, mu = c(0, 0), Sigma = diag(2))
> S2 <- mvrnorm(n = 75, mu = c(0, 0), Sigma = diag(2))
> fasano.franceschini.test(S1, S2, seed = 1, verbose = FALSE)
```

```
Fasano–Franceschini Test
```

```
data: S1 and S2
D = 1425, p-value = 0.4653
```

We next draw two samples from bivariate normal distributions with identical covariance matrices but different locations. The test rejects the null hypothesis at an $\alpha = 0.05$ significance level.

```
> S3 <- mvrnorm(n = 40, mu = c(0, 0), Sigma = diag(2))
> S4 <- mvrnorm(n = 42, mu = c(1, 1), Sigma = diag(2))
> fasano.franceschini.test(S3, S4, seed = 2, verbose = FALSE)
```

Fasano–Franceschini Test

```
data: S3 and S4
D = 1932, p-value = 0.001832
```

The test can take a while to run when the sample sizes or the dimension of the data are large, in which case it is useful to use multiple threads to speed up computation.

```
> S5 <- mvrnorm(n = 1000, mu = c(1, 3, 5), Sigma = diag(3) + 1)
> S6 <- mvrnorm(n = 600, mu = c(1, 3, 5), Sigma = diag(3))
> fasano.franceschini.test(S5, S6, seed = 3, threads = 4)
```

Fasano–Franceschini Test

```
data: S5 and S6
D = 263800, p-value = 0.0007002
```

Note that the number of threads used does not affect the results. In particular, as long as the same seed is used, the same p -value is returned for any number of threads.

```
> fasano.franceschini.test(S5, S6, seed = 3, threads = 1)
```

Fasano–Franceschini Test

```
data: S5 and S6
D = 263800, p-value = 0.0007002
```

5 Comparison with other R packages

In this section, we compare the **fasano.franceschini.test** package with three other CRAN packages that perform multivariate two-sample goodness-of-fit tests.

5.1 Peacock.test

The **Peacock.test** package (Xiao, 2016) provides functions to compute Peacock’s test statistic (Peacock, 1983) in two and three dimensions. As no function is provided to compute p -values, we cannot directly compare the performance of this package with the **fasano.franceschini.test** package. However, a treatment of the power of both Peacock and Fasano–Franceschini tests can be found in both the primary literature (Peacock, 1983; Fasano and Franceschini, 1987) and in a subsequent benchmarking paper (Lopes et al., 2007), which found that the two tests have similar power across a variety of alternatives.

5.2 cramer

The **cramer** package (Franz, 2019) implements the two-sample test described in Baringhaus and Franz (2004), which the authors refer to as the Cramér test. The Cramér test statistic is based on the Euclidean inter-point distances between the two samples, and is given by

$$T_{m,n} = \frac{mn}{m+n} \left(\frac{2}{mn} \sum_{i=1}^m \sum_{j=1}^n \phi \left(\left\| \mathbf{x}_i - \mathbf{y}_j \right\|_2^2 \right) - \frac{1}{m^2} \sum_{i,j=1}^m \phi \left(\left\| \mathbf{x}_i - \mathbf{x}_j \right\|_2^2 \right) - \frac{1}{n^2} \sum_{i,j=1}^n \phi \left(\left\| \mathbf{y}_i - \mathbf{y}_j \right\|_2^2 \right) \right)$$

for samples $\mathbf{X}_1, \dots, \mathbf{X}_m$ and $\mathbf{Y}_1, \dots, \mathbf{Y}_n$. The default kernel function is $\phi(x) = \sqrt{x}/2$, although several other choices are implemented (see the documentation for more details). Several randomization

methods are provided to compute p -values, including bootstrapping (the default) and a permutation test.

5.3 diproperm

The **diproperm** package (Allmon et al., 2021) implements the DiProPerm test introduced by Wei et al. (2016). A binary linear classifier is first trained to determine a separating hyperplane between the two samples. The data are then projected onto the normal vector to the hyperplane, and the test statistic is taken to be a univariate statistic of the projected data (by default the absolute difference of means). As in the **fasano.franceschini.test** package, significance is determined using a permutation test.

5.4 Power comparison

To compare the **fasano.franceschini.test** package with the **cramer** and **diproperm** packages, we performed power analyses using three classes of alternatives: location alternatives, where the means of the marginals are varied; dispersion alternatives, where the variances of the marginals are varied; and copula alternatives, where the marginals remain fixed but the copula joining them is varied.

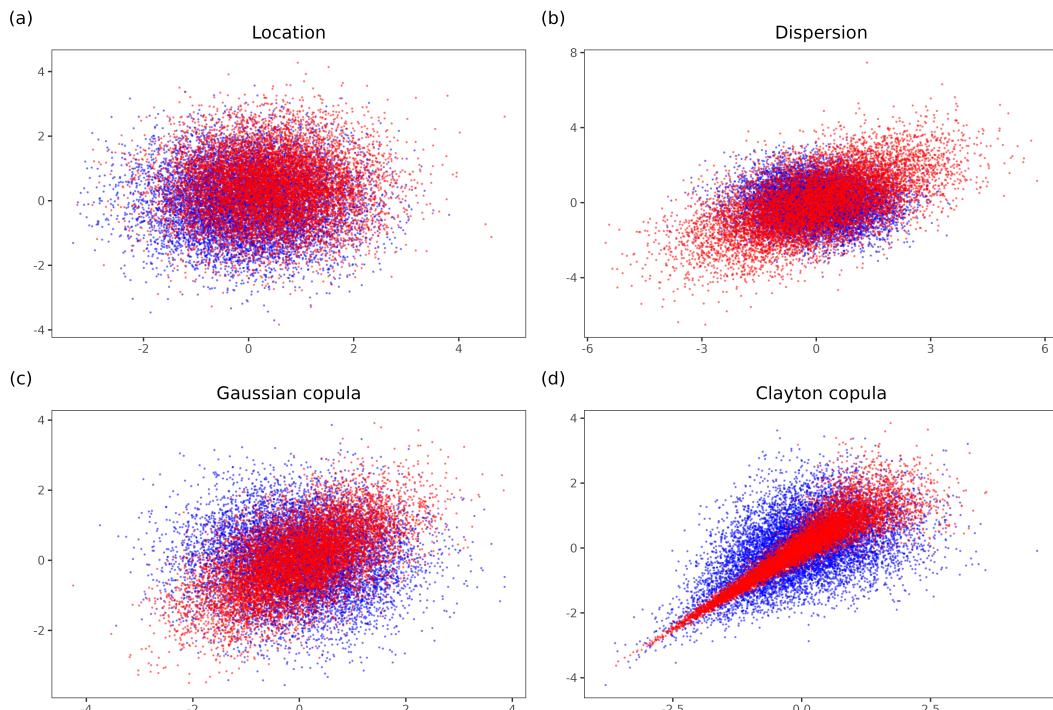


Figure 4: Visualization of the distributions used in power analyses. Each plot shows two samples consisting of 10000 points each. The first sample S_1 is shown in blue, and the second sample S_2 is shown in red. (a) $S_1 \sim N_2(\mathbf{0}, \mathbf{I}_2)$ and $S_2 \sim N_2(\mathbf{0.4}, \mathbf{I}_2)$. (b) $S_1 \sim N_2(\mathbf{0}, \mathbf{I}_2)$ and $S_2 \sim N_2(\mathbf{0}, \mathbf{I}_2 + 1.5)$. (c) $S_1 \sim G_2(0)$ and $S_2 \sim G_2(0.6)$. (d) $S_1 \sim C_2(1)$ and $S_2 \sim C_2(8)$.

For location and dispersion alternatives, we used multivariate normal distributions. We denote the d -dimensional normal distribution with mean $\mu \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$ by $N_d(\mu, \Sigma)$, and sample from it using the **MASS** package (Ripley, 2021). The $d \times d$ identity matrix, which is sometimes used as a covariance matrix, is denoted by \mathbf{I}_d . For copula alternatives, we consider the Gaussian copula with correlation matrix

$$[\mathbf{P}(\rho)]_{ij} = \begin{cases} \rho, & i \neq j \\ 1, & i = j \end{cases}$$

and the Clayton copula with parameter $\theta \in [-1, \infty) \setminus \{0\}$. We denote the d -dimensional distribution with standard normal marginals joined by a Gaussian copula with correlation matrix $\mathbf{P}(\rho)$ by $G_d(\rho)$. We denote the d -dimensional distribution with standard normal marginals joined by a Clayton copula with parameter θ by $C_d(\theta)$. Both distributions are sampled from using the **copula** package (Hofert et al., 2022). Examples of distributions in each of these four families are shown in Figure 4.

In the following analyses, power was approximated using 1000 replications, a significance level of $\alpha = 0.05$ was used, all samples were of size 40, and all R functions implementing tests were called using their default arguments. Although we aimed to cover a wide range of distributions in this analysis, absent any theoretical results concerning these three tests, we cannot guarantee that the results here are generalizable to different sampling distributions or sample sizes.

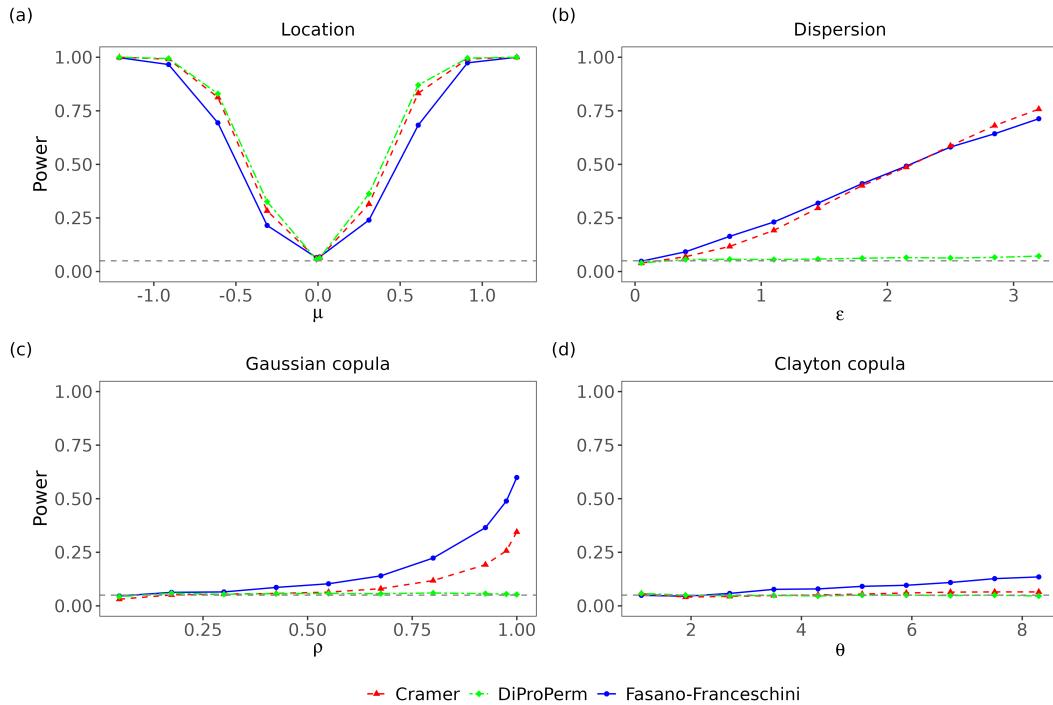


Figure 5: Comparison of power of the Fasano–Franceschini, Cramér, and DiProPerm tests on various bivariate alternatives. (a) Location alternatives, with $S_1 \sim N_2(\mathbf{0}, \mathbf{I}_2)$ and $S_2 \sim N_2(\mu, \mathbf{I}_2)$. (b) Dispersion alternatives, with $S_1 \sim N_2(\mathbf{0}, \mathbf{I}_2)$ and $S_2 \sim N_2(\mathbf{0}, \mathbf{I}_2 + \varepsilon)$. (c) Gaussian copula alternatives, with $S_1 \sim G_2(0)$ and $S_2 \sim G_2(\rho)$. (d) Clayton copula alternatives, with $S_1 \sim C_2(1)$ and $S_2 \sim C_2(\theta)$.

We first examined the power of the tests on various bivariate alternatives (Figure 5). All three tests had similar power across location alternatives, although the Cramér and DiProPerm tests did outperform the Fasano–Franceschini test. Across dispersion alternatives, the Cramér and Fasano–Franceschini tests had very similar powers. On Gaussian copula alternatives, the Fasano–Franceschini test had a consistently higher power than the Cramér test. This was also the case with Clayton copula alternatives, although none of the tests were able to achieve high power. The DiProPerm test was unable to achieve a power above the significance level of $\alpha = 0.05$ on any of the dispersion or copula alternatives. This is likely due to the fact that in these instances, there is significant overlap between the high density regions of the two sampling distributions, making it difficult to find a separating hyperplane between samples drawn from them.

We next examined how the power of the three tests varied when the two sampling distributions were kept fixed but the dimension of the data increased (Figure 6). On the location alternative, the Cramér and DiProPerm tests again outperformed the Fasano–Franceschini test. In particular, for $d > 5$, the Fasano–Franceschini steadily lost power as the dimension increased whereas the other tests gained power. On the dispersion alternative, the Cramér and Fasano–Franceschini tests had nearly identical powers through to $d = 5$, but for higher dimensions the Cramér test consistently outperformed the Fasano–Franceschini test. On the other hand, for both the Gaussian and Clayton copula alternatives the Fasano–Franceschini test had a much higher power than the Cramér test. The DiProPerm test was still unable to attain a power above the significance level on the dispersion alternatives or either of the copula alternatives.

Overall, the Cramér and DiProPerm tests performed better than the Fasano–Franceschini test on location alternatives, especially as dimension increased. On dispersion alternatives, the Fasano–Franceschini and Cramér tests had comparable performance for low dimensions, but the Cramér test maintained a higher power for high dimensions. However, in these cases the marginal distributions differ, and thus a multivariate test is not strictly necessary as univariate tests could be applied to the marginals independently (with a multiple testing correction) to detect differences between the multivariate distributions. On copula alternatives, where a multivariate test is necessary, the Fasano–

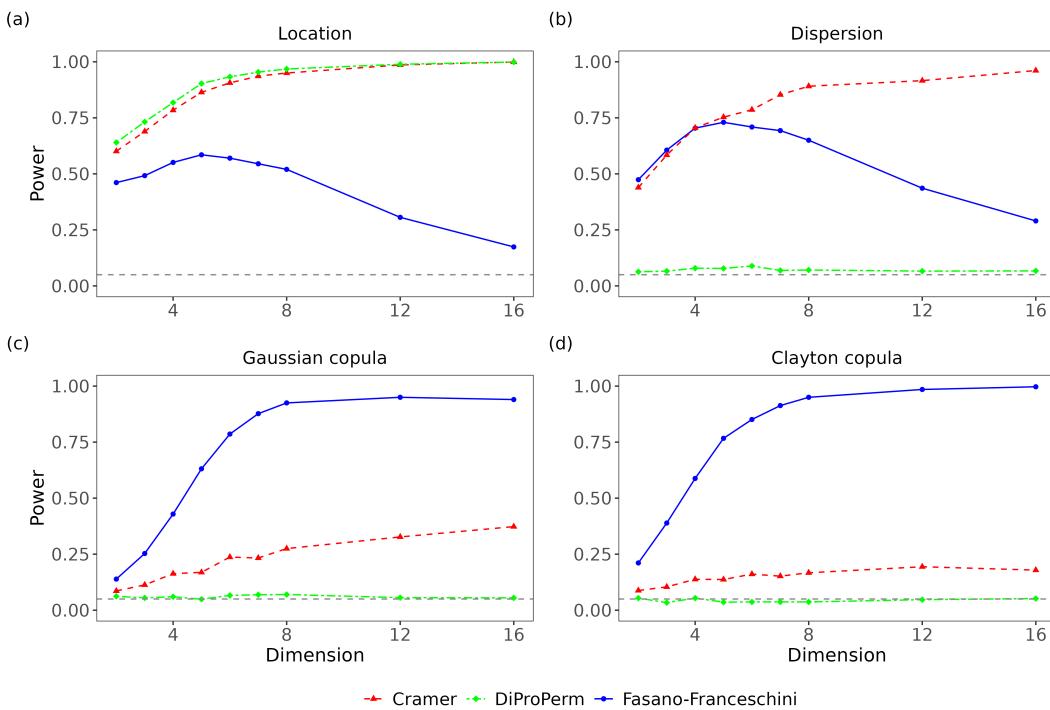


Figure 6: Comparison of power of the Fasano–Franceschini, Cramér, and DiProPerm tests on fixed alternatives as the dimension of the data increases. (a) Location alternative, with $S_1 \sim N_d(\mathbf{0}, \mathbf{I}_d)$ and $S_2 \sim N_d(\mathbf{0.4}, \mathbf{I}_d)$. (b) Dispersion alternative, with $S_1 \sim N_d(\mathbf{0}, \mathbf{I}_d)$ and $S_2 \sim N_d(\mathbf{0}, \mathbf{I}_d + 1.5)$. (c) Gaussian copula alternative, with $S_1 \sim G_d(0)$ and $S_2 \sim G_d(0.6)$. (d) Clayton copula alternative, with $S_1 \sim C_d(1)$ and $S_2 \sim C_d(8)$.

Franceschini test consistently outperformed both the Cramér and DiProPerm tests. Thus while the Fasano–Franceschini did not achieve the highest power in every case, we believe it to be the best choice as a general purpose multivariate two-sample goodness-of-fit test.

6 Summary

This paper introduces the `fasano.franceschini.test` package, an R implementation of the multivariate two-sample goodness-of-fit test described by Fasano and Franceschini (1987). We provide users with a computationally efficient test that is applicable to data of any dimension and of any type (continuous, discrete, or mixed), and that demonstrates competitive performance with similar R packages. Complete package documentation and source code are available via the Comprehensive R Archive Network (CRAN) at <https://cran.r-project.org/web/packages/fasano.franceschini.test> and the package website at <https://braunlab-nu.github.io/fasano.franceschini.test>.

7 Computational details

The results in this paper were obtained using R 4.2.0 with the packages `fasano.franceschini.test` 2.2.1, `diproperm` 0.2.0, `cramer` 0.9-3, `MASS` 7.3-60, `copula` 1.1-2, and `microbenchmark` 1.4.10 (Mersmann, 2023). Plots were generated using `ggplot2` 3.4.2 (Wickham et al., 2023) and `patchwork` 1.1.1 (Pedersen, 2020). All computations were done using the Quest high performance computing facility at Northwestern University.

8 Acknowledgments

This research was supported in part through the computational resources and staff contributions provided for the Quest high performance computing facility at Northwestern University which is jointly supported by the Office of the Provost, the Office for Research, and Northwestern University

Information Technology. Funding for this work was provided by NIH/NIA R01AG068579, Simons Foundation 597491-RWC01, and NSF 1764421-01.

References

- J. Allaire, R. Francois, K. Ushey, G. Vandenbrouck, M. Geelnard, and Intel. *RcppParallel: Parallel Programming Tools for 'Rcpp'*, 2022. URL <https://CRAN.R-project.org/package=RcppParallel>. R package version 5.1.5. [p164]
- A. G. Allmon, J. Marron, and M. G. Hudgens. *diproperm: Conduct Direction-Projection-Permutation Tests and Display Plots*, 2021. URL <https://CRAN.R-project.org/package=diproperm>. R package version 0.2.0. [p166]
- S. Atasoy, L. Roseman, M. Kaelen, M. L. Kringelbach, G. Deco, and R. L. Carhart-Harris. Connectome-harmonic decomposition of human brain activity reveals dynamical repertoire re-organization under LSD. *Scientific Reports*, 7(1):1–18, 2017. URL <https://doi.org/10.1038/s41598-017-17546-0>. [p159]
- L. Baringhaus and C. Franz. On a new multivariate two-sample test. *Journal of Multivariate Analysis*, 88(1):190–206, 2004. ISSN 0047-259X. URL [https://doi.org/10.1016/S0047-259X\(03\)00079-4](https://doi.org/10.1016/S0047-259X(03)00079-4). [p165]
- J. L. Bentley. Decomposable searching problems. *Information Processing Letters*, 8(5):244–251, 1979. ISSN 0020-0190. URL [https://doi.org/10.1016/0020-0190\(79\)90117-0](https://doi.org/10.1016/0020-0190(79)90117-0). [p161]
- F. Chiang, O. Mazdiyasni, and A. AghaKouchak. Amplified warming of droughts in southern united states in observations and model simulations. *Science Advances*, 4(8):eaat2380, 2018. URL <https://doi.org/10.1126/sciadv.aat2380>. [p159]
- M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer Berlin Heidelberg, 3rd edition, 2008. ISBN 978-3-540-77974-2. URL <https://doi.org/10.1007/978-3-540-77974-2>. [p161, 164]
- D. Eddelbuettel, R. Francois, J. Allaire, K. Ushey, Q. Kou, N. Russell, I. Ucar, D. Bates, and J. Chambers. *Rcpp: Seamless R and C++ Integration*, 2022. URL <https://CRAN.R-project.org/package=Rcpp>. R package version 1.0.9. [p164]
- G. Fasano and A. Franceschini. A multidimensional version of the Kolmogorov-Smirnov test. *Monthly Notices of the Royal Astronomical Society*, 225(1):155–170, 03 1987. ISSN 0035-8711. URL <https://doi.org/10.1093/mnras/225.1.155>. [p159, 160, 162, 165, 168]
- C. Franz. *cramer: Multivariate Nonparametric Cramer-Test for the Two-Sample-Problem*, 2019. URL <https://CRAN.R-project.org/package=cramer>. R package version 0.9-3. [p165]
- J. M. Hahne, M. A. Schweisfurth, M. Koppe, and D. Farina. Simultaneous control of multiple functions of bionic hand prostheses: Performance and robustness in end users. *Science Robotics*, 3(19):eaat3630, 2018. URL <https://doi.org/10.1126/scirobotics.aat3630>. [p159]
- J. Hemerik and J. Goeman. Exact testing with random permutations. *TEST*, 27(4):811–825, Dec 2018. ISSN 1863-8260. URL <https://doi.org/10.1007/s11749-017-0571-1>. [p163]
- C. Heuchenne and G. Mordant. Using space filling curves to compare two multivariate distributions with distribution-free tests. *Journal of Computational and Applied Mathematics*, 416:114494, Dec. 2022. ISSN 0377-0427. URL <https://doi.org/10.1016/j.cam.2022.114494>. [p159]
- W. Hoeffding. The Large-Sample Power of Tests Based on Permutations of Observations. *The Annals of Mathematical Statistics*, 23(2):162–192, 1952. ISSN 0003-4851. URL <https://doi.org/10.1214/aoms/1177729436>. [p163]
- M. Hofert, I. Kojadinovic, M. Maechler, and J. Yan. *copula: Multivariate Dependence with Copulas*, 2022. URL <https://CRAN.R-project.org/package=copula>. R package version 1.1-0. [p166]
- A. Justel, D. Peña, and R. Zamar. A multivariate Kolmogorov-Smirnov test of goodness of fit. *Statistics & Probability Letters*, 35(3):251–259, 1997. ISSN 0167-7152. URL [https://doi.org/10.1016/S0167-7152\(97\)00020-5](https://doi.org/10.1016/S0167-7152(97)00020-5). [p159]

- S. Kaczanowska, D. W. Beury, V. Gopalan, A. K. Tycko, H. Qin, M. E. Clements, J. Drake, C. Nwanze, M. Murgai, Z. Rae, W. Ju, K. A. Alexander, J. Kline, C. F. Contreras, K. M. Wessel, S. Patel, S. Hannenhalli, M. C. Kelly, and R. N. Kaplan. Genetically engineered myeloid cells rebalance the core immune suppression program in metastasis. *Cell*, 184(8):2033–2052.e21, 2021. ISSN 0092-8674. URL <https://doi.org/10.1016/j.cell.2021.02.048>. [p159]
- A. N. Kolmogorov. Sulla Determinazione Empirica di Una Legge di Distribuzione. *Giornale dell'Istituto Italiano degli Attuari*, 4:83–91, 1933a. [p159]
- A. N. Kolmogorov. Über die Grenzwertsätze der Wahrscheinlichkeitsrechnung. *Bull. Acad. Sci. URSS*, 3:363–372, 1933b. URL <https://www.mathnet.ru/eng/im5009>. [p159]
- R. H. C. Lopes, I. Reid, and P. R. Hobson. The two-dimensional Kolmogorov-Smirnov test. In *XI International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, 2007. URL <https://bura.brunel.ac.uk/handle/2438/1166>. [p159, 165]
- O. Mersmann. *microbenchmark: Accurate Timing Functions*, 2023. URL <https://CRAN.R-project.org/package=microbenchmark>. R package version 1.4.10. [p168]
- M. Naaman. On the tight constant in the multivariate Dvoretzky–Kiefer–Wolfowitz inequality. *Statistics & Probability Letters*, 173:109088, 2021. ISSN 0167-7152. URL <https://doi.org/10.1016/j.spl.2021.109088>. [p159]
- E. Ness-Cohn, M. Iwanaszko, W. L. Kath, R. Allada, and R. Braun. TimeTrial: An Interactive Application for Optimizing the Design and Analysis of Transcriptomic Time-Series Data in Circadian Biology Research. *Journal of Biological Rhythms*, 35(5):439–451, 2020. URL <https://doi.org/10.1177/0748730420934672>. PMID: 32613882. [p162]
- J. A. Peacock. Two-dimensional goodness-of-fit testing in astronomy. *Monthly Notices of the Royal Astronomical Society*, 202(3):615–627, 03 1983. ISSN 0035-8711. URL <https://doi.org/10.1093/mnras/202.3.615>. [p159, 165]
- T. L. Pedersen. *patchwork: The Composer of Plots*, 2020. URL <https://CRAN.R-project.org/package=patchwork>. R package version 1.1.1. [p168]
- W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, USA, 3rd edition, 2007. ISBN 0521880688. URL <https://doi.org/10.1145/1874391.187410>. [p162]
- A. Ramdas, R. F. Barber, E. J. Candes, and R. J. Tibshirani. Permutation tests using arbitrary permutation distributions. 2022. URL <https://doi.org/10.48550/arXiv.2204.13581>. [p163]
- B. Ripley. *MASS: Support Functions and Datasets for Venables and Ripley's MASS*, 2021. URL <https://CRAN.R-project.org/package=MASS>. R package version 7.3-54. [p166]
- N. V. Smirnov. Sur la distribution de ω^2 (criterium de M.R. v. Mises). *Com. Rend. Acad. Sci. (Paris)*, 202: 449–452, 1936. [p159]
- N. V. Smirnov. On the distribution of the mises ω^2 criterion [in Russian]. *Rec. Math. N.S. [Mat. Sbornik]*, 2:973–993, 1937. URL <https://www.mathnet.ru/eng/sm5636>. [p159]
- N. V. Smirnov. On the deviations of the empirical distribution curve [in Russian]. *Rec. Math. N.S. [Mat. Sbornik]*, 6(48):3–26, 1939. URL <https://www.mathnet.ru/eng/sm5810>. [p159]
- N. V. Smirnov. Approximate laws of distribution of random variables from empirical data. *Uspehi Matem. Nauk*, 10:179–206, 1944. [p159]
- N. V. Smirnov. Table for estimating the goodness of fit of empirical distributions. *The Annals of Mathematical Statistics*, 1948. ISSN 0003-4851. URL <https://doi.org/10.1214/aoms/1177730256>. [p159]
- S. Wei, C. Lee, L. Wicher, and J. S. Marron. Direction-Projection-Permutation for High-Dimensional Hypothesis Tests. *Journal of Computational and Graphical Statistics*, 25(2):549–569, 2016. URL <https://doi.org/10.1080/10618600.2015.1027773>. [p166]
- L. Weihs. C++ Range Tree Data Structure, 2020. URL <https://github.com/Lucaweihs/range-tree>. [p164]

- H. Wickham, W. Chang, L. Henry, T. L. Pedersen, K. Takahashi, C. Wilke, K. Woo, H. Yutani, and D. Dunnington. *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*, 2023. URL <https://CRAN.R-project.org/package=ggplot2>. R package version 3.4.2. [p168]
- F. Wong and J. J. Collins. Evidence that coronavirus superspreading is fat-tailed. *Proceedings of the National Academy of Sciences*, 117(47):29416–29418, 2020. URL <https://doi.org/10.1073/pnas.2018490117>. [p159]
- Y. Xiao. *Peacock.test: Two and Three Dimensional Kolmogorov-Smirnov Two-Sample Tests*, 2016. URL <https://CRAN.R-project.org/package=Peacock.test>. R package version 1.0. [p165]

Connor Puritz

*Department of Engineering Sciences and Applied Mathematics, Northwestern University
Evanston, IL 60208*
ORCID: 0000-0001-7602-0444
connorpuritz2025@u.northwestern.edu

Elan Ness-Cohn

*Department of Molecular Biosciences, Northwestern University
Evanston, IL 60208*
ORCID: 0000-0002-3935-6667
elan.ness-cohn@northwestern.edu

Rosemary Braun

*Department of Molecular Biosciences, Northwestern University
Evanston, IL 60208*
ORCID: 0000-0001-9668-9866
rbraun@northwestern.edu

Bayesian Inference for Multivariate Spatial Models with INLA

by Francisco Palmí-Perales, Virgilio Gómez-Rubio, Roger S. Bivand, Michela Cameletti, and Håvard Rue

Abstract Bayesian methods and software for spatial data analysis are well-established now in the broader scientific community generally and in the spatial data analysis community specifically. Despite the wide application of spatial models, the analysis of multivariate spatial data using the integrated nested Laplace approximation through its R package (R-INLA) has not been widely described in the existing literature. Therefore, the main objective of this article is to demonstrate that R-INLA is a convenient toolbox to analyse different types of multivariate spatial datasets. This will be illustrated by analysing three datasets which are publicly available. Furthermore, the details and the R code of these analyses are provided to exemplify how to fit models to multivariate spatial datasets with R-INLA.

1 Introduction

In recent times, spatial data analysis has increased in popularity due to the widespread availability of different types of geo-referenced data. Spatial models will exploit the geographical information in the data so that spatial dependence is exploited, among other things, to build better predictive models. There is also a growing interest (and application) of multivariate models as often spatial data incorporates multiple variables. For instance, healthcare, econometrics, climate and other fields require multivariate models. One of the main advantages of multivariate models is their ability to find similar spatial patterns in the different response variables. This is the main motivation of this work.

Analysing spatial data poses a number of methodological and computational challenges. In this regard, Bayesian inference has been particularly successful as it has provided data analysts with both a modelling framework and estimation methods for spatial data analysis (see, for example, Banerjee et al., 2014). Bayesian inference for spatial models has often relied upon Markov Chain Monte Carlo (MCMC, Gilks et al., 1995) algorithms, which provide samples from the joint posterior distribution of the model parameters. However flexible, MCMC algorithms have proved slow when dealing with large datasets or highly parameterized models.

For this reason, the integrated nested Laplace approximation (INLA, Rue et al., 2009) has become an alternative approach to fit Bayesian hierarchical models. The advantage of INLA is its ability to approximate the marginal posterior distribution of the different hyperparameters and latent effects of a model that could be parametrised as a latent Gaussian model (Rue and Held, 2005) in less time than MCMC algorithms. This methodology has been implemented in the INLA R package. To avoid confusion between the INLA methodology and the INLA R package, we refer to the package as **R-INLA** (Rue et al., 2020).

The different spatial models which can be fit with **R-INLA** have been summarised by several authors (Lindgren et al., 2015; Blangiardo and Cameletti, 2015; Bakka et al., 2018). For a recent review, the reader is referred to Chapter 7 of Gómez-Rubio (2020). Krainski et al. (2019) provide an exhaustive tutorial about how to fit advanced spatial models in **R-INLA** using the stochastic partial differential equation approach (SPDE, Lindgren et al., 2011) to estimate continuous processes. Additionally, INLA can be combined with other algorithms such as Markov Chain Monte Carlo (MCMC, Gilks et al., 1995) techniques in order to fit models which can not be fit solely with INLA (Gómez-Rubio and Palmí-Perales, 2019).

Therefore, the main objective of this article is to describe how to analyse any multivariate spatial dataset using **R-INLA** in a Bayesian framework. Multivariate spatial models have been studied by several authors. For example, Van Lieshout and Baddeley (1999) describe dependence between multivariate point patterns by proposing novel summary statistics. Additionally, multivariate log-Gaussian Cox processes have been used to analyse multivariate point patterns (Diggle et al., 2013; Waagepetersen et al., 2016; Gómez-Rubio et al., 2015). Furthermore, several studies have been published for analysing multivariate lattice data. For example, in MacNab (2018) an insight into the generalization of univariate models to multivariate models is extensively discussed and in Martínez-Beneito et al. (2017) a framework to analyse multiple response variables is proposed in the context of disease mapping. A review of the multivariate spatial models in disease mapping is performed in Martínez-Beneito

and Botella-Rocamora (2019). These works rely on MCMC methods to develop their examples. In the context of spatial modeling, these methods can lead to a high computational burden.

The R programming ([R Core Team, 2022](#)) language offers a wide range of standalone packages for analysing spatial datasets. Several of them focus on a particular type of spatial data. For instance, point pattern analysis can be performed with [spatstat](#) (Baddeley et al., 2015) and [spatialkernel](#) (Gómez-Rubio et al., 2017). Geostatistical data can be modeled using [gstat](#) (Pebesma and Wesseling, 1998; Pebesma, 2004) or [spBayes](#) (Finley et al., 2007, 2015). Other R packages such as [CARBayes](#) (Lee, 2013) are designed to analyse lattice data.

The remainder of the manuscript is organised as follows. First, a short introduction to the INLA methodology and how the **R-INLA** package fits models to multivariate data is provided. Next, a brief description of the different multivariate models is detailed in the case of areal, continuous and point pattern data in three different sections. Each of these sections discusses the models to be fit, including the prior distribution choice and the structure of the data, and includes an example on how to analyse multivariate spatial data with **R-INLA**. Finally, a brief summary of the conclusions of this work appears in the last part of this manuscript.

2 Integrated nested Laplace approximation

Bayesian inference has been usually performed with MCMC. However, these methods have a high computational cost in a spatial modelling context, mainly because of the simulation-based approach they use to estimate the large number of latent effects present in spatial models. A good alternative is the integrated nested Laplace approximation (INLA, Rue et al., 2009) as it is able to accurately and efficiently approximate posterior distributions of high-dimensional models in a short computing time.

The INLA methodology focuses on the analysis of latent Gaussian models ([Rue and Held, 2005](#)). In essence, a latent Gaussian model is a Bayesian hierarchical model in which the mean parameter of the likelihood is linked to a linear predictor composed of different additive latent effects; the distribution of the vector of latent effects follows a multivariate Gaussian distribution (more details can be found in, for example, Chapter 2 of [Rue and Held, 2005](#)). Latent Gaussian models comprise a large group of statistical models such as many spatial and spatio-temporal models. The **R-INLA** package implements the INLA method in a flexible way, so that models with a number of latent effects can be fit. Additionally, INLA can be combined with other algorithms in order to fit other models ([Gómez-Rubio and Palmí-Perales, 2019](#)).

Gaussian Markov random fields (GMRF) with sparse precision matrices can be used to approximate Gaussian latent fields due to their conditional independence properties. The sparse precision matrices allow computationally efficient methods ([Rue and Held, 2005](#)). This is the reason of the low computational costs of INLA which is one of its main benefits. More details about how to implement Bayesian inference with INLA can be found in ([Gómez-Rubio, 2020](#)).

Model fitting with **R-INLA** is done via the `inla()` function, which works like the `glm()` and `gam()` functions. The model to fit is defined using a formula (that may include different types of random effects in the right-hand side). For example, to fit a Poisson regression with response variable `y` and covariate `x`, both in data frame `d`, the code is:

```
inla(y ~ x, family = "poisson", data = d)
```

The output includes the posterior marginal distribution of the model parameters, as well as summary statistics computed from these distributions. In its example, default priors have been used, but **R-INLA** can work with a good number of prior distributions.

In the previous example, `y` represents the vector of observations of a single variable. However, INLA can deal with multivariate responses so that they can be analyzed jointly. Fitting models with a multivariate responses requires the storage of variables in a particular format. Let us consider the simplest scenario in which a dataset with D variables have been measured in a study region divided into n areas. In order to analyse this dataset with **R-INLA**, a matrix with D columns and $D \times n$ rows has to be built. Specifically, this matrix will store the n values of the first variable in the first column, from the first to the n -th row, then the following values of this column will be NA. Then, the data of the

second variable would be placed in the second column between the $n + 1$ and the $2n$ rows. The rest of the values of the second column will be filled with NA's. Data for other variables will be added to the matrix accordingly.

As a toy example, we will take $n = 2$ and $D = 3$, so that the original dataset is structured as a 2×3 matrix given by

$$\begin{bmatrix} 1 & 4 & 3 \\ 2 & 6 & 5 \end{bmatrix}.$$

Following the above procedure, the matrix that should be passed to **R-INLA** would be the following:

$$\begin{bmatrix} 1 & \text{NA} & \text{NA} \\ 2 & \text{NA} & \text{NA} \\ \text{NA} & 4 & \text{NA} \\ \text{NA} & 6 & \text{NA} \\ \text{NA} & \text{NA} & 3 \\ \text{NA} & \text{NA} & 5 \end{bmatrix}.$$

This data format must be used in the case of the multivariate response but also for multivariate covariates. In particular, multivariate covariates will be included in the model so that values in a particular column only affect values in the same column of the response, i.e., the model will include a specific coefficient for each variable in the response. In the R code provided together with this paper (as a supplementary material), we have provided function `create_multivariate_data()` that will take a matrix or several vectors of data and create the appropriate matrix for R-INLA. When the covariate is expected to have the same effect for all the response variables (i.e., the coefficient will be the same), a single vector with the values of the covariates will be used. This is better explained in the examples that follow.

3 Multivariate Lattice Data

Areal (or lattice) data are obtained when the spatial data are observed on regions with defined boundaries. In this case, the domain is divided into (non-overlapping) areas in which the data are collected. It is usually considered that two areas are neighbours if they share a common boundary. This adjacency structure is often included to account for spatial autocorrelation ([Banerjee et al., 2014](#)).

When the values of several variables are recorded in each area the resulting data become a multivariate lattice dataset. The joint analysis of the spatial distribution of several variables allows to detect similar (spatial) patterns among some of these variables ([Banerjee et al., 2014](#), Chapter 10) while estimating the spatial effects. We will illustrate the analysis of this type of data using a Poisson regression model, which is commonly employed in spatial epidemiology to analyse count data. Other similar models can be proposed for binary or continuous outcomes.

Given the d -th variable of interest (with $d = 1, \dots, D$) and area i (with $i = 1, \dots, n$), the response of interest $Y_{d,i}$ can be modeled using a Poisson distribution with mean $\mu_{d,i}$:

$$Y_{d,i} \sim \text{Po}(\mu_{d,i}).$$

The mean is usually modeled as a linear predictor made up of several terms and transformed via a convenient link function. The choice of these terms depends on the available data and the analyst's understanding of the data generating process. For instance, one option can be described as

$$\psi(\mu_{d,i}) = \alpha_d + u_i + v_{d,i}.$$

Here, $\psi(\cdot)$ is a link function (e.g., natural logarithm), α_d is a variable-specific intercept, u_i a shared (between all or a group of variables) spatial term, and $v_{d,i}$ a variable-specific random effect. Note that restrictions may be required on $v_{d,i}$ to make all effects identifiable ([Rue and Held, 2005](#)).

In the context of disease mapping, the usual variables of interest are the counts of mortality or incidence of different diseases over the study region. Now d represents the specific disease, therefore, following the above structure, the observed number of cases of disease d in area i , $Y_{d,i}$, can be modeled as

$$Y_{d,i} \sim \text{Po}(\mu_{d,i} = E_{d,i} \cdot \theta_{d,i})$$

$$\log(\theta_{d,i}) = \alpha_d + u_i + v_{d,i}$$

where $E_{d,i}$ and $\theta_{d,i}$ are the expected number of cases and the relative risk of disease d in area i , respectively. As before, α_d is a disease specific intercept (to account for differences in the total number of observed cases), u_i a shared spatial term (which does not depend on the disease) and $v_{d,i}$ a disease-specific spatial random effect.

In the case of multivariate lattice data, a prior distribution should be assigned to the dispersion parameter of each spatial effect. Some authors have discussed the most appropriate vague prior distributions in these cases. For instance, Gelman (2006) suggests to avoid inverse Gamma distributions on the precision and propose some alternatives. In this example, flat uniform prior distributions are assigned to the standard deviation parameters (see Section 5.3 in Gómez-Rubio, 2020).

Several authors (see, for example, Martínez-Beneito, 2013, and the references therein) have proposed different approaches for modelling multiple diseases in space and time. Gómez-Rubio et al. (2019) propose a separable spatio-temporal model with weighted shared components that can be used to detect diseases with similar patterns. In Palmí-Perales et al. (2021), the authors have developed an R package (**INLAMSM**) which builds on top of **R-INLA** to provide some of the most common multivariate model for lattice data.

3.1 Example: spatial analysis of several diseases

Gómez-Rubio and Palmí-Perales (2019) study the variation of spatial risk of three types of cancer in peninsular Spain. In particular, they consider oral cavity, esophagus and stomach cancer, all at the province level. In order to assess similar spatial variation that may lead to the identification of shared risk factors, models with shared and disease specific spatial patterns can be proposed.

Data are available in a RData file available from GitHub at https://github.com/becarioprecario/INLAMCMC_spatial_examples (see Gómez-Rubio and Palmí-Perales, 2019, for details). The following code creates the response variable (by stacking the three vectors of observed cases) as well as the expected counts. Note that due to the different structure of the variables involved, data are stored in a list object instead of a data.frame. Function `create_multivariate_data()` used below is available in the code provided as supplementary material.

```
# Load data
load("dismap_sim_data.RData")

# Set shorter names
names(OyE.sim)[1:6] <- c("Obs.Cb", "Esp.Cb", "Obs.Eso", "Esp.Eso", "Obs.Est", "Esp.Est")

# Create a dataset for INLA (n x 3)
n <- nrow(OyE.sim)

d <- list(OBS =
  create_multivariate_data(as.data.frame(OyE.sim)[, c("Obs.Cb", "Obs.Eso", "Obs.Est")])
)

# Expected cases
d$EXP <- c(OyE.sim$Esp.Cb, OyE.sim$Esp.Eso, OyE.sim$Esp.Est)
```

As an example, we will consider a model in which the log-relative risk of oral cavity cancer ($\theta_{o,i}$) is modeled as the sum of an intercept (α_o), and a shared spatial term (u_i). An intrinsic conditional auto-regressive (ICAR, Banerjee et al., 2014) is assigned to this shared term following the model described for areal (or lattice) data in the previous section. Furthermore, log-relative risks of esophagus ($\theta_{e,i}$) and stomach cancer ($\theta_{s,i}$) are modeled using disease-specific intercepts (α_e and α_s , respectively) plus the shared spatial term and cancer-specific ICAR spatial terms ($v_{e,i}$ and $v_{s,i}$, respectively). Specifically, the relative risks are defined as follows:

$$\begin{aligned}\log(\theta_{o,i}) &= \alpha_o + u_i \\ \log(\theta_{e,i}) &= \alpha_e + u_i + v_{e,i} \\ \log(\theta_{s,i}) &= \alpha_s + u_i + v_{s,i} \quad i = 1, \dots, n\end{aligned}$$

The disease-specific spatial terms can be used to assess departures from the shared spatial term. The chosen prior distributions for the standard deviation of all the effects are improper flat uniform distributions.

The model formula is defined in the model below. The `rf` term represents the disease-specific intercepts (the value of the above α s). Latent effects of type `copy` (see, for example, Section 6.5.1 in Gómez-Rubio, 2020) are used to define shared terms u_i in the model. Furthermore, spatial latent effects have the uniform prior for their standard deviation defined in object `prior.prec`.

```
# Formulas for the model
form <- OBS ~ -1 + rf +
  f(copy1, model = "besag", graph = W, hyper = list(prec = prior.prec)) +
  f(copy2, copy = "copy1", fixed = TRUE) +
  f(copy3, copy = "copy1", fixed = TRUE) +
  f(spatial2, model = "besag", graph = W, hyper = list(prec = prior.prec)) +
  f(spatial3, model = "besag", graph = W, hyper = list(prec = prior.prec))
```

Finally the model is fit with **R-INLA** using the code below. Note how the `family` argument takes a vector of three elements as this likelihood has three components (one for each disease).

```
res <- inla(formula = form, data = d, family = rep("poisson", 3), E = d$EXP)
```

Figure 1 shows the different spatial terms in the model. The shared term represents the spatial variation in the risk of oral cavity cancer and also serves as a baseline for the other types of cancer. The esophagus-specific spatial term is quite mild, which indicates that these two types of cancer have a very similar spatial pattern. The stomach-specific spatial term shows that some provinces in the center of the country have a higher mortality from stomach cancer as compared to oral cavity/esophagus cancer.

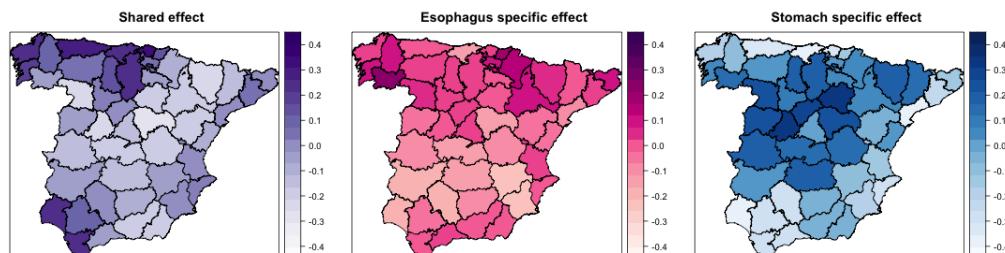


Figure 1: Posterior means of the shared spatial pattern (left), the esophagus-specific spatial pattern (middle) and stomach-specific spatial pattern (right).

4 Multivariate Continuous Data

Datasets for variables that vary continuously in space contain observations which are geographically referenced, i.e., both the value and where it is collected (e.g., the coordinates) appear in the dataset. Then, the spatial variation of the different variables is estimated using geostatistical models.

Similarly to lattice data, geostatistical multivariate models can be fit with **R-INLA** by sharing common terms. Therefore, the first variable can be modelled so that the mean includes a shared spatial term assumed to be a Gaussian process with a covariance defined using a Matérn function, and all the other variables can depend on this shared spatial term plus specific spatial effects. Hence, for example,

D variables of interest (Y_d for $d = 0, \dots, D - 1$), with a general likelihood function $P(\cdot)$, measured at n different locations can be written as:

$$Y_{i,d} \sim P(\mu_{i,d}).$$

Then, the mean of the baseline variable ($\mu_{i,0}$) will be modelled as a sum of an intercept (α_0) and a shared spatial effect ($u_{i,0}$). Furthermore, the mean of observation i and variable d ($\mu_{i,d}$) will be modeled through an intercept for each variable (α_d), the shared spatial effect ($u_{i,0}$) and a specific spatial effect ($u_{i,d}$) as follows:

$$\begin{aligned} \mu_{i,0} &= \alpha_0 + u_{i,0}; & i &= 1, \dots, n \\ \mu_{i,d} &= \alpha_d + u_{i,0} + u_{i,d} & i &= 1, \dots, n; d = 1, \dots, D - 1 \end{aligned}$$

Here, $u_{i,0}$ represents the shared (between variables) spatial term, while $u_{i,d}$ ($d \geq 1$) represents specific terms that can be used to assess departures from the shared spatial term. These random effect terms are assumed to be Gaussian processes with covariance defined using a Matérn covariance function, which for a generic spatial random effect $u(s)$ is defined as:

$$\text{Cov}(u_p, u_l) = \text{Cov}(u(s_p), u(s_l)) = \frac{\sigma^2}{\Gamma(\lambda)2^{\lambda-1}} (\kappa \|s_p - s_l\|)^{\lambda} K_{\lambda}(\kappa \|s_p - s_l\|),$$

where $\|s_p - s_l\|$ represents the Euclidean distance between points s_p and s_l , σ^2 is the marginal variance of the latent Gaussian process, κ is the scaling parameter, which is related to the range, K_{λ} represents the modified Bessel function of the second kind, and λ is the order term, which measures the smoothness of the process. Furthermore, this latent Gaussian field $u(s)$ can be approximated using the stochastic partial differential equation (SPDE) approach described in [Lindgren et al. \(2011\)](#). This approach relies on expressing the random effect $u(s)$ as the solution of an SPDE which is approximated through (an appropriate choice of) deterministic basis functions defined in a triangulation of the domain:

$$u(s) = \sum_{k=1}^m \phi_k(s) w_k.$$

Here, $\phi_k(s)$ are the basis functions (pairwise linear functions), m is the total number of nodes (triangle vertices) and w_k are zero-mean Gaussian distributed weights. For more details, the reader is referred to [Lindgren et al. \(2011\)](#).

In this example, all the measurements of the different variables are obtained from the same n locations. However, the measurements of each variable can be from different locations of the study region. Furthermore, INLA is not able to estimate any cross-covariance using the proposed model as dependence among the variables relies on the shared spatial term. A multivariate geostatistical analysis can be performed using the R package **gstat**. However, the models implemented in this R package are based on a classic and frequentist statistical approach.

Regarding the priors, the SPDE approximation requires setting a prior distribution to the nominal range, r , and the nominal standard deviation, σ . The nominal range is the distance at which the correlation is 0.1. Penalized complexity prior distributions (PC-priors, [Simpson et al., 2017](#)) can be chosen for both parameters. In a nutshell, PC-priors are based on the idea of penalising the complexity from a simple baseline model (that is, a model in which the parameter has been fixed to a particular reference value), i.e., the prior density is related to the distance from a baseline model. A remarkable benefit of the PC-priors are their high intuitiveness in their definition as they are set by stating values for the probabilities of the parameters, in particular, $P(r < r_0)$ and $P(\sigma > \sigma_0)$ for certain values of r_0 and σ_0 .

The prior on the range is set such that r is lower than half the maximum distance (d_m) between any two points in the study region, with high probability, i.e., $P(r < d_m/2) \approx 1$. As for the standard deviation, following the example of [Sørbye et al. \(2019\)](#) we set an upper limit U_{α} such that σ is less than this limit with high probability. Therefore, the probability that the standard deviation is greater than this upper limit ($P(\sigma > U_{\alpha})$) is set to be almost 0.

4.1 Example: spatial distribution of heavy metals

The **meuse** dataset in the **gstat** package gives the locations and measurements of topsoil heavy metals collected in a flood plain by the Meuse river close to the village of Stein (located in Netherlands). After

loading the `gstat` package, the data can be loaded using the following code:

```
# Load the data
data(meuse)

# Create the spatial object
coordinates(meuse) <- ~ x + y
proj4string(meuse) <- CRS("+init=epsg:28992")
```

These measurements are highly correlated and we will explore in this example how to fit geostatistical models with the SPDE approach. Note that observations do not need to be in a regular grid. Instead of a grid, a mesh is defined to apply the SPDE approach. The boundary of the study region is stored in object `meuse.bdy` (see accompanying code). The definition of the mesh is done using the coordinates of the boundary of the area following the code below:

```
# Create the mesh
mesh <- inla.mesh.2d(boundary = meuse.bdy, loc = coordinates(meuse),
max.edge = c(250, 500), offset = c(250, 500), n = c(32, 32))
```

The left plot in Figure 2 shows the mesh built with the above code for this example.

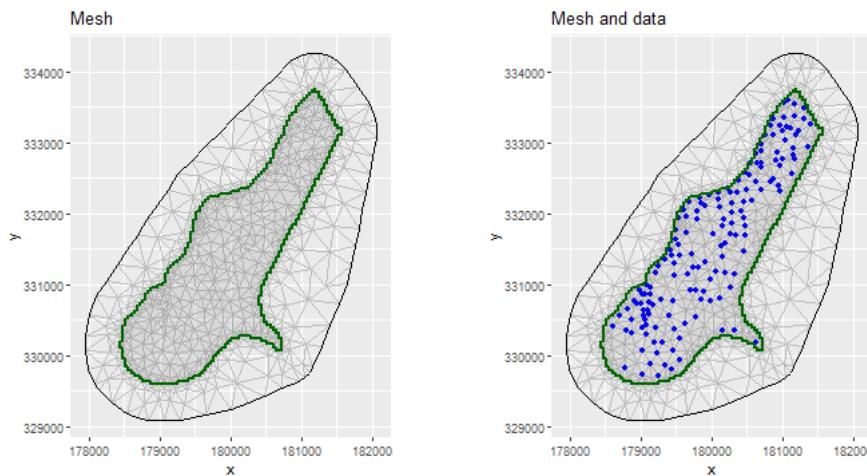


Figure 2: Mesh used in the estimation of the concentration of heavy metals around river Meuse, only with the boundary of the study region (left) and jointly with the survey locations (right).

In particular, the model will consider concentrations of lead and zinc. Concentrations have been measured at different locations which are displayed in the right plot in Figure 2. However, data on the concentrations of other metals is also available within the same dataset. Then, a model with all the metal concentrations could be handled following a similar structure. Values (in the original dataset) are considered in the log-scale. These log-transformed concentrations ($\log(y_l)$ and $\log(y_z)$, respectively) are assumed to be normally distributed. The mean of the log-concentration of lead is modeled using an intercept α_l plus a Gaussian process with a Matérn covariance, $u_{i,s}$, while the mean of the log-concentration of zinc is modeled using an intercept, α_z , the shared spatial effect, $u_{i,s}$, plus another spatial Gaussian process with Matérn covariance, $u_{i,z}$. This will allow us to assess differences in the spatial distribution of the concentration of both heavy metals. Furthermore, the Euclidian distance from the location of each measurement to the closest point of the river ($dist$) is included in both linear predictors as a covariate, each with a different slope parameter (β_l and β_z).

Hence, the model can be written as:

$$\begin{aligned}\log(y_l) &\sim N(\mu_{i,l}, \sigma_l) \\ \log(y_z) &\sim N(\mu_{i,z}, \sigma_z)\end{aligned}$$

where $\mu_{i,l}$ and $\mu_{i,z}$ represent the mean of the concentration of lead and zinc, respectively and which are modelled as follows:

$$\begin{aligned}\mu_{i,l} &= \alpha_l + \beta_l dist_i + u_{i,s}; & i = 1, \dots, n \\ \mu_{i,z} &= \alpha_z + \beta_z dist_i + u_{i,s} + u_{i,z} & i = 1, \dots, n\end{aligned}$$

In this case, the prior choices for the nominal range (r) and standard deviation (σ) for both the shared and zinc-specific spatial random effects are:

$$\begin{aligned}P(r < 2394.16) &= 0.95, \\ P(\sigma > 1000) &= 0.05.\end{aligned}$$

In plain words, this represents the belief that substantial correlations (> 0.1) between points more than half the maximum distance (2394.16 meters) apart are unlikely, and so are standard deviations greater than 1,000. These prior distributions are specified using the `inla.spde2.pcmatern()` function:

```
spde <- inla.spde2.pcmatern(mesh = mesh,
prior.range = c(2394.16, 0.95), prior.sigma = c(1000, 0.05))
```

The SPDE approximation estimates the spatial random effects at the vertices of the mesh, so that estimates at any other point are based on the estimates at the vertices of the triangle that contains the point. The position of this point inside the triangle is identified using barycentric coordinates (see [Krainski et al., 2019](#), for details). The projector matrix contains all these coordinates for all the points in the dataset and it is obtained with function `inla.spde.make.A()`:

```
A.m <- inla.spde.make.A(mesh = mesh, loc = coordinates(meuse))
```

In order to fit models using the SPDE approach, data must be *stacked* using the appropriate format. Helper function `inla.stack()` can be used to build a stack object which contains the data (properly structured), the effects considered in the model, and the projector matrix. Specifically, a stack object will be built to model each variable. Then, all these objects will be put together into a single stack object so that a joint model can be fit.

In all the stacks the name of the response variable will be the same. The values will be given in a matrix with as many columns as the number of variables and as many rows as the number of observations. In the first stack, the values of the first response variable will be in the first column and the rest of columns will be filled with NA's. The other stacks and corresponding response variables will be defined analogously.

Regarding fixed and other latent effects included in the linear predictor, these can take the same name across stacks when the effect is shared for different response variables. When the effect of the same covariate or latent effect needs to be different across response variables, then different names must be used. See the R code below for more details.

The data are structured using the `inla.stack()` function specifying three elements: the data, the projector matrix, the different effects of the linear predictor and a tag to identify each part of the dataset in the final stack. This tag is used when retrieving particular results from the output.

```
# Create the stack object for lead
stk.lead <- inla.stack(
  data = list(log.y = cbind(log(meuse$lead), NA)),
  A = list(A.m, 1),
  effects = list(spatial.field.lead = 1:spde$n.spde,
    data.frame(Intercept.lead = 1, dist.lead = meuse$dist)),
  tag = "Lead")

# Create the stack object for zinc
stk.zinc <- inla.stack(
  data = list(log.y = cbind(NA, log(meuse$zinc))),
  A = list(A.m, A.m, 1),
  effects = list(
    spatial.field.zinc = 1:spde$n.spde, base.copy.zinc = 1:nv,
    data.frame(Intercept.zinc = 1, dist.zinc = meuse$dist)),
  tag = "Zinc")
```

A projector matrix and a stack for the prediction grid have also been created as follows:

```
# Create the projector matrix for the prediction
A.pr <- inla.spde.make.A(mesh = mesh, loc = coordinates(meuse.grid))

# Prepare the data for the prediction
y.pred <- matrix(NA, nrow = nrow(meuse.grid), ncol = 2)

# Build predicting stack for lead
stk.lead.pr <- inla.stack(
  data = list(log.y = y.pred),
  A = list(A.pr, 1),
  effects = list(spatial.field.lead = 1:spde$n.spde,
    data.frame(Intercept.lead = 1, dist.lead = meuse.grid$dist)),
  tag = "Lead.pred")

# Build predicting stack for zinc
stk.zinc.pr <- inla.stack(
  data = list(log.y = y.pred),
  A = list(A.pr, A.pr, 1),
  effects = list(
    spatial.field.zinc = 1:spde$n.spde, base.copy.zinc = 1:nv,
    data.frame(Intercept.zinc = 1, dist.zinc = meuse.grid$dist)),
  tag = "Zinc.pred")
```

Next, a stack object is built for the shared and zinc-specific effects in order to study the spatial patterns of these effects:

```
# Stack for the shared effect
stk.shared <- inla.stack(
  data = list(log.y = y.pred),
  A = list(A.pr),
  effects = list(spatial.field.lead = 1:spde$n.spde),
  tag = "Shared")

# Stack for the specific sp effect zinc
stk.zinc.spec <- inla.stack(
  data = list(log.y = y.pred),
  A = list(A.pr),
  effects = list(spatial.field.zinc = 1:spde$n.spde),
  tag = "Zinc.spec")
```

All the stack objects are put together in a single joint stack object using the `inla.stack()` function:

```
# Put all the stacks together
join.stack <- inla.stack(
  stk.lead, stk.zinc,
  stk.zinc.pr, stk.lead.pr,
  stk.shared, stk.zinc.spec)
```

The model formula is defined below. The latent effect of type `copy` is used to define the shared term of the model.

```
# Formulas for the model
form <- log.y ~ -1 + Intercept.lead + Intercept.zinc + dist.lead + dist.zinc +
  f(spatial.field.lead, model = spde) +
  f(spatial.field.zinc, model = spde) +
  f(base.copy.zinc, copy = "spatial.field.lead", fixed = TRUE)
```

Finally, the model is fit with **R-INLA**. Note how the family argument takes a vector of two "gaussian" elements (one for each heavy metal concentration). Furthermore, note that the data are obtained from the joint stack with the `inla.stack.data()` function and that the projector matrix `A` is also obtained using the `inla.stack.A()` function.

```
meuse.res <- inla(formula = form, verbose = FALSE,
  data = inla.stack.data(join.stack, spde = spde),
  family = rep("gaussian", 2),
  control.family = list(zero.prec, zero.prec),
  control.predictor = list(A = inla.stack.A(join.stack), compute = TRUE),
  control.compute = list(dic = TRUE, waic = TRUE, cpo = TRUE, mlik = TRUE, po = TRUE))
```

Given that the structure of the data used for model fitting is a stack now, particular parameter estimates can be accessed through function `inla.stack.index()`. This function can provide the indices to access the entries of specific data or latent effects in the summaries provided by **R-INLA**. For example, the following code can be used to obtain the posterior means of the fitted values obtained with the effects in the 'Lead.pred' stack:

```
idx.lead <- inla.stack.index(join.stack, 'Lead.pred')$data
meuse.grid$lead.pr <- meuse.res$summary.fitted.values[idx.lead, 'mean']
```

Figure 3 shows point estimates (posterior means) of the log-concentration of lead and zinc; the posterior mean of the shared and the zinc-specific effect are also shown. Note the similar spatial pattern across both metals.

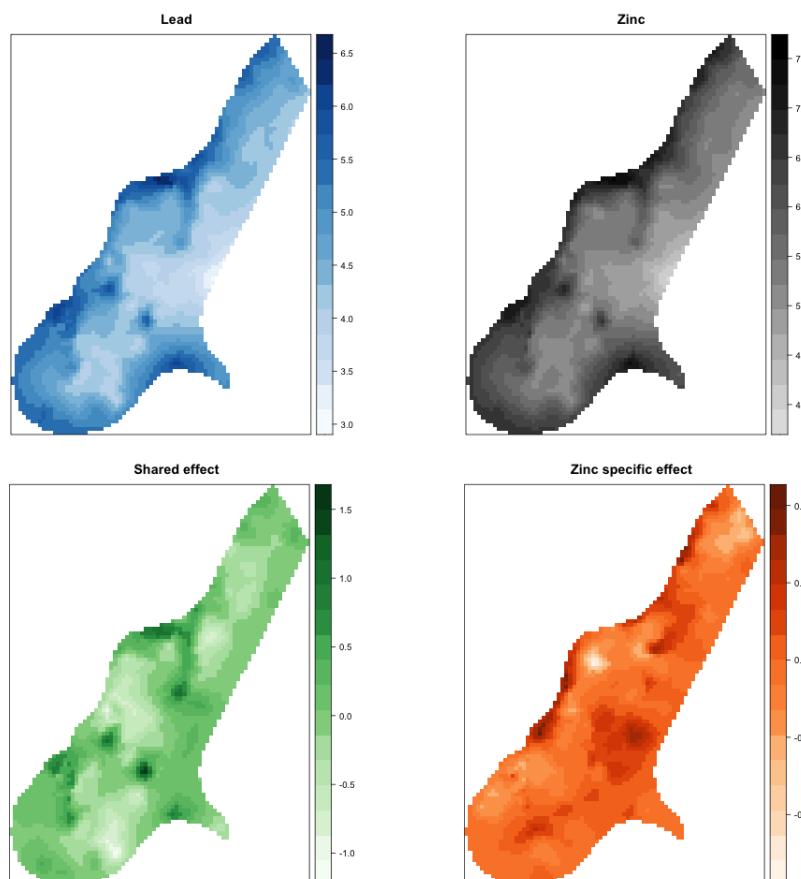


Figure 3: Estimates of the posterior means of the log-concentration of lead (top-left) and zinc (top-right). Estimates of the posterior mean of the shared spatial effect (bottom-left) and the zinc-specific spatial effect (bottom-right).

5 Multivariate point patterns

A point pattern is defined as a group of points (geographically located) which are a single realization of a stochastic process called point process. A multivariate point pattern can be defined as a group of several point patterns where each point pattern has a different origin, i.e., each point pattern is caused

by different processes. These are also referred to a specific case of *marked* point pattern (Baddeley et al., 2015) where each point pattern is labelled with a categorical mark.

In a completely random point process, points appear independently of each other and uniformly over the study region. This is also known as a homogeneous Poisson process with (constant) intensity λ , which measures the average number of points per unit area. It is also possible to consider a spatially varying intensity, $\lambda(s)$ (with s being a point of the study region), so that the process becomes an *inhomogeneous* Poisson process. The model can be extended to more complex point patterns (see, for example, Baddeley et al., 2015).

Several methods have been used to model the intensity function $\lambda(s)$. A complete spatial randomness scenario will not be considered here, that is, the intensity function of each point pattern will not be considered as constant over the study area. Specifically, intensities will be considered as continuous processes over the entire study region and they will be modeled using log-Gaussian Cox processes (Møller et al., 1998; Diggle et al., 2013). Log-Gaussian Cox processes can be fit by including spatial terms using the SPDE approach implemented in INLA (Simpson et al., 2016). The analysis of the intensity as a continuous function over the study region is similar to the case of multivariate geostatistics.

Given K point patterns in a region \mathcal{D} , an example of how to structure a multivariate point patterns model is:

$$\begin{aligned} \log(\lambda_0(s)) &= \alpha_0 + u_0(s); & s \in \mathcal{D} \\ \log(\lambda_j(s)) &= \alpha_j + u_0(s) + u_j(s) & s \in \mathcal{D}; j = 1, \dots, K - 1 \end{aligned}$$

where $\lambda_0(s)$ is the intensity of the baseline point pattern and $\lambda_j(s)$ the intensity function of the j th point pattern. Moreover, α_0, α_j terms represent the intercepts and $u_0(s), u_j(s)$ are the spatial effects. Specifically, $u_0(s)$ is the shared (by the different point patterns) spatial term and $u_j(s)$ are the specific spatial terms which model the differences between each point pattern and the baseline.

In spatial epidemiology, the goal is often to discern whether a distribution of cases follows the spatial distribution of a set of controls, or whether it depends on exposure to pollution sources or other risk factors such as pollution (Palmí-Perales et al., 2021). This is an application of the model described here, in which the log-intensity of the controls can be modeled using a shared spatial term and the log-intensity of the cases can include this shared spatial term plus a disease-specific spatial term. Furthermore, the linear predictor can include other terms to account for risk factors.

Priors are set similarly as in the example for continuous spatial data as the spatial random effects $u_j(s)$, $j = 0, \dots, K - 1$ are actually modeled as SPDE latent effects. Other parameters of effects included in the model (e.g., fixed effects) will be assigned priors accordingly.

5.1 Example: forest fires in Castilla-La Mancha (Spain)

The **spatstat** package contains the **clmfires** dataset. This dataset records the occurrence of forest fires in the region of Castilla-La Mancha (Spain) from 1998 to 2007. These forest fires are classified by four different causes: lightning, accidental, intentional, and other fires (Figure 4). After loading the packages, the data are accessed using the `data()` function:

```
#Load and display the data
data("clmfires")
```

Specifically, in this example, the intensities of the different types of forest fires are estimated by considering lightning fires as the baseline pattern. Furthermore, INLA is used to assess the similarities or differences among their spatial patterns. First of all, the mesh is built using the coordinates of the boundary of the dataset (`bdy.SP`) using the code below.

```
mesh <- inla.mesh.2d(
  boundary = list(bdy.SP, NULL), cutoff = 2, max.edge = c(20, 50),
  min.angle = 27, offset = c(1, 50), n=c(16,16))
```

Following the model structure for multivariate point patterns detailed above, the log-intensity of the lightning fires $\lambda_l(s)$ will be modeled using an intercept, α_l , and a spatial Gaussian effect with Matérn covariance, $u_l(s)$ as follows

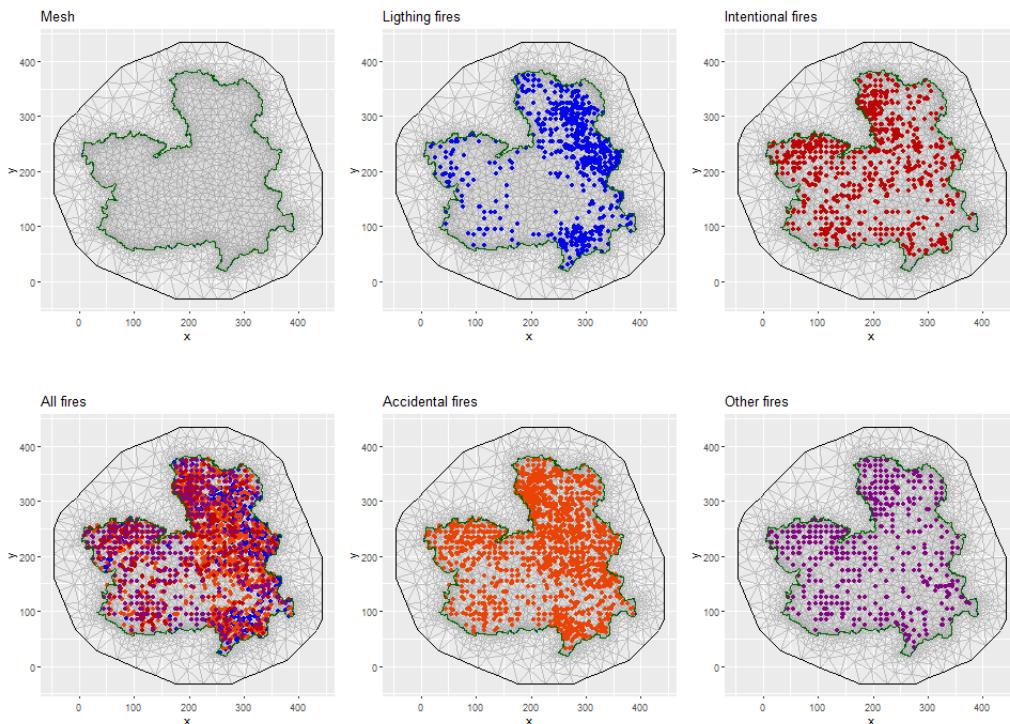


Figure 4: The mesh used is shown alone (top-left) and with all the fire types (bottom-left). Furthermore, the mesh is also displayed for each fire type separately: lightning fires (top-middle), accidental fires (bottom-middle), intentional fires (top-right), and other fires (bottom-right).

$$\log \lambda_l(s) = \alpha_l + u_l(s); \quad s \in D$$

Spatial effect $u_l(s)$ will also be shared in the linear predictor of the other types of fire. Similarly, the log-intensity of the accidental, intentional, and other fires (which includes unknown cause) will be modeled using specific intercepts, plus the shared spatial effect, plus a specific spatial effect as follows:

$$\begin{aligned} \log \lambda_a(s) &= \alpha_a + u_l(s) + u_a(s); \quad s \in D \\ \log \lambda_i(s) &= \alpha_i + u_l(s) + u_i(s); \quad s \in D \\ \log \lambda_o(s) &= \alpha_o + u_l(s) + u_o(s) \quad s \in D \end{aligned}$$

Following the prior specification section, the chosen PC-priors in this example are:

$$\begin{aligned} P(r < 200) &= 0.95 \\ P(\sigma > 100) &= 0.05 \end{aligned}$$

where a nominal range higher than half the maximum distance (e.g., 200 kilometres) of the domain is unlikely. Similarly for the nominal standard deviation, its prior assumes that it is really unlikely that it is higher than 100 in this context. These prior distributions have been specified using function `inla.spde2.pcmatern()`:

```
spde <- inla.spde2.pcmatern(mesh = mesh,
  prior.range = c(200, 0.95), prior.sigma = c(10, 0.05))
```

Here, argument `prior.range` sets the prior for the range and the argument `prior.sigma` sets the prior for the standard deviation of the spatial effect.

As this model includes SPDE latent effects, data must be put together using `stack` objects. In the case of point patterns, the data included in the `stack` function is different from the geostatistical example as point pattern data has to be detailed following a specific structure (see below). Model fitting now relies on the methods described in Simpson et al. (2016). In this case, two elements will be

included in a list for storing the data of each point pattern.

The first element of the list will be a matrix with $N_v + N_i$ rows and K columns where N_v is the number of vertices of the SPDE mesh, N_i is the total number of points of the i -th point pattern and K is the number of different point patterns. The matrix of the stack of the i -th point pattern will be filled with NA's except for the i -th column. This i -th column will contain firstly N_v zeros corresponding to the points of the mesh. After these zeros, there will be N_i ones corresponding to the N_i points of the i -th point pattern.

The second element of the list will be a vector containing an offset. Specifically, the length of this vector is also $N_v + N_i$ where the first N_v elements will contain the "weights" of the mesh points. The reader is referred to [Simpson et al. \(2016\)](#) for more details about this approximation. The rest of the values will be zeros.

These two elements have to be created for each stack of each point pattern. Then, as before, the stacks are combined in a single stack object. As a toy example, consider a mesh with $N_v = 3$, two point patterns ($K = 2$) with three and four points, respectively ($N_1 = 3$ and $N_2 = 4$). The weights associated to the three mesh points will be 2.3, 4.3 and 6.2. Then, the data passed to the first and second stack are:

$$\begin{bmatrix} 0 & \text{NA} \\ 0 & \text{NA} \\ 0 & \text{NA} \\ 1 & \text{NA} \\ 1 & \text{NA} \\ 1 & \text{NA} \end{bmatrix}, \begin{bmatrix} 2.3 \\ 4.3 \\ 6.2 \\ 0 \\ 0 \\ 0 \end{bmatrix}; \quad \begin{bmatrix} \text{NA} & 0 \\ \text{NA} & 0 \\ \text{NA} & 0 \\ \text{NA} & 1 \\ \text{NA} & 1 \\ \text{NA} & 1 \end{bmatrix}, \begin{bmatrix} 2.3 \\ 4.3 \\ 6.2 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

As stated above, when analysing point patterns it is necessary to assign some weights to the points of the mesh. This is done by creating a Voronoi tessellation using these points, so that the area of the associated polygon becomes the associated weight. The following code illustrates how to obtain the Voronoi tessellation and the associated weights:

```
library(deldir)
dd <- deldir(mesh$loc[, 1], mesh$loc[, 2])
# Create a list of tiles in a tessellation
mytiles <- tile.list(dd)

# Boundary as a polygon
pl.study <- as(bdy, "gpc.poly")
# Area of the study area
area.poly(pl.study)

# Compute weights as the area of the polygon given as an
# intersection between Voronoi tiles and domain polygon
w <- unlist(lapply(mytiles,
  function(p) area.poly(
    intersect(as(cbind(p$x,p$y), "gpc.poly"), pl.study)
  )
))
)
```

These computed weights are introduced as the expected weights on the mesh points. Furthermore, the data should be structured as follows:

```
# Data for the stack function: lightning fires
e.lig <- c(w, rep(0, n.lig))
y.lig <- matrix(NA, nrow = nv + n.lig, ncol = n.pp)
y.lig[, 1] <- rep(0:1, c(nv, n.lig))

# Data for the stack function: accidental fires
e.acc <- c(w, rep(0, n.acc))
y.acc <- matrix(NA, nrow = nv + n.acc, ncol = n.pp)
y.acc[, 2] <- rep(0:1, c(nv, n.acc))

# Data for the stack function: intentional fires
```

```

e.int <- c(w, rep(0, n.int))
y.int <- matrix(NA, nrow = nv + n.int, ncol = n.pp)
y.int[, 3] <- rep(0:1, c(nv, n.int))

# Data for the stack function: other fires
e.oth <- c(w, rep(0, n.oth))
y.oth <- matrix(NA, nrow = nv + n.oth, ncol = n.pp)
y.oth[, 4] <- rep(0:1, c(nv, n.oth))

```

Another element of the SPDE approach is the projector matrix that represents each point using barycentric coordinates from the mesh points (as explained above). When analysing multivariate point patterns, this matrix has two parts: one for the mesh points (named `imat`), which is a diagonal matrix as every mesh point matches itself when using barycentric coordinates, and the other for the points of the point pattern (named `lmat`). Then, the projector matrix is the combination of these two matrices:

```

#imat: define imat
imat <- Diagonal(nv, rep(1, nv))

#lmat: define lmat
lmat.lig <- inla.spde.make.A(mesh, pts.lig)
lmat.acc <- inla.spde.make.A(mesh, pts.acc)
lmat.int <- inla.spde.make.A(mesh, pts.int)
lmat.oth <- inla.spde.make.A(mesh, pts.oth)

#Projector matrix: Put together imat and lmat
A.lig <- rbind(imat, lmat.lig)
A.acc <- rbind(imat, lmat.acc)
A.int <- rbind(imat, lmat.int)
A.oth <- rbind(imat, lmat.oth)

```

Once all the elements of the stack object are set, the `inla.stack()` function is used to build the different stack objects. For instance, the stack objects of two (out of four) types of forest fires are shown here below:

```

# Create the stack for the lighting fires
stk.lig <- inla.stack(
  data = list(y = y.lig, e = e.lig),
  A = list(A.lig, 1),
  effects = list(spatial.field.lig = s.index.lig,
    data.frame(Intercept.lig = rep(1, n.lig)))
  ),
tag = "Lighting")

# Create the stack for the accidental fires
stk.acc <- inla.stack(
  data = list(y = y.acc, e = e.acc),
  A = list(A.acc, A.acc, 1),
  effects = list(
    base.copy.acc = 1:nv,
    spatial.field.acc = s.index.acc,
    data.frame(Intercept.acc = rep(1, n.acc)))
  ),
tag = "Accidental")

```

As in the example for continuous spatial data, the stack objects for the predictions have to be built following the same structure (their name will end with ".pr" in the R code). Finally, all the stack objects are joined into a single joint stack object as follows:

```

# All stacks together
join.stack <- inla.stack(
  stk.lig, stk.acc, stk.int, stk.oth,
  stk.lig.pr, stk.acc.pr, stk.int.pr, stk.oth.pr,
  stk.shared, stk.acc.spec, stk.int.spec, stk.oth.spec)

```

Note that in this example separate stack objects have been created for estimating the intensity of each of the different types of forest types, the shared spatial pattern (stack `stk.shared`) and the three different specific spatial effects (stacks `stk.acc.spec`, `stk.int.spec` and `stk.oth.spec`).

The model formula is defined here below, where the latent effects of type `copy` are used to define the shared terms of the model:

```
form <- y ~ -1 + Intercept.lig + Intercept.acc + Intercept.int + Intercept.oth +
  f(spatial.field.lig, model = spde) +
  f(spatial.field.acc, model = spde) +
  f(base.copy.acc, copy = "spatial.field.lig", fixed = TRUE) +
  f(spatial.field.int, model = spde) +
  f(base.copy.int, copy = "spatial.field.lig", fixed = TRUE) +
  f(spatial.field.oth, model = spde) +
  f(base.copy.oth, copy = "spatial.field.lig", fixed = TRUE)
```

The model is fit with **R-INLA** using the code below. Note how the `family` argument takes a vector of four "poisson" elements (one for each fire type).

```
pp.res <- inla(formula = form, verbose = FALSE,
  data = inla.stack.data(join.stack, spde = spde),
  family = rep("poisson", 4),
  control.predictor = list(A = inla.stack.A(join.stack), compute = TRUE, link = 1),
  control.compute = list(dic = TRUE, waic = TRUE, cpo = TRUE, mlik = TRUE, po = TRUE)
)
```

Figure 5 shows the posterior mean of the intensity of each type of the forest fires. A different spatial pattern can be seen for each fire type. Lightning fires mostly appear in the east part of the region, while the other three types of forest fires are more likely to appear in the west and the central parts of the region. Additionally, the posterior means of the shared and specific spatial effects can also be seen in Figure 6.

6 Discussion

In this paper, we have shown how to fit multivariate spatial models using the **R-INLA** package. In particular, the details of how to analyse each spatial data type (lattice, continuous, and point patterns) have been given. Furthermore, we have illustrated the application of these models using three datasets: simulated data on mortality by three types of cancer (available from https://github.com/becarioprecario/INLAMCMC_spatial_examples), the `clmfires` dataset in the `spatstat` package and the `meuse` dataset in the `gstat` package. Furthermore, more complex spatial and spatio-temporal models to multivariate data can be fitted with **R-INLA** (see, for example, [Palmí-Perales et al., 2021](#)).

The main goal of this work has been to illustrate how to perform multivariate spatial Bayesian inference using **R-INLA**. In particular, we have paid attention to the different steps required to create the necessary data structures for model fitting. The advantage of **R-INLA** compared with alternatives is its computational efficiency; particularly, MCMC-based methods can struggle in the computationally demanding high-dimensional setting of spatial data sets. Hence, it has been shown that **R-INLA** is a useful and a worthwhile toolbox for fitting multivariate spatial models. Additionally, the necessary R scripts to reproduce the examples are available at <https://github.com/FranciscoPalmíPerales/Mult-Sp-INLA>.

7 Acknowledgements

This work has been supported by grants SBPLY/17/180501/000491 and SBPLY/21/180501/000241, funded by Consejería de Educación, Cultura y Deportes (JCCM, Spain) and Fondo Europeo de Desarrollo Regional, grant MTM2016-77501-P and PID2019-106341GB-I00, funded by Ministerio de Economía y Competitividad (Spain), grants PID2019-106341GB-I00 and PID2022-136455NB-I00, funded by Ministerio de Ciencia e Innovación (Spain), and grant CIAICO/2022/165, funded by Dirección General de Ciencia e Investigación (Generalitat Valenciana). F. Palmí-Perales was supported by a doctoral scholarship awarded by the University of Castilla-La Mancha (Spain) and by grant PID2021-128228NB-I00 funded by Ministerio de Ciencia e innovación.

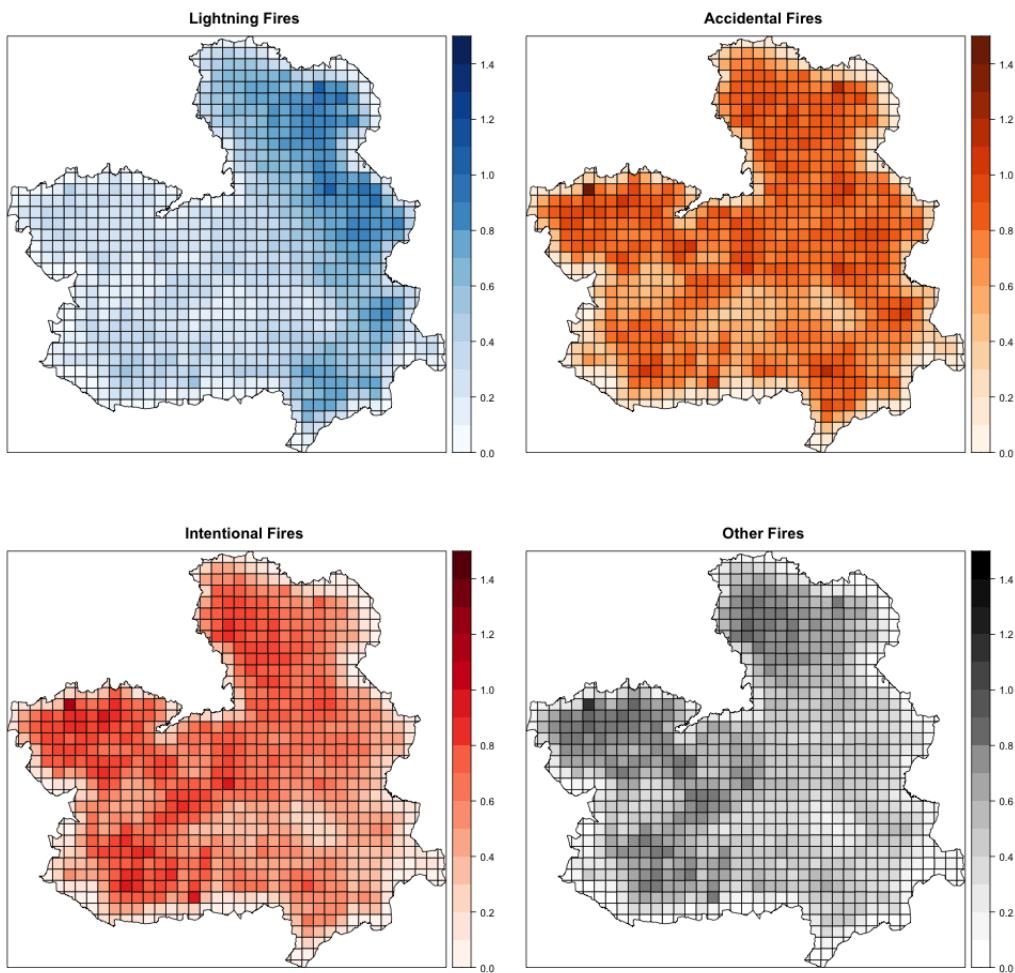


Figure 5: Posterior mean of the intensity of the lightning (top-left), accidental (top-right), intentional (bottom-left) and other (bottom-right) fires.

References

- A. Baddeley, E. Rubak, and R. Turner. *Spatial Point Patterns: Methodology and Applications with R*. Chapman & Hall/CRC Press, London, 2015. URL <http://www.crcpress.com/Spatial-Point-Patterns-Methodology-and-Applications-with-R/Baddeley-Rubak-Turner/9781482210200/>. [p173, 182]
- H. Bakka, H. Rue, G.-A. Fuglstad, A. Riebler, D. Bolin, E. Krainski, D. Simpson, and F. Lindgren. Spatial modelling with R-INLA: A review. *WIREs Comput Stat*, 10(6):1–24, 2018. URL <https://doi.org/10.1002/wics.1443>. [p172]
- S. Banerjee, B. P. Carlin, and A. E. Gelfand. *Hierarchical modeling and analysis for spatial data*. CRC press, 2014. [p172, 174, 175]
- M. Blangiardo and M. Cameletti. *Spatial and spatio-temporal Bayesian models with R-INLA*. John Wiley & Sons, 2015. [p172]
- P. J. Diggle, P. Moraga, B. Rowlingson, and B. M. Taylor. Spatial and spatio-temporal log-Gaussian Cox processes: Extending the geostatistical Paradigm. *Statistical Science*, 28(4):542 – 563, 2013. URL <https://doi.org/10.1214/13-STS441>. [p172, 182]
- A. O. Finley, S. Banerjee, and B. P. Carlin. spBayes: An R package for univariate and multivariate hierarchical point-referenced spatial models. *Journal of Statistical Software*, 19(4):1, 2007. URL <https://www.jstatsoft.org/index.php/jss/article/view/v019i04>. [p173]

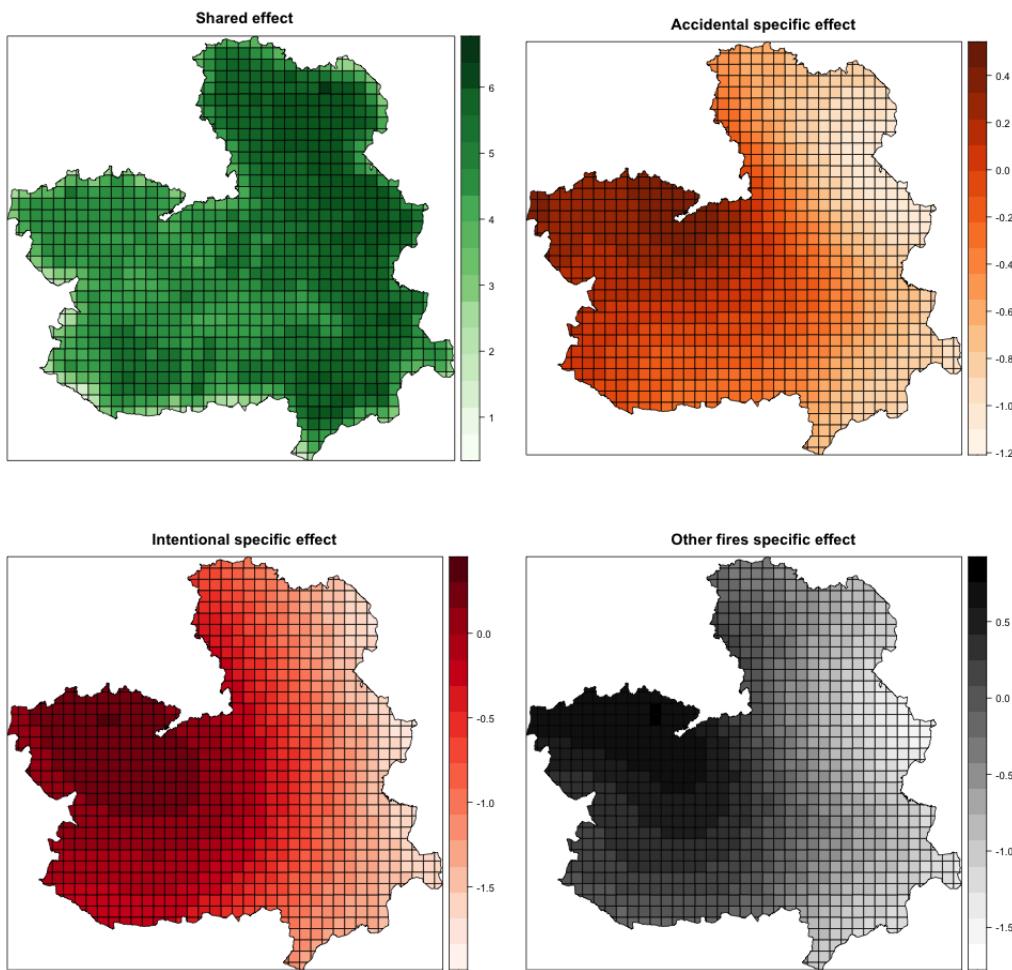


Figure 6: Posterior mean of the shared spatial effect (top-left), the accidental-specific (top-right), the intentional-specific (bottom-left) and the other-specific (bottom-right) spatial effects.

- A. O. Finley, S. Banerjee, and A. E. Gelfand. spBayes for large univariate and multivariate point-referenced spatio-temporal data models. *Journal of Statistical Software*, 63(13):1–28, 2015. URL <http://www.jstatsoft.org/v63/i13/>. [p173]
- A. Gelman. Prior distributions for variance parameters in hierarchical models (comment on article by Browne and Draper). *Bayesian Analysis*, 1(3):515 – 534, 2006. URL <https://doi.org/10.1214/06-BA117A>. [p175]
- W. R. Gilks, S. Richardson, and D. J. Spiegelhalter. *Markov Chain Monte Carlo in practice*. Chapman & Hall, 1995. [p172]
- V. Gómez-Rubio. *Bayesian inference with INLA*. CRC Press, 2020. [p172, 173, 175, 176]
- V. Gómez-Rubio and F. Palmí-Perales. Multivariate posterior inference for spatial models with the integrated nested Laplace approximation. *Journal of the Royal Statistical Society, Series C*, 68(1): 199–215, 2019. URL <https://doi.org/10.1111/rssc.12292>. [p172, 173, 175]
- V. Gómez-Rubio, M. Cameletti, and F. Finazzi. Analysis of massive marked point patterns with stochastic partial differential equations. *Spatial Statistics*, 14:179–196, 2015. URL <https://doi.org/10.1016/j.spasta.2015.06.003>. [p172]
- V. Gómez-Rubio, P. Zheng, P. Diggle, D. C. Sterrett, R. D. Peng, D. Murdoch, and B. Rowlingson. *spatialkernel: Non-Parametric Estimation of Spatial Segregation in a Multivariate Point Process*, 2017. URL <https://CRAN.R-project.org/package=spatialkernel>. R package version 0.4-23. [p173]

- V. Gómez-Rubio, F. Palmí-Perales, G. López-Abente, R. Ramis-Prieto, and P. Fernández-Navarro. Bayesian joint spatio-temporal analysis of multiple diseases. *SORT*, 1:51–74, 2019. URL <https://doi.org/10.2436/20.8080.02.79>. [p175]
- E. T. Krainski, V. Gómez-Rubio, H. Bakka, A. Lenzi, D. Castro-Camilo, D. Simpson, F. Lindgren, and H. Rue. *Advanced Spatial Modeling with Stochastic Partial Differential Equations Using R and INLA*. Chapman & Hall/CRC, Boca Raton, FL, 2019. [p172, 179]
- D. Lee. CARBayes: An R package for Bayesian spatial modeling with conditional autoregressive priors. *Journal of Statistical Software*, 55(13):1–24, 2013. URL <https://www.jstatsoft.org/v55/i13/>. [p173]
- F. Lindgren, H. Rue, and J. Lindström. An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(4):423–498, 2011. URL <https://doi.org/10.1111/j.1467-9868.2011.00777.x>. [p172, 177]
- F. Lindgren, H. Rue, et al. Bayesian spatial modelling with R-INLA. *Journal of Statistical Software*, 63(19):1–25, 2015. URL <https://doi.org/10.18637/jss.v063.i19>. [p172]
- Y. C. MacNab. Some recent work on multivariate Gaussian Markov random fields. *Test*, 27(3):497–541, 2018. URL <https://doi.org/10.1007/s11749-018-0605-3>. [p172]
- M. Martínez-Beneito. A general modelling framework for multivariate disease mapping. *Biometrika*, 100(3):539–553, 2013. URL <https://doi.org/10.1093/biomet/ast023>. [p175]
- M. A. Martínez-Beneito and P. Botella-Rocamora. *Disease Mapping: From Foundations to Multidimensional Modeling*. CRC Press, 2019. [p172]
- M. A. Martínez-Beneito, P. Botella-Rocamora, and S. Banerjee. Towards a multidimensional approach to Bayesian disease mapping. *Bayesian analysis*, 12(1):239, 2017. URL <https://doi.org/10.1214/16-BA995>. [p172]
- J. Møller, A. R. Syversveen, and R. P. Waagepetersen. Log Gaussian Cox Processes. *Scandinavian Journal of Statistics*, 25(3):451–482, 1998. URL <https://doi.org/10.1111/1467-9469.00115>. [p182]
- F. Palmí-Perales, V. Gómez-Rubio, and M. A. Martínez-Beneito. Bayesian Multivariate Spatial Models for Lattice Data with INLA. *Journal of Statistical Software*, 98(2):1–29, 2021. URL <https://www.jstatsoft.org/index.php/jss/article/view/v098i02>. [p175, 186]
- F. Palmí-Perales, V. Gómez-Rubio, G. López-Abente, R. Ramis, J. M. Sanz-Anquela, and P. Fernández-Navarro. Approximate Bayesian inference for multivariate point pattern analysis in disease mapping. *Biometrical Journal*, 63(3):632–649, 2021. URL <https://doi.org/10.1002/bimj.201900396>. [p182]
- E. J. Pebesma. Multivariable geostatistics in S: the gstat package. *Computers & Geosciences*, 30(7):683–691, 2004. [p173]
- E. J. Pebesma and C. G. Wesseling. Gstat: A Program for Geostatistical Modelling, Prediction and Simulation. *Computers & Geosciences*, 24(1):17–31, 1998. [p173]
- R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2022. URL <https://www.R-project.org/>. [p173]
- H. Rue and L. Held. *Gaussian Markov Random Fields. Theory and Applications*. Chapman & Hall, New York, 2005. [p172, 173, 174]
- H. Rue, S. Martino, and N. Chopin. Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *Journal of the royal statistical society: Series b (statistical methodology)*, 71(2):319–392, 2009. URL <https://doi.org/10.1111/j.1467-9868.2008.00700.x>. [p172, 173]
- H. Rue, F. Lindgren, D. Simpson, S. Martino, E. Teixeira Krainski, H. Bakka, A. Riebler, and G.-A. Fuglstad. *INLA: Full Bayesian Analysis of Latent Gaussian Models using Integrated Nested Laplace Approximations*, 2020. R package version 20.03.17. [p172]
- D. Simpson, J. Illian, F. Lindgren, S. H. Sørbye, and H. Rue. Going off grid: Computationally efficient inference for log-Gaussian Cox processes. *Biometrika*, 103(1):49–70, 2016. URL <https://doi.org/10.1093/biomet/asv064>. [p182, 183, 184]

- D. Simpson, H. Rue, A. Riebler, T. G. Martins, and S. H. Sørbye. Penalising model component complexity: A principled, practical approach to constructing priors. *Statistical Science*, 32(1):1–28, 2017. ISSN 08834237, 21688745. URL <http://www.jstor.org/stable/26408114>. [p177]
- S. H. Sørbye, J. B. Illian, D. P. Simpson, D. Burslem, and H. Rue. Careful prior specification avoids incautious inference for log-Gaussian Cox point processes. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 68(3):543–564, 2019. URL <https://doi.org/10.1111/rssc.12321>. [p177]
- M. N. M. Van Lieshout and A. J. Baddeley. Indices of dependence between types in multivariate point patterns. *Scandinavian Journal of Statistics*, 26(4):511–532, 1999. URL <https://doi.org/10.1111/1467-9469.00165>. [p172]
- R. Waagepetersen, Y. Guan, A. Jalilian, and J. Mateu. Analysis of multispecies point patterns by using multivariate log-Gaussian Cox processes. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 65(1):77–96, 2016. ISSN 00359254, 14679876. URL <http://www.jstor.org/stable/24773017>. [p172]

Francisco Palmí-Perales

*Department of Statistics and Operational Research, Faculty of Mathematics, Universitat de València
C/ Dr. Moliner, 50, 46100
Burjassot, Spain
0000-0002-0751-7315
Francisco.Palmi@uv.es*

Virgilio Gómez-Rubio

*Department of Mathematics, E.T.S. Ingeniería Industrial-Albacete, Universidad de Castilla-La Mancha
Av. de España, s/n, 02071
Albacete, Spain
0000-0002-4791-3072
Virgilio.Gomez@uclm.es*

Roger S. Bivand

*Department of Economics, Norwegian School of Economics
Helleveien 30, N-5045
Bergen, Norway
0000-0003-2392-6140
Roger.Bivand@nhh.no*

Michela Cameletti

*Department of Economics, Università degli studi di Bergamo
Via dei Caniana 2. IT-24127
Bergamo, Italy
0000-0002-6502-7779
michela.cameletti@unibg.it*

Håvard Rue

*King Abdullah University of Science and Technology
Thuwal, Saudi Arabia
0000-0002-0222-1881
haavard.rue@kaust.edu.sa*

Two-Stage Sampling Design and Sample Selection with the R Package R2BEAT

by Giulio Barcaroli, Andrea Fasulo, Alessio Guandalini, and Marco D. Terribili

Abstract R2BEAT ("R 'to' Bethel Extended Allocation for Two-stage sampling") is an R package for the optimal allocation of a sample. Its peculiarity lies in properly addressing allocation problems for two-stage and complex sampling designs with multi-domain and multi-purpose aims. This is common in many official and non-official statistical surveys, therefore R2BEAT could become an essential tool for planning a sample survey. The functions implemented in R2BEAT allow the use of different workflows, depending on the available information on one or more interest variables. The package covers all the phases, from the optimization of the sample to the selection of the Primary and Secondary Stage Units. Furthermore, it provides several outputs for evaluating the allocation results.

1 Introduction

Sample surveys carried out by National Statistical Institutes (NSIs) and by other institutions have multi-domain and multi-purpose objectives, so they have to provide accurate estimates for different parameters and different domains (i.e. geographical areas such as national, regional, and more).

Surveys have budgetary and logistical constraints, so they must be carefully planned to provide high-quality estimates for parameters of interest. In this context, several decisions need to be made, such as sample size, stratification, allocation of sampling units among the strata and multiple-stage.

These decisions may not be trivial and can strongly affect all the following steps of the survey and, moreover, the quality of the results. This justifies the care and attention typically given in the literature. A seminal work in this perspective is that of Kish (1965) and, later, different approaches have been proposed for defining optimal sampling strategies, i.e. maximizing data quality within budgetary constraints (see among the others Cochran, 1977).

The proposed package, R2BEAT (standing for R "to" Bethel Extended Allocation for Two-stage), fits into this context and fills a gap within the range of statistical software concerning sample size allocation, thereby introducing an improvement into the R community.

There are several R packages for allocating a stratified sample, such as surveyplanning (Breidaks et al., 2020), PracTools (Valliant et al., 2020), samplesize4surveys (Rojas, 2020), optimStrat (Bueno, 2020) and SamplingStrata (Barcaroli, 2014). But R2BEAT, extending the methodology implemented in the Italian National Institute of Statistic's open-source software called Mauss-R ("Multivariate Allocation of Units in Sampling Surveys") (Barcaroli et al., 2020), enables us also to compute the optimal allocation among strata for two-stage and complex sampling design considering both multivariate and multi-domain cases.

In the following section the methodological aspects, underlying the package and its functions, will be presented in detail: the optimal allocation of the sample and its selection will be illustrated. In the third section will be shown how to prepare, organize and check the input data needed by the package for allocating the whole sample size among strata and finally select the units. A case study on a synthetic dataset will be used as an example to test the package functions, and also for a comparison to the other two packages handling two-stage sample design (PracTools and samplesize4surveys). Finally, the concluding remarks will point out that R2BEAT can allocate the sample more efficiently than the other available multistage allocation software, while also being strongly flexible, generalizable, and integrated with software that manages all the different phases of the statistical data production process.

2 Methodological aspects

2.1 The optimal allocation

Let us consider a population U of size N ($k = 1, \dots, N$) partitioned in H subgroups, U_h ($h = 1, \dots, H$), called strata. Hence, each stratum contains N_h elements, where N_h is assumed to be known such that $\sum_{h=1}^H N_h = N$.

The strata can be defined in different ways on the basis of one or more qualitative variables known for all the units in the population.

Then, we assume, at least for the moment, to be interested in investigating the mean of just one y variable in the population U ,

$$\mu_y = \frac{\sum_{k \in U} y_k}{N} \quad (1)$$

where y_k is the value of the y variable observed on the k -th unit in the population U . The y variable could be a quantitative variable or dichotomous, that is $y \in \{0, 1\}$. Please note that, even when y is a dichotomous variable, expression (1) holds and μ_y is equal to the proportion of units in the population for which $y = 1$.

Furthermore, assume we want to estimate μ_y through a probabilistic sample s of size n with the estimator

$$\hat{Y} = \frac{\hat{Y}_{HT}}{N} = \frac{\sum_{k \in s} y_k d_k}{N} \quad (2)$$

where \hat{Y}_{HT} is the Horvitz-Thompson estimator for the total (Horvitz and Thompson, 1952) in which d_k is the design weight usually equal to the inverse of the first order inclusion probability.

The sample size of a survey, n , is usually exogenous information, dictated by budget and, sometimes, by logistic constraints associated with the unit k in the sample. Then, in practice, the problem comes down to the allocation of the n units in the H strata, such that $\sum_{h=1}^H n_h = n$.

Therefore, let us define

$$\mu_{hy} = \frac{\sum_U y_k \mathbf{1}_h}{N_h} \quad (3)$$

the mean of the y in each stratum where $\mathbf{1}_h$ is the membership indicator for the unit k in the stratum h .

In the same way, expression (2) can be easily adapted for estimating μ_{hy} , that is

$$\hat{Y}_h = \frac{\hat{Y}_{HT,h}}{N_h} = \frac{\sum_{k \in s_h} y_k d_k}{N_h}, \quad (4)$$

where s_h is the sample in the stratum h . The sampling variance estimator of \hat{Y}_h is given by

$$\widehat{\text{var}}(\hat{Y}_h) = \frac{1}{n_h - 1} \left(\frac{1}{n_h} - \frac{1}{N_h} \right) \sum_{k \in h} (y_{hk} - \bar{y}_h)^2, \quad (5)$$

where \bar{y}_h is the sample mean of the variable y in the stratum h .

The mean of y in (1) can be written also as $\mu_y = \sum_{h=1}^H (N_h/N) \mu_{hy}$ and, consequently, \hat{Y} in (2) as $\hat{Y} = \sum_{h=1}^H (N_h/N) \hat{Y}_h$. Therefore, the sampling variance estimator for \hat{Y} is $\widehat{\text{var}}(\hat{Y}) = \sum_{h=1}^H (N_h/N)^2 \widehat{\text{var}}(\hat{Y}_h)$.

When there is no information on y , the sample size to be allocated to each stratum, n_h , can be assigned by performing uniform or proportional allocation.

Uniform allocation assigns an equal number of sampling units to each stratum, that is $n_h^{UNIF} = n/H$.

More often, we want the sample size assigned to strata in the sample to be proportional to the sizes of the strata in the population, that is $n_h^{PROP} = n (N_h/N)$ where N_h/N is the weight of the stratum in the population with $\sum_{h=1}^H N_h/N = 1$. If the size is the same for all strata ($N_1 = \dots = N_h = \dots = N_H = N/H$), n_h^{PROP} comes down to n_h^{UNIF} .

When there is information in the population strata on y and in particular on its variance, S_{yh}^2 , a more favorable allocation can be performed. Alternatively, it is possible to consider also a proxy variable highly correlated with y . In this case, Tschprow (1923) demonstrated that the optimal allocation can be obtained by

$$n_h^{OPT} = n \frac{\frac{N_h}{N} \sqrt{S_{yh}^2}}{\sum_{h=1}^H \frac{N_h}{N} \sqrt{S_{yh}^2}}$$

However, this result, also published by Neyman (1934), is more often referred to as Neyman's allocation. The rationale behind the optimal allocation is that strata with more weight and in which y has much more variability need many more observations to reach better estimates. If the variance is the same in all the strata ($S_1^2 = \dots = S_h^2 = \dots = S_H^2$), n_h^{OPT} comes down to n_h^{PROP} .

The computation of the population variance is a crucial point in the optimal allocation. A dis-

tinction between the types of variables and the sources from which they can be obtained is needed. When y is a dichotomous variable available from a population register, its population variance can be computed as

$$S_{yh}^2 = p_h \times (1 - p_h) \quad (6)$$

where p_h is the proportion of units with $y = 1$ in the population strata. In the case of a quantitative variable, S_{yh}^2 is equal to $S_{yh}^2 = \left[\sum_{k \in U_h} (y_k - \mu_{yh})^2 \right] / N_h$.

When there is no population register, information on the variability can be obtained from a sample survey or a pilot survey which has previously been carried out. Let us assume to have collected the y variable, or at least its proxy variable, on a sample s^* . Then, (6) can be computed just by replacing p_h with

$$\hat{p}_h = \frac{\sum_{k \in s_h^*} y_k w_k}{\sum_{k \in s_h^*} w_k}, \quad (7)$$

that is the related estimate for each stratum obtained from the sample s^* . In (7) w_k is the sampling weight associated with the unit k in the sample s^* .

Instead, when y is a quantitative variable, $S_{yh}^2 = \hat{M}_h^2 - \hat{Y}_h^2$ where $\hat{M}_h^2 = (\sum_{k \in s_h^*} y_k^2 w_k) / N$ and $\hat{Y}_h = (\sum_{k \in s_h^*} y_k w_k) / N$ are the quadratic mean and the arithmetic mean estimated on the sample s^* in the h -th stratum, respectively.

The optimal allocation for just one y variable is of little practical use unless the various variables under study are highly correlated. This is because an allocation that is optimal for one characteristic is generally far from being optimal for others.

Therefore, several works have been devoted to solving the problem when more than one variable of interest has to be measured on each sampled unit. All the contributions can be classified into two main approaches: the "average variance" and convex programming.

The methods under the "average variance" approach consist of defining a weight for each variable to consider, computing a weighted average of the stratum variance and finding the optimal allocation using the "average variance" which results. They are computationally simple, intuitive and can be solved under fixed cost assumption. However, the choice of the weights is completely arbitrary and the optimal properties are not clear (see, e.g., Kish, 1976, for more details).

Instead, the other approach includes methods that use convex programming to find the minimum cost allocation, satisfying the fixed constraints regarding the variances of all the sampling variables. The obtained allocation is actually optimal, but sometimes it can exceed the budgetary constraints.

The most important method in the convex programming approach is the Bethel algorithm (Bethel, 1989) which extends the Neyman allocation to the multivariate case, providing the optimal allocation in the strata, in terms of surveying cost, according to a set of variables observed in a multidomain context.

In particular, when we are interested in investigating the mean of more than one y variable (quantitative or dichotomous), namely $y_1, \dots, y_i, \dots, y_I$, the optimal allocation problem reduces, in practice, to a minimum optimization problem of a convex function under a set of linear constraints

$$\begin{cases} C = \min \\ \widehat{CV}(\hat{Y}_{i,h}) \leq \delta(\hat{Y}_{i,h}) & i = 1, \dots, I \\ h = 1, \dots, H \end{cases} \quad (8)$$

where C is the global cost of the survey to be minimized and $\widehat{CV}(\hat{Y}_{i,h})$ is the estimate of the relative error. The estimate of the relative error,

$$\widehat{CV}(\hat{Y}_{i,h}) = \frac{\sqrt{\widehat{\text{var}}(\hat{Y}_{i,h})}}{\hat{Y}_{i,h}}, \quad (9)$$

is the ratio between the estimate of the sampling variance for the mean estimator of y_i variable ($i = 1, \dots, I$) in the stratum h given by expression (5) and the related estimate. In this case, $\widehat{CV}(\hat{Y}_{i,h})$ is called expected error and it must be less than or equal to the precision constraints defined by the user or by regulation, $\delta(\hat{Y}_{i,h})$.

Bethel (1989) demonstrates that the solution to this optimization problem exists and can be obtained

through an algorithm that applies the Lagrange multipliers method. The solution is a real number, so it must be rounded to provide an integer stratum sample size. The rounding clearly causes some deviations from the solution that, however, do not affect its optimality (Cochran, 1977).

This framework also works in the case of the multi-domain problem. Usually, estimates of a survey are disseminated for the whole population and sub-domains, for instance for geographical areas. Then, it is useful to define the optimal allocation also taking into account these outcomes of the survey.

Sub-domain estimation is actually a long-established theory (Särndal et al., 2003). Expression (1) can be easily adapted just by introducing the sub-domain membership indicator variable, $\mathbf{1}_{k,d}$, which equals 1 for each the unit k in the domain d and 0 otherwise, that is $\mu_y^d = (\sum_{U_d} y_k \mathbf{1}_{k,d}) / N_d$ where N_d is the population size in the domain d ($d = 1, \dots, D$). It is important to point out, that domains must be an aggregation of strata and thus should not split strata. Then, it is sufficient to consider the domain estimates in the minimum optimization problem in (8) and use Bethel's algorithm for deriving the multivariate allocation in the multi-domain case.

However, in official statistics, especially for household surveys, two-stage sampling designs are usually adopted. Two-stage sampling is based on a double sampling procedure: one on the primary stage units (PSUs) and another on the second stage units (SSUs). For instance, in the household survey, the PSUs are the municipalities, which are selected first. Then, in each selected municipality, a sample of households - the SSU - can be selected.

Two-stage sampling permits more complex sampling strategies and, moreover, it helps in the organization and cost reduction of data collection, because it reduces the interviewer's travels. However, this economic saving is counterbalanced with a loss of efficiency of the estimates. In fact, each additional stage of selection usually entails an increase of the sampling variance of the mean estimator. This increase can be assessed by the design effect ($deff$) that measures how much the sampling variance of \hat{Y}_i , under the adopted sampling design (des), is inflated with respect to a simple random sample (srs), with the same sample size. An estimate of the design effect can be given by the expression:

$$deff(\hat{Y}_i) = \widehat{\text{var}}(\hat{Y}_i)_{des} / \widehat{\text{var}}(\hat{Y}_i)_{srs}$$

A rough approximation of the $deff$ can be obtained when the clusters have the same sample size and the same inclusion probability (Cicchitelli et al., 1992),

$$deff(\hat{Y}_i) = 1 + \rho_i (b - 1) \quad (10)$$

where b is the average cluster (i.e. PSU) size in terms of the final sampling units and ρ_i is the intra-class correlation within the cluster (PSU) for the variable y_i ($i = 1, \dots, I$).

The intra-class correlation provides a measure of data clustering in PSUs and SSUs. In general, if ρ_i is close to 1, the clustering is high and it is convenient to collect only a few units in the cluster. Instead, if ρ_i is close to 0, the collection of units from the same cluster does not affect the efficiency of the estimates.

Also for computing ρ_i , we can distinguish whether a population register in which the y_i variables ($i = 1, \dots, I$), or at least their proxies, are available or not. In the former case, a good approximation, given in Cicchitelli et al. (1992), is

$$\rho_i = 1 - (D_{w_i} / D_{y_i}) \quad (11)$$

where $D_{w_i} = \sum_{h=1}^H \sum_{k=1}^{N_h} (y_{i,k} - \mu_{y_{i,h}})^2$ is the deviance within clusters and $D_{y_i} = \sum_{k \in U} (y_{i,k} - \mu_{y_i})^2$ is the global deviance of the y_i variable. Remember that $D_{y_i} = D_{w_i} + D_{b_i}$, where $D_{b_i} = \sum_{h=1}^H N_h (\mu_{y_{i,h}} - \mu)^2$, is the deviance between clusters. Therefore, $0 \leq \rho_i \leq 1$.

Instead, ρ_i can be estimated from a sample with the expression (10)

$$\hat{\rho}_i = (deff_i - 1) / (b - 1). \quad (12)$$

Here we consider, directly, a more general expression for the estimate of the $deff$ in terms of the intra-class correlation coefficient. This expression refers to a typical situation in household surveys where PSUs are assigned to Self-Representing (SR) strata, that is they are included for sure in the sample, or to Not-Self-Representing (NSR) strata, where they are selected by chance. In practice, this assignment is usually performed by comparing the measure of the size of PSUs to the threshold (see, e.g., Hansen et al., 1953):

$$\lambda = (\bar{m} \Delta) / f \quad (13)$$

where \bar{m} is the minimum number of SSUs to be interviewed in each selected PSU, $f = n/N$ is the sampling fraction and Δ is the average dimension of the SSU in terms of elementary survey units.

Then, Δ must be set to 1 if, for the survey, the selection units are the same as the elementary units (that is, household-household or individuals-individuals), whereas it must be set equal to the average dimension of the households if the elementary units are individuals, while the selection units are the households. PSUs with a measure of size exceeding the threshold are identified as SR, while the remaining PSUs are identified as NSR.

Then, the extended expression of d_{eff} (see, among others, Rojas, 2016) is

$$d_{eff}(\hat{Y}_i) = \frac{n}{N^2} \left\{ \frac{N_{SR}^2}{n_{SR}} [1 + (\rho_{i,SR} (b_{SR} - 1))] + \frac{N_{NSR}^2}{n_{NSR}} [1 + (\rho_{i,NSR} (b_{NSR} - 1))] \right\} \quad (14)$$

where, for SR and NSR strata,

- N_{SR} and N_{NSR} are the population sizes;
- n_{SR} and n_{NSR} are the sample sizes;
- $\rho_{i,SR}$ and $\rho_{i,NSR}$ are the intra-class correlation coefficients for the variable i ($i = 1, \dots, I$);
- b_{SR} and b_{NSR} are the average PSU size in terms of the final sampling units.

Of course, if there are no SR strata, this expression simplifies to Equation (10). The design effect is equal to 1 under the *srs* design and increases for each additional stage of selection, due to the intra-class correlation coefficient which is, usually, positive.

The intra-class correlation coefficient for NSR can be computed with expression (11) or (12) whether population register data are available or not. It is not necessary to compute the intra-class correlation coefficient for SR strata because just one PSU is selected and the intra-class correlation is 1 by definition.

Therefore, under a two-stage sample design for determining the optimal allocation, the number of PSUs and SSUs must be determined.

A solution which makes iterative use of the Bethel algorithm has been proposed by Falorsi et al. (1998). In fact, at the first iteration, the Bethel algorithm is applied. The optimal allocation for a stratified simple sampling design is obtained. Then, this allocation is used to update the threshold in (13) and the design effect in (14). A new design effect is computed and used in turn to inflate the S_h^2 (or equivalently \hat{S}_h^2). It is used as input in the next iteration in which the Bethel algorithm is used again. The obtained allocation is used again to update the threshold and the design effect, and a new allocation is found. The process is iterated until the difference between two consecutive iterations is lower than a predefined threshold.

However, as pointed out by Waters and Chester (1987), different combinations yield the same variance and can satisfy the precision constraints, $\delta(\hat{Y}_{i,h})$. The optimal solution strongly depends on the budgetary constraints that limit the SSUs and the data collection organization that influences the maximum number of PSUs that can be managed.

All this discussion holds when you want to use the *HT* estimator. But, currently, the most applied estimator for the NSIs survey is the calibrated estimator (Deville and Särndal, 1992). The calibrated estimator, through the use of auxiliary variables, usually provides better estimates than *HT*. The use of a different estimator from the *HT* can be considered since the allocation phase, by accounting for the estimator effect and following the procedure explained above

An estimate of the estimator effect (*effst*) is given by

$$effst(\hat{Y}_i) = \frac{\text{var}(\hat{Y}_i)}{\text{var}(\hat{Y}_{i,HT})}. \quad (15)$$

It measures how much the sampling variance of the applied estimator under the adopted design is inflated or deflated with respect to the sampling variance of the *HT* estimator, on the same sample design.

2.2 Sample selection

Once the optimal allocation is defined, the selection of sampling units must be performed.

In the case of a stratified two-stage sampling design two sampling selections need to be done: one for PSUs and one for SSUs.

In each stratum, the PSUs are split into SR and NSR according to a size threshold (13). PSUs with a measure of size exceeding the threshold are identified as SR, included for sure in the sample and each

Algorithm 1: R2BEAT optimal allocation of PSUs and SSUs in sampling strata**Input :**

- a. precision constraints in terms of CV;
- b. information on sampling strata (mean and stdev of target variables, N , ...);
- c. information on previous design: $deff$, $effst$, ρ ;
- d. information on PSUs in sampling strata (measure of size);
- e. minimum number of SSUs per PSU;

Output:

- a. for each stratum: number of PSUs and SSUs to be selected;
- b. expected CVs for target estimates;
- c. sensitivity of the solution;

REM First iteration;

1. input deff is used to inflate standard deviations of target variables in sampling strata;
2. optimal allocation of SSUs in sampling strata is obtained by applying the Bethel algorithm as if it were a one-stage sampling design;
3. the number of PSUs is determined on the basis of the minimum number of SSUs per PSU;
4. the threshold for determination of self-representing PSUs is calculated;
5. new deff is calculated and used to update the standard deviations of target variables in sampling strata;

REM Next iterations;

while not convergence **do**

1. optimal allocation of SSUs in sampling strata is obtained by applying the Bethel algorithm;
2. the number of PSUs is determined on the basis of the minimum number of SSUs per PSU;
3. the threshold for determination of self-representing PSUs is calculated;
4. new deff is calculated and used to update standard deviations of target variables in sampling strata;
5. the iteration stops if
 - a. the difference between the sample sizes of two iterations is lower than 5 (default value) or
 - b. the maximum of defts (square root of deffs) largest differences is lower than 0.06 (default value) or
 - c. the number of iterations is higher than 20 (default value);

end

of them constitutes an independent sub-stratum. Therefore, the probability that they are included in the sample (inclusion probability, π_I) is always equal to 1.

It is possible that that it can happen that no PSU has a measure of size higher than the threshold: this can happen for example when we consider as PSUs the census enumeration areas, whose distribution of the measure of size is about uniform; on the contrary, it is unlikely to happen when we consider municipalities.

The remaining PSUs, NSR-PSUs, are ordered by their measure of the size and divided into finer strata (*sub-strata*) whose sizes are approximately equal to the threshold multiplied by the number of PSUs to be selected in each stratum. In this way, sub-strata are composed of PSUs having size as homogeneous as possible.

The PSUs in each stratum can be selected in different ways. However, the selection of a fixed number of PSUs per stratum is usually carried out with Sampford's method (unequal probabilities, without replacement, fixed sample size).

Finally, the SSUs must be drawn in the selected PSU. Also in this case the SSU can be selected in different ways. In most cases, they are selected through a systematic sampling design that shares several properties with the *srs*.

Then, the design weight for the unit k in the h -th strata in the ℓ -th PSU is equal to the inverse of the product of the first stage and the second stage inclusion probabilities, $d_k = \frac{1}{\pi_{ij}} \frac{1}{\pi_{ij}}$. The design weights sum up to the population size, $\sum_{k \in s} d_k = N$, and are almost constant in each stratum, which means that the sample is self-weighting.

Algorithm 2: R2BEAT selection of PSUs

Input :

- a. The output of the allocation step (function `beat.2st`) (universe of PSUs, measure of PSUs, number of PSUs and SSUs to be selected in each stratum, threshold);

Output:

- a. universe of PSUs with stratum, sub-stratum, PSU first order inclusion probability, PSU weight, flag sample, and number of SSUs to be selected in each PSU;
- b. sample of PSUs (flag sample=1) with stratum, sub-stratum, PSU first order inclusion probability, PSU weight, number of SSUs to be selected in each PSU;
- c. statistics related to the sample of PSUs at stratum level;

REM creation of *sub-strata* and selection of PSUs;

1. in each stratum, PSUs are sorted in descending order according to their measure of size;
 2. the measure of size of PSUs are compared with the threshold;
 3. PSUs with a measure of size exceeding the threshold are identified as SR, included for sure in the sample and constitute an independent sub-stratum;
 4. the remaining PSUs, NSR-PSUs, are ordered in decreasing way by their measure of size and aggregated into finer strata (*sub-strata*);
 5. *sub-strata* are created adding PSUs (still in descending order of measure of size) for which the sum of the measure of size of the *sub-strata* is approximately equal to the threshold multiplied by the number of PSUs to be selected in each stratum;
 6. in each *sub-stratum* a fixed number of PSUs per stratum are usually selected with Sampford's method (unequal probabilities, without replacement, fixed sample size);
-

3 Structure of the package

The R2BEAT package provides functions for drawing complex sample designs using an optimal allocation also performing the selection of the PSUs and SSUs. To install the latest release version of R2BEAT from CRAN, type `install.packages("R2BEAT")` within R. The current development version can be downloaded and installed from GitHub by executing `devtools::install_github("barcaroli/R2BEAT")`.

The workflow to draw and select a complex sample using R2BEAT is: (1) prepare the input data, (2) check the input data, (3) define the design and obtain the allocation, and (4) select the final sample units.

3.1 Prepare the input data

As will be illustrated in detail in the next sub-sections the R2BEAT package provides functions to define one-stage stratified sample design (**beat.1st**) and two-stage stratified sample design (**beat.2st**). The preparation of the input dataset changes whether the former or the latter sample design will be adopted.

In the case of a multivariate optimal allocation for different domains in a stratified one-stage sample design, the function **beat.1st** can be used. This function requires two inputs, a data frame containing survey strata information (**stratif**) and a data frame of expected CV for each domain and each variable (**errors**). No functions to prepare these inputs are provided by the package but it is possible to follow the example dataset **stratif** and **error** to properly create the input datasets for the function **beat.1st**.

In the case of a two-stage design, two functions are provided by the package to help in the creation of the input data for the function **beat.2st**. There are two functions because two different scenarios are possible, depending on the initial information available:

1. Only the sampling frame is available, no previous rounds of the survey have been carried out. In this scenario, a strict condition on the information content of the sampling frame must hold: values of the sample target surveys (or of their proxy correlated variables) are available for each unit in the frame. This can be accomplished by considering the previous census, or by using administrative registers. In this scenario, the function **prepareInputToAllocation1** can be used to create the input dataframes **stratif**, **rho**, **deft**, **effst**, **des_file** and **psu_file**.

2. Together with a sampling frame containing the units of the population of reference, also a previous round of the sampling survey to be planned is available. The **prepareInputToAllocation2** produces the same outputs of **prepareInputToAllocation1**, but it requires the design and/or calibrated objects of the previous sample survey, obtained using the ReGenesee package (Zardetto, 2015).

The function **sensitivity_min_SSU** allows analyzing the different results in terms of first stage size (number of PSUs) and second stage size (number of SSUs), obtained when varying the values of the minimum number of SSUs to be selected in each PSU.

When using a previous survey, the 'strata' dataframe is automatically obtained by estimating all the variables in it, including the 'N', that is the population in strata. Being an estimate, these values can differ from the ones obtained by the 'des' dataframe (obtained by aggregating PSUs values). It is up to the user to decide if PSUs derived N values are more reliable than those obtained by the survey, and, in case, to assign those to the 'strata' dataframe. To check the coherence between the estimated population in the strata (**stratif**) and the population calculated by the PSUs dataset (**des_file**), the function **check_input** is provided to the users. This function compares the strata sizes giving information about the differences and replacing the estimated stratum size with the stratum population calculated by the PSUs dataset.

3.2 Defining the design and determining the allocation

The package allows performing the optimal allocation for both one-stage and two-stage stratified sampling.

The first one is implemented within the function **beat.1st** and computes a multivariate optimal allocation for different domains in a one-stage stratified sample design. As described in the previous section, in a one-stage stratified sample design there are only two inputs to be provided to **beat.1st**: the dataframes **stratif** and **errors**. Besides these two mandatory inputs, it is also possible to indicate the minimum number of sampling units to be selected in each stratum, by default set equal to 2.

The function **beat.2st** performs the same multivariate optimal allocation for different domains considering stratified two-stage design. Together with the input data **stratif** and **errors** other mandatory input are:

- **des_file**: dataframe containing a row per each stratum, with information on total population, the values of the **delta** parameter (equal to the mean number of final SSUs contained in clusters to be selected, for instance, the mean number of individuals in a household), and the minimum number of SSUs to be selected in each PSU;
- **psu_file**: dataframe containing information on each PSUs (identifier, stratum, measure of size).
- **rho**: dataframe contains a row per each stratum with the intra-class correlation coefficient both for self representing and non-self representing PSUs.

Is also possible to provide optional information about:

- **deft_start**: dataframe containing a row per each stratum with the starting values for the square root of the design effect in the stratum of each variable of interest.
- **effst**: dataframe containing a row per each stratum with the estimator effect for each variable of interest.

The functions **beat.1st** and **beat.2st** produce lists with respectively 4 and 9 items.

The **beat.1st** output contains:

1. **n**: a vector with the optimal sample size for each stratum;
2. **file_strata**: a dataframe corresponding to the input dataframe **stratif** with the *n* optimal sample size column added;
3. **alloc**: a dataframe with optimal (**ALLOC**), proportional (**PROP**), equal (**EQUAL**) sample size allocation;
4. **sensitivity**: a data frame with a summary of planned coefficients of variation (**Planned CV**), the expected ones under the given optimal allocation (**Actual CV**), and the sensitivity at 10% for each domain and each variable. Sensitivity can be a useful tool to help in finding the best allocation, as it provides a hint of the expected sample size variation for a 10% change in planned CVs.

Together with the previous outputs, the function **beat.2st** produces also:

1. **iterations**: a dataframe that for each iteration of the Bethel algorithm provides a summary with the number of PSUs (**PSU_Total**), distinguished between SR (**PSU_SR**) and NSR (**PSU_NSR**), plus the number of SSUs;
2. **planned**: a dataframe with the planned coefficients of variation for each variable in each domain.
3. **expected**: a dataframe with a summary of expected coefficients of variation under the given optimal allocation for each target variable in each domain;
4. **deft_c**: a dataframe with the design effect for each variable in each domain in each iteration. Note that **DEFT1_0 - DEFTn_0** is always equal to 1 if **deft_start** is NULL; otherwise it is equal to **deft_start**. While **DEFT1 - DEFTn** are the final design effect related to the given allocation.
5. **param_alloc**: a vector with a resume of all the parameters given for the allocation.

3.3 Sample units selection

Once the allocation for the primary and secondary sampling stage units has been defined, it is possible to use two functions for the selection of the final sampling units.

The function **select_PSU** allows the users to select the PSUs allocated in each stratum, using the Sampford method, as implemented by the **UPsampford** function of R package **sampling** (Tillé and Matei, 2021).

The input of this function is the output of the **beat.2st** function.

The output of the function is a list containing the following items:

1. **universe_PSU**: a dataframe that reports the whole universe of PSUs, with the inner strata formed for the selection;
2. **sample_PSU**: a dataframe containing the selected PSUs, with the indication, for each of them, of how many SSUs must be selected;
3. **PSU_stats**: a table containing summary information on selected PSUs.

In the last step, the selection of a sample of SSUs has to be carried out. The function **select_SSU** allows selecting a sample of SSUs from the population frame, based on the SSUs allocated to each selected PSUs.

The input datasets are two:

1. **df**: the dataframe containing the final sampling units;
2. **PSU_sampled**: the dataframe containing selected PSUs, corresponding to the second item of the output of the **select_PSU** function.

The function **select_SSU** returns a dataframe containing the selection of the **df** dataframe, enriched with information about the first stage inclusion probability, the second stage inclusion probability, the final inclusion probability (the product of the first stage and the second stage inclusion probabilities) and the design weights.

4 Illustrative examples

To illustrate how to implement workflows making use of **R2BEAT** functions, we will consider two scenarios, depending on the initial setting:

1. only the sampling frame is available, no previous rounds of the survey have been carried out;
2. together with a sampling frame containing the units of the population of reference, a previous round of the sampling survey to be planned is available;

In both cases, we assume that the sampling frame contains information on the final sampling units, together with the indication of the PSUs to which each unit belongs. In the first scenario, a stricter condition on the information content of the sampling frame must hold: values of the sample target surveys (or of their proxy correlated variables) must be available for each unit in the frame. This can be accomplished by considering a previous census, or by imputing values using predictive models. In the following paragraphs, we will show only a subset of the code necessary to produce the final results, the relevant part of it¹.

4.1 Scenario 1 workflow

In this scenario, it is assumed that a sampling frame is available. We consider a frame (**pop.RData**), containing 2,258,507 units:

```
'data.frame': 2258507 obs. of 13 variables:
 $ region      : Factor w/ 3 levels "north","center",...: 1 1 1 1 1 1 1 1 1 ...
 $ province    : Factor w/ 6 levels "north_1","north_2",...: 1 1 1 1 1 1 1 1 1 ...
 $ municipality: num  1 1 1 1 1 1 1 1 1 ...
 $ id_hh       : Factor w/ 963018 levels "H1","H10","H100",...: 1 1 1 2 3 3 3 3 1114 1114 ...
 $ id_ind      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ stratum     : Factor w/ 24 levels "1000","2000",...: 12 12 12 12 12 12 12 12 12 12 ...
 $ stratum_label: chr  "north_1_6" "north_1_6" "north_1_6" "north_1_6" ...
 $ sex         : int  1 2 1 2 1 1 2 2 1 1 ...
 $ cl_age      : Factor w/ 8 levels "(0,14]", "(14,24]",...: 3 7 8 5 4 6 6 4 4 1 ...
 $ active      : num  1 1 0 1 1 1 1 1 0 ...
 $ income_hh   : num  30488 30488 30488 21756 29871 ...
 $ unemployed  : num  0 0 0 0 0 0 0 0 0 ...
 $ inactive    : num  0 0 1 0 0 0 0 0 1 ...
```

covering a (synthetic) population of reference, with basic information (geographical and demographic variables):

- region: the NUTS2 identifier;
- province: the NUTS3 identifier;
- municipality: identifier of the municipality, that plays the role of the PSU identifier;
- id_hh: the household identifier;
- id_ind: the individual identifier;
- stratum and stratum_label: identifier of the initial strata (provinces);
- sex and cl_age: demographic information on individuals.

together with information that is related to the sampling survey we want to design:

¹In order to reproduce the processing related to these examples, datasets and R scripts are downloadable from the link <https://zenodo.org/records/10183968>.

- **active, inactive, unemployed**: binary variables indicating the occupation status of the individual;
- **income_hh**: household income.

We suppose that the values of these variables have been made available by a different source (for instance a census) or by predicting them with a model-based approach. In any case the uncertainty related to these values should be taken into account, by correctly evaluating the anticipated variance related to the models used for the predictions when producing the **strata** dataset (Baillargeon and Rivest, 2011, p. 59).

Anyway, in the following, we will not consider this issue, as we want only to illustrate how it is possible to automatically derive all the inputs required by the next steps.

Step 1: preparation of the inputs for the optimal sample design

The function **prepareInputToAllocation1** allows preparing all the inputs required by the optimal allocation step under this first scenario. This function requires the attribution of values to the following parameters: **samp_frame**, **id_PSU**, **id_SSU**, **strata_var**, **target_vars**, **deff_var**, **domain_var**, **delta** (average dimension of the SSU in terms of elementary survey units), **minimum** (minimum number of SSUs to be interviewed in each selected PSU).

About the values of these parameters, the choices are almost always driven by the content and structure of the sampling frame, except for **minimum**. To choose a suitable value for this parameter, the function **sensitivity_min_SSU** allows performing a sensitivity analysis, showing how the first and second stage sample sizes vary by varying their values:

```
> sens_min_SSU <- sensitivity_min_SSU (
+   samp_frame=pop,
+   id_PSU="municipality",
+   id_SSU="id_ind",
+   strata_var="stratum",
+   target_vars=c("income_hh","active","inactive","unemployed"),
+   deff_var="stratum",
+   domain_var="region",
+   delta=1,
+   deff_sugg=1.5,
+   min=30,
+   max=80)
```

This function calculates 11 different pairs of values for the number of PSUs and SSU as resulting from the allocation step, starting with the value '30' assigned to the parameter **minimum**, ending with the value '80'. The results are reported in Figure 1.

On the basis of the results of the sensitivity analysis, we can set the minimum (for instance 50) in the **prepareInputToAllocation1** function:

```
> inp <- prepareInputToAllocation1(
+   samp_frame = pop,
+   id_PSU = "municipality",
+   id_SSU = "id_ind",
+   strata_var = "stratum",
+   target_vars = c("income_hh","active","inactive","unemployed"),
+   deff_var = "stratum",
+   domain_var = "region",
+   delta = delta,
+   minimum = 50,
+   deff_sugg = 1.5)
Computations are being done on population data
Number of strata:  24
... of which with only one unit:  0
```

The output of this function (**inp**) is a list composed by the following dataframes: **strata**, **deff**, **effst**, **rho**, **psu_file**, **des_file**. These will be the inputs for the optimal allocation step (with the exception of the **deff**), which is produced only for documentation.

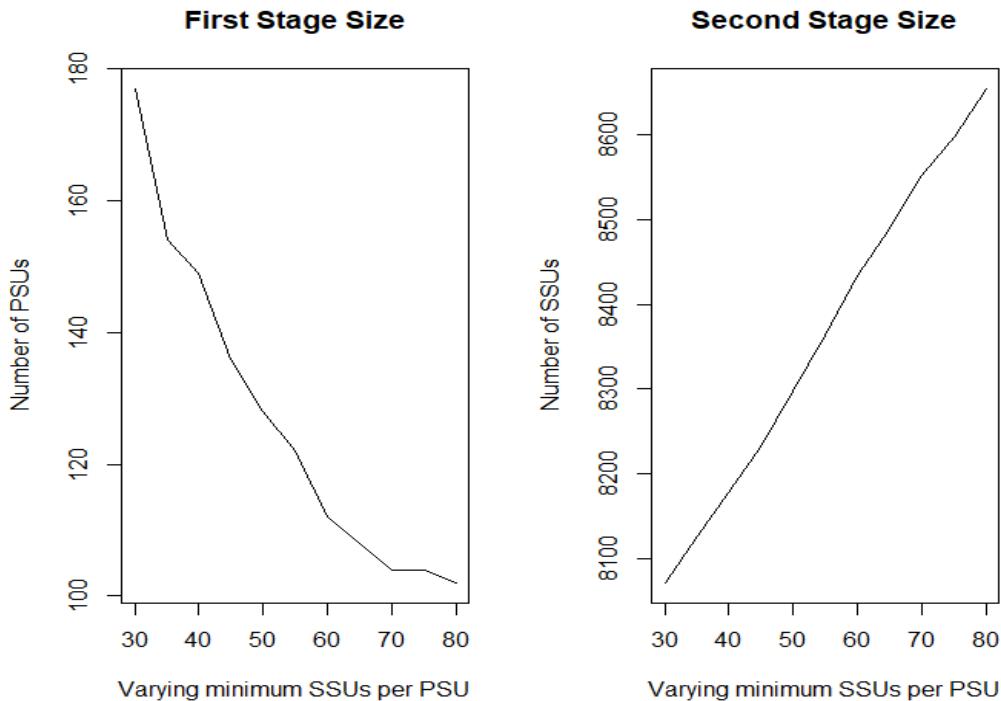


Figure 1: Sensitivity analysis for **minimum** parameter.

Step 2: optimization of PSUs and SSUs allocation

It is now possible to execute the optimization step of the sample design.

First of all, we define the set of precision constraints on the target variables:

```
DOM CV1 CV2 CV3 CV4
1 DOM1 0.02 0.03 0.03 0.05
2 DOM2 0.03 0.06 0.06 0.08
```

We interpret the values of the CVs in this way: the maximum expected coefficient of variation for the first target variable (**household income**) is 2% at the national level and 3% at the regional level; for **active** and **inactive** the expected maximum values of CV is 3% at the national level and 6% at the regional level; finally, for **unemployed** it is 5% at the national level and 8% at the regional level.

The optimization step is performed by executing the **beat.2s** function:

```
> inp1$desfile$MINIMUM <- 50
> alloc1 <- beat.2st(stratif = inp1$strata,
+                     errors = cv,
+                     des_file = inp1$des_file,
+                     psu_file = inp1$psu_file,
+                     rho = inp1$rho,
+                     deft_start = NULL,
+                     effst = inp1$effst,
+                     minPSUstrat = 2,
+                     minnumstrat = 50
+                     )
iterations PSU_SR PSU NSR PSU Total   SSU
1          0     0     0      0    7887
2          1     31    104    135  8328
3          2     39    104    143  8318
4          3     38    104    142  8321
```

This design is characterized by 142 PSUs (of which 38 self-representative, SR, and 104 non self-representative, NSR) and 8,321 SSUs.

Step 3: selection of PSUs and SSUs

We can now proceed in selecting the PSUs:

```
> sample_1st <- select_PSU(alloc, type="ALLOC", pps=TRUE, plot=TRUE)
> sample_1st$PSU_stats
   STRATUM PSU PSU_SR PSU_NSR   SSU SSU_SR SSU_NSR
1     1000    2      2       0  286    286       0
2     2000    9      3       6 452    152    300
3     3000    4      0       4  200       0    200
...
23    23000    4      0       4  200       0    200
24    24000    2      0       2 100       0    100
25 Total 142    38      104 9421   4221    5200
```

A discrepancy can be noted between the number of SSUs determined by the allocation step and the one produced by the selection of PSUs. This is because the selection of PSUs controls that the minimum number of SSUs to be allocated in each selected PSU is compliant with the minimum, in our case equal to 50: if not, this minimum is assigned. This is why the total number of SSUs increases from 8,320 to 9,421.

Selected PSUs are contained in the `sample_PSU` element of the output list:

```
> head(sample_1st$sample_PSU)
  PSU_ID STRATUM stratum SR nSR PSU_final_sample_unit Pik weight_1st weight_2st  weight
1    330    1000 1000-1  1  0                      207    1      1 706.0966 706.0966
2    309    1000 1000-2  1  0                      72     1      1 706.1806 706.1806
3     51   10000 10000-0  1  0                     171    1      1 196.8480 196.8480
4     11   10000 10000-1  1  0                     96     1      1 196.9688 196.9688
5     40   10000 10000-2  1  0                     79     1      1 197.9494 197.9494
6     13   10000 10000-3  1  0                     72     1      1 198.3750 198.3750
```

With this input, we can proceed to select the sample of final units. The distribution of PSUs and SSUs in the different strata is reported in Figure 2.

```
> PSU_sampled <- sample_1st$sample_PSU
> samp <- select_SSU(df=pop,
+                       PSU_code="municipality",
+                       SSU_code="id_ind",
+                       PSU_sampled)

-----
Total PSUs = 142
Total SSUs = 9421
-----
```

Step 4: verify compliance with precision constraints

The function `eval_2stage` allows verifying the compliance of the two-stage sample design to the set of precision constraints, by selecting a given number of different samples (in our case, 500) from the sampling frame, producing the estimates for each sample, and calculating over them the coefficients of variation for each target estimate.

We apply twice the function, first for the national level:

```
> # Domain level = national
> domain_var <- "one"
> set.seed(1234)
> eval111 <- eval_2stage(df,
+                         PSU_code,
+                         SSU_code,
+                         domain_var,
+                         target_vars,
+                         sample_1st$sample_PSU,
```

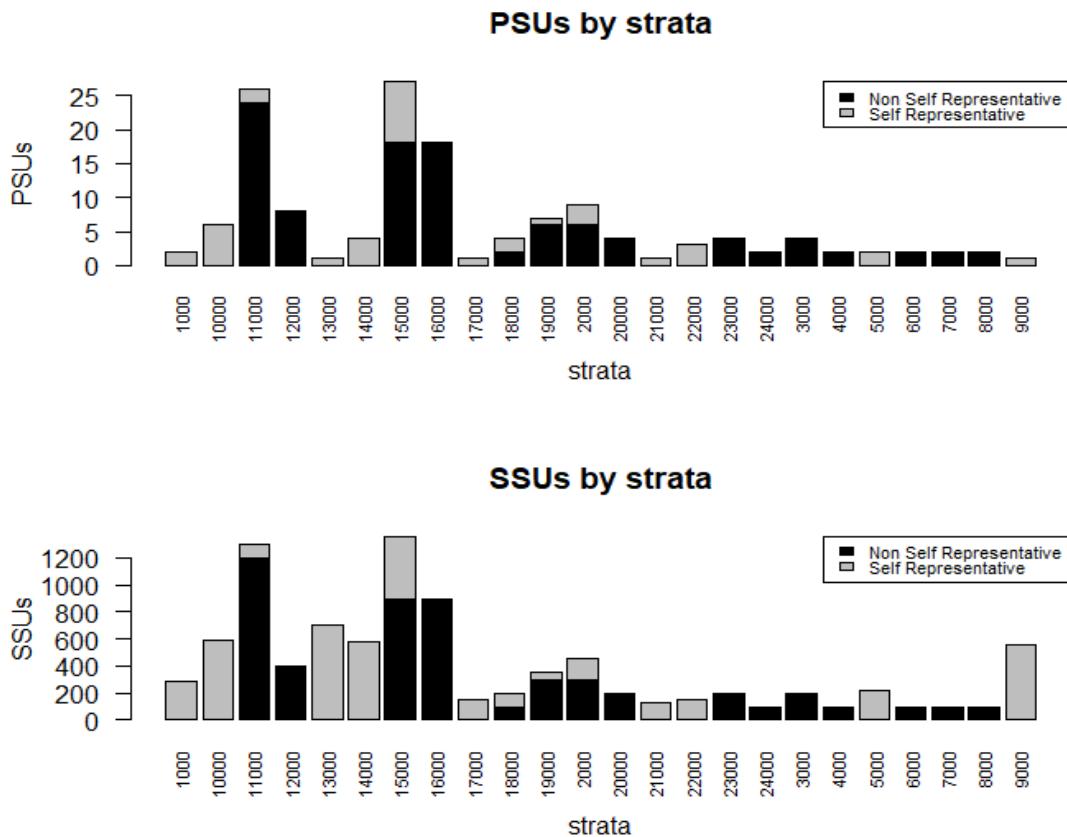


Figure 2: Allocation of PSUs and SSUs (scenario 1).

```

+
+                               nsampl=100,
+
+                               writeFiles=FALSE,
+
+                               progress=FALSE)
> eval11$coeff_var
      CV1      CV2      CV3      CV4   dom
1 0.0093  0.0099  0.025  0.0356 DOM1

```

then, at the regional level:

```

> # Domain level = regional
> domain_var <- "region"
> set.seed(1234)
> eval12 <- eval_2stage(df,
+                         PSU_code,
+                         SSU_code,
+                         domain_var,
+                         target_vars,
+                         sample_1st$sample_PSU,
+                         nsampl=100,
+                         writeFiles=FALSE,
+                         progress=FALSE)
> eval12$coeff_var
      CV1      CV2      CV3      CV4   dom
1 0.0105  0.0061  0.0224  0.0702 DOM1
2 0.0241  0.0171  0.0452  0.0652 DOM2
3 0.0281  0.0279  0.0533  0.0446 DOM3

```

We recall that the precision constraints had been set equal to 2% for the first variable, 3% for the second and third, and 5% for the fourth, at national level; and respectively to 3% and 6% and 8% at regional level. We can see that the computed CVs are all compliant.

4.2 Scenario 2 workflow

Together with the availability of a sampling frame, containing the same information presented in the previous scenario, we assume also the availability of at least one previous round of the survey. For sake of simplicity, we assume that the previous round sample is the same selected in scenario 1. We assume also that the values of the four target variables are the observed ones after the data collection. Having set the above conditions, the main difference with scenario 1 is that, instead of choosing in a somewhat arbitrarily way the values of the inputs required by the optimal allocation step, we can derive them directly from the collected survey data.

Step 1: processing and analysis of survey data

In this step, we proceed to perform the usual phases of calibration and production of the estimates. In doing that, we make use of the R package **ReGenesees**.

First, we describe the sample design:

```
> ## Sample design description
> sample$stratum_2 <- as.factor(sample$stratum_2) # stratum as factor, required by the next function
> sample.des <- e.svydesign(sample,
+                           ids = ~ municipality + id_ss,
+                           strata = ~ stratum_2,
+                           self.rep.str = ~ SR,
+                           weights = ~ weight,
+                           check.data = TRUE)
```

obtaining the **sample.des** object. Then we proceed with the calibration step:

```
> ## Calibration with known totals
> totals <- pop.template(sample.des,
+                         calmodel = ~ sex : cl_age,
+                         partition = ~ region)
> totals <- fill.template(pop, totals, mem.frac = 10)
> sample.cal <- e.calibrate(sample.des,
+                             totals,
+                             calmodel = ~ sex : cl_age,
+                             partition = ~ region,
+                             calfun = "logit",
+                             bounds = c(0.676, 1.279),
+                             aggregate.stage = 2,
+                             force = FALSE)
```

obtaining the **sample.cal** object.

These two objects are what is needed to obtain, in an automated way, all the inputs required by the optimization step.

Step 2: preparation of the inputs for the optimal sample design

The preparation of all the inputs required by the optimization step is a straightforward operation by using the **prepareInputToAllocation2** function:

```
> inp <- prepareInputToAllocation2(
+   samp_frame = pop,                      # sampling frame
+   RGdes = sample.des,                    # ReGenesees design object
+   RGcal = sample.cal,                   # ReGenesees calibrated object
+   id_PSU = "municipality",             # identification variable of PSUs
+   id_SSU = "id_ss",                     # identification variable of SSUs
+   strata_vars = "stratum",              # strata variables
+   target_vars = c("income_ss", "active", "inactive", "unemployed"), # target variables
+   deff_vars = "stratum",                # deff variables
+   domain_vars = "region",               # domain variables
+   delta = 1,                            # Average number of SSUs for each selection unit
+   minimum = 50)                         # Minimum number of SSUs to be selected in each PSU
```

The configuration of the output is the same as that seen in scenario 1 for the function `prepareInputToAllocation1`.

Step 3: optimization of PSUs and SSUs allocation

The optimal allocation of PSUs and SSUs is obtained in the same way as in the first scenario:

```
> set.seed(1234)
> inp2$des_file$MINIMUM <- 50
> alloc2 <- beat.2st(stratif = inp2$strata,
+                      errors = cv,
+                      des_file = inp2$des_file,
+                      psu_file = inp2$psu_file,
+                      rho = inp2$rho,
+                      deft_start = NULL,
+                      effst = inp2$effst,
+                      minnumstrat = 2,
+                      minPSUstrat = 2)
   iterations PSU_SR PSU NSR PSU Total   SSU
1             0     0     0      0  9546
2             1    71    92    163  8475
3             2    40   108    148  8406
4             3    38   108    146  8404
```

Step 4: selection of PSUs and SSUs

The selection of first and second stage units proceeds in exactly the same way as in scenario 1, first selecting the PSUs, and then the SSUs.

```
> sample_1st <- select_PSU(alloc2, type="ALLOC", pps=TRUE)
> sample_1st$PSU_stats
  STRATUM PSU PSU_SR PSU_NSR   SSU SSU_SR SSU_NSR
1     1000   2     2     0  276   276     0
2     2000  10     6     4  517   317   200
3     3000   4     0     4  200     0   200
...
24    24000   2     0     2  100     0   100
25  Total  146    38    108 9580  4180  5400
>
> samp <- select_SSU(df=pop,
+                       PSU_code="municipality",
+                       SSU_code="id_ind",
+                       PSU_sampled=sample_1st$sample_PSU,
+                       verbose=TRUE)
-----
Total PSUs = 144
Total SSUs = 9465
```

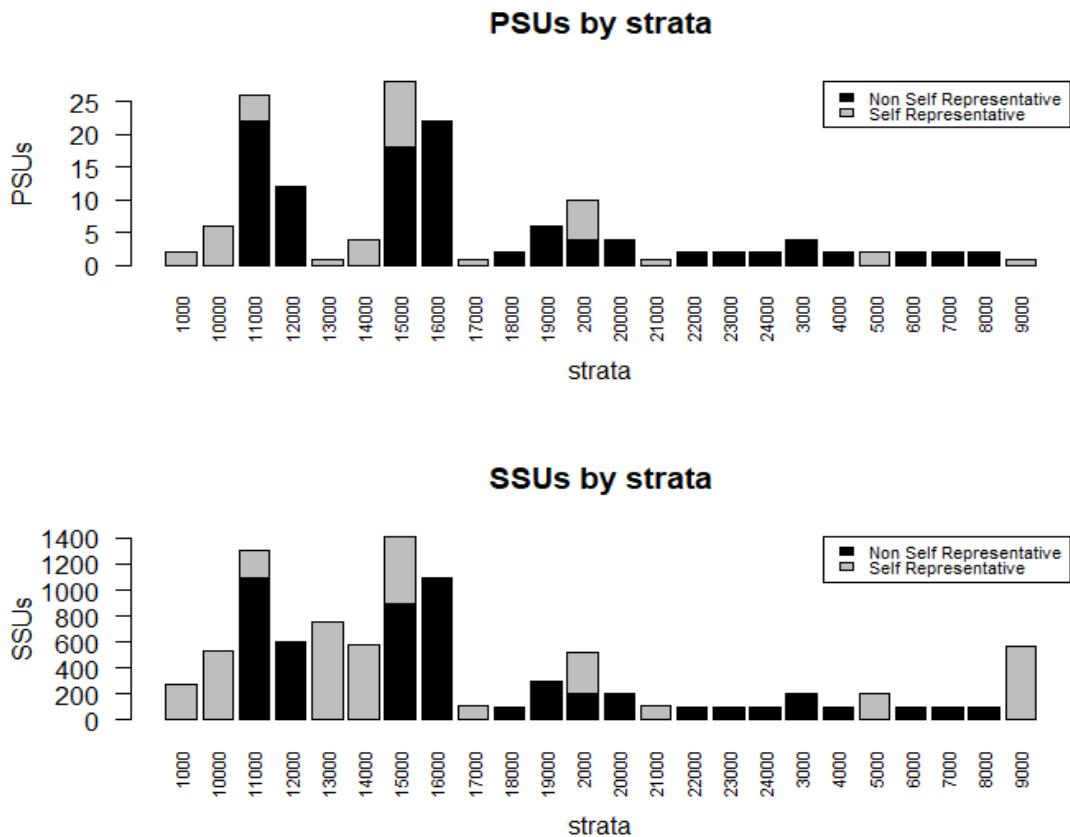
The distribution of PSUs and SSUs in the different strata is reported in Figure 3. It can be seen that the relative distribution of both units in the strata is quite similar to the one obtained in scenario 1.

Step 5: verify the compliance to precision constraints

As in the previous scenario, the final check consists in verifying the compliance of the optimized design to the precision constraints.

We, therefore, apply the function `eval_2stage`, first for the national level:

```
> # Domain level = national
> domain_var <- "one"
> eval21 <- eval_2stage(df,
+                        PSU_code,
```

**Figure 3:** Allocation of PSUs and SSUs (scenario 2).

```

+           SSU_code,
+           domain_var,
+           target_vars,
+           PSU_sampled=sample_1st$sample_PSU,
+           nsampl=100)
> eval21$coeff_var
   CV1    CV2    CV3    CV4   dom
1 0.0102 0.0095 0.0232 0.0354 DOM1

```

then, at regional level:

```

> # Domain level = regional
> domain_var <- "region"
> eval22 <- eval_2stage(df,
+                         PSU_code,
+                         SSU_code,
+                         domain_var,
+                         target_vars,
+                         PSU_sampled=sample_1st$sample_PSU,
+                         nsampl=100)
> eval22$coeff_var
   CV1    CV2    CV3    CV4   dom
1 0.0108 0.0067 0.0243 0.0824 DOM1
2 0.0240 0.0202 0.0514 0.0671 DOM2
3 0.0275 0.0353 0.0633 0.0417 DOM3

```

In this case, the expected precision is slightly higher than the precision constraints in two cases: the fourth variable in domain 1, and the third variable in domain 3. This can be due to the fact that in this scenario the variance of the target variables in strata (on the basis of which the total sample

size and the allocation are determined) is not derived from the sampling frame (as in scenario 1), but estimated from a previous round of the survey.

5 Comparison with other software

To evaluate the performance of R2BEAT, in this section we compare it to the other two R packages, i.e.:

1. the package **PracTools** (Valliant et al., 2020) implements many of the procedures described in Valliant et al. (2015), including those regarding the design of multistage samples;
2. the package **samplesize4surveys** (Rojas, 2020) allows to calculate the sample size for complex surveys.

First, we briefly illustrate, for both packages, the functions covering the two-stage sampling design, then we apply them to the same case seen in scenario 1, finally comparing the obtained results ².

5.1 R package PracTools

Valliant et al. (2015) describe (pages 231-234) a method for the optimal allocation of two-stage sampling when numbers of sample PSUs and elements per PSU are adjustable (which is our case).

This method is implemented in the R function **clusOpt2** in the **PracTools** package. This function computes the number of PSUs and the number of final units for each PSU for a two-stage sample which uses *srs* at each stage or probability proportional to size with replacement (*ppswr*) at the first stage and *srs* at the second.

This function requires the indication of a number of parameters, among which:

- C1: unit cost per PSU
- C2: unit cost per SSU
- delta: homogeneity measure
- unit.rv: unit relvariance
- k: ratio of B2+W2 to unit relvariance
- CV0: target CV
- tot.cost: total budget for variable costs
- cal.sw: indicates if the optimization has to be run for a fixed total budget, or for the target CV0

The function **BW2stagePPS** computes the population values of B2, W2, and delta whose meaning is explained in Valliant et al. (2015) (page 222).

The method is univariate: the optimization can be performed by indicating only one variable. The whole code required for the case described in scenario 1 is given here:

```
> load("pop.RData")
> library(PracTools)
> # Probabilities of inclusion (I stage)
> pp <- as.numeric(table(pop$municipality))/nrow(pop)
> # variable income_hh
> bw <- BW2stagePPS(pop$income_hh, pp, psuID=pop$municipality)
> bw
      B2          W2  unit relvar        B2+W2         k       delta
0.04075893  0.79538674  0.83601766  0.83614567 1.00015312  0.04874621
> des <- clusOpt2(C1=130,
+                   C2=1,
+                   delta=bw[6],
+                   unit.rv=bw[3],
+                   k=bw[5],
+                   CV0=0.02,
+                   tot.cost=NULL,
+                   cal.sw=2)
> des
C1 = 130
```

²In order to reproduce the processing related to the evaluation of the different software, datasets and R scripts are downloadable from the link <https://zenodo.org/records/10184220>

```
C2 = 1
delta = 0.04874621
unit relvar = 0.8360177
k = 1.000153
cost = 25499.72
m.opt = 141.4
n.opt = 50.4
CV = 0.02
> sample_size <- des$m.opt*des$n.opt
> sample_size
7126.56
```

In running the function, we have indicated that the optimization step was to be carried out having a target CV of 2% for the variable **income_hh**. As there is no way to directly indicate a desired minimum number of SSUs per PSU, we managed to obtain the desired value of 50 by indicating a couple of values 130 and 1 respectively for C1 and C2. As a result, the number of PSUs is 141 and the number of SSUs is 7,127.

5.2 R package samplesize4surveys

This package offers two functions to compute a grid of possible sample sizes for estimating single means (**ss2s4m**) or single proportions (**ss2s4p**) under two-stage sampling designs.

The required parameters are the following:

- N: the population size
- mu: the value of the estimated mean of a variable of interest
- sigma: the value of the estimated standard deviation of a variable of interest
- conf: the statistical confidence
- delta: the maximum relative margin of error that can be allowed for the estimation
- M: number of clusters in the population
- to: (integer) maximum number of final units to be selected per cluster
- rho: the intraclass correlation coefficient

Here is the code we used in the case of the target variable **income_hh**:

```
> load("pop.RData")
> PSU <- length(unique(pop$municipality))
> pop_strata <- as.numeric(table(pop$stratum))
> rho <- 0.04875369 # value taken from scenario 1 analysis
> ss2s4m(N = nrow(pop),
+         mu = mean(pop$income_hh),
+         sigma = sd(pop$income_hh),
+         delta = 0.02 * 1.96,
+         M = PSU,
+         to = 50,
+         rho = sum(rho$RHO_NAR1*pop_strata) / sum(pop_strata))
50 3.388931 142 50 7061
```

we obtain a design characterized by a total sample size of 7,061, with 142 PSUs.

Concerning the way we indicated the value of the parameter **rho**, we made use of the value of the intra-class correlation coefficient computed in scenario 1 by **R2BEAT**, not considering domains and strata.

In order to compare the 2% precision constraint expressed in terms of coefficient of variation, as the package requires the margin of error, we multiply the value of the CV by a z-value equal to 1.96, to obtain the ratio between the semi-width of the confidence interval and the estimate of the mean of the parameter.

The use of the function **ss2s4p**, applicable for the other three variables, is practically the same.

5.3 Comparison of results

We refer to the scenario 1 setting.

We consider the same precision levels for the four variables for the unique domain, set respectively to 2%, 3%, 3% and 5%.

We apply another constraint for all three packages, that is, we want to select a minimum number of final units in each PSU, set to 50.

There is no problem in doing that for package **samplesize4surveys**, by setting the parameter **to** equal to 50: the last value of the final grid is the result we want. Moreover, there is no loss in the optimality of the solution in doing that, because the sample sizes obtained for further values are increasingly higher.

As for **PracTools**, it is more complicated because there is no direct way to set this constraint. In any case, we manage to do that, by varying the value of C1 (leaving C2 equal to 1) until we find the solution with the nearest value of **n.opt** to 50.

A final consideration regarding the application of **R2BEAT**: in this setting, to be comparable with the other packages (that are univariate and mono-domain), it has been applied in a simplified way, that is, one variable per time (univariate), and no different domains and strata in the sampling frame. By so doing, **R2BEAT** yields obviously different results from those seen in scenario 1.

Table 1: Two-stage sample design obtained by different packages.

Variable	PracTools		R2BEAT		samplesize4surveys	
	PSUs	SSUs	PSUs	SSUs	PSUs	SSUs
active	49	2459	37	2030	49	2436
inactive	90	4395	68	4338	88	4391
income_hh	141	7127	79	5140	142	7061
unemployed	406	19956	149	10884	402	20058

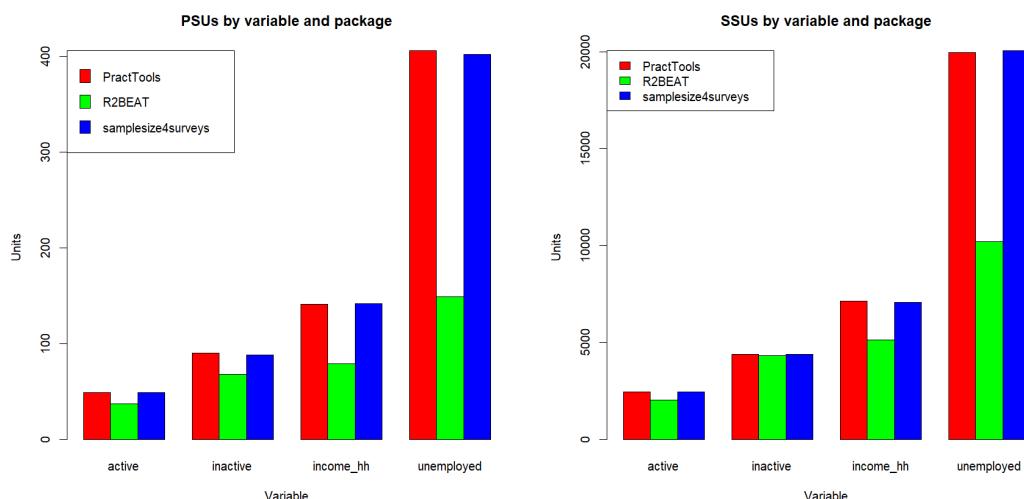


Figure 4: Sample sizes by packages.

In Table 1 and in Figure 4 are reported the results obtained by the three packages. For checking the reliability of the results, also a simulation has been performed to prove that **R2BEAT** provides sample sizes that always allow to obtain CVs below the planned ones.

Going in-depth with the comparison among the packages, it can be seen that **PracTools** and **samplesize4surveys** have similar behaviors and provide very close results, while **R2BEAT** always provide samples with smaller sample sizes, both in terms of PSUs and SSUs.

The differences related to the SSUs are smaller but there are two extremes. For "inactive" all three packages yield almost the same results in terms of SSUs, while for "unemployed" **R2BEAT** outperforms the other two competitors almost halving their sample sizes.

Juxtaposing Table 1 and 2, it is possible to see that the differences can be related to the intraclass correlation coefficient, ρ . The differences are maximum in the case of the variable "unemployed"

which has the higher ρ , and is almost negligible in the case of the variable “inactive” for which ρ is close to 0.

Focusing on the different uses that the three packages make of ρ , the differences in the final allocation results arise. In **PracTools** and **samplesize4surveys**, ρ is only involved in the direct calculation of the optimal number of both PSUs and SSUs, once and for all. Furthermore, in **samplesize4surveys**, ρ is used only to calculate the d_{eff} , in turn, used to inflate the variance of the estimate but, also in this case, once and for all. While, in **R2BEAT**, ρ is involved in the partition of the PSUs in self-representative (SR) and non-self-representative (NSR).

This partition, updated every iteration, takes ρ into account each time. In fact, the iterative procedure - and of course the use of ρ in each iteration - makes the results more stable and explores, among all the suitable solutions, the more efficient in terms of PSUs and SSUs.

Summarising, the higher the value of the ρ , the higher the efficiency of the **R2BEAT** algorithm that makes extensive use of the ρ values.

Table 2: Intraclass correlation coefficient (ρ) for active, inactive, income, and unemployed in the whole population.

Variable	ρ
active	0.0656
inactive	0.0021
income_hh	0.0488
unemployed	0.1263.

6 How to manage the total non-response

Sample size and allocation in strata provided by **R2BEAT** ensure compliance with the precision constraints. Unfortunately, in practice, the planned sample size is affected by total non-response which reduces it and, consequently, weakens the precision and efficiency of the estimates. Therefore, it is important to consider the total non-response when planning sample surveys and to implement strategies to preserve the precision of the estimates.

The most common methods to carry out this task at the planning stage are *oversampling* and *substitutions*.

Oversampling is a technique used to increase the sample size of a population in a survey according to the total non-response rate in the population strata (nrr_h , with $0 \leq nrr_h \leq 1$ and $h = 1, \dots, H$) for recovering, at the end of the data collection phase, the desired sample size and the expected precision requirements. These rates are usually derived from previous survey occasions, carried out with the same data collection technique and under similar conditions, or on reliable assumptions made by those who design the survey. For embedding this procedure in **R2BEAT**, before performing the selection of the SSUs, it is sufficient to multiply the column **PSU_final_sample_unit** of the **data.frame sample_PSU** provided as the output of the function **select_PSU** by $1 + nrr_h$. If the response rates are not available at the strata level but at a higher domain level (for instance, for domains that are aggregations of strata), it is possible to assume a constant non-response rate for the strata belonging to the same domain. Of course, the more precise the knowledge of response rates, the more accurate the oversampling procedure adopted will be.

Instead, substitutions involve replacing non-responding units with other ones which have been “a priori” selected. Sometimes even more than one substitute for a single SSU is singled out. The selection of substitute units can be carried out in different ways (see, for a brief but complete review [Lynn, 2004](#)). If a simple random substitution of the SSUs, before performing the selection of the SSU, it is sufficient to multiply the column **PSU_final_sample_unit** of the **data.frame sample_PSU** provided as the output of the function **select_PSU** by the chosen number of substitutes. Otherwise, if a non-random substitution or a stratified substitution, unlike in the previous case, the selection of SSUs requires additional information on the SSUs and, therefore, the use of ad hoc solutions.

However, it is important to point out that substitutions and oversampling aim to preserve the sample size inflating it to withstand the non-response. However, it could be anyway not enough to avoid total non-response bias side effects on the final estimates. Therefore, non-response treatment methods must be applied after the data collection phase (see, e.g., [Särndal and Lundström, 2005](#); [Little and Rubin, 2019](#)).

7 Concluding remarks

Methodology, completeness, and efficiency can be considered the main strengths of the R package R2BEAT.

The methodology based on the application of the Bethel algorithm ensures that the resulting sample designs are compliant with the precision constraints set on the target estimates of a given survey.

The completeness can be considered both:

- in terms of applicability: R2BEAT applies to stratified, multipurpose and multidomain, one-stage and two-stage sampling surveys;
- in terms of coverage of the user needs: the package covers all the steps of a complex sample design, from the setting of precision constraints, the determination of the total sample size and of the allocation in the strata and the selection of the sampling units, distinctly according to the stage of selection.

Moreover, a set of functions generates the input required by the optimization step starting from the availability of a sampling frame and/or a previous round of the survey, thus allowing the users to fully harness all the available information with minimal effort. Other functions help them to perform analysis and checks on the obtained allocations and the selected sample.

Finally, we demonstrated the efficiency of the package by comparing it with two other competitor packages in terms of the results obtained in a case study: on equal errors, the sample size determined by R2BEAT is always lower, in terms of both Primary and Secondary Stage Units (PSUs and SSUs).

In future work, considering that a limitation of the package is its applicability to only mean parameters, the extension to more complex parameters will be investigated.

References

- S. Baillargeon and L.-P. Rivest. The construction of stratified designs in R with the package *stratification*. *Survey Methodology*, 37(1):53–65, 2011. [p201]
- G. Barcaroli. *SamplingStrata*: An R package for the optimization of stratified sampling. *Journal of Statistical Software*, 61(4):1–24, 2014. [p191]
- G. Barcaroli, T. Buglielli, and C. D. Vitiis. *MAUSS-R: Multivariate Allocation of Units in Sampling Surveys*, 2020. URL <https://www.istat.it/en/methods-and-tools/methods-and-it-tools/design-design-tools/mauss-r>. R package version 2.4. [p191]
- J. W. Bethel. Sample allocation in multivariate surveys. *Survey methodology*, 15(1):47–57, 1989. [p193]
- J. Breidaks, M. Liberts, and J. Jukams. *surveyplanning: Survey planning tools*. Riga, Latvia, 2020. URL <https://csblatvia.github.io/surveyplanning/>. R package version 4.0. [p191]
- E. Bueno. *optimStrat: Choosing the Sample Strategy*, 2020. URL <https://CRAN.R-project.org/package=optimStrat>. R package version 2.3. [p191]
- G. Cicchitelli, A. Herzl, and G. E. Montanari. *Il campionamento statistico*. Il mulino, Bologna, 1992. [p194]
- W. G. Cochran. *Sampling techniques*. John Wiley & Sons, 1977. [p191, 194]
- J.-C. Deville and C.-E. Särndal. Calibration estimators in survey sampling. *Journal of the American statistical Association*, 87(418):376–382, 1992. [p195]
- P. D. Falorsi, M. Ballin, C. De Vitiis, and G. Scepi. Principi e metodi del software generalizzato per la definizione del disegno di campionamento nelle indagini sulle imprese condotte dall’istat. *Statistica Applicata*, 10(2):235–257, 1998. [p195]
- M. H. Hansen, W. N. Hurwitz, and W. G. Madow. Sample survey methods and theory. Vol. I and II, Methods and applications. 1953. [p194]
- D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association*, 47(260):663–685, 1952. [p192]
- L. Kish. *Survey sampling*. John Wiley & Sons, Inc., New York, 1965. [p191]

- L. Kish. Optima and proxima in linear sample designs. *Journal of the Royal Statistical Society: Series A (General)*, 139(1):80–95, 1976. [p193]
- R. J. Little and D. B. Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2019. [p211]
- P. Lynn. The use of substitution in surveys. *The Survey Statistician*, 49:14–16, 2004. [p211]
- J. Neyman. On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection. *Journal of the Royal Statistical Society*, 97(4):558–625, 1934. [p192]
- H. A. G. Rojas. *Estrategias de muestreo: diseño de encuestas y estimación de parámetros*. Ediciones de la U, 2016. [p195]
- H. A. G. Rojas. *samplesize4surveys: Sample Size Calculations for Complex Surveys*, 2020. URL <https://CRAN.R-project.org/package=samplesize4surveys>. R package version 4.1.1. [p191, 208]
- C.-E. Särndal and S. Lundström. *Estimation in surveys with nonresponse*. John Wiley & Sons, 2005. [p211]
- C.-E. Särndal, B. Swensson, and J. Wretman. *Model assisted survey sampling*. Springer Science & Business Media, 2003. [p194]
- Y. Tillé and A. Matei. *sampling: Survey Sampling*, 2021. URL <https://CRAN.R-project.org/package=sampling>. R package version 2.9. [p199]
- A. Tschprow. On the two different aspects of the representative method: the method of stratified son the mathematical expectation of the moments of frequency distributions in the case of correlated observation sampling and the method of purposive selection. *Metron*, 2:646–683, 1923. [p192]
- R. Valliant, J. A. Dever, and F. Kreuter. *Practical Tools for Designing and Weighting Survey Samples*. Springer, 2015. [p208]
- R. Valliant, J. A. Dever, and F. Kreuter. *PracTools: Tools for Designing and Weighting Survey Samples*, 2020. URL <https://CRAN.R-project.org/package=PracTools>. R package version 1.2.2. [p191, 208]
- J. R. Waters and A. J. Chester. Optimal allocation in multivariate, two-stage sampling designs. *The American Statistician*, 41(1):46–50, 1987. [p195]
- D. Zardetto. ReGenesees: An advanced R system for calibration, estimation and sampling error assessment in complex sample surveys. *Journal of Official Statistics*, 31(2):177–203, 2015. [p198]

Giulio Barcaroli
Independent consultant
via Monte delle Gioie 29 - 00199 Roma
Italy
gbarcaroli@gmail.com

Andrea Fasulo
Italian National Institute of Statistics
via Cesare Balbo 16 - 00184 Roma
Italy
andrea.fasulo@istat.it

Alessio Guandalini
Italian National Institute of Statistics
via Cesare Balbo 16 - 00184 Roma
Italy
alessio.guandalini@istat.it

Marco D. Terribili
Italian National Institute of Statistics
via Cesare Balbo 16 - 00184 Roma
Italy
marco.terribili@istat.it

fnets: An R Package for Network Estimation and Forecasting via Factor-Adjusted VAR Modelling

by Dom Owens, Haeran Cho, and Matteo Barigozzi

Abstract Vector autoregressive (VAR) models are useful for modelling high-dimensional time series data. This paper introduces the package `fnets`, which implements the suite of methodologies proposed by (Barigozzi, Cho, and Owens 2023) for the network estimation and forecasting of high-dimensional time series under a factor-adjusted vector autoregressive model, which permits strong spatial and temporal correlations in the data. Additionally, we provide tools for visualising the networks underlying the time series data after adjusting for the presence of factors. The package also offers data-driven methods for selecting tuning parameters including the number of factors, the order of autoregression, and thresholds for estimating the edge sets of the networks of interest in time series analysis. We demonstrate various features of `fnets` on simulated datasets as well as real data on electricity prices.

1 Introduction

Vector autoregressive (VAR) models have been popularly adopted for modelling time series data across many disciplines including economics (Koop, 2013), finance (Barigozzi and Brownlees, 2019), neuroscience (Kirch et al., 2015), and systems biology (Shojaie and Michailidis, 2010). By fitting VAR models to data, we can infer dynamic interdependence between the variables and forecast future values. In particular, by inferring the non-zero elements of the VAR parameter matrices, we can find a network representation of the data which embeds Granger causal linkages. Besides, by estimating the precision matrix (inverse of the covariance matrix) of the VAR innovations, we can define a network representing their contemporaneous dependencies by means of partial correlations. Finally, the inverse of the long-run covariance matrix of the data simultaneously captures lead-lag and contemporaneous co-movements of the variables. For further discussions on the network interpretation of VAR modelling, we refer to Dahlhaus (2000), Eichler (2007), Billio et al. (2012) and Barigozzi and Brownlees (2019).

Fitting VAR models to the data can quickly become a high-dimensional problems since the number of parameters grows quadratically with the dimensionality of the data. There exists a mature literature on ℓ_1 -regularisation methods for estimating VAR models in high dimensions under suitable sparsity assumptions on the parameters (Basu and Michailidis, 2015; Han et al., 2015; Kock and Callot, 2015; Medeiros and Mendes, 2016; Nicholson et al., 2020; Liu and Zhang, 2021). Consistency of such methods is derived under the assumption that the spectral density matrix of the data has bounded eigenvalues. However, in many applications, the datasets exhibit strong serial and cross-sectional correlations which leads to the violation of this assumption. As a motivating example, we introduce a dataset of node-specific prices in the PJM (Pennsylvania, New Jersey and Maryland) power pool area in the United States, see [Energy price data](#) for further details. Figure 1 demonstrates that the leading eigenvalue of the long-run covariance matrix (i.e. spectral density matrix at frequency 0) increases linearly as the dimension of the data increases, which implies the presence of latent common factors in the panel data (Forni et al., 2000). Additionally, the left panel of Figure 2 shows the inadequacy of fitting a VAR model to such data under the sparsity assumption via ℓ_1 -regularisation methods, unless the presence of strong correlations is accounted for by a *factor-adjustment* step as in the right panel.

Barigozzi et al. (2023) propose the FNETS method for factor-adjusted VAR modelling of high-dimensional, second-order stationary time series. Under their proposed model, the data is decomposed into two latent components such that the *factor-driven* component accounts for pervasive leading, lagging or contemporaneous co-movements of the variables, while the remaining *idiosyncratic* dynamic dependence between the variables is modelled by a sparse VAR process. Then, FNETS provides tools for inferring the networks underlying the latent VAR process and forecasting.

In this paper, we present an R package named `fnets` which implements the FNETS method. It provides a range of user-friendly tools for estimating and visualising the networks representing the interconnectedness of time series variables, and for producing forecasts. In addition, `fnets` includes a range of methods for selecting tuning parameters ranging from the number of factors and the VAR order, to regularisation and thresholding parameters adopted for producing sparse and interpretable networks. The main routine of `fnets` outputs an object of S3 class `fnets` which is supported by a `plot` method for network visualisation and a `predict` method for time series forecasting.

There exist several packages for fitting VAR models and their extensions to high-dimensional time series, see `LSVAR` (Bai, 2021), `sparsevar` (Vazzoler, 2021), `nets` (Brownlees, 2020), `mgm` (Haslbeck

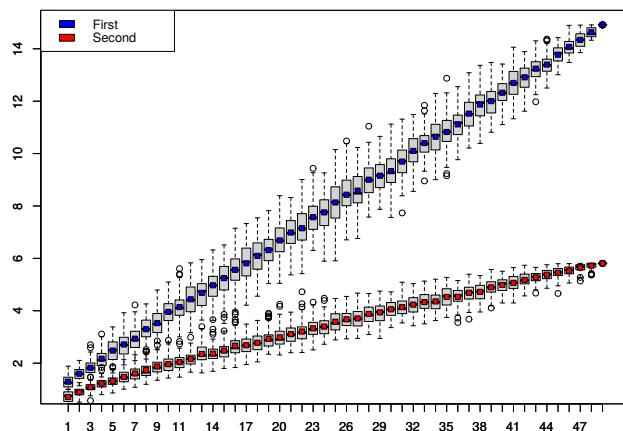


Figure 1: Box plots of the two largest eigenvalues (y -axis) of the long-run covariance matrix estimated from the energy price data collected between 01/01/2021 and 19/07/2021 ($n = 200$), see [Real data example](#) for further details. Cross-sections of the data are randomly sampled 100 times for each given dimension $p \in \{2, \dots, 50\}$ (x -axis) to produce the box plots.

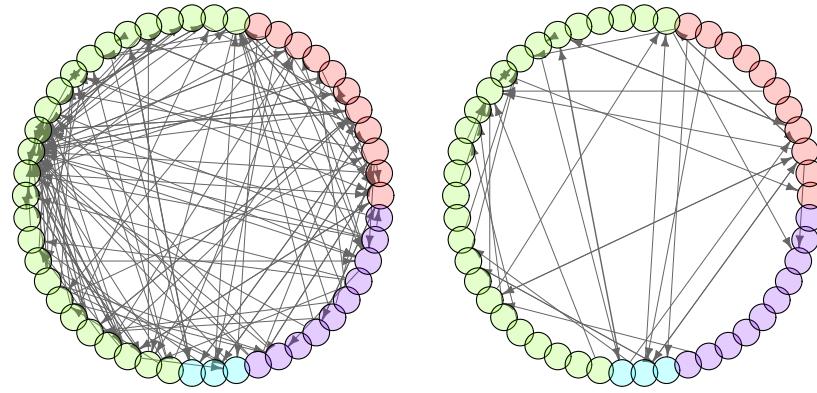


Figure 2: Granger causal networks defined in (5) obtained from fitting a VAR(1) model to the energy price data analysed in Figure 1, without (left) and with (right) the factor adjustment step outlined in [FNETS: Network estimation](#). Edge weights (proportional to the size of coefficient estimates) are visualised by the width of each edge, and the nodes are coloured according to their groupings, see [Real data example](#) for further details.

and Waldorp, 2020), **graphicalVAR** (Epskamp et al., 2018), **BigVAR** (Nicholson et al., 2017), and **bigtime** (Wilms et al., 2021). There also exist R packages for time series factor modelling such as **dfms** (Krantz and Bagdziunas, 2023) and **sparseDFM** (Mosley et al., 2023), and **FAVAR** (Bernanke et al., 2005) for Bayesian inference of factor-augmented VAR models. The advantage of **fnets** over the above-mentioned packages is its ability to handle strong cross-sectional and serial correlations in the data via factor-adjustment step performed in the frequency domain. In addition, the FNETS method operates under the most general approach to high-dimensional time series factor modelling termed the Generalised Dynamic Factor (GDFM), first proposed in Forni et al. (2000) and further investigated in Forni et al. (2015). Accordingly, **fnets** is the first R package to provide tools for high-dimensional panel data analysis under the GDFM, such as fast computation of spectral density and autocovariance matrices via the Fast Fourier Transform, but it is flexible enough to allow for more restrictive static factor models. While there exist some packages for network-based time series modelling (e.g. **GNAR**, Knight et al., 2020), we highlight that the goal of **fnets** is to learn the networks underlying a time series and does not require a network as an input.

2 FNETS methodology

In this section, we introduce the factor-adjusted VAR model and describe the FNETS methodology proposed in Barigozzi et al. (2023) for network estimation and forecasting of high-dimensional time

series. We limit ourselves to describing the key steps of FNets and refer to the above paper for its comprehensive treatment.

2.1 Factor-adjusted VAR model

A zero-mean, p -variate process ξ_t follows a VAR(d) model if it satisfies

$$\xi_t = \sum_{\ell=1}^d \mathbf{A}_\ell \xi_{t-\ell} + \boldsymbol{\Gamma}^{1/2} \boldsymbol{\varepsilon}_t, \quad (1)$$

where $\mathbf{A}_\ell \in R^{p \times p}$, $1 \leq \ell \leq d$, determine how future values of the series depend on the past values. For the p -variate random vector $\boldsymbol{\varepsilon}_t = (\varepsilon_{1t}, \dots, \varepsilon_{pt})^\top$, we assume that ε_{it} are independently and identically distributed (i.i.d.) for all i and t with $E(\varepsilon_{it}) = 0$ and $\text{Var}(\varepsilon_{it}) = 1$. Then, the positive definite matrix $\boldsymbol{\Gamma} \in R^{p \times p}$ is the covariance matrix of the innovations $\boldsymbol{\Gamma}^{1/2} \boldsymbol{\varepsilon}_t$.

In the literature on factor modelling of high-dimensional time series, the factor-driven component exhibits strong cross-sectional and/or serial correlations by ‘loading’ finite-dimensional vectors of factors linearly. Among many time series factor models, the GDFM (Forni et al., 2000) provides the most general approach where the p -variate factor-driven component χ_t admits the following representation

$$\chi_t = \mathcal{B}(L)\mathbf{u}_t = \sum_{\ell=0}^{\infty} \mathbf{B}_\ell \mathbf{u}_{t-\ell} \text{ with } \mathbf{u}_t = (u_{1t}, \dots, u_{qt})^\top \text{ and } \mathbf{B}_\ell \in R^{p \times q}, \quad (2)$$

for some fixed q , where L stands for the lag operator. The q -variate random vector \mathbf{u}_t contains the common factors which are loaded across the variables and time by the filter $\mathcal{B}(L) = \sum_{\ell=0}^{\infty} \mathbf{B}_\ell L^\ell$, and it is assumed that u_{jt} are i.i.d. with $E(u_{jt}) = 0$ and $\text{Var}(u_{jt}) = 1$. The model (2) reduces to a static factor model (Bai, 2003; Stock and Watson, 2002; Fan et al., 2013), when $\mathcal{B}(L) = \sum_{\ell=0}^s \mathbf{B}_\ell L^\ell$ for some finite integer $s \geq 0$. Then, we can write

$$\chi_t = \boldsymbol{\Lambda} \mathbf{F}_t \text{ where } \mathbf{F}_t = (\mathbf{u}_t^\top, \dots, \mathbf{u}_{t-s}^\top)^\top \text{ and } \boldsymbol{\Lambda} = [\mathbf{B}_0, \dots, \mathbf{B}_s] \quad (3)$$

with $r = q(s+1)$ as the dimension of static factors \mathbf{F}_t . Throughout, we refer to the models (2) and (3) as *unrestricted* and *restricted* to highlight that the latter imposes more restrictions on the model.

Barigozzi et al. (2023) propose a factor-adjusted VAR model under which we observe a zero-mean, second-order stationary process $\mathbf{X}_t = (X_{1t}, \dots, X_{pt})^\top$ for $t = 1, \dots, n$, that permits a decomposition into the sum of the unobserved components ξ_t and χ_t , i.e.

$$\mathbf{X}_t = \xi_t + \chi_t. \quad (4)$$

We assume that $E(\varepsilon_{it} u_{jt'}) = 0$ for all i, j, t and t' as is commonly assumed in the literature, such that $E(\xi_{it} \chi_{i't'}) = 0$ for all $1 \leq i, i' \leq p$ and $t, t' \in \mathbb{Z}$.

2.2 Networks

Under (4), it is of interest to infer three types of networks representing the interconnectedness of \mathbf{X}_t after factor adjustment. Let $\mathcal{V} = \{1, \dots, p\}$ denote the set of vertices representing the p cross-sections. Then, the VAR parameter matrices, $\mathbf{A}_\ell = [A_{\ell,ij'}]$, $1 \leq i, i' \leq p$, encode the directed network $\mathcal{N}^G = (\mathcal{V}, \mathcal{E}^G)$ representing Granger causal linkages, where the set of edges are given by

$$\mathcal{E}^G = \{(i, i') \in \mathcal{V} \times \mathcal{V} : A_{\ell,ii'} \neq 0 \text{ for some } 1 \leq \ell \leq d\}. \quad (5)$$

Here, the presence of an edge $(i, i') \in \mathcal{E}^G$ indicates that $\xi_{i',t-\ell}$ Granger causes ξ_{it} at some lag $1 \leq \ell \leq d$ (Dahlhaus, 2000).

The second network contains undirected edges representing contemporaneous cross-sectional dependence in VAR innovations $\boldsymbol{\Gamma}^{1/2} \boldsymbol{\varepsilon}_t$, denoted by $\mathcal{N}^C = (\mathcal{V}, \mathcal{E}^C)$. We have $(i, i') \in \mathcal{E}^C$ if and only if the partial correlation between the i -th and i' -th elements of $\boldsymbol{\Gamma}^{1/2} \boldsymbol{\varepsilon}_t$ is non-zero, which in turn is given by $-\delta_{ii'} / \sqrt{\delta_{ii} \cdot \delta_{i'i'}}$ where $\boldsymbol{\Gamma}^{-1} = \boldsymbol{\Delta} = [\delta_{ii'}]$, $1 \leq i, i' \leq p$ (Peng et al., 2009). Hence, the set of edges for \mathcal{N}^C is given by

$$\mathcal{E}^C = \left\{ (i, i') \in \mathcal{V} \times \mathcal{V} : i \neq i' \text{ and } -\frac{\delta_{ii'}}{\sqrt{\delta_{ii} \cdot \delta_{i'i'}}} \neq 0 \right\}, \quad (6)$$

Finally, we can summarise the aforementioned lead-lag and contemporaneous relations between

the variables in a single, undirected network $\mathcal{N}^L = (\mathcal{V}, \mathcal{E}^L)$ by means of the long-run partial correlations of ξ_t . Let $\Omega = [\omega_{ii'}, 1 \leq i, i' \leq p]$ denote the inverse of the zero-frequency spectral density (a.k.a. long-run covariance) of ξ_t , which is given by $\Omega = 2\pi\mathcal{A}^\top(1)\Delta\mathcal{A}(1)$ with $\mathcal{A}(z) = \mathbf{I} - \sum_{\ell=1}^d \mathbf{A}_\ell z^\ell$. Then, the long-run partial correlation between the i -th and i' -th elements of ξ_t , is obtained as $-\omega_{ii'} / \sqrt{\omega_{ii} \cdot \omega_{i'i'}}$ (Dahlhaus, 2000), so the edge set of \mathcal{N}^L is given by

$$\mathcal{E}^L = \left\{ (i, i') \in \mathcal{V} \times \mathcal{V} : i \neq i' \text{ and } -\frac{\omega_{ii'}}{\sqrt{\omega_{ii} \cdot \omega_{i'i'}}} \neq 0 \right\}. \quad (7)$$

2.3 FNETS: Network estimation

We describe the three-step methodology for estimating the networks \mathcal{N}^G , \mathcal{N}^C and \mathcal{N}^L . Throughout, we assume that the number of factors, either q under the more general model in (2) or r under the restricted model in (3), and the VAR order d , are known, and discuss its selection in [Tuning parameter selection](#).

Step 1: Factor adjustment

The autocovariance (ACV) matrices of ξ_t , denoted by $\Gamma_\xi(\ell) = \mathbb{E}(\xi_{t-\ell}\xi_t^\top)$ for $\ell \geq 0$ and $\Gamma_\xi(\ell) = (\Gamma_\xi(-\ell))^\top$ for $\ell < 0$, play a key role in network estimation. Since ξ_t is not directly observed, we propose to adjust for the presence of the factor-driven χ_t and estimate $\Gamma_\xi(\ell)$. For this, we adopt a frequency domain-based approach and perform the dynamic principal component analysis (PCA). Spectral density matrix $\Sigma_x(\omega)$ of a time series $\{\mathbf{X}_t\}_{t \in \mathbb{Z}}$ aggregates information of its ACV $\Gamma_x(\ell)$, $\ell \in \mathbb{Z}$, at a specific frequency $\omega \in [-\pi, \pi]$, and is obtained by the Fourier transform $\Sigma_x(\omega) = (2\pi)^{-1} \sum_{\ell=-\infty}^{\infty} \Gamma_x(\ell) \exp(-i\ell\omega)$ where $i = \sqrt{-1}$. Denoting the sample ACV matrix of \mathbf{X}_t at lag ℓ by

$$\widehat{\Gamma}_x(\ell) = \frac{1}{n} \sum_{t=\ell+1}^n \mathbf{X}_{t-\ell} \mathbf{X}_t^\top \text{ when } \ell \geq 0 \quad \text{and} \quad \widehat{\Gamma}_x(\ell) = (\widehat{\Gamma}_x(-\ell))^\top \text{ when } \ell < 0,$$

we estimate the spectral density of \mathbf{X}_t by

$$\widehat{\Sigma}_x(\omega_k) = \frac{1}{2\pi} \sum_{\ell=-m}^m K\left(\frac{\ell}{m}\right) \widehat{\Gamma}_x(\ell) \exp(-i\ell\omega_k), \quad (8)$$

where $K(\cdot)$ denotes a kernel, m the kernel bandwidth (for its choice, see [Tuning parameter selection](#)) and $\omega_k = 2\pi k / (2m + 1)$ the Fourier frequencies. We adopt the Bartlett kernel as $K(\cdot)$, which ensures positive semi-definiteness of $\widehat{\Sigma}_x(\omega)$ and also $\widehat{\Gamma}_x(\ell)$ estimating $\Gamma_\xi(\ell)$ obtained as described below.

Performing PCA on $\widehat{\Sigma}_x(\omega_k)$ at each ω_k , we obtain the estimator of the spectral density matrix of χ_t as $\widehat{\Sigma}_\chi(\omega_k) = \sum_{j=1}^q \widehat{\mu}_{x,j}(\omega_k) \widehat{\mathbf{e}}_{x,j}(\omega_k) (\widehat{\mathbf{e}}_{x,j}(\omega_k))^*$, where $\widehat{\mu}_{x,j}(\omega_k)$ denotes the j -th largest eigenvalue of $\widehat{\Sigma}_x(\omega_k)$, $\widehat{\mathbf{e}}_{x,j}(\omega_k)$ its associated eigenvector, and for any vector $\mathbf{a} \in \mathbb{C}^n$, we denote its transposed complex conjugate by \mathbf{a}^* . Then taking the inverse Fourier transform of $\widehat{\Sigma}_\chi(\omega_k)$, $-m \leq k \leq m$, leads to an estimator of $\Gamma_\chi(\ell)$, the ACV matrix of χ_t , as

$$\widehat{\Gamma}_\chi(\ell) = \frac{2\pi}{2m+1} \sum_{k=-m}^m \widehat{\Sigma}_\chi(\omega_k) \exp(i\ell\omega_k) \quad \text{for } -m \leq \ell \leq m.$$

Finally, we estimate the ACV of ξ_t by

$$\widehat{\Gamma}_\xi(\ell) = \widehat{\Gamma}_x(\ell) - \widehat{\Gamma}_\chi(\ell). \quad (9)$$

When we assume the restricted factor model in (3), the factor-adjustment step is simplified as it suffices to perform PCA in the time domain, i.e. eigenanalysis of the sample covariance matrix $\widehat{\Gamma}_x(0)$. Denoting the eigenvector of $\widehat{\Gamma}_x(0)$ associated with its j -th largest eigenvalue by $\widehat{\mathbf{e}}_{x,j}$, we obtain $\widehat{\Gamma}_\xi(\ell) = \widehat{\Gamma}_x(\ell) - \widehat{\mathbf{E}}_x \widehat{\mathbf{E}}_x^\top \widehat{\Gamma}_x(\ell) \widehat{\mathbf{E}}_x \widehat{\mathbf{E}}_x^\top$ where $\widehat{\mathbf{E}}_x = [\widehat{\mathbf{e}}_{x,j}, 1 \leq j \leq r]$.

Step 2: Estimation of \mathcal{N}^G

Recall from (5) that \mathcal{N}^G , representing Granger causal linkages, has its edge set determined by the VAR transition matrices \mathbf{A}_ℓ , $1 \leq \ell \leq d$. By the Yule-Walker equation, we have $\boldsymbol{\beta} = [\mathbf{A}_1, \dots, \mathbf{A}_d]^\top =$

$\mathbf{G}(d)^{-1}\mathbf{g}(d)$, where

$$\mathbf{G}(d) = \begin{bmatrix} \Gamma_\xi(0) & \Gamma_\xi(-1) & \dots & \Gamma_\xi(-d+1) \\ \Gamma_\xi(1) & \Gamma_\xi(0) & \dots & \Gamma_\xi(-d+2) \\ & & \ddots & \\ \Gamma_\xi(d-1) & \Gamma_\xi(d-2) & \dots & \Gamma_\xi(0) \end{bmatrix} \quad \text{and} \quad \mathbf{g}(d) = \begin{bmatrix} \Gamma_\xi(1) \\ \Gamma_\xi(2) \\ \vdots \\ \Gamma_\xi(d) \end{bmatrix}. \quad (10)$$

We propose to estimate β as a regularised Yule-Walker estimator based on $\widehat{\mathbf{G}}(d)$ and $\widehat{\mathbf{g}}(d)$, each of which is obtained by replacing $\Gamma_\xi(\ell)$ with $\widehat{\Gamma}_\xi(\ell)$, see (9), in the definition of $\mathbf{G}(d)$ and $\mathbf{g}(d)$.

For any matrix $\mathbf{M} = [m_{ij}] \in R^{n_1 \times n_2}$, let $|\mathbf{M}|_1 = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} |m_{ij}|$, $|\mathbf{M}|_\infty = \max_{1 \leq i \leq n_1} \max_{1 \leq j \leq n_2} |m_{ij}|$ and $\text{tr}(\mathbf{M}) = \sum_{i=1}^{n_1} m_{ii}$ when $n_1 = n_2$. We consider two estimators of β . Firstly, we adopt a Lasso-type estimator which solves an ℓ_1 -regularised M -estimation problem

$$\widehat{\beta}^{\text{las}} = \arg \min_{\mathbf{M} \in \mathbb{R}^{pd \times p}} \text{tr} \left(\mathbf{M}^\top \widehat{\mathbf{G}}(d) \mathbf{M} - 2\mathbf{M}^\top \widehat{\mathbf{g}}(d) \right) + \lambda |\mathbf{M}|_1 \quad (11)$$

with a tuning parameter $\lambda > 0$. In the implementation, we solve (11) via the fast iterative shrinkage-thresholding algorithm (FISTA, Beck and Teboulle, 2009). Alternatively, we adopt a constrained ℓ_1 -minimisation approach closely related to the Dantzig selector (DS, Candes and Tao, 2007):

$$\widehat{\beta}^{\text{DS}} = \arg \min_{\mathbf{M} \in \mathbb{R}^{pd \times p}} |\mathbf{M}|_1 \quad \text{subject to} \quad \left| \widehat{\mathbf{G}}(d) \mathbf{M} - \widehat{\mathbf{g}}(d) \right|_\infty \leq \lambda \quad (12)$$

for some tuning parameter $\lambda > 0$. We divide (12) into p sub-problems and obtain each column of $\widehat{\beta}^{\text{DS}}$ via the simplex algorithm (using the function `lp` in `IpSolve` (Berkelaar et al., 2020)), which is performed in parallel with `doParallel` and `foreach` (Microsoft and Weston, 2022a,b).

Barigozzi et al. (2023) establish the consistency of both $\widehat{\beta}^{\text{las}}$ and $\widehat{\beta}^{\text{DS}}$ but, as is typically the case for ℓ_1 -regularisation methods, they do not achieve exact recovery of the support of β . Hence we propose to estimate the edge set of \mathcal{N}^G by thresholding the elements of $\widehat{\beta}$ with some threshold $t > 0$, where either $\widehat{\beta} = \widehat{\beta}^{\text{las}}$ or $\widehat{\beta} = \widehat{\beta}^{\text{DS}}$, i.e.

$$\widetilde{\beta}(t) = \left[\widehat{\beta}_{ij} \cdot \mathbb{I}_{\{|\widehat{\beta}_{ij}| > t\}}, 1 \leq i \leq pd, 1 \leq j \leq p \right]. \quad (13)$$

We discuss cross validation and information criterion methods for selecting λ , and a data-driven choice of t , in Tuning parameter selection.

Step 3: Estimation of \mathcal{N}^C and \mathcal{N}^L

From the definitions of \mathcal{N}^C and \mathcal{N}^L given in (6) and (7), their edge sets are obtained by estimating $\Delta = \Gamma^{-1}$ and $\Omega = 2\pi\mathcal{A}^\top(1)\Delta\mathcal{A}(1)$. Suppose that we are given $\widehat{\beta} = [\widehat{\mathbf{A}}_1, \dots, \widehat{\mathbf{A}}_d]^\top$, some estimator of the VAR parameter matrices obtained as in either (11) or (12). Then, a natural estimator of Γ arises from the Yule-Walker equation $\Gamma = \Gamma_\xi(0) - \sum_{\ell=1}^d \mathbf{A}_\ell \Gamma_\xi(\ell) = \Gamma_\xi(0) - \beta^\top \mathbf{g}$, as $\widehat{\Gamma} = \widehat{\Gamma}_\xi(0) - \widehat{\beta}^\top \widehat{\mathbf{g}}$. In high dimensions, it is not feasible or recommended to directly invert $\widehat{\Gamma}$ to estimate Δ . Therefore, we adopt a constrained ℓ_1 -minimisation method motivated by the CLIME methodology of Cai et al. (2011).

Specifically, the CLIME estimator of Δ is obtained by first solving

$$\check{\Delta} = \arg \min_{\mathbf{M} \in \mathbb{R}^{p \times p}} |\mathbf{M}|_1 \quad \text{subject to} \quad \left| \widehat{\Gamma} \mathbf{M} - \mathbf{I} \right|_\infty \leq \eta, \quad (14)$$

and applying a symmetrisation step to $\check{\Delta} = [\check{\delta}_{ii'}, 1 \leq i, j \leq p]$ as

$$\widehat{\Delta} = [\widehat{\delta}_{ii'}, 1 \leq i, i' \leq p] \text{ with } \widehat{\delta}_{ii'} = \check{\delta}_{ii'} \cdot \mathbb{I}_{\{|\check{\delta}_{ii'}| \leq |\check{\delta}_{i'i}|\}} + \check{\delta}_{i'i} \cdot \mathbb{I}_{\{|\check{\delta}_{i'i}| < |\check{\delta}_{ii'}|\}}. \quad (15)$$

for some tuning parameter $\eta > 0$. Cai et al. (2016) propose ACLIME, which improves the CLIME estimator by selecting the parameter η in (15) adaptively. It first produces the estimators of the diagonal entries δ_{ii} , $1 \leq i \leq p$, as in (15) with $\eta_1 = 2\sqrt{\log(p)/n}$ as the tuning parameter. Then these estimates are used for adaptive tuning parameter selection in the second step. We provide the full description of the ACLIME estimator along with the details of its implementation in `ACLIME estimator` of the Appendix.

Given the estimators $\widehat{\mathcal{A}}(1) = \mathbf{I} - \sum_{\ell=1}^d \widehat{\mathbf{A}}_\ell$ and $\widehat{\Delta}$, we estimate Ω by $\widehat{\Omega} = 2\pi\widehat{\mathcal{A}}^\top(1)\widehat{\Delta}\widehat{\mathcal{A}}(1)$. In Barigozzi et al. (2023), $\widehat{\Delta}$ and $\widehat{\Omega}$ are shown to be consistent in ℓ_∞ - and ℓ_1 -norms under suitable sparsity assumptions. However, an additional thresholding step as in (13) is required to guarantee consistency

in estimating the support of Δ and Ω and consequently the edge sets of \mathcal{N}^C and \mathcal{N}^L . We discuss data-driven selection of these thresholds and η in [Tuning parameter selection](#).

2.4 FNETS: Forecasting

Following the estimation procedure, FNETS performs forecasting by estimating the best linear predictor of \mathbf{X}_{n+a} given \mathbf{X}_t , $t \leq n$, for a fixed integer $a \geq 1$. This is achieved by separately producing the best linear predictors of χ_{n+a} and ξ_{n+a} as described below, and then combining them.

Forecasting the factor-driven component

For given $a \geq 0$, the best linear predictor of χ_{n+a} given \mathbf{X}_t , $t \leq n$, under (2) is

$$\chi_{n+a|n} = \sum_{\ell=0}^{\infty} \mathbf{B}_{\ell+a} \mathbf{u}_{n-\ell}.$$

[Forni et al. \(2015\)](#) show that the model (2) admits a low-rank VAR representation with \mathbf{u}_t as the innovations under mild conditions, and [Forni et al. \(2017\)](#) propose the estimators of \mathbf{B}_ℓ and \mathbf{u}_t based on this representation which make use of the estimators of the ACV of χ_t obtained as described in Step 1. Then, a natural estimator of $\chi_{n+a|n}$ is

$$\hat{\chi}_{n+a|n}^{\text{unr}} = \sum_{\ell=0}^K \hat{\mathbf{B}}_{\ell+a} \hat{\mathbf{u}}_{n-\ell} \quad (16)$$

for some truncation lag K . We refer to $\hat{\chi}_{n+a|n}^{\text{unr}}$ as the *unrestricted* estimator of $\chi_{n+a|n}$ as it is obtained without imposing any restrictions on the factor model (2).

When χ_t admits the static representation in (3), we can show that $\chi_{n+a|n} = \mathbf{\Gamma}_\chi(-a) \mathbf{E}_\chi \mathbf{M}_\chi^{-1} \mathbf{E}_\chi^\top \chi_n$, where $\mathbf{M}_\chi \in R^{r \times r}$ is a diagonal matrix with the r eigenvalues of $\mathbf{\Gamma}_\chi(0)$ on its diagonal and $\mathbf{E}_\chi \in R^{p \times r}$ the matrix of the corresponding eigenvectors; see Section 4.1 of [Barigozzi et al. \(2023\)](#) and also [Forni et al. \(2005\)](#). This suggests an estimator

$$\hat{\chi}_{n+a|n}^{\text{res}} = \hat{\mathbf{\Gamma}}_\chi(-a) \hat{\mathbf{E}}_\chi \hat{\mathbf{M}}_\chi^{-1} \hat{\mathbf{E}}_\chi^\top \mathbf{X}_n, \quad (17)$$

where $\hat{\mathbf{M}}_\chi$ and $\hat{\mathbf{E}}_\chi$ are obtained from the eigendecomposition of $\hat{\mathbf{\Gamma}}_\chi(0)$. We refer to $\hat{\chi}_{n+a|n}^{\text{res}}$ as the *restricted* estimator of $\chi_{n+a|n}$. As a by-product, we obtain the in-sample estimators of χ_t , $t \leq n$, as $\hat{\chi}_{t|n} = \hat{\chi}_t$, with either of the two estimators in (16) and (17).

Forecasting the latent VAR process

Once the VAR parameters are estimated either as in (11) or (12), we produce an estimator of $\xi_{n+a|n} = \sum_{\ell=1}^d \mathbf{A}_\ell \xi_{n+a-\ell}$, the best linear predictor of ξ_{n+a} given \mathbf{X}_t , $t \leq n$, as

$$\hat{\xi}_{n+a|n} = \sum_{\ell=1}^{\max(1,a)-1} \hat{\mathbf{A}}_\ell \hat{\xi}_{n+a-\ell|n} + \sum_{\ell=\max(1,a)}^d \hat{\mathbf{A}}_\ell \hat{\xi}_{n+a-\ell}. \quad (18)$$

Here, $\hat{\xi}_{n+1-\ell} = \mathbf{X}_{n+1-\ell} - \hat{\chi}_{n+1-\ell}$ denotes the in-sample estimator of $\xi_{n+1-\ell}$, which may be obtained with either of the two (in-sample) estimators of the factor-driven component in (16) and (17).

3 Tuning parameter selection

3.1 Factor numbers q and r

The estimation and forecasting tools of the FNETS methodology require the selection of the number of factors, i.e. q under the unrestricted factor model in (2), and r under the restricted, static factor model in (3). Under (2), there exists a large gap between the q leading eigenvalues of the spectral density matrix of \mathbf{X}_t and the remainder which diverges with p (see also Figure 1). We provide two methods for selecting the factor number q , which make use of the postulated eigengap using $\hat{\mu}_{x,j}(\omega_k)$, $1 \leq j \leq p$, the eigenvalues of the spectral density estimator of \mathbf{X}_t in (8) at a given Fourier frequency ω_k , $-m \leq k \leq m$.

Hallin and Liška (2007) propose an information criterion for selecting the number of factors under the model (2) and further, a methodology for tuning the multiplicative constant in the penalty. Define

$$\text{IC}(b, c) = \log \left(\frac{1}{p} \sum_{j=b+1}^p \frac{1}{2m+1} \sum_{k=-m}^m \widehat{\mu}_{x,j}(\omega_k) \right) + b \cdot c \cdot \text{pen}(n, p), \quad (19)$$

where $\text{pen}(n, p) = \min(p, m^2, \sqrt{n/m})^{-1/2}$ by default (for other choices of the information criterion, see Appendix A), and $c > 0$ a constant. Provided that $\text{pen}(n, p) \rightarrow 0$ sufficiently slowly, for an arbitrary value of c , the factor number q is consistently estimated by the minimiser of $\text{IC}(b, c)$ over $b \in \{0, \dots, \bar{q}\}$, with some fixed \bar{q} as the maximum allowable number of factors. However, this is not the case in finite sample, and Hallin and Liška (2007) propose to simultaneously select q and c . First, we identify $\widehat{q}(n_l, p_l, c) = \arg \min_{0 \leq b \leq \bar{q}} \text{IC}(n_l, p_l, b, c)$ where $\text{IC}(n_l, p_l, b, c)$ is constructed analogously to $\text{IC}(b, c)$, except that it only involves the sub-sample $\{X_{it}, 1 \leq i \leq p_l, 1 \leq t \leq n_l\}$, for sequences $0 < n_1 < \dots < n_L = n$ and $0 < p_1 < \dots < p_L = p$. Then, denoting the sample variance of $\widehat{q}(n_l, p_l, c)$, $1 \leq l \leq L$, by $S(c)$, we select $\widehat{q} = \widehat{q}(n, p, \widehat{c})$ with \widehat{c} corresponding to the second interval of stability with $S(c) = 0$ for the mapping $c \mapsto S(c)$ as c increases from 0 to some c_{\max} (the first stable interval is where \widehat{q} is selected with a very small value of c). Figure 3 plots $\widehat{q}(n, p, c)$ and $S(c)$ for varying values of c obtained from a dataset simulated in Data simulation. In the implementation of this methodology, we set $n_l = n - (L-l)\lfloor n/20 \rfloor$ and $p_l = \lfloor 3p/4 + lp/40 \rfloor$ with $L = 10$, and $\bar{q} = \min(50, \lfloor \sqrt{\min(n-1, p)} \rfloor)$.

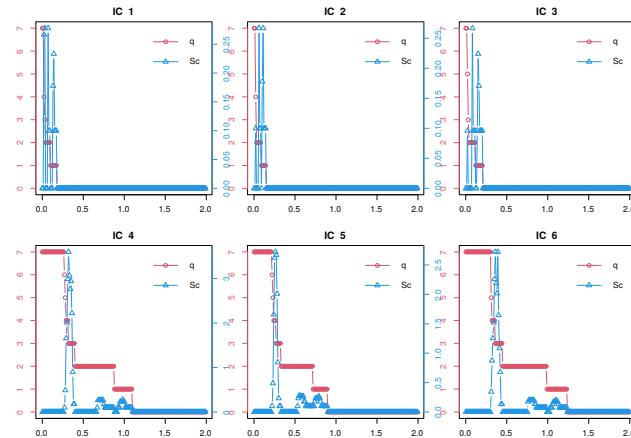


Figure 3: Plots of c against $\widehat{q}(n, p, c)$ (in circles, y -axis on the left) and $S(c)$ (in triangles, y -axis on the right) with the six IC (see Appendix A) implemented in the function `factor.number` of `fnets`, on a dataset simulated as described in Data simulation (with $n = 500$, $p = 50$ and $q = 2$). With the default choice of IC in (19) (IC₅), we obtain $\widehat{q} = \widehat{q}(n, p, \widehat{c}) = 2$ correctly estimating $q = 2$.

Alternatively, we can adopt the ratio-based estimator $\widehat{q} = \arg \min_{1 \leq b \leq \bar{q}} \text{ER}(b)$ proposed in Avarucci et al. (2022), where

$$\text{ER}(b) = \left(\sum_{k=-m}^m \widehat{\mu}_{x,b+1}(\omega_k) \right)^{-1} \left(\sum_{k=-m}^m \widehat{\mu}_{x,b}(\omega_k) \right). \quad (20)$$

These methods are readily modified to select the number of factors r under the restricted factor model in (3), by replacing $(2m+1)^{-1} \sum_{k=-m}^m \widehat{\mu}_{x,j}(\omega_k)$ with $\widehat{\mu}_{x,m}$, the j -th largest eigenvalues of the sample covariance matrix $\widehat{\Gamma}_x(0)$. We refer to Bai and Ng (2002) and Alessi et al. (2010) for the discussion of the information criterion-based method in this setting, and Ahn and Horenstein (2013) for that of the eigenvalue ratio-based method.

3.2 Threshold t

Motivated by Liu et al. (2021), we propose a method for data-driven selection of the threshold t , which is applied to the estimators of \mathbf{A}_ℓ , $1 \leq \ell \leq d$, Δ or Ω for estimating the edge sets of \mathcal{N}^G , \mathcal{N}^C or \mathcal{N}^L , respectively, see also (13).

Let $\mathbf{B} = [b_{ij}] \in R^{m \times n}$ denote a matrix for which a threshold is to be selected, i.e. \mathbf{B} may be either $\widehat{\beta} = [\widehat{\mathbf{A}}_1, \dots, \widehat{\mathbf{A}}_d]^\top$, $\widehat{\Delta}_0$ ($\widehat{\Delta}$ with diagonals set to zero), or $\widehat{\Omega}_0$ ($\widehat{\Omega}$ with diagonals set to zero), obtained

from Steps 2 and 3 of FNets. We work with $\widehat{\Delta}_0$ and $\widehat{\Omega}_0$ since we do not threshold the diagonal entries of $\widehat{\Delta}$ and $\widehat{\Omega}$. As such estimators have been shown to achieve consistency in ℓ_∞ -norm, we expect there exists a large gap between the entries of \mathbf{B} corresponding to true positives and false positives. Further, it is expected that the number of edges reduces at a faster rate when increasing the threshold from 0 towards this (unknown) gap, compared to when increasing the threshold from the gap to $|\mathbf{B}|_\infty$. Therefore, we propose to identify this gap by casting the problem as that of locating a single change point in the trend of the ratio of edges to non-edges,

$$\text{Ratio}_k = \frac{|\mathbf{B}(t_k)|_0}{\max(N - |\mathbf{B}(t_k)|_0, 1)}, \quad k = 1, \dots, M.$$

Here, $\mathbf{B}(t) = [b_{ij} \cdot \mathbb{I}_{\{|b_{ij}| > t\}}]$, $|\mathbf{B}(t)|_0 = \sum_{i=1}^{m_1} \sum_{j=1}^{m_2} \mathbb{I}_{\{|b_{ij}| > t\}}$ and $\{t_k, 1 \leq k \leq M : 0 = t_1 < t_2 < \dots < t_M = |\mathbf{B}|_\infty\}$ denotes a sequence of candidate threshold values. We recommend using an exponentially growing sequence for $\{t_k\}_{k=1}^M$ since the size of the false positive entries tends to be very small. The quantity N in the denominator of Ratio_k is set as $N = p^2d$ when $\mathbf{B} = \widehat{\beta}$, and $N = p(p-1)$ when $\mathbf{B} = \widehat{\Delta}_0$ or $\mathbf{B} = \widehat{\Omega}_0$. Then, from the difference quotient

$$\text{Diff}_k = \frac{\text{Ratio}_k - \text{Ratio}_{k-1}}{t_k - t_{k-1}}, \quad k = 2, \dots, M,$$

we compute the cumulative sum (CUSUM) statistic

$$\text{CUSUM}_k = \sqrt{\frac{k(M-k)}{M}} \left| \frac{1}{k} \sum_{l=2}^k \text{Diff}_l - \frac{1}{M-k} \sum_{l=k+1}^M \text{Diff}_l \right|, \quad k = 2, \dots, M-1,$$

and select $t_{\text{ada}} = t_{k^*}$ with $k^* = \arg \max_{2 \leq k \leq M-1} \text{CUSUM}_k$. For illustration, Figure 4 plots Ratio_k and CUSUM_k against candidate thresholds for the dataset simulated in [Data simulation](#).

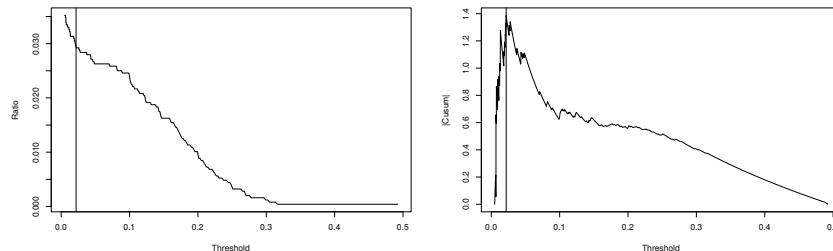


Figure 4: Ratio_k (left) and CUSUM_k (right) plotted against t_k when $\mathbf{B} = \widehat{\beta}^{\text{las}}$ obtained from the data simulated in [Data simulation](#) with $n = 500$ and $p = 50$, as a Lasso estimator of the VAR parameter matrix, with the selected t_{ada} denoted by the vertical lines.

3.3 VAR order d , λ and η

[Step 2](#) and [Step 3](#) of the network estimation methodology of FNets involve the selection of the tuning parameters λ and η (see [\(11\)](#), [\(12\)](#) and [\(14\)](#)) and the VAR order d . While there exist a variety of methods available for VAR order selection in fixed dimensions ([Lütkepohl, 2005](#), Chapter 4), the data-driven selection of d in high dimensions remains largely unaddressed with a few exceptions ([Nicholson et al., 2020](#); [Krampe and Margaritella, 2021](#); [Zheng, 2022](#)). We suggest two methods for jointly selecting λ and d for Step 2. The first method is also applicable for selecting η in Step 3.

Cross validation

Cross validation (CV) methods have been popularly adopted for tuning parameter and model selection. [Bergmeir et al. \(2018\)](#) study the usage of a conventional CV procedure that randomly partitions the data, in the time series settings when the model is correctly specified. However, such arguments do not apply to our problem since the VAR process is latent. Instead, we propose to adopt a modified CV procedure that bears resemblance to out-of-sample evaluation or rolling forecasting validation ([Wang and Tsay, 2021](#)), for simultaneously selecting d and λ in Step 2. For this, the data is partitioned into L folds, $\mathcal{I}_l = \{n_l^\circ + 1, \dots, n_{l+1}^\circ\}$ with $n_l^\circ = \min(l\lceil n/L \rceil, n)$, $1 \leq l \leq L$, and each fold is split into a training set $\mathcal{I}_l^{\text{train}} = \{n_l^\circ + 1, \dots, \lceil(n_l^\circ + n_{l+1}^\circ)/2\rceil\}$ and a test set $\mathcal{I}_l^{\text{test}} = \mathcal{I}_l \setminus \mathcal{I}_l^{\text{train}}$. On each fold, β is

estimated from $\{\mathbf{X}_t, t \in \mathcal{I}_l^{\text{train}}\}$ as either the Lasso (11) or the Dantzig selector (12) estimators with λ as the tuning parameter and some b as the VAR order, say $\hat{\beta}_l^{\text{train}}(\lambda, b)$, using which we compute the CV measure

$$\begin{aligned} \text{CV}(\lambda, b) = \sum_{l=1}^L \text{tr} & \left(\hat{\Gamma}_{\xi,l}^{\text{test}}(0) - (\hat{\beta}_l^{\text{train}}(\lambda, b))^{\top} \hat{\mathbf{g}}_l^{\text{test}}(b) - \right. \\ & \left. (\hat{\mathbf{g}}_l^{\text{test}}(b))^{\top} \hat{\beta}_l^{\text{train}}(\lambda, b) + (\hat{\beta}_l^{\text{train}}(\lambda, b))^{\top} \hat{\mathbf{G}}_l^{\text{test}}(b) \hat{\beta}_l^{\text{train}}(\lambda, b) \right), \end{aligned}$$

where $\hat{\Gamma}_{\xi,l}^{\text{test}}(\ell)$, $\hat{\mathbf{G}}_l^{\text{test}}(b)$ and $\hat{\mathbf{g}}_l^{\text{test}}(b)$ are generated analogously as $\hat{\Gamma}_{\xi}(\ell)$, $\hat{\mathbf{G}}(b)$ and $\hat{\mathbf{g}}(b)$, respectively, from the test set $\{\mathbf{X}_t, t \in \mathcal{I}_l^{\text{test}}\}$. Although we do not directly observe ξ_t , the measure $\text{CV}(\lambda, b)$ gives an approximation of the prediction error. Then, we select $(\hat{\lambda}, \hat{d}) = \arg \min_{\lambda \in \Lambda, 1 \leq b \leq \bar{d}} \text{CV}(\lambda, b)$, where Λ is a grid of values for λ , and $\bar{d} \geq 1$ is a pre-determined upper bound on the VAR order. A similar approach is taken for the selection of η with a Burg matrix divergence-based CV measure:

$$\text{CV}(\eta) = \sum_{l=1}^L \text{tr} \left(\hat{\Delta}_l^{\text{train}}(\eta) \hat{\Gamma}_l^{\text{test}} \right) - \log \left| \hat{\Delta}_l^{\text{train}}(\eta) \hat{\Gamma}_l^{\text{test}} \right| - p.$$

Here, $\hat{\Delta}_l^{\text{train}}(\eta)$ denotes the estimator of Δ with η as the tuning parameter from $\{\mathbf{X}_t, t \in \mathcal{I}_l^{\text{train}}\}$, and $\hat{\Gamma}_l^{\text{test}}$ the estimator of Γ from $\{\mathbf{X}_t, t \in \mathcal{I}_l^{\text{test}}\}$, see Step 3 for the descriptions of the estimators. In the numerical results reported in Simulations, the sample size is relatively small (ranging between $n = 200$ and $n = 500$ while $p \in \{50, 100, 200\}$ and the number of parameters increasing with p^2), and we set $L = 1$ which returns reasonably good performance. When more observations are available, relative to the dimensionality, we may use the number of folds greater than one.

Extended Bayesian information criterion

Alternatively, to select the pair (λ, d) in Step 2, we propose to use the extended Bayesian information criterion (eBIC) of Chen and Chen (2008), originally proposed for variable selection in high-dimensional linear regression. Let $\tilde{\beta}(\lambda, b, t_{\text{ada}})$ denote the thresholded version of $\hat{\beta}(\lambda, b)$ as in (13) with the threshold t_{ada} chosen as described in Threshold t . Then, letting $s(\lambda, b) = |\tilde{\beta}(\lambda, b, t_{\text{ada}})|_0$, we define

$$\begin{aligned} \text{eBIC}_{\alpha}(\lambda, b) &= \frac{n}{2} \log(\mathcal{L}(\lambda, b)) + s(\lambda, b) \log(n) + 2\alpha \log \binom{bp^2}{s(\lambda, b)}, \quad \text{where} \\ \mathcal{L}(\lambda, b) &= \text{tr} \left(\hat{\mathbf{G}}(b) - (\tilde{\beta}(\lambda, b))^{\top} \hat{\mathbf{g}}(b) - (\hat{\mathbf{g}}(b))^{\top} \tilde{\beta}(\lambda, b) + (\tilde{\beta}(\lambda, b))^{\top} \hat{\mathbf{G}}(b) \tilde{\beta}(\lambda, b) \right). \end{aligned} \quad (21)$$

Then, we select $(\hat{\lambda}, \hat{d}) = \arg \min_{\lambda \in \Lambda, 1 \leq b \leq \bar{d}} \text{eBIC}_{\alpha}(\lambda, b)$. The constant $\alpha \in (0, 1)$ determines the degree of penalisation which may be chosen from the relationship between n and p . Preliminary simulations suggest that $\alpha = 0$ is a suitable choice for the dimensions (n, p) considered in our numerical studies.

3.4 Other tuning parameters

Motivated by theoretical results reported in Barigozzi et al. (2023), we select the kernel bandwidth for Step 1 of FNets as $m = \lfloor 4(n/\log(n))^{1/3} \rfloor$. In forecasting the factor-driven component as in (16), we set the truncation lag at $K = 20$, as it is expected that the elements of \mathbf{B}_{ℓ} decay rapidly as ℓ increases for short-memory processes.

4 Package overview

fnets is available from the Comprehensive R Archive Network (CRAN). The main function, `fnets`, implements the FNets method for the input data and returns an object of S3 class `fnets`. `fnets.var` implements Step 2 of the FNets methodology estimating the VAR parameters only, and is applicable directly for VAR modelling of high-dimensional time series; its outputs are of class `fnets`. `fnets.factor.model` performs factor modelling under either of the two models (2) and (3), and returns an object of class `fm`. We provide `predict` methods for the objects of classes `fnets` and `fm`, and a `plot` method for the `fnets` class objects. Prior to using these functions to fit VAR models, we recommend to perform a unit root test and, if necessary, transform the time series such that it is stationary. In this section, we demonstrate how to use the functions included with the package.

4.1 Data simulation

For illustration, we generate an example dataset of $n = 500$ and $p = 50$, following the model described in (4). **fnets** provides functions for this purpose. For given n and p , the function `sim.var` generates the VAR(1) process following (1) with $d = 1$, Γ as supplied to the function ($\Gamma = \mathbf{I}$ by default), and \mathbf{A}_1 generated as described in [Simulations](#). The function `sim.unrestricted` generates the factor-driven component under the unrestricted factor model in (2) with q dynamic factors ($q = 2$ by default) and the filter $\mathcal{B}(L)$ generated as in model (C1) of [Simulations](#).

```
set.seed(111)
n <- 500
p <- 50
x <- sim.var(n, p)$data + sim.unrestricted(n, p)$data
```

Throughout this section, we use the generated dataset for demonstrating the use of **fnets**, unless specified otherwise. There also exists `sim.restricted` which generates the factor-driven component under the restricted factor model in (3). For all data simulation functions, the default is to use the standard normal distribution when generating \mathbf{u}_t and ε_t . However, by specifying the argument `heavy = TRUE`, the innovations are generated from $\sqrt{3/5} \cdot t_5$, the t -distribution with 5 degrees of freedom scaled to have unit variance. The package also comes attached with pre-generated datasets `data.restricted` and `data.unrestricted`.

4.2 Calling fnets with default parameters

The function `fnets` can be called with the $n \times p$ data matrix `x` as the only input, which sets all other arguments to their default choices. It then performs the factor-adjustment under the unrestricted model in (2) with q estimated by minimising the IC in (19). The VAR parameter matrix is estimated via the Lasso estimator in (11), with $d = 1$ as the VAR order, and the tuning parameters λ and η chosen via CV, without any thresholding step. This returns an object of class `fnets` whose entries are described in Table 1.

```
fnets(x)

Factor-adjusted vector autoregressive model with
n: 500, p: 50
Factor-driven common component -----
Factor model: unrestricted
Factor number: 2
Factor number selection method: ic
Information criterion: IC5
Idiosyncratic VAR component -----
VAR order: 1
VAR estimation method: lasso
Tuning method: cv
Threshold: FALSE
Non-zero entries: 95/2500
Long-run partial correlations -----
LRPC: TRUE
```

4.3 Calling fnets with optional parameters

We can also specify the arguments of `fnets` to control how Steps 1–3 of FNETS are to be performed. The full model call is as follows:

```
out <- fnets(x, center = TRUE, fm.restricted = FALSE,
  q = c("ic", "er"), ic.op = NULL, kern.bw = NULL,
  common.args = list(factor.var.order = NULL, max.var.order = NULL, trunc.lags = 20,
    n.perm = 10), var.order = 1, var.method = c("lasso", "ds"),
  var.args = list(n.iter = NULL, n.cores = min(parallel::detectCores() - 1, 3)),
  do.threshold = FALSE, do.lrpc = TRUE, lrpc.adaptive = FALSE,
  tuning.args = list(tuning = c("cv", "bic"), n.folds = 1, penalty = NULL,
    path.length = 10))
)
```

Table 1: Entries of S3 objects of class fnets

Name	Description	Type
q	Factor number	integer
spec	Spectral density matrices for \mathbf{X}_t , χ_t and ξ_t (when fm.restricted = FALSE)	list
acv	Autocovariance matrices for \mathbf{X}_t , χ_t and ξ_t	list
loadings	Estimates of \mathbf{B}_ℓ , $0 \leq \ell \leq K$ (when fm.restricted = FALSE) or Λ (when fm.restricted = TRUE)	array
factors	Estimates of $\{\mathbf{u}_t\}$ (when fm.restricted = FALSE) or $\{\mathbf{F}_t\}$ (when fm.restricted = TRUE)	array
idio.var	Estimates of \mathbf{A}_ℓ , $1 \leq \ell \leq d$, and Γ , and d and λ used	list
lrc	Estimates of Δ , Ω , (long-run) partial correlations and η used	list
mean.x	Sample mean vector	vector
var.method	Estimation method for \mathbf{A}_ℓ (input parameter)	string
do.lrc	Whether to estimate the long-run partial correlations (input parameter)	Boolean
kern.bw	Kernel bandwidth (when fm.restricted = FALSE, input parameter)	double

Here, we discuss a selection of input arguments. The center argument will de-mean the input. fm.restricted determines whether to perform the factor-adjustment under the restricted factor model in (3) or not. If the number of factors is known, we can specify q with a non-negative integer. Otherwise, it can be set as "ic" or "er", which specifies either (19) or (20) as the factor number estimator, respectively. When q = "ic", setting the argument ic.op as an integer between 1 and 6 specifies the choice of the IC (see Appendix A) where the default is ic.op = 5. kern.bw takes a positive integer which specifies the bandwidth to be used in Step 1 of FNets. The list common.args specifies arguments for estimating \mathbf{B}_ℓ and \mathbf{u}_t under (2), and relates to the low-rank VAR representation of χ_t under the unrestricted factor model. var.order specifies a vector of positive integers to be considered in VAR order selection. var.method determines the method for VAR parameter estimation, which can be either "lasso" (for the estimator in (11)) or "ds" (for that in (12)). The list var.args takes additional parameters for Step 2 of FNets, such as the number of gradient descent steps (n.iter, when var.method = "lasso") or the number of cores to use for parallel computing (n.cores, when var.method = "ds"). do.threshold specifies whether to threshold the estimators of \mathbf{A}_ℓ , $1 \leq \ell \leq d$, Δ and Ω . It is possible to perform Steps 1–2 of FNets only without estimating Δ and Ω by setting do.lrc = FALSE. If do.lrc = TRUE, lrc.adaptive specifies whether to use the non-adaptive estimator in (14) or the ACLIME estimator. The list tuning.args supplies arguments to the CV or eBIC procedures, including the number of folds L (n.folds), the eBIC parameter α (penalty, see (21)) and the length of the grid of values for λ and/or η (path.length). Finally, it is possible to set only a subset of the arguments of common.args, var.args and tuning.args whereby the unspecified arguments are set to their default values.

The factor adjustment (Step 1) and VAR parameter estimation (Step 2) functionalities can be accessed individually by calling fnets.factor.model and fnets.var, respectively. The latter is equivalent to calling fnets with q = 0 and do.lrc = FALSE. The former returns an object of class fm which contains the entries of the fnets object in Table 1 that relate to the factor-driven component only.

4.4 Network visualisation

Using the plot method available for the objects of class fnets, we can visualise the Granger network \mathcal{N}^G induced by the estimated VAR parameter matrices (see the left panel of Figure 5):

```
plot(out, type = "granger", display = "network")
```

With display = "network", the function plots an igraph object from the **igraph** package (Csardi et al., 2006). Setting the argument type to "pc" or "lrc", we can visualise \mathcal{N}^C given by the partial correlations of VAR innovations or \mathcal{N}^L given by the long-run partial correlations of ξ_t . By setting display = "heatmap", we can visualise the networks as a heat map instead, with colour indicating edge weights. This plot relies on the **fields** package (Douglas Nychka et al., 2021) and **RColorBrewer** (Neuwirth, 2022). We plot \mathcal{N}^L as a heat map in the right panel of Figure 5 using the following command:

```
plot(out, type = "lrc", display = "heatmap")
```

It is also possible to directly produce an igraph object from the objects of class fnets via the network method as:

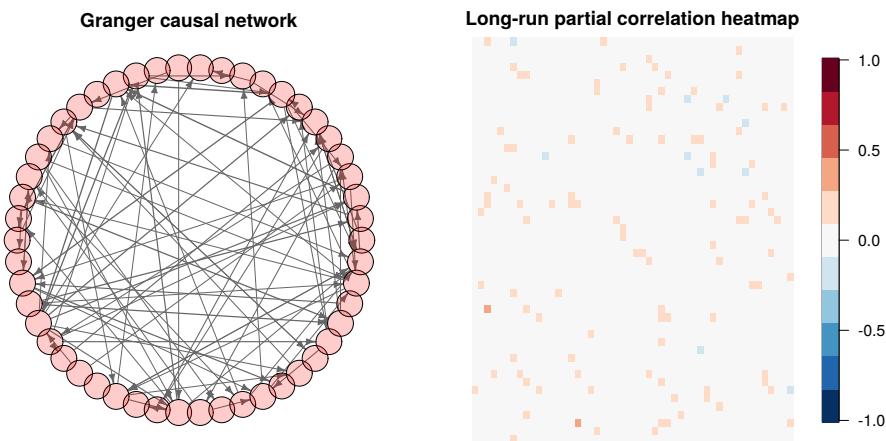


Figure 5: Estimated networks for simulated data described in [Data simulation](#). Left: Granger causal network \mathcal{N}^G . A directed arrow from node i to node i' indicates that variable i Granger causes node i' , and the width of the arrow indicates the edge weight or estimated coefficient. Right: Long-run partial correlation network \mathcal{N}^L with edge weights (i.e. partial correlations) visualised by the colour.

```
g <- network(out, type = "granger")$network
plot(g, layout = igraph::layout_in_circle(g),
      vertex.color = grDevices::rainbow(1, alpha = 0.2), vertex.label = NA,
      main = "Granger causal network")
```

This produces a plot identical to the left panel of Figure 5 using the `igraph` object `g`.

4.5 Forecasting

The `fnets` objects also implement the `predict` method with which we can forecast the input data `n.ahead` steps. For example, we can produce a one-step ahead forecast of \mathbf{X}_{n+1} as

```
pr <- predict(out, n.ahead = 1, fc.restricted = TRUE)
pr$forecast
```

The argument `fc.restricted` specifies whether to use the estimator $\hat{\chi}_{n+h|n}^{\text{res}}$ in (17) generated under a restricted factor model (3), or $\hat{\chi}_{n+h|n}^{\text{unr}}$ in (16) generated without such a restriction. Table 2 lists the entries from the output from `predict.fnets`. We can similarly produce forecasts from `fnets` objects output from `fnets.var`, or `fms` objects from `fnets.factor.model`.

Table 2: Entries of the output from `predict.fnets`

Name	Description	Type
<code>forecast</code>	$h \times p$ matrix containing the h -step ahead forecasts of \mathbf{X}_t	matrix
<code>common.predict</code>	A list containing	list
<code>\$is</code>	$n \times p$ matrix containing the in-sample estimator of χ_t	
<code>\$fc</code>	$h \times p$ matrix containing the h -step ahead forecasts of χ_t	
<code>\$h</code>	Input parameter	
<code>\$r</code>	Factor number (only produced when <code>fc.restricted = TRUE</code>)	
<code>idio.predict</code>	A list containing <code>is</code> , <code>fc</code> and <code>h</code> , see <code>common.predict</code>	list
<code>mean.x</code>	Sample mean vector	vector

4.6 Factor number estimation

It may be of interest to estimate the number of factors (if any) in the input dataset, independent of any estimation procedure. The function `factor.number` provides access to the two methods for selecting q described in [Factor numbers \$q\$ and \$r\$](#) . The following code calls the information criterion-based factor number estimation method in (19), and prints the output:

```

fn <- factor.number(x, fm.restricted = FALSE)
print(fn)

Factor number selection
Factor model: unrestricted
Method: Information criterion
Number of factors:
IC1: 2
IC2: 2
IC3: 3
IC4: 2
IC5: 2
IC6: 2

```

Calling `plot(fn)` returns Figure 3 which visualises the factor number estimators from six information criteria implemented. Alternatively, we call the eigenvalue ratio-based method in (20) as

```
fn <- factor.number(x, method = "er", fm.restricted = FALSE)
```

In this case, `plot(fn)` produces a plot of $ER(b)$ against the candidate factor number $b \in \{1, \dots, \bar{q}\}$.

4.7 Visualisation of tuning parameter selection procedures

The method for threshold selection discussed in [Threshold t](#) is implemented by the `threshold` function, which returns objects of `threshold` class supported by `print` and `plot` methods.

```

th <- threshold(out$idio.var$beta)
th

```

```

Thresholded matrix
Threshold: 0.0297308643
Non-zero entries: 62/2500

```

The call `plot(th)` generates Figure 4. Additionally, we provide tools for visualising the tuning parameter selection results adopted in Steps 2 and 3 of FNets (see [VAR order d, \$\lambda\$ and \$\eta\$](#)). These tools are accessible from both `fnets` and `fnets.var` by calling the `plot` method with the argument `display = "tuning"`, e.g.

```

set.seed(111)
n <- 500
p <- 10
x <- sim.var(n, p)$data
out1 <- fnets(x, q = 0, var.order = 1:3, tuning.args = list(tuning = "cv"))
plot(out1, display = "tuning")

```

This generates the two plots reported in Figure 6 which visualise the CV errors computed as described in [Cross validation](#) and, in particular, the left plot shows that the VAR order is correctly selected by this approach. When `tuning.args` contains `tuning = "bic"`, the results from the eBIC method described in [Extended Bayesian information criterion](#) adopted in Step 2, is similarly visualised in place of the left panel of Figure 6.

5 Simulations

[Barigozzi et al. \(2023\)](#) provide comprehensive simulation results on the estimation and forecasting performance of FNets in comparison with competing methodologies. Therefore in this paper, we focus on assessing the performance of the methods for selecting tuning parameters such as the threshold and VAR order discussed in [Tuning parameter selection](#). Additionally in [Appendix B](#), we compare the adaptive and the non-adaptive estimators in estimating Δ and also investigate how their performance is carried over to estimating Ω .

5.1 Settings

We consider the following data generating processes for the factor-driven component χ_t :

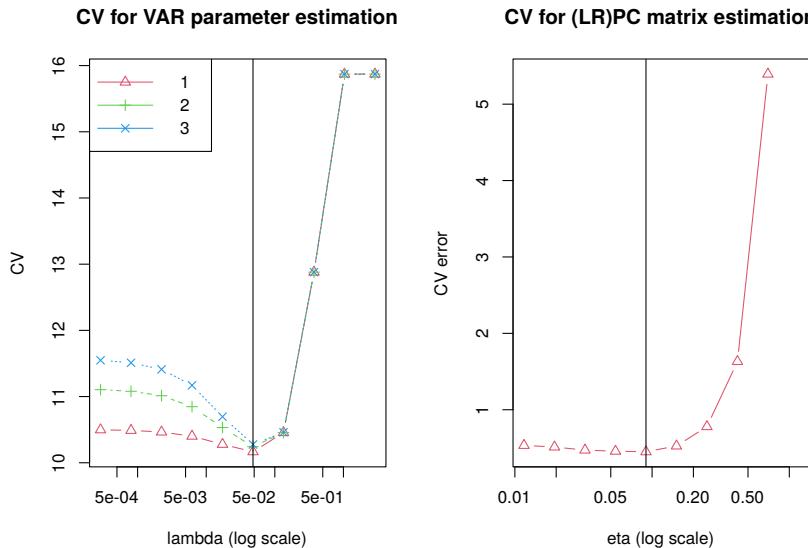


Figure 6: Plots of $\text{CV}(\lambda, b)$ against λ with $b \in \{1, 2, 3\}$ (left) and $\text{CV}(\eta)$ against η (right). Vertical lines denote where the minimum CV measure is attained with respect to λ and η , respectively.

- (C1) Taken from Forni et al. (2017), χ_{it} is generated as a sum of AR processes $\chi_{it} = \sum_{j=1}^q a_{ij}(1 - \alpha_{ij}L)^{-1}u_{jt}$ with $q = 2$, where $u_{jt} \sim_{\text{iid}} \mathcal{N}(0, 1)$, $a_{ij} \sim_{\text{iid}} \mathcal{U}[-1, 1]$ and $\alpha_{ij} \sim_{\text{iid}} \mathcal{U}[-0.8, 0.8]$ with $\mathcal{U}[a, b]$ denoting a uniform distribution. Then, χ_t does not admit a static representation in (3).
- (C2) $\chi_t = \mathbf{0}$, i.e. the VAR process is directly observed as $\mathbf{X}_t = \boldsymbol{\xi}_t$.

For generating a VAR(d) process $\boldsymbol{\xi}_t$, we first generate a directed Erdős-Renyi random graph $\mathcal{N} = (\mathcal{V}, \mathcal{E})$ on $\mathcal{V} = \{1, \dots, p\}$ with the link probability $1/p$, and set entries of \mathbf{A}_d such that $A_{d,ii'} = 0.275$ when $(i, i') \in \mathcal{E}$ and $A_{d,ii'} = 0$ otherwise. Also, we set $\mathbf{A}_\ell = \mathbf{O}$ for $\ell < d$. The VAR innovations are generated as below.

- (E1) Gaussian with the covariance matrix $\Gamma = \Delta^{-1} = \mathbf{I}$.
- (E2) Gaussian with the covariance matrix $\Gamma = \Delta^{-1}$ such that $\delta_{ii} = 1$, $\delta_{i,i+1} = \delta_{i+1,i} = 0.6$, $\delta_{i,i+2} = \delta_{i+2,i} = 0.3$, and $\delta_{ii'} = 0$ for $|i - i'| \geq 3$.

For each setting, we generate 100 realisations.

5.2 Results: Threshold selection

We assess the performance of the adaptive threshold. We generate χ_t as in (C1) and fix $d = 1$ for generating $\boldsymbol{\xi}_t$ and further, treat d as known. We consider $(n, p) \in \{(200, 50), (200, 100), (500, 100), (500, 200)\}$. Then we estimate Ω using the thresholded Lasso estimator of \mathbf{A}_1 (see (11) and (13)) with two choices of thresholds, $t = t_{\text{ada}}$ generated as described in Threshold t and $t = 0$. To assess the performance of $\hat{\Omega} = [\hat{\omega}_{ii'}]$ in recovering the support of $\Omega = [\omega_{ii'}]$, i.e. $\{(i, i') : \omega_{ii'} \neq 0\}$, we plot the receiver operating characteristic (ROC) curves of the true positive rate (TPR) against false positive rate (FPR), where

$$\text{TPR} = \frac{|\{(i, i') : \hat{\omega}_{ii'} \neq 0 \text{ and } \omega_{ii'} \neq 0\}|}{|\{(i, i') : \omega_{ii'} \neq 0\}|} \quad \text{and} \quad \text{FPR} = \frac{|\{(i, i') : \hat{\omega}_{ii'} \neq 0 \text{ and } \omega_{ii'} = 0\}|}{|\{(i, i') : \omega_{ii'} = 0\}|}.$$

Figure 7 plots the ROC curves averaged over 100 realisations when $t = t_{\text{ada}}$ and $t = 0$. When $\Delta = \mathbf{I}$ under (E1), we see little improvement from adopting t_{ada} as the support recovery performance is already good even without thresholding. However, when $\Delta \neq \mathbf{I}$ under (E2), the adaptive threshold leads to improved support recovery especially when the sample size is large. Tables 3 and 4 in Appendix C additionally report the errors in estimating \mathbf{A}_1 and Ω with and without thresholding, where we see little change is brought by thresholding. In summary, we conclude that the estimators already perform reasonably well without thresholding, and the adaptive threshold t_{ada} brings marginal improvement in support recovery which is of interest in network estimation.

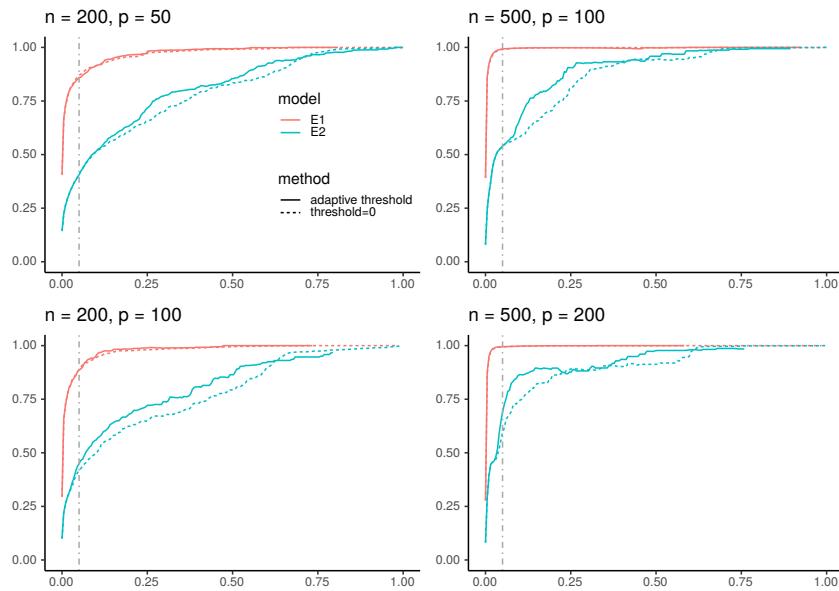


Figure 7: ROC curves of TPR against FPR for $\tilde{\beta}(t)$ (13) (with $\hat{\beta} = \hat{\beta}^{\text{las}}$) when $t = t_{\text{ada}}$ and $t = 0$ in recovering the support of Ω , averaged over 100 realisations. Vertical lines indicate $FPR = 0.05$

5.3 Results: VAR order selection

We compare the performance of the CV and eBIC methods proposed in VAR order d, λ and η for selecting the order of the VAR process. Here, we consider the case when $\chi_t = 0$ (setting (C2)) and when ξ_t is generated under (E1) with $d \in \{1, 3\}$. We set $(n, p) \in \{(200, 10), (200, 20), (500, 10), (500, 20)\}$ where the range of p is in line with the simulation studies conducted in the relevant literature (see e.g. Zheng (2022)). We consider $\{1, 2, 3, 4\}$ as the candidate VAR orders. Figure 8 and Table 5 in Appendix C show that CV works reasonably well regardless of $d \in \{1, 3\}$, with slightly better performance observed together with the DS estimator. On the other hand, eBIC tends to over-estimate the VAR order when $d = 1$ while under-estimating it when $d = 3$, and hence is less reliable compared to the CV method.

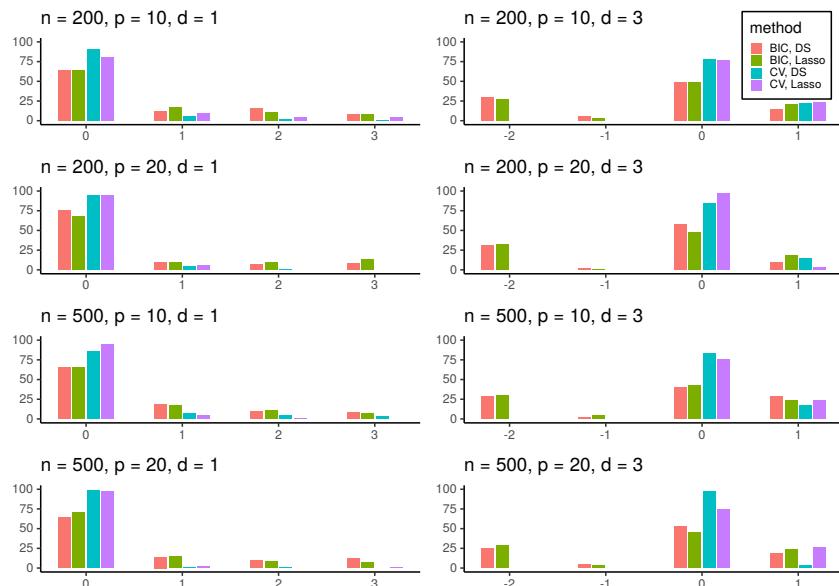


Figure 8: Box plots of $\hat{d} - d$ over 100 realisations when the VAR order is selected by the CV and eBIC methods in combination with the Lasso (11) and the DS (12) estimators.

6 Real-world data example

6.1 Energy price data

Compared with physical commodities, electricity is more difficult to store, and this results in high volatility and seasonality in spot prices (Han et al., 2022). Global market deregulation has increased the volume of electricity trading, which promotes the development of better forecasting and risk management methods. We analyse a dataset of node-specific prices in the PJM (Pennsylvania, New Jersey and Maryland) power pool area in the United States, accessed using dataminer2.pjm.com. There are four node types in the panel, which are Zone, Aggregate, Hub, and Extra High Voltage (EHV) (for definitions, names, and types of the $p = 50$ nodes, see Tables 9 and 8 in Appendix D). The series we model is the sum of the real time congestion price and marginal loss price or, equivalently, the difference between the spot price at a given location and the overall system price, where the latter can be thought of as an observed factor in the local spot price. These are obtained as hourly prices and then averaged over each day as per Maciejowska and Weron (2013). We remove any short-term seasonality by subtracting a separate mean for each day of the week. Since the energy prices may take negative values, we adopt the inverse hyperbolic sine transformation as in Uniejewski et al. (2017) for variance stabilisation.

6.2 Network estimation

We analyse the data collected between 01/01/2021 and 19/07/2021 ($n = 200$). The information criterion in (19) selects a single factor ($\hat{q} = 1$), and $\hat{d} = 1$ is selected by CV. See Figure 9 for the heat maps visualising the three networks \mathcal{N}^G , \mathcal{N}^C and \mathcal{N}^L described in Networks, which are produced by `fnets`.

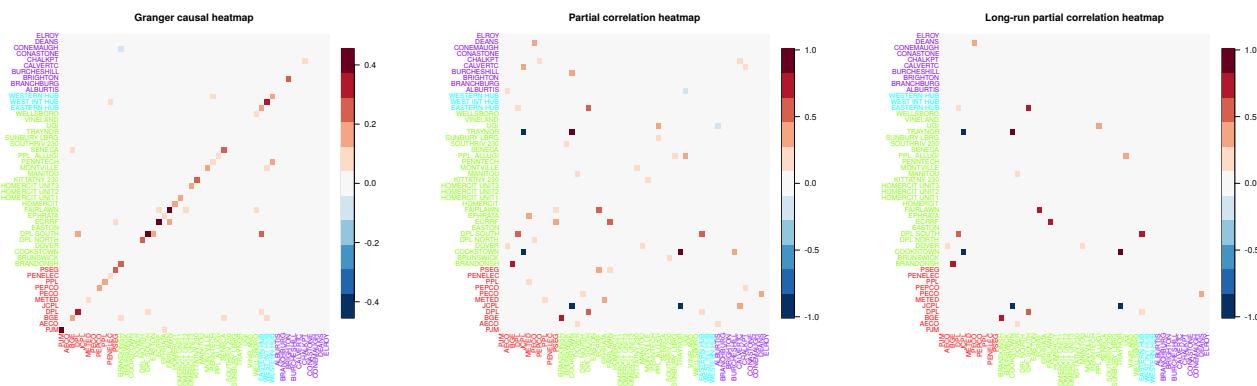


Figure 9: Heat maps of the three networks underlying the energy price data collected over the period 01/01/2021–19/07/2021. Left: \mathcal{N}^G obtained with the Lasso estimator (11) combined with the adaptive threshold t_{ada} . Middle: \mathcal{N}^C obtained with the ACLIME estimator of Δ . Right: \mathcal{N}^L obtained by combining the estimators of VAR parameters and Δ . In the axis labels, Zone-type nodes are coloured in red, Aggregate-types in green, Hub-types in blue and EHV-types in purple.

The non-zero entries of the VAR parameter matrix estimates tend to take positive values, indicating that high energy prices are persistent and spill over to other nodes. Considering the node types, Hub-type nodes (blue) tend to have out-going edges to nodes of different types, which reflects the behaviour of the electrical transmission system. Some Zone-type nodes (red) have several in-coming edges from Aggregate-types (green) and Hub-types, while EHV-types (purple) have few edges in \mathcal{N}^G , which carries forward to \mathcal{N}^L where we observe that those Zone-type nodes have strong long-run correlations with other nodes while EHV-types do not.

7 Summary

We introduce the R package `fnets` which implements the FNets methodology proposed by Barigozzi et al. (2023) for network estimation and forecasting of high-dimensional time series exhibiting strong correlations. The package further implements several data-driven methods for selecting tuning parameters, and provides tools for high-dimensional time series factor modelling under the GDFM. The efficacy of our package is demonstrated on both real and simulated datasets.

1 Appendix A: Information criteria for factor number selection

Here we list information criteria for factor number estimation which are implemented in **fnets** and accessible by the functions **fnets**, **fnets.factor.model** and **factor.number** by setting the argument **ic.op** at an integer belonging to $\{1, \dots, 6\}$. When **fm.restricted = FALSE**, we have

- IC₁: $\left(\frac{1}{p} \sum_{j=b+1}^p \frac{1}{2m+1} \sum_{k=-m}^m \hat{\mu}_{x,j}(\omega_k)\right) + b \cdot c \cdot (m^{-2} + \sqrt{m/n} + p^{-1}) \cdot \log(\min(p, m^2, \sqrt{n/m}))$,
- IC₂: $\left(\frac{1}{p} \sum_{j=b+1}^p \frac{1}{2m+1} \sum_{k=-m}^m \hat{\mu}_{x,j}(\omega_k)\right) + b \cdot c \cdot (\min(p, m^2, \sqrt{n/m}))^{-1/2}$,
- IC₃: $\left(\frac{1}{p} \sum_{j=b+1}^p \frac{1}{2m+1} \sum_{k=-m}^m \hat{\mu}_{x,j}(\omega_k)\right) + b \cdot c \cdot (\min(p, m^2, \sqrt{n/m}))^{-1} \cdot \log(\min(p, m^2, \sqrt{n/m}))$,
- IC₄: $\log\left(\frac{1}{p} \sum_{j=b+1}^p \frac{1}{2m+1} \sum_{k=-m}^m \hat{\mu}_{x,j}(\omega_k)\right) + b \cdot c \cdot (m^{-2} + \sqrt{m/n} + p^{-1}) \cdot \log(\min(p, m^2, \sqrt{n/m}))$,
- IC₅: $\log\left(\frac{1}{p} \sum_{j=b+1}^p \frac{1}{2m+1} \sum_{k=-m}^m \hat{\mu}_{x,j}(\omega_k)\right) + b \cdot c \cdot (\min(p, m^2, \sqrt{n/m}))^{-1/2}$,
- IC₆: $\log\left(\frac{1}{p} \sum_{j=b+1}^p \frac{1}{2m+1} \sum_{k=-m}^m \hat{\mu}_{x,j}(\omega_k)\right) + b \cdot c \cdot (\min(p, m^2, \sqrt{n/m}))^{-1} \cdot \log(\min(p, m^2, \sqrt{n/m}))$.

When **fm.restricted = TRUE**, we use one of

- IC₁: $\left(\frac{1}{p} \sum_{j=b+1}^p \hat{\mu}_{x,j}\right) + b \cdot c \cdot (n+p)/(np) \cdot \log(np/(n+p))$,
- IC₂: $\left(\frac{1}{p} \sum_{j=b+1}^p \hat{\mu}_{x,j}\right) + b \cdot c \cdot (n+p)/(np) \cdot \log(np/(n+p))$,
- IC₃: $\left(\frac{1}{p} \sum_{j=b+1}^p \hat{\mu}_{x,j}\right) + b \cdot c \cdot \log(\min(n,p))/(\min(n,p))$,
- IC₄: $\log\left(\frac{1}{p} \sum_{j=b+1}^p \hat{\mu}_{x,j}\right) + b \cdot c \cdot (n+p)/(np) \cdot \log(np/(n+p))$,
- IC₅: $\log\left(\frac{1}{p} \sum_{j=b+1}^p \hat{\mu}_{x,j}\right) + b \cdot c \cdot (n+p)/(np) \cdot \log(np/(n+p))$,
- IC₆: $\log\left(\frac{1}{p} \sum_{j=b+1}^p \hat{\mu}_{x,j}\right) + b \cdot c \cdot \log(\min(n,p))/(\min(n,p))$.

Whether **fm.restricted = FALSE** or not, the default choice is **ic.op = 5**.

2 Appendix B: ACLIME estimator

We provide a detailed description of the adaptive extension of the CLIME estimator of Δ in (14), extending the methodology proposed in Cai et al. (2016) for precision matrix estimation in the independent setting. Let $\widehat{\Gamma}^* = \widehat{\Gamma} + n^{-1}\mathbf{I}$ and $\eta_1 = 2\sqrt{\log(p)/n}$.

Step 1: Let $\check{\Delta}^{(1)} = [\check{\delta}_{ii'}^{(1)}]$ be the solution to

$$\begin{aligned} \check{\Delta}_{\cdot i'}^{(1)} &= \arg \min_{\mathbf{m} \in \mathbb{R}^p} |\mathbf{m}|_1 \quad \text{subject to} \\ &\left|(\widehat{\Gamma}^* \mathbf{m} - \mathbf{e}_{i'})_i\right| \leq \eta_1 (\widehat{\gamma}_{ii} \vee \widehat{\gamma}_{i'i'}) m_{i'} \quad \forall 1 \leq i \leq p \text{ and } m_{i'} > 0, \end{aligned} \quad (22)$$

for $i' = 1, \dots, p$. Then we obtain truncated estimates

$$\widehat{\delta}_{ii}^{(1)} = \delta_{ii}^{(1)} \cdot \mathbb{I}_{\{|\widehat{\gamma}_{ii}| \leq \sqrt{n/\log(p)}\}} + \sqrt{\frac{\log(p)}{n}} \cdot \mathbb{I}_{\{|\widehat{\gamma}_{ii}| > \sqrt{n/\log(p)}\}}.$$

Step 2: We obtain

$$\check{\Delta}_{\cdot i'}^{(2)} = \arg \min_{\mathbf{m} \in \mathbb{R}^p} |\mathbf{m}|_1 \quad \text{subject to} \quad \left|(\widehat{\Gamma}^* \mathbf{m} - \mathbf{e}_{i'})_i\right| \leq \eta_2 \sqrt{\widehat{\gamma}_{ii} \widehat{\delta}_{i'i'}^{(1)}} \quad \forall 1 \leq i \leq p,$$

where $\eta_2 > 0$ is a tuning parameter. Since $\check{\Delta}^{(2)}$ is not guaranteed to be symmetric, the final estimator is obtained after a symmetrisation step:

$$\widehat{\Delta}_{ada} = [\widehat{\delta}_{ii'}, 1 \leq i, i' \leq p] \text{ with } \widehat{\delta}_{ii'}^{(2)} = \check{\delta}_{ii'}^{(2)} \cdot \mathbb{I}_{\{|\check{\delta}_{ii'}^{(2)}| \leq |\check{\delta}_{i'i'}^{(2)}|\}} + \check{\delta}_{i'i}^{(2)} \cdot \mathbb{I}_{\{|\check{\delta}_{i'i}^{(2)}| < |\check{\delta}_{ii'}^{(2)}|\}}. \quad (23)$$

The constraints in (22) incorporate the parameter in the right-hand side. To use linear programming software to solve this, we formulate the constraints for each $1 \leq i' \leq p$ as

$$\begin{aligned} \forall 1 \leq i \leq p, \quad & ((\widehat{\Gamma}^* - Q^{i'})\mathbf{m} - \mathbf{e}_{i'})_i \leq 0, \\ \forall 1 \leq i \leq p, \quad & -((\widehat{\Gamma}^* + Q^{i'})\mathbf{m} - \mathbf{e}_{i'})_i \leq 0, \\ & m_{i'} > 0. \end{aligned}$$

where $Q^{i'}$ has entries $q_{ii'} = \eta_1(\widehat{\gamma}_{ii'} \vee \widehat{\gamma}_{i'i'})$ in column i' and 0 elsewhere.

3 Appendix C: Additional simulation results

3.1 Threshold selection

Tables 3 and 4 report the errors in estimating \mathbf{A}_1 and $\boldsymbol{\Omega}$ when the threshold $t = t_{\text{ada}}$ or $t = 0$ is applied to the estimator of \mathbf{A}_1 obtained by either the Lasso (11) or the DS (12) estimators. With a matrix γ as an estimand we measure the estimation error of its estimator $\widehat{\gamma}$ using the following (scaled) matrix norms:

$$L_F = \frac{\|\widehat{\gamma} - \gamma\|_F}{\|\gamma\|_F} \quad \text{and} \quad L_2 = \frac{\|\widehat{\gamma} - \gamma\|}{\|\gamma\|}.$$

Table 3: Errors in estimating \mathbf{A}_1 with $t \in \{0, t_{\text{ada}}\}$ in combination with the Lasso (11) and the DS (12) estimators, measured by L_F and L_2 , averaged over 100 realisations (with standard errors reported in brackets). We also report the average TPR when FPR = 0.05 and the corresponding standard error. See [Results: Threshold selection](#) in the main text for further information.

Model	n	p	$t = 0$						$t = t_{\text{ada}}$					
			$\widehat{\beta}^{\text{las}}$			$\widehat{\beta}^{\text{DS}}$			$\widehat{\beta}^{\text{las}}$			$\widehat{\beta}^{\text{DS}}$		
			TPR	L_F	L_2	TPR	L_F	L_2	TPR	L_F	L_2	TPR	L_F	L_2
(E1)	200	50	0.9681 (0.050)	0.6234 (0.081)	0.7204 (0.118)	0.8991 (0.096)	0.4299 (0.280)	0.3747 (0.225)	0.9413 (0.112)	0.6226 (0.088)	0.7204 (0.121)	0.6932 (0.216)	0.4487 (0.256)	0.3960 (0.206)
		100	0.9398 (0.091)	0.6696 (0.096)	0.8113 (0.096)	0.8810 (0.094)	0.5772 (0.449)	0.4362 (0.271)	0.8832 (0.182)	0.6710 (0.108)	0.8132 (0.100)	0.6491 (0.246)	0.6025 (0.418)	0.4642 (0.250)
	500	100	0.9990 (0.003)	0.4648 (0.054)	0.6682 (0.094)	0.9304 (0.065)	0.2740 (0.158)	0.2604 (0.138)	0.9971 (0.010)	0.4608 (0.056)	0.6645 (0.095)	0.7237 (0.199)	0.2806 (0.133)	0.2699 (0.111)
	500	200	0.9986 (0.003)	0.5068 (0.058)	0.7729 (0.081)	0.9167 (0.076)	0.3680 (0.196)	0.3882 (0.134)	0.9964 (0.006)	0.5023 (0.061)	0.7637 (0.082)	0.7095 (0.256)	0.3889 (0.187)	0.4014 (0.126)
(E2)	200	50	0.9595 (0.053)	0.6375 (0.077)	0.7075 (0.094)	0.8828 (0.107)	0.4673 (0.324)	0.4280 (0.255)	0.9442 (0.064)	0.6356 (0.079)	0.7079 (0.096)	0.6720 (0.212)	0.4835 (0.303)	0.4433 (0.241)
		100	0.9624 (0.072)	0.6200 (0.079)	0.6909 (0.089)	0.8093 (0.100)	0.4519 (0.385)	0.4090 (0.251)	0.9435 (0.093)	0.6175 (0.082)	0.6913 (0.090)	0.5903 (0.182)	0.4765 (0.371)	0.4324 (0.243)
	500	100	0.9970 (0.006)	0.4657 (0.056)	0.5553 (0.076)	0.9304 (0.089)	0.3434 (0.158)	0.3621 (0.153)	0.9958 (0.008)	0.4638 (0.058)	0.5525 (0.077)	0.8384 (0.182)	0.3370 (0.140)	0.3634 (0.144)
	500	200	0.9981 (0.003)	0.4702 (0.065)	0.5658 (0.091)	0.9205 (0.088)	0.3684 (0.182)	0.3740 (0.162)	0.9945 (0.014)	0.4686 (0.068)	0.5665 (0.093)	0.8154 (0.205)	0.3663 (0.159)	0.3803 (0.145)

Table 4: Errors in estimating $\boldsymbol{\Omega}$ with $t \in \{0, t_{\text{ada}}\}$ applied to the estimator of \mathbf{A}_1 in combination with the Lasso (11) and the DS (12) estimators, measured by L_F and L_2 , averaged over 100 realisations (with standard errors reported in brackets). We also report the average TPR when FPR = 0.05 and the corresponding standard error. See [Results: Threshold selection](#) in the main text for further information.

Model	n	p	$t = 0$						$t = t_{\text{ada}}$					
			$\widehat{\beta}^{\text{las}}$			$\widehat{\beta}^{\text{DS}}$			$\widehat{\beta}^{\text{las}}$			$\widehat{\beta}^{\text{DS}}$		
			TPR	L_F	L_2	TPR	L_F	L_2	TPR	L_F	L_2	TPR	L_F	L_2
(E1)	200	50	0.8714 (0.108)	0.4143 (0.048)	0.5553 (0.066)	0.8622 (0.119)	0.4217 (0.054)	0.5691 (0.070)	0.8685 (0.118)	0.4145 (0.049)	0.5559 (0.067)	0.8640 (0.121)	0.4217 (0.055)	0.5695 (0.070)
		100	0.8827 (0.084)	0.4320 (0.050)	0.5890 (0.072)	0.8961 (0.080)	0.4379 (0.046)	0.5949 (0.065)	0.8684 (0.139)	0.4326 (0.052)	0.5892 (0.074)	0.8867 (0.120)	0.4386 (0.048)	0.5960 (0.066)
	500	100	0.9909 (0.016)	0.3311 (0.031)	0.4916 (0.069)	0.9886 (0.021)	0.3391 (0.036)	0.4989 (0.065)	0.9928 (0.015)	0.3303 (0.032)	0.4901 (0.069)	0.9901 (0.018)	0.3380 (0.037)	0.4975 (0.066)
	500	200	0.9942 (0.009)	0.3520 (0.038)	0.5287 (0.054)	0.9916 (0.018)	0.3511 (0.045)	0.5400 (0.065)	0.9954 (0.008)	0.3512 (0.039)	0.5273 (0.055)	0.9672 (0.129)	0.3528 (0.055)	0.5399 (0.072)
(E2)	200	50	0.4074 (0.073)	0.7831 (0.089)	0.8353 (0.072)	0.4027 (0.087)	0.7942 (0.079)	0.8335 (0.034)	0.4063 (0.072)	0.7832 (0.089)	0.8353 (0.072)	0.4045 (0.089)	0.7943 (0.079)	0.8336 (0.034)
		100	0.4178 (0.091)	0.8406 (0.108)	0.8690 (0.036)	0.3541 (0.107)	0.9119 (0.126)	0.8879 (0.045)	0.4486 (0.091)	0.8407 (0.108)	0.8690 (0.036)	0.4038 (0.123)	0.9120 (0.126)	0.8880 (0.045)
	500	100	0.5405 (0.111)	0.8267 (0.125)	0.8118 (0.047)	0.5632 (0.122)	0.7910 (0.166)	0.7953 (0.062)	0.5406 (0.111)	0.8267 (0.125)	0.8117 (0.047)	0.5628 (0.123)	0.7910 (0.166)	0.7951 (0.062)
	500	200	0.5951 (0.175)	0.8713 (0.165)	0.8519 (0.088)	0.6487 (0.159)	0.8184 (0.182)	0.8259 (0.090)	0.6918 (0.148)	0.8713 (0.165)	0.8519 (0.088)	0.7101 (0.122)	0.8184 (0.182)	0.8258 (0.090)

3.2 VAR order selection

Table 5 reports the results of VAR order estimation over 100 realisations.

Table 5: Distribution of $\hat{d} - d$ over 100 realisations when the VAR order is selected by the CV and eBIC methods in combination with the Lasso (11) and the DS (12) estimators, see [Results: VAR order selection](#) in the main text for further information.

d	n	p	CV				eBIC											
			$\hat{\beta}^{\text{las}}$		$\hat{\beta}^{\text{DS}}$		$\hat{\beta}^{\text{las}}$		$\hat{\beta}^{\text{DS}}$									
			0	1	2	3	0	1	2	3	0	1						
1	200	10	81	10	4	5	91	6	2	1	64	17	11	8	64	12	16	8
	200	20	94	6	0	0	94	5	1	0	68	10	9	13	75	10	7	8
	500	10	94	5	1	0	86	7	4	3	65	17	11	7	65	18	9	8
	500	20	97	2	0	1	98	1	1	0	70	15	8	7	64	14	10	12
			-2	-1	0	1	-2	-1	0	1	-2	-1	0	1	-2	-1	0	1
3	200	10	0	0	77	23	0	0	78	22	27	3	49	21	30	6	49	15
	200	20	0	0	97	3	0	0	85	15	32	1	48	19	31	2	58	9
	500	10	0	0	76	24	0	0	83	17	30	4	43	23	29	2	40	29
	500	20	0	0	74	26	0	0	97	3	29	3	45	23	25	4	53	18

3.3 CLIME vs. ACLIME estimators

We compare the performance of the adaptive and non-adaptive estimators for the VAR innovation precision matrix Δ and its impact on the estimation of Ω , the inverse of the long-run covariance matrix of the data (see [Step 3](#)). We generate χ_t as in (C1), fix $d = 1$ and treat it as known and consider $(n, p) \in \{(200, 50), (200, 100), (500, 100), (500, 200)\}$.

In Tables 6 and 7, we report the errors of Δ and Ω . We consider both the Lasso (11) and DS (12) estimators of VAR parameters, and CLIME and ACLIME estimators for Δ , which lead to four different estimators for Δ and Ω , respectively. Overall, we observe that with increasing n , the performance of all estimators improve according to all metrics regardless of the scenarios (E1) or (E2), while increasing p has an adverse effect. The two methods perform similarly in setting (E1) when $\Delta = \mathbf{I}$. There is marginal improvement for adopting the ACLIME estimator noticeable under (E2), particularly in TPR. Figures 10 and 11 shows the ROC curves for the support recovery of Δ and Ω when the Lasso estimator is used.

Table 6: Errors in estimating Δ using CLIME and ACLIME estimators, measured by L_F and L_2 , averaged over 100 realisations (with standard errors reported in brackets). We also report the average TPR when FPR = 0.05 and the corresponding standard errors.

Model	n	p	CLIME						ACLIME					
			$\hat{\beta}^{\text{las}}$			$\hat{\beta}^{\text{DS}}$			$\hat{\beta}^{\text{las}}$			$\hat{\beta}^{\text{DS}}$		
			TPR	L_F	L_2	TPR	L_F	L_2	TPR	L_F	L_2	TPR	L_F	L_2
(E1)	200	50	1.000 (0.000)	0.215 (0.047)	0.489 (0.223)	1.000 (0.000)	0.220 (0.047)	0.497 (0.182)	1.000 (0.002)	0.207 (0.043)	0.472 (0.173)	1.000 (0.000)	0.209 (0.041)	0.469 (0.116)
		100	1.000 (0.000)	0.235 (0.036)	0.513 (0.089)	1.000 (0.000)	0.241 (0.036)	0.521 (0.107)	1.000 (0.000)	0.223 (0.033)	0.507 (0.084)	1.000 (0.000)	0.228 (0.034)	0.518 (0.099)
	500	100	1.000 (0.000)	0.181 (0.022)	0.458 (0.062)	1.000 (0.000)	0.183 (0.029)	0.466 (0.087)	1.000 (0.000)	0.176 (0.022)	0.452 (0.052)	1.000 (0.000)	0.178 (0.028)	0.458 (0.069)
	500	200	1.000 (0.000)	0.198 (0.027)	0.510 (0.066)	1.000 (0.000)	0.193 (0.035)	0.492 (0.065)	1.000 (0.000)	0.187 (0.026)	0.505 (0.056)	1.000 (0.000)	0.182 (0.033)	0.489 (0.057)
(E2)	200	50	0.659 (0.058)	0.422 (0.101)	0.816 (0.654)	0.662 (0.057)	0.391 (0.031)	0.608 (0.144)	0.682 (0.055)	0.397 (0.056)	0.706 (0.351)	0.687 (0.054)	0.380 (0.030)	0.600 (0.176)
	200	100	0.639 (0.044)	0.417 (0.039)	0.695 (0.205)	0.637 (0.042)	0.420 (0.043)	0.720 (0.249)	0.669 (0.041)	0.404 (0.037)	0.663 (0.162)	0.668 (0.039)	0.405 (0.037)	0.684 (0.193)
	500	100	0.730 (0.035)	0.372 (0.097)	0.764 (0.828)	0.726 (0.039)	0.499 (1.101)	1.708 (7.586)	0.735 (0.032)	0.358 (0.038)	0.650 (0.322)	0.734 (0.031)	0.361 (0.056)	0.718 (0.517)
	500	200	0.729 (0.028)	0.370 (0.035)	0.711 (0.355)	0.728 (0.028)	0.362 (0.035)	0.736 (0.384)	0.737 (0.023)	0.363 (0.026)	0.647 (0.239)	0.737 (0.024)	0.354 (0.028)	0.673 (0.279)

Table 7: Errors in estimating Ω using CLIME and ACLIME estimators of Δ , measured by L_F and L_2 , averaged over 100 realisations (with standard errors reported in brackets). We also report the average TPR when FPR = 0.05 and the corresponding standard errors.

Model	n	p	CLIME						ACLIME					
			$\hat{\beta}^{\text{las}}$			$\hat{\beta}^{\text{DS}}$			$\hat{\beta}^{\text{las}}$			$\hat{\beta}^{\text{DS}}$		
			TPR	L_F	L_2	TPR	L_F	L_2	TPR	L_F	L_2	TPR	L_F	L_2
(E1)	200	50	0.871 (0.108)	0.415 (0.050)	0.557 (0.070)	0.862 (0.119)	0.422 (0.055)	0.571 (0.080)	0.867 (0.106)	0.411 (0.051)	0.558 (0.088)	0.856 (0.114)	0.417 (0.053)	0.570 (0.083)
		100	0.883 (0.084)	0.432 (0.050)	0.589 (0.072)	0.896 (0.080)	0.438 (0.046)	0.595 (0.065)	0.868 (0.088)	0.423 (0.048)	0.583 (0.077)	0.883 (0.085)	0.429 (0.045)	0.587 (0.061)
	500	100	0.991 (0.016)	0.331 (0.031)	0.492 (0.069)	0.989 (0.021)	0.339 (0.036)	0.499 (0.065)	0.991 (0.015)	0.328 (0.033)	0.490 (0.070)	0.989 (0.019)	0.337 (0.036)	0.498 (0.067)
	500	200	0.994 (0.009)	0.352 (0.038)	0.529 (0.054)	0.992 (0.018)	0.351 (0.045)	0.540 (0.065)	0.994 (0.009)	0.344 (0.038)	0.525 (0.056)	0.990 (0.014)	0.342 (0.044)	0.537 (0.068)
(E2)	200	50	0.509 (0.078)	0.532 (0.071)	0.724 (0.243)	0.510 (0.068)	0.514 (0.043)	0.664 (0.137)	0.504 (0.071)	0.518 (0.055)	0.679 (0.162)	0.507 (0.063)	0.506 (0.043)	0.658 (0.141)
		100	0.511 (0.059)	0.541 (0.047)	0.683 (0.082)	0.513 (0.065)	0.542 (0.051)	0.695 (0.093)	0.509 (0.062)	0.531 (0.045)	0.674 (0.084)	0.504 (0.061)	0.531 (0.046)	0.679 (0.084)
	500	100	0.640 (0.066)	0.450 (0.072)	0.655 (0.402)	0.624 (0.079)	0.544 (0.866)	1.099 (3.714)	0.642 (0.059)	0.441 (0.036)	0.597 (0.118)	0.637 (0.060)	0.440 (0.047)	0.617 (0.204)
	500	200	0.670 (0.045)	0.461 (0.041)	0.630 (0.116)	0.658 (0.043)	0.450 (0.040)	0.630 (0.117)	0.677 (0.041)	0.456 (0.036)	0.612 (0.075)	0.661 (0.037)	0.445 (0.037)	0.605 (0.082)

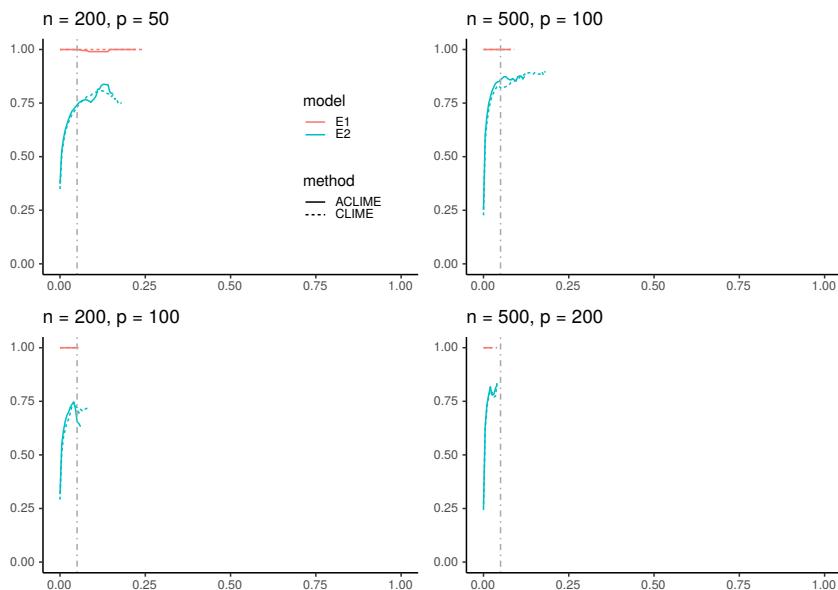


Figure 10: ROC curves of TPR against FPR for $\hat{\Delta}$ with CLIME and ACLIME estimators in recovering the support of Δ , averaged over 100 realisations. Vertical lines indicate FPR = 0.05.

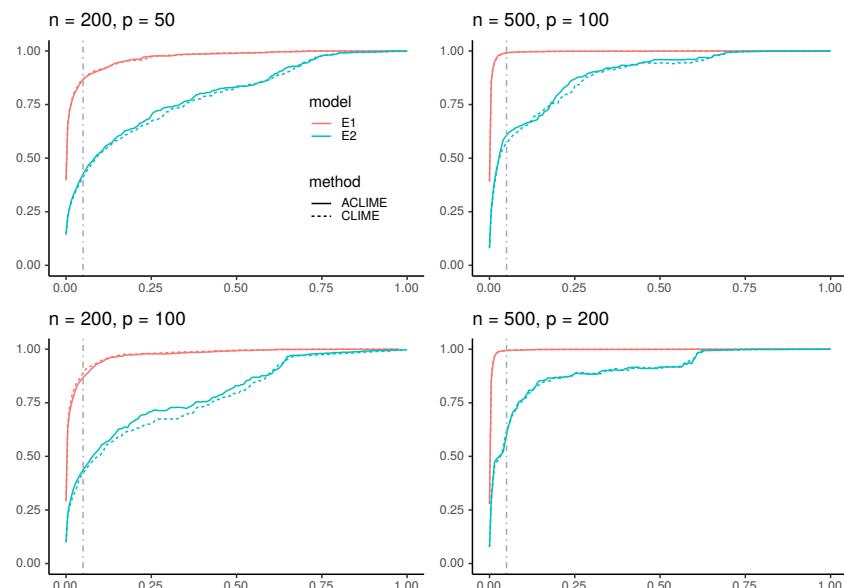


Figure 11: ROC curves of TPR against FPR for $\hat{\Omega}$ with CLIME and ACLIME estimators in recovering the support of Ω , averaged over 100 realisations. Vertical lines indicate $FPR = 0.05$.

4 Appendix D: Dataset information

Table 8 defines the four node types in the panel. Table 9 describes the dataset analysed in Real data example.

Table 8: Node type definitions for energy price data.

Name	Definition
Zone	A transmission owner's area within the PJM Region.
Aggregate	A group of more than one individual bus into a pricing node (pnode) that is considered as a whole in the Energy Market and other various systems and Markets within PJM.
Hub	A group of more than one individual bus into a regional pricing node (pnode) developed to produce a stable price signal in the Energy Market and other various systems and Markets within PJM.
Extra High Voltage (EHV)	Nodes at 345kV and above on the PJM system.

References

- S. C. Ahn and A. R. Horenstein. Eigenvalue ratio test for the number of factors. *Econometrica*, 81(3):1203–1227, 2013. URL <https://doi.org/10.3982/ECTA8968>. [p220]
- L. Alessi, M. Barigozzi, and M. Capasso. Improved penalization for determining the number of factors in approximate factor models. *Statistics & Probability Letters*, 80(23–24):1806–1813, 2010. URL <https://doi.org/10.1016/j.spl.2010.08.005>. [p220]
- M. Avarucci, M. Cavicchioli, M. Forni, and P. Zaffaroni. The main business cycle shock(s): Frequency-band estimation of the number of dynamic factors. CEPR Discussion Paper No. DP17281, 2022. URL <https://dx.doi.org/10.2139/ssrn.3970658>. [p220]
- J. Bai. Inferential theory for factor models of large dimensions. *Econometrica*, 71(1):135–171, 2003. URL <https://doi.org/10.1111/1468-0262.00392>. [p216]
- J. Bai and S. Ng. Determining the number of factors in approximate factor models. *Econometrica*, 70:191–221, 2002. URL <https://doi.org/10.1111/1468-0262.00273>. [p220]
- P. Bai. *LSVAR: Estimation of low rank plus sparse structured vector auto-regressive (VAR) model*, 2021. URL <https://CRAN.R-project.org/package=LSVAR>. R package version 1.2. [p214]
- M. Barigozzi and C. Brownlees. Nets: Network estimation for time series. *Journal of Applied Econometrics*, 34:347–364, 2019. URL <https://doi.org/10.1002/jae.2676>. [p214]
- M. Barigozzi, H. Cho, and D. Owens. Fnets: Factor-adjusted network estimation and forecasting for high-dimensional time series. *Journal of Business & Economic Statistics*, pages 1–13, 2023. URL <https://doi.org/10.1080/07350015.2023.2257270>. [p214, 215, 216, 218, 219, 222, 226, 229]
- S. Basu and G. Michailidis. Regularized estimation in sparse high-dimensional time series models. *The Annals of Statistics*, 43:1535–1567, 2015. URL <10.1214/15-AOS1315>. [p214]
- A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009. URL <https://doi.org/10.1137/080716542>. [p218]
- C. Bergmeir, R. J. Hyndman, and B. Koo. A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics & Data Analysis*, 120:70–83, 2018. [p221]
- M. Berkelaar et al. *lpSolve: Interface to ‘Lpsolve’ v. 5.5 to solve linear/integer programs*, 2020. URL <https://CRAN.R-project.org/package=lpSolve>. R package version 5.6.15. [p218]
- B. S. Bernanke, J. Boivin, and P. Eliasz. Measuring the effects of monetary policy: a factor-augmented vector autoregressive (favar) approach. *The Quarterly Journal of Economics*, 120(1):387–422, 2005. URL <https://doi.org/10.1162/0033553053327452>. [p215]
- M. Billio, M. Getmansky, A. W. Lo, and L. Pelizzon. Econometric measures of connectedness and systemic risk in the finance and insurance sectors. *Journal of Financial Economics*, 104(3):535–559, 2012. URL <https://doi.org/10.1016/j.jfineco.2011.12.010>. [p214]

- C. Brownlees. *nets: Network estimation for time series*, 2020. URL <https://CRAN.R-project.org/package=nets>. R package version 0.9.1. [p214]
- T. Cai, W. Liu, and X. Luo. A constrained ℓ_1 minimization approach to sparse precision matrix estimation. *Journal of the American Statistical Association*, 106(494):594–607, 2011. URL <https://doi.org/10.1198/jasa.2011.tm10155>. [p218]
- T. T. Cai, W. Liu, and H. H. Zhou. Estimating sparse precision matrix: Optimal rates of convergence and adaptive estimation. *The Annals of Statistics*, 44(2):455–488, 2016. URL [10.1214/13-AOS1171](https://doi.org/10.1214/13-AOS1171). [p218, 230]
- E. Candes and T. Tao. The dantzig selector: Statistical estimation when p is much larger than n. *The Annals of Statistics*, 35(6):2313–2351, 2007. URL [10.1214/00905360600001523](https://doi.org/10.1214/00905360600001523). [p218]
- J. Chen and Z. Chen. Extended bayesian information criteria for model selection with large model spaces. *Biometrika*, 95(3):759–771, 2008. URL <https://doi.org/10.1093/biomet/asn034>. [p222]
- G. Csardi, T. Nepusz, et al. The igraph software package for complex network research. *InterJournal, complex systems*, 1695(5):1–9, 2006. [p224]
- R. Dahlhaus. Graphical interaction models for multivariate time series. *Metrika*, 51(2):157–172, 2000. URL <https://doi.org/10.1007/s001840000055>. [p214, 216, 217]
- Douglas Nychka, Reinhard Furrer, John Paige, and Stephan Sain. fields: Tools for spatial data, 2021. URL <https://github.com/dnnychka/fieldsRPackage>. R package version 14.1. [p224]
- M. Eichler. Granger causality and path diagrams for multivariate time series. *Journal of Econometrics*, 137(2):334–353, 2007. URL <http://dx.doi.org/10.1016/j.jeconom.2005.06.032>. [p214]
- S. Epskamp, L. J. Waldorp, R. Möttus, and D. Borsboom. The gaussian graphical model in cross-sectional and time-series data. *Multivariate Behavioral Research*, 53(4):453–480, 2018. URL <https://doi.org/10.1080/00273171.2018.1454823>. [p215]
- J. Fan, Y. Liao, and M. Mincheva. Large covariance estimation by thresholding principal orthogonal complements. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 75(4), 2013. URL <https://doi.org/10.1111/rssb.12016>. [p216]
- M. Forni, M. Hallin, M. Lippi, and L. Reichlin. The generalized dynamic-factor model: Identification and estimation. *Review of Economics and Statistics*, 82(4):540–554, 2000. URL <http://dx.doi.org/10.1162/003465300559037>. [p214, 215, 216]
- M. Forni, M. Hallin, M. Lippi, and L. Reichlin. The generalized dynamic factor model: one-sided estimation and forecasting. *Journal of the American statistical association*, 100(471):830–840, 2005. URL <https://doi.org/10.1198/016214504000002050>. [p219]
- M. Forni, M. Hallin, M. Lippi, and P. Zaffaroni. Dynamic factor models with infinite-dimensional factor spaces: One-sided representations. *Journal of Econometrics*, 185(2):359–371, 2015. URL <https://doi.org/10.1016/j.jeconom.2013.10.017>. [p215, 219]
- M. Forni, M. Hallin, M. Lippi, and P. Zaffaroni. Dynamic factor models with infinite-dimensional factor space: Asymptotic analysis. *Journal of Econometrics*, 199(1):74–92, 2017. URL <https://doi.org/10.1016/j.jeconom.2017.04.002>. [p219, 227]
- M. Hallin and R. Liška. Determining the number of factors in the general dynamic factor model. *Journal of the American Statistical Association*, 102(478):603–617, 2007. URL <https://doi.org/10.1198/01621450600001275>. [p219, 220]
- F. Han, H. Lu, and H. Liu. A direct estimation of high dimensional stationary vector autoregressions. *Journal of Machine Learning Research*, 16(97):3115–3150, 2015. URL <http://jmlr.org/papers/v16/han15a.html>. [p214]
- L. Han, I. Cribben, and S. Trueck. Extremal dependence in australian electricity markets. *arXiv preprint arXiv:2202.09970*, 2022. URL <https://doi.org/10.48550/arXiv.2202.09970>. [p229]
- J. M. Haslbeck and L. J. Waldorp. mgm: Estimating time-varying mixed graphical models in high-dimensional data. *Journal of Statistical Software*, 93:1–46, 2020. URL <https://doi.org/10.18637/jss.v093.i08>. [p214]

- C. Kirch, B. Muhsal, and H. Ombao. Detection of changes in multivariate time series with application to eeg data. *Journal of the American Statistical Association*, 110:1197–1216, 2015. URL <https://doi.org/10.1080/01621459.2014.957545>. [p214]
- 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. [p215]
- A. B. Kock and L. Callot. Oracle inequalities for high dimensional vector autoregressions. *Journal of Econometrics*, 186(2):325–344, 2015. URL <https://doi.org/10.1016/j.jeconom.2015.02.013>. [p214]
- G. M. Koop. Forecasting with medium and large bayesian vars. *Journal of Applied Econometrics*, 28(2):177–203, 2013. URL <https://doi.org/10.1002/jae.1270>. [p214]
- J. Krampe and L. Margaritella. Dynamic factor models with sparse var idiosyncratic components. *arXiv preprint arXiv:2112.07149*, 2021. URL <https://doi.org/10.48550/arXiv.2112.07149>. [p221]
- S. Krantz and R. Bagdziunas. *dfms: Dynamic Factor Models*, 2023. URL <https://sebkrantz.github.io/dfms/>. R package version 0.2.1. [p215]
- B. Liu, X. Zhang, and Y. Liu. Simultaneous change point inference and structure recovery for high dimensional gaussian graphical models. *Journal of Machine Learning Research*, 22(274):1–62, 2021. URL <https://www.jmlr.org/papers/volume22/20-327/20-327.pdf>. [p220]
- L. Liu and D. Zhang. Robust estimation of high-dimensional vector autoregressive models. *arXiv preprint arXiv:2109.10354*, 2021. [p214]
- H. Lütkepohl. *New introduction to multiple time series analysis*. Springer Science & Business Media, 2005. [p221]
- K. Maciejowska and R. Weron. Forecasting of daily electricity spot prices by incorporating intra-day relationships: Evidence from the uk power market. In *2013 10th International Conference on the European Energy Market (EEM)*, pages 1–5. IEEE, 2013. URL [10.1109/EEM.2013.6607314](https://doi.org/10.1109/EEM.2013.6607314). [p229]
- M. C. Medeiros and E. F. Mendes. ℓ_1 -regularization of high-dimensional time-series models with non-gaussian and heteroskedastic errors. *Journal of Econometrics*, 191(1):255–271, 2016. URL <https://doi.org/10.1016/j.jeconom.2015.10.011>. [p214]
- Microsoft and S. Weston. *doParallel: Foreach parallel adaptor for the 'parallel' package*, 2022a. URL <https://CRAN.R-project.org/package=doParallel>. R package version 1.0.17. [p218]
- Microsoft and S. Weston. *foreach: Provides foreach looping construct*, 2022b. URL <https://CRAN.R-project.org/package=foreach>. R package version 1.5.2. [p218]
- L. Mosley, T.-S. Chan, and A. Gibberd. sparseDFM: An R package to estimate dynamic factor models with sparse loadings. *arXiv preprint arXiv:2303.14125*, 2023. URL <https://doi.org/10.48550/arXiv.2303.14125>. [p215]
- E. Neuwirth. *RColorBrewer: ColorBrewer palettes*, 2022. URL <https://CRAN.R-project.org/package=RColorBrewer>. R package version 1.1-3. [p224]
- W. Nicholson, D. Matteson, and J. Bien. Bigvar: Tools for modeling sparse high-dimensional multi-variate time series. *arXiv preprint arXiv:1702.07094*, 2017. URL <https://doi.org/10.48550/arXiv.1702.07094>. [p215]
- W. B. Nicholson, I. Wilms, J. Bien, and D. S. Matteson. High dimensional forecasting via interpretable vector autoregression. *Journal of Machine Learning Research*, 21(166):1–52, 2020. URL <https://jmlr.org/papers/v21/19-777.html>. [p214, 221]
- J. Peng, P. Wang, N. Zhou, and J. Zhu. Partial correlation estimation by joint sparse regression models. *Journal of the American Statistical Association*, 104(486):735–746, 2009. URL [10.32614/RJ-2021-023](https://doi.org/10.32614/RJ-2021-023). [p216]
- A. Shojaie and G. Michailidis. Discovering graphical granger causality using the truncating lasso penalty. *Bioinformatics*, 26(18):i517–i523, 2010. URL <https://doi.org/10.1093/bioinformatics/btq377>. [p214]
- J. H. Stock and M. W. Watson. Forecasting using principal components from a large number of predictors. *Journal of the American Statistical Association*, 97(460):1167–1179, 2002. URL <https://doi.org/10.1198/016214502388618960>. [p216]

- B. Uniejewski, R. Weron, and F. Ziel. Variance stabilizing transformations for electricity spot price forecasting. *IEEE Transactions on Power Systems*, 33(2):2219–2229, 2017. URL <http://dx.doi.org/10.1109/TPWRS.2017.2734563>. [p229]
- S. Vazzoler. *sparsevar: Sparse VAR/VECM Models Estimation*, 2021. URL <https://CRAN.R-project.org/package=sparsevar>. R package version 0.1.0. [p214]
- D. Wang and R. S. Tsay. Rate-optimal robust estimation of high-dimensional vector autoregressive models. *arXiv preprint arXiv:2107.11002*, 2021. URL <https://doi.org/10.48550/arXiv.2107.11002>. [p221]
- I. Wilms, S. Basu, J. Bien, and D. Matteson. *bigtime: Sparse estimation of large time series models*, 2021. URL <https://cran.r-project.org/package=bigtime>. R package version 0.2.1. [p215]
- Y. Zheng. An interpretable and efficient infinite-order vector autoregressive model for high-dimensional time series. *arXiv preprint arXiv:2209.01172*, 2022. URL <https://doi.org/10.48550/arXiv.2209.01172>. [p221, 228]

Dom Owens

School of Mathematics, University of Bristol

Supported by EPSRC Centre for Doctoral Training (EP/S023569/1)

domowens1@gmail.com

Haeran Cho

School of Mathematics, University of Bristol

Supported by the Leverhulme Trust (RPG-2019-390)

haeran.cho@bristol.ac.uk

Matteo Barigozzi

Department of Economics, Università di Bologna

Supported by MIUR (PRIN 2017, Grant 2017TA7TYC)

matteo.barigozzi@unibo.it

Table 9: Names, IDs and Types for the 50 power nodes in the energy price dataset.

Name	Node ID	Node Type
PJM	1	ZONE
AECO	51291	ZONE
BGE	51292	ZONE
DPL	51293	ZONE
JCPL	51295	ZONE
METED	51296	ZONE
PECO	51297	ZONE
PEPCO	51298	ZONE
PPL	51299	ZONE
PENELEC	51300	ZONE
PSEG	51301	ZONE
BRANDONSH	51205	AGGREGATE
BRUNSWICK	51206	AGGREGATE
COOKSTOWN	51211	AGGREGATE
DOVER	51214	AGGREGATE
DPL NORTH	51215	AGGREGATE
DPL SOUTH	51216	AGGREGATE
EASTON	51218	AGGREGATE
ECRRF	51219	AGGREGATE
EPHRATA	51220	AGGREGATE
FAIRLAWN	51221	AGGREGATE
HOMERCIT	51229	AGGREGATE
HOMERCIT UNIT1	51230	AGGREGATE
HOMERCIT UNIT2	51231	AGGREGATE
HOMERCIT UNIT3	51232	AGGREGATE
KITTATNY 230	51238	AGGREGATE
MANITOU	51239	AGGREGATE
MONTVILLE	51241	AGGREGATE
PENNTECH	51246	AGGREGATE
PPL_ALLUGI	51252	AGGREGATE
SENECA	51255	AGGREGATE
SOUTHRIV 230	51261	AGGREGATE
SUNBURY LBRG	51270	AGGREGATE
TRAYNOR	51277	AGGREGATE
UGI	51279	AGGREGATE
VINELAND	51280	AGGREGATE
WELLSBORO	51285	AGGREGATE
EASTERN HUB	51217	HUB
WEST INT HUB	51287	HUB
WESTERN HUB	51288	HUB
ALBURTIS	52443	EHV
BRANCHBURG	52444	EHV
BRIGHTON	52445	EHV
BURCHESHILL	52446	EHV
CALVERTC	52447	EHV
CHALKPT	52448	EHV
CONASTONE	52449	EHV
CONEMAUGH	52450	EHV
DEANS	52451	EHV
ELROY	52452	EHV

Coloring in R's Blind Spot

by Achim Zeileis and Paul Murrell

Abstract Prior to version 4.0.0 R had a poor default color palette (using highly saturated red, green, blue, etc.) and provided very few alternative palettes, most of which also had poor perceptual properties (like the infamous rainbow palette). Starting with version 4.0.0 R gained a new and much improved default palette and, in addition, a selection of more than 100 well-established palettes are now available via the functions `palette.colors()` and `hcl.colors()`. The former provides a range of popular qualitative palettes for categorical data while the latter closely approximates many popular sequential and diverging palettes by systematically varying the perceptual hue, chroma, and luminance (HCL) properties in the palette. This paper provides a mix of contributions including an overview of the new color functions and the palettes they provide along with advice about which palettes are appropriate for specific tasks, especially with regard to making them accessible to viewers with color vision deficiencies.

1 Introduction

Color can be a very effective way to distinguish between different groups within a data visualization. Color is a “preattentive” visual feature, meaning that groups are identified rapidly and without conscious effort (Ware 2012). For example, it is trivial to identify the two groups of points in the scatterplot in Figure 1.

Employing color to represent values on a continuous numeric scale will be less successful (Cleveland and McGill 1984), but color can still be useful to convey additional variables when more effective visual features, such as location, have already been used. For example, color might be used to fill in different regions on a map, as demonstrated in the right hand plot of Figure 1.

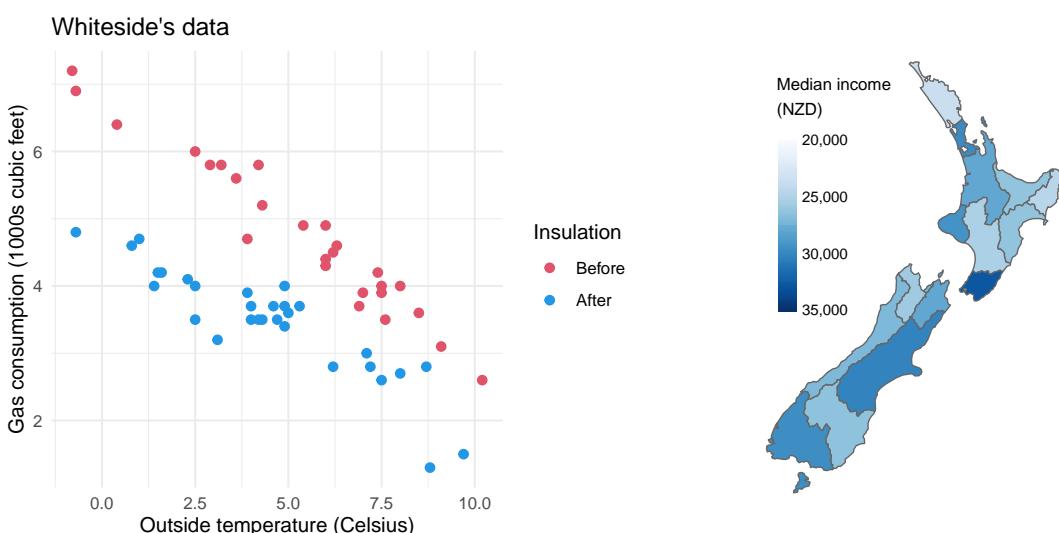


Figure 1: Typical usage of color for coding qualitative/categorical information (left) and quantitative/continuous information (right). Left: Scatter plot of weekly gas consumption by outside temperature before and after installing house insulation. Right: Choropleth map of median income in the 16 regions of New Zealand in 2018.

R provides several ways to specify a color: by name (e.g., "red"); by hexadecimal RGB code (e.g., "#FF0000"); or by integer (e.g., 2). When we specify an integer, that provides an index into a default set of colors; the color 2 means the second color in the default set of colors.

However, a more important task than specifying one particular color is the task of specifying a set of colors to use in combination with each other. For example, in the left panel of Figure 1, we need two colors that are very easily perceived as different from each other. In the right panel of Figure 1, we require a set of colors that appear to change monotonically, e.g., from darker to lighter.

We call this the problem of selecting a good *palette* of colors. What we need to generate is a vector of R colors, e.g., `c("red", "blue")`, `c("#FF0000", "#0000FF")`, or `c(2, 4)`.

2 A brief history of R palettes

Early versions of R provided very few functions for choosing colors from readily available palettes. The palettes that were provided, although standard at the time they were implemented, have meanwhile been widely recognized as being rather poor.

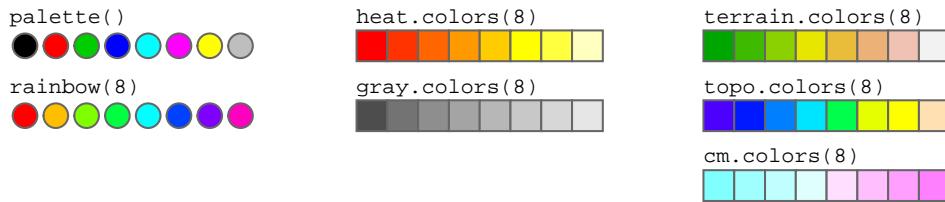


Figure 2: Old base R palettes. At top left is the old default palette (prior to version 4.0.0), consisting largely of highly saturated primary colors or combinations thereof. Below that is the rainbow palette of different highly saturated hues. The middle column shows the old sequential palettes, with heat colors again being highly saturated. The last column shows an old diverging palette plus two palettes motivated by shadings of geographic maps.

The `palette()` function generates a vector of eight colors. These provide the default set of colors that an integer color specification selects from and can be used for coding *categorical* information. The output below shows what R produced prior to version 4.0.0, along with a *swatch* of color circles.

```
palette()
#> [1] "black"    "red"      "green3"   "blue"     "cyan"     "magenta" "yellow"
#> [8] "gray"
```

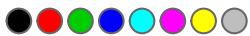


Figure 2 depicts this old default `palette()` (top-left) along with other old base R palettes using swatches of circles or rectangles that are filled with the corresponding colors. The other palette functions all take an argument *n* to generate that number of colors (possibly along with further arguments that allow for certain customizations):

- `heat.colors()`, `terrain.colors()`, `topo.colors()`, and `gray.colors()` can be used as *sequential* palettes for ordered or numeric information.
- `cm.colors()` can be used as a *diverging* palette for values that are distributed around a “neutral” value, such as zero.
- `rainbow()` implements the infamous rainbow (or “jet”) palette that was widely used (possibly with restrictions of the hue range) for all types of variables: *categorical*, *sequential*, and *diverging*.

All of these palettes – except `gray.colors()` – have poor perceptual properties. The colors are highly saturated, which can be distracting and overly stimulating, and the colors are unbalanced with respect to chroma and luminance, which means that they have unequal visual impact (Lonsdale and Lonsdale 2019; Bartram, Patra, and Stone 2017; Etchebehere and Fedorovskaya 2017). In addition, the palettes do not perform well for viewers with some form of colorblindness (about 10% of the male population, Ware 2012). Most of the palettes also use sequences of hues obtained in the RGB (red-green-blue) space or simple derivations thereof like HSV (hue-saturation-value) or HLS (hue-lightness-saturation), which leads to clustering of colors at the red, green, and blue primaries.

Although these limitations have been well known for some time, no changes were made to these palettes provided by the core R graphics system for a number of years. There were various reasons for this including the following:

- In R version 2.1.0, Thomas Lumley added the `colorRampPalette()` function. This made it easier to generate a palette, though the user is still required to select, for example, start and end colors from which a palette of colors can then be interpolated.
- Better palettes became available via packages on CRAN (Comprehensive R Archive Network) starting with `RColorBrewer` (Neuwirth 2022, first published on CRAN in 2002), later `colorspace` (Ihaka 2003; Zeileis, Hornik, and Murrell 2009), and more recently `viridis` (Garnier 2023), `rcartocolor` (Nowosad 2023b), `scico` (Pedersen and Cramer 2023), and `cols4all` (Tennekes 2023),

among many others. In most cases, these make palettes available in R that were developed elsewhere, e.g., [ColorBrewer.org](#) (Harrouer and Brewer 2003), [CARTOCOLORS](#) (CARTO 2023), the [scientific color maps](#) of Crameri, Shephard, and Heron (2020), and the Viridis palettes for [matplotlib](#) in Python (Smith and Van der Walt 2015).

- Higher-level graphics systems like [ggplot2](#) (Wickham 2016) and [lattice](#) (Sarkar 2008) developed their own color themes.

3 A new set of R palettes

On the road to R version 4.0.0 an attempt was made to address the limited and deficient set of palettes in base R and to add a range of modern color palettes. In particular, `palette()` has a new improved default color palette, `palette.colors()` provides further well-established qualitative palettes (Zeileis et al. 2019), and `hcl.colors()` provides a wide range of qualitative, sequential, and diverging palettes obtained by a standardized approach in the so-called HCL (hue-chroma-luminance) space (Wikipedia 2023); see Zeileis and Murrell (2019) and Zeileis et al. (2020).

3.1 A new default color `palette()`

The default color palette in R – the default set of colors that can be specified by integer index – has been replaced. The new palette follows the same basic hues as the old default palette, but the palette is less saturated overall and reduces the size of changes in chroma and luminance across the palette. This produces a calmer and less distracting palette with a more even visual impact. An attempt has also been made to improve the discriminability of the colors in the default palette for colorblind viewers. The output (and swatches) below show what R produces from version 4.0.0 onwards.

```
palette()
```

```
#> [1] "black"    "#DF536B"  "#61D04F"  "#2297E6"  "#28E2E5"  "#CD0BBC"  "#F5C710"
#> [8] "gray62"
```



3.2 The `palette.colors()` function

The `palette.colors()` function, new in R 4.0.0, provides a way to access several other predefined palettes (see also Figure 7). All of these are *qualitative palettes* so they are appropriate for encoding qualitative (categorical) variables. In other words, these palettes are appropriate for differentiating between groups. By default `palette.colors()` returns the "Okabe-Ito" (Okabe and Ito 2008) palette. This palette was designed to be very robust under color vision deficiencies, so the different colors in this palette should be easily distinguishable for all viewers.

```
palette.colors()
```

```
#> [1] "#000000" "#E69F00" "#56B4E9" "#009E73" "#F0E442" "#0072B2" "#D55E00"
#> [8] "#CC79A7" "#999999"
```



The first argument to `palette.colors()` is a number of colors. Each palette has a fixed number of colors, but we can ask for fewer or, with `recycle = TRUE`, we can get more colors by recycling. For example, the following code just requests the first four colors from the "Okabe-Ito" palette.

```
palette.colors(4)
```

```
#> [1] "#000000" "#E69F00" "#56B4E9" "#009E73"
```



Note that up to R version 4.2.x some `palette.colors()`, including "Okabe-Ito", provide *named* output but starting from 4.3.0 all output is unnamed by default.

The following code requests ten colors from the "Okabe-Ito" palette. That palette only contains nine colors, but because `recycle = TRUE`, a tenth color is provided by recycling the first color (black) from the palette.

```
palette.colors(10, recycle = TRUE)

#> [1] "#000000" "#E69F00" "#56B4E9" "#009E73" "#F0E442" "#0072B2" "#D55E00"
#> [8] "#CC79A7" "#999999" "#000000"
```



The second argument to `palette.colors()` is the palette to select colors from. For example, the following code requests the first four colors from the "R4" palette (the new default in `palette()`).

```
palette.colors(4, palette = "R4")

#> [1] "#000000" "#DF536B" "#61D04F" "#2297E6"
```



3.3 The `hcl.colors()` function

The `hcl.colors()` function was added in R 3.6.0, with the range of supported palettes slowly expanded over time. This function provides access to another range of palettes, including sequential and diverging palettes for representing continuous variables. As with `palette.colors()`, the first argument is a number of colors to generate and the second specifies a palette to generate colors from. The `hcl.pals()` function provides a full list of the available palette names that we can choose from.

```
hcl.colors(8, palette = "Blues 3")

#> [1] "#00366C" "#005893" "#007BC0" "#5E9BD8" "#91BAEB" "#BAD5FA" "#DDECFF"
#> [8] "#F9F9F9"
```



One difference with `hcl.colors()` is that the palette we are selecting colors from is *not* a fixed set of colors. Instead, the palettes in `hcl.colors()` are a path within HCL colorspace. For each dimension – hue, chroma, and luminance – a palette can have a constant value, a monotonic trajectory, or a triangular trajectory. For example, the trajectories for the "Blues 3" palette are shown in Figure 3. The palette is (almost) constant in the hue dimension yielding different shades of (almost) the same blue. The palette is monotonically increasing in the luminance dimension, so the blues vary from very dark to very light. Finally, the palette has a triangular trajectory in the chroma dimension, so the blues are more colorful towards the middle of the palette. The trajectories do not involve exactly straight lines because in some cases a power curve is employed and in other cases the palette has to be adjusted to remain within the range of representable colours – see Zeileis, Hornik, and Murrell (2009) and Ihaka et al. (2023) for more details.

Because the palettes from `hcl.colors()` are based on a continuous path in HCL space, we can select as many colors as we like. For example, the following code generates five colors from the multi-hue sequential palette "YlGnBu" (see also Figure 6) and nine colors from the diverging palette "Purple-Green" (see also Figure 12).

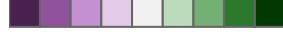
```
hcl.colors(5, palette = "YlGnBu")

#> [1] "#26185F" "#007EB3" "#18BDB0" "#BCE9C5" "#FCFFDD"
```



```
hcl.colors(9, palette = "Purple-Green")
```

```
#> [1] "#492050" "#90529C" "#C490CF" "#E4CAE9" "#F1F1F1" "#BCDABC" "#72B173"
#> [8] "#2C792D" "#023903"
```



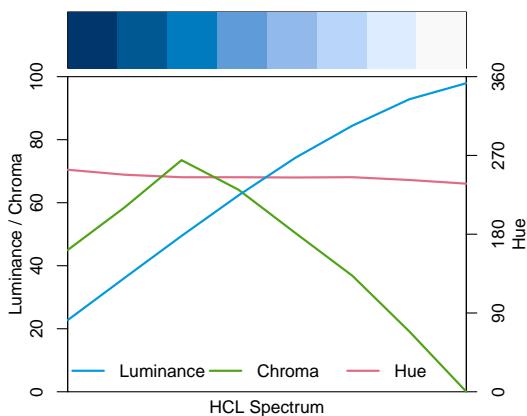


Figure 3: Hue, chroma, and luminance paths for the "Blues 3" palette. This plot is created by the `colorspace::specplot()` function. We can see that hue is held constant in this palette, while luminance increases monotonically and chroma peaks towards the middle of the palette.

3.4 Illustrations

To illustrate the benefits of the new color palettes, Figure 4 shows several versions of a time series plot, depicting four different European stock indexes during most of the 1990s (`EuStockMarkets` data). The plots compare the old "R3" default palette with the new "R4" default and the new qualitative palette "Okabe-Ito". These can all be selected using `palette.colors()`. The first row shows the "R3" default using a typical color legend in the top left corner; the second column shows an emulation of a kind of red-green color blindness known as deutanopia using the `colorspace` package (based on Machado, Oliveira, and Fernandes 2009). The second row uses the "R4" palette and the third row uses "Okabe-Ito"; both with direct labels for the different time series instead of a color legend.

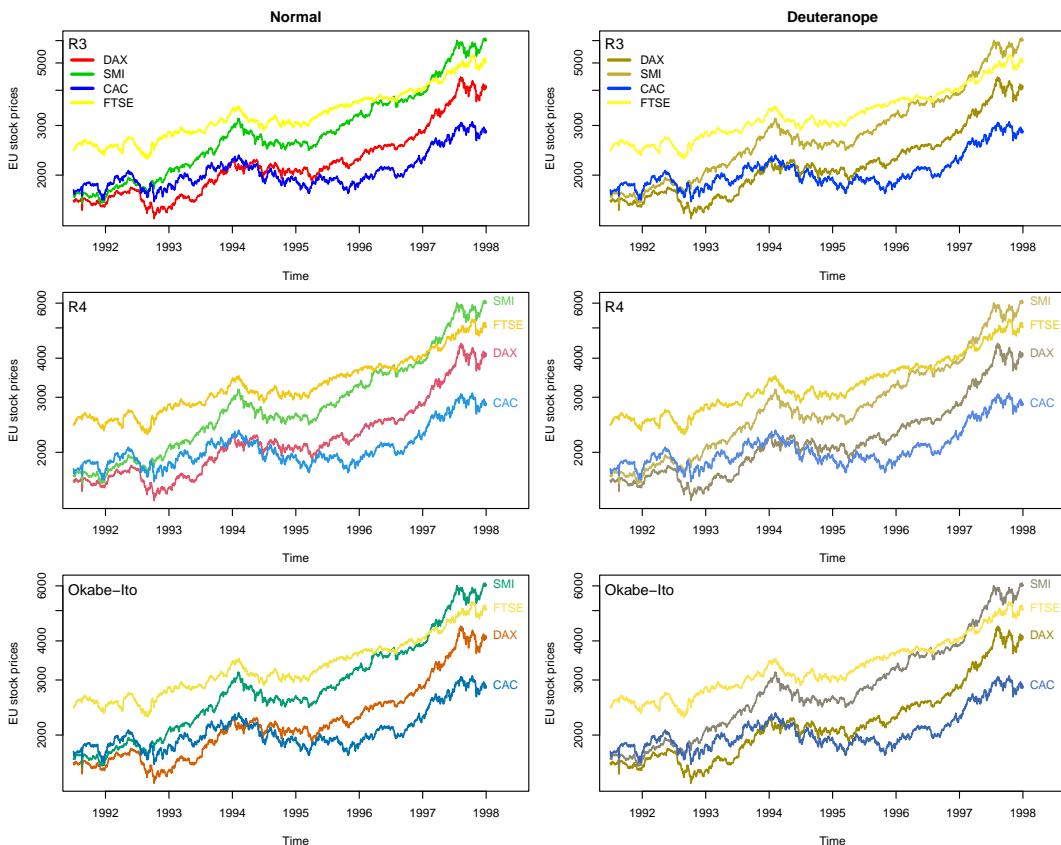


Figure 4: Time series line plot of `EuStockMarkets`. Rows: Old "R3" default palette (top), new "R4" default palette (middle), "OkabeIto" palette (bottom), designed to be robust under color vision deficiencies. Columns: Normal vision (left) and emulated deutanope vision (right). A color legend is used in the first row and direct labels in the other rows.

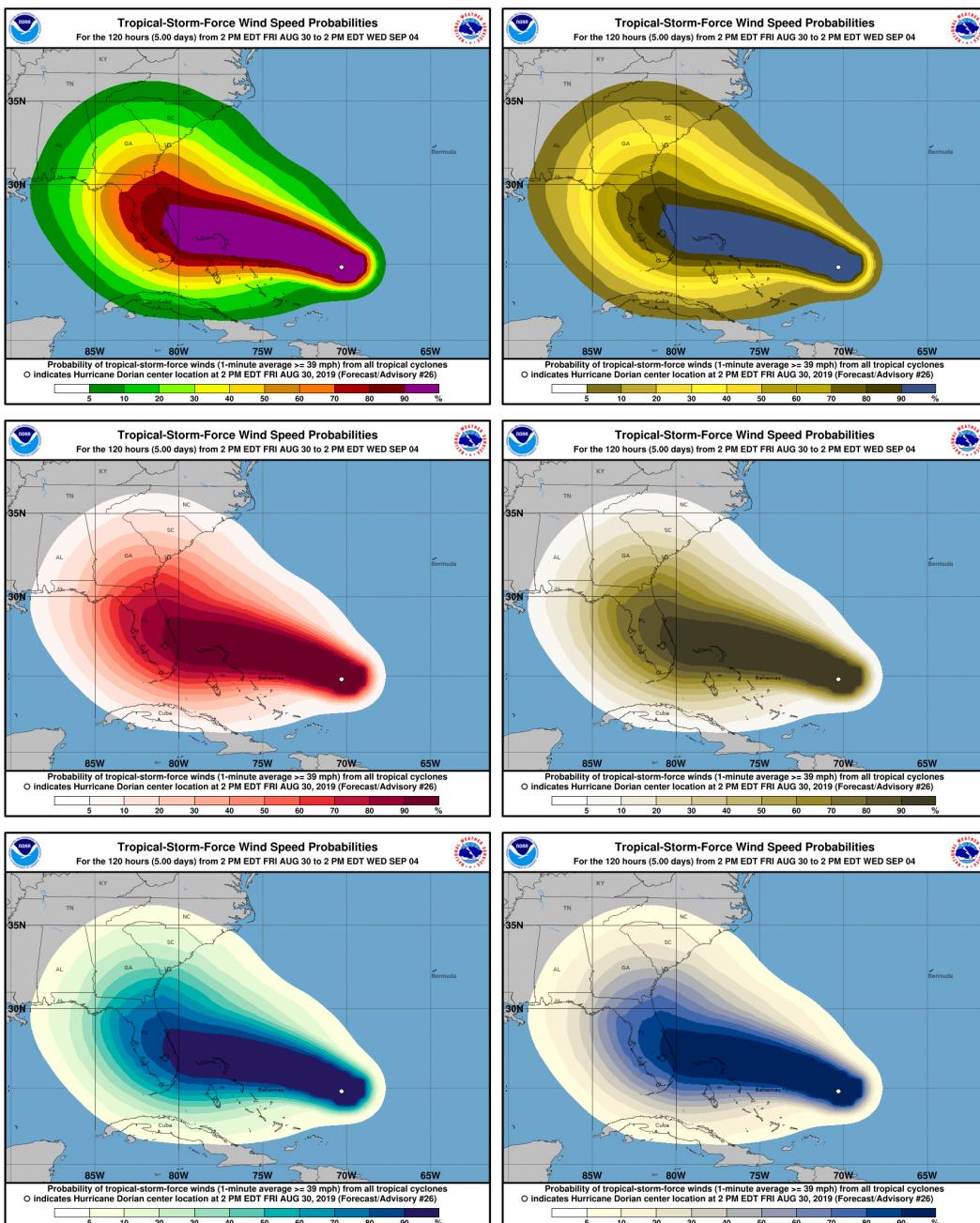


Figure 5: Probability of wind speeds $> 39 \text{ mph}$ (63 km h^{-1}) during hurricane Dorian in 2019. On the left is the original image (top row) and two reproductions using the "Reds" (middle) and "YlGnBu" (bottom) multi-hue sequential palettes. On the right are emulations of how the images on the left might appear to a colorblind viewer.

We can see that the "R3" colors are highly saturated and they vary in luminance. For example, the yellow line is noticeably lighter than the others. Furthermore, for deutanope viewers, the DAX and the SMI lines are difficult to distinguish from each other (exacerbated by the use of a color legend that makes matching the lines to labels almost impossible). Moreover, the FTSE line is more difficult to distinguish from the white background, compared to the other lines.

The "R4" palette is an improvement: the luminance is more even and the colors are less saturated, plus the colors are more distinguishable for deutanope viewers (aided by the use of direct color labels instead of a legend). The "Okabe-Ito" palette works even better, particularly for deutanope viewers.

To illustrate an application of the new sequential color palettes for use with continuous data, Figure 5 shows several versions of a weather map that was produced by the National Oceanic and Atmospheric Administration (and infamously misinterpreted by a former President of The United

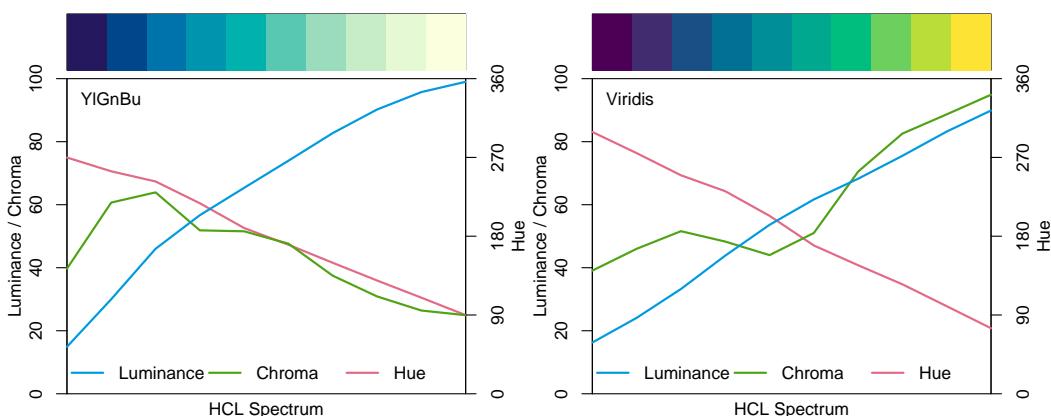


Figure 6: Hue, chroma, and luminance paths for the "YlGnBu" (left) and "Viridis" (right) palettes. These plots are created by the `colorspace::specplot()` function. For "YlGnBu" we can see that hue changes from blue to yellow, luminance increases monotonically, and chroma has a small peak in the blue range and then decreases with luminance. "Viridis", on the other hand, has almost the same trajectory for both hue and luminance, but chroma increases for the light colors.

States, see Zeileis and Stauffer 2019). The top row shows the original image along with an emulation of deutanopia in the second column. The middle row uses the sequential palette "Reds" that can be selected using `hcl.colors()` and the bottom row uses the sequential palette "YlGnBu", which is also available via `hcl.colors()`.

The weather map is intended to convey the probability of wind speeds > 39 mph during hurricane Dorian, 2019-08-30–2019-09-04. The probabilities are highest in the central magenta region and lowest in the outer green regions. The original image does not convey the information very well because there is a non-monotonic change in luminance (from dark to light and back to dark); the high saturation across all of the colors is also distracting. These issues persist for deutanope viewers, plus any benefit of a red (danger!) to green (safe) change in hue is lost.

The "Reds" version of the image conveys the information more clearly by relating the monotonic changes in probability to monotonic changes in luminance. Hue is fairly constant in this palette and the saturation peaks towards the middle, which is similar to the "Blues_3" palette shown in Figure 3, just with a different narrow range of hues. The deutanope version retains this advantage.

The "YlGnBu" version of the image is also more effective than the original. This palette employs a much broader range of hues and varies chroma along with luminances so that the dark colors have higher chroma and the light colors lower chroma (see Figure 6). This still clearly conveys the order from light to dark but additionally yields more distinguishable colors, making it easier to associate contour bands with the legend. Note that the "YlGnBu" palette is similar to the very popular "Viridis" palette (also shown in Figure 6 on the right), with almost the same hue and luminance trajectories. However, an important advantage of the "YlGnBu" palette in this visualization is that the light colors have low chroma and thus signal low risk better than the light colors in the "Viridis" palette which have very high chroma. Finally, we remark that the "YlGnBu" version does lose the benefit of red (danger!) at high probabilities; an alternative would be to use the "Purple-Yellow" multi-hue palette instead, a variation of which was used by Zeileis and Stauffer (2019).

The following sections describe the full range of new color palettes in more detail. A much more condensed overview of the new functions and palettes that are available and some suggestions for robust default palettes are given in Section 6.

4 A gallery of palettes

This section goes through all of the color palettes that are now available in base R (without using any additional packages). There is some discussion of the background for the palettes, strengths and weaknesses of different palettes, and appropriate uses of the palettes.

4.1 The `palette.colors()` function

The `palette.colors()` function provides a range of qualitative palettes (see Figure 7 for an overview). The first argument to the `palette.colors()` function specifies the number of colors to return and the

palette argument allows us to select the palette of colors to choose from. As previously mentioned, the default palette is "Okabe-Ito", which has very good perceptual properties. The "R4" palette specifies the new R default palette which is also returned by palette() by default. As previously mentioned, this was constructed to have reasonable perceptual properties, including accommodation for color vision deficiencies (see Zeileis et al. 2019 for more details). The accompanying palette.pals() function returns a character vector of the available palette names.

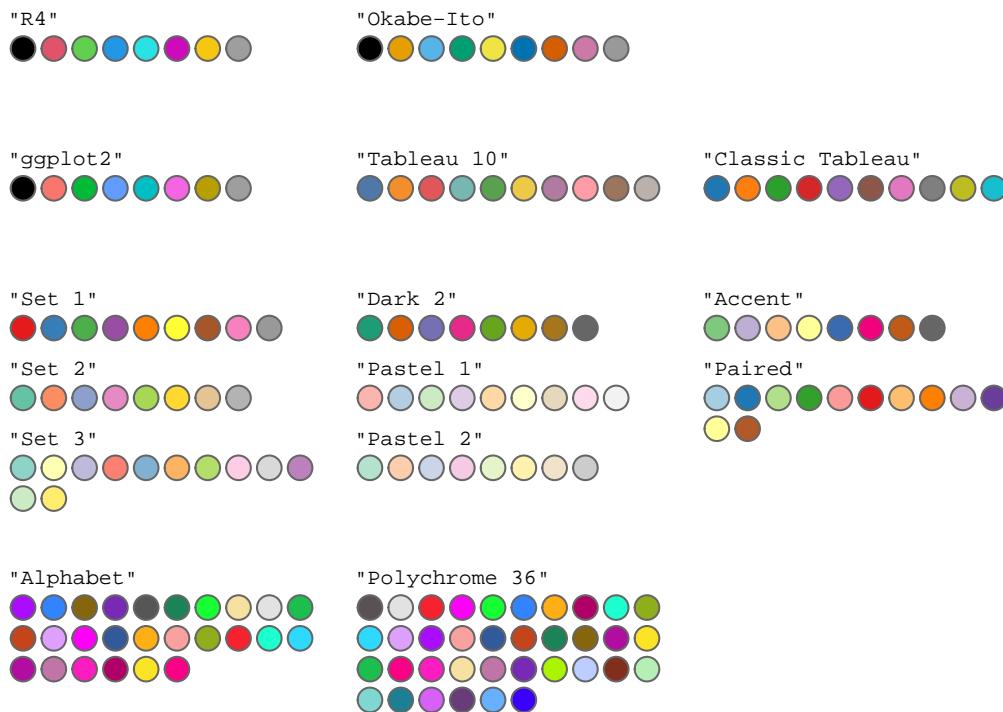


Figure 7: New qualitative palettes in base R available from the palette.colors() function. The label above each swatch shows the argument to provide to palette.colors() to produce the set of colors. The palette at top-left is the new default that is also produced by palette(). The "Okabe-Ito" palette is the default that is produced by palette.colors() (with no arguments).

```
palette.pals()
```

```
#> [1] "R3"          "R4"           "ggplot2"        "Okabe-Ito"
#> [5] "Accent"       "Dark 2"        "Paired"         "Pastel 1"
#> [9] "Pastel 2"     "Set 1"         "Set 2"          "Set 3"
#> [13] "Tableau 10"   "Classic Tableau" "Polychrome 36"  "Alphabet"
```

Each of the predefined palettes can be set as the default palette by passing the palette name to the palette() function. For example, the following code sets the Okabe-Ito palette as the default palette.

```
palette("Okabe-Ito")
```

There are several palettes that have been taken from the color schemes of the **ggplot2** package and Tableau (Tableau Software, LLC 2021), both well-established graphics systems.

The "Tableau 10" palette represents the redesign of the default palette in Tableau (Stone 2016). These palettes may be useful to emulate the look and feel of plots from those other systems.

Several palettes come from ColorBrewer.org (Harrower and Brewer 2003) and were originally designed for filling regions on maps. However, they are also useful for filling regions within data visualizations like bar plots, density plots, and heatmaps, among others. Two of these palettes are a bit different because they deliberately contain darker and lighter colors: the "Accent" palette may be useful to emphasize one or more categories over the others; the "Paired" palette may be useful to represent more than one categorical variable via color, e.g., different types of treatment as well as high vs. low levels of each treatment.

Finally, there are two palettes from the [Polychrome](#) package (Coombes et al. 2019). These are much larger palettes, with colors chosen to be evenly spread throughout HCL colorspace. The "Polychrome_36" palette represents the largest set of colors that could be generated while still being visually distinguishable. The "Alphabet" palette is a smaller, but still large, set (one for each letter of the alphabet). These palettes may be useful if we are attempting to represent a very large number of categories at once. The result is unlikely to be easy to interpret, but these palettes will provide the best chance.

4.2 The `hcl.colors()` function

The `hcl.colors()` function provides qualitative, sequential, and diverging palettes that are derived from certain trajectories of the perceptual properties – hue, chroma, and luminance (HCL). Most of the resulting palettes have one or more desirable perceptual properties:

- **Colorblind-safe:** This means that the palette retains its perceptual properties for colorblind users.
- **Perceptual order:** This means that there is a perceived ordering of the colors, typically arising from a monotonic change from light to dark or vice versa.
- **Perceptual uniformity:** This means that if we take a small step along the path of the palette in HCL space, the perceived difference between the two colors will be the same anywhere along the path.
- **Perceptual balance:** This means that, for example, while there are changes in hue and chroma, luminance remains pretty much the same, so no color stands out from the others.

These properties are very difficult to achieve in a single palette, which is one reason why there are multiple palettes available. Furthermore, different properties will be more or less important depending on the data being displayed and the point that a data visualization is attempting to make. For example, perceptual balance is not desirable when we want to highlight a particular point or category of interest; in that scenario we explicitly want some colors to have a greater visual impact than others. The choice of palette may also depend on how many colors are needed. For example, a palette with a light gray, a medium color, and a full color may still work effectively on a white background if the light gray group is less important and is just provided in the background for reference.

Perceptual order and colorblind-safety are closely linked because the easiest approach to obtaining a colorblind-safe palette is by using a monotonic change in luminance. All of the sequential palettes in `hcl.colors()` in fact have this property and are colorblind-safe to a certain degree, though the effectiveness depends on the range of luminance within the palette. A quick way to check a palette for colorblind-safety is via `colorspace::swatchplot(pal, cvd = TRUE)`, where `pal` is a palette of colors. More elaborate tools are provided by the package [colorblindcheck](#) (Nowosad 2023a).

The [colorspace](#) package also provides functions like `sequential_hcl()` and `diverging_hcl()` to generate palettes by defining a custom set of hue, chroma, and luminance trajectories, e.g., based on specific hues that have inherent meanings for a particular data set.

Qualitative palettes

The qualitative palettes available from `hcl.colors()` are shown in Figure 8. The common feature of these palettes is that they only vary hue while using the same chroma and luminance for all of their colors. One drawback to this approach is that fewer easily distinguishable colors can be generated from these palettes.

The first five palettes are inspired by the ColorBrewer.org palettes of the same name. They employ different fixed levels of chroma and luminance and span the full hue range. Most of these palettes are also available as a fixed set of colors via `palette.colors()`. There are two key differences: First, chroma and luminance are fixed in `hcl.colors()` but typically vary somewhat in `palette.colors()`. The former has the advantage that the colors are more balanced. The latter has the advantage that more sufficiently different colors can be obtained. Second, `hcl.colors()` will return `n` colors interpolated from the full range of hues, whereas `palette.colors()` will return the first `n` colors from a fixed set.

The ColorBrewer.org palettes were designed with good perceptual properties in mind, but also relied on expert opinion and trial and error. This means that a little more care should be taken when selecting one of the ColorBrewer-inspired HCL-based palettes because, for example, they are often not colorblind-safe. The ColorBrewer.org palettes are also available in R via the [RColorBrewer](#) package, which includes a facility for generating ColorBrewer palettes that are colorblind-safe.

The remaining four palettes are taken from Ihaka (2003). These palettes keep chroma and luminance fixed and restrict the range of hues (blues and greens for "Cold" and reds and oranges for "Warm").

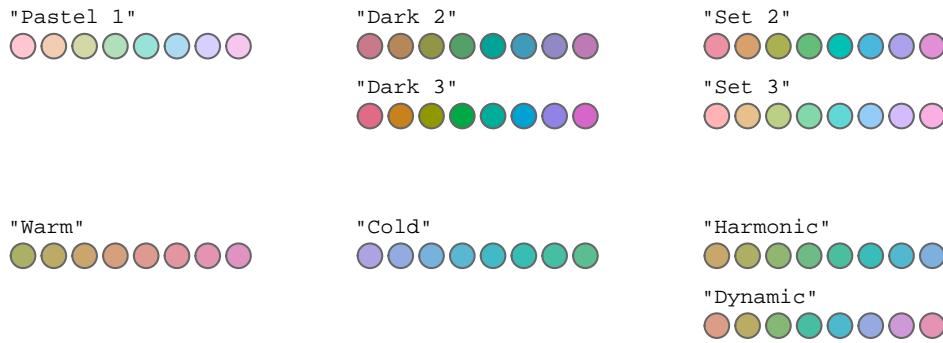


Figure 8: The qualitative palettes that are available with the `hcl.colors()` function.

Holding chroma and luminance fixed means that the visual impact is even across the palette. This makes these palettes appropriate if all categories in a variable have equal importance, but, as with the ColorBrewer.org emulations, they are not colorblind-safe and they will not be appropriate for grayscale printing.

When palettes are employed for shading areas in statistical displays (e.g., in bar plots, pie charts, or regions in maps), lighter colors (with moderate chroma and high luminance) such as "Pastel 1" or "Set 3" are typically less distracting. By contrast, when coloring points or lines, colors with a higher chroma are often required: On a white background a moderate luminance as in "Dark 2" or "Dark 3" usually works better while on a black/dark background the luminance should be higher as in "Set 3" for example.

Single-hue sequential palettes

We divide sequential palettes into single-hue (this section) and multi-hue palettes (the next section).

Single-hue sequential palettes vary only from dark/colorful to light/gray, with a constant underlying hue. Figure 3 provides a good example of the hue, chroma, and luminance trajectories for these palettes. Certain hues will be more appropriate for representing data on specific concepts, such as green for "vegetation" and red for "temperature".

Figure 9 shows the sequential palettes that hold hue largely constant. All of these palettes have a large monotonic variation in luminance, typically from dark to light. This is also typically accompanied by a change in chroma from more colorful to less. The result is a palette that makes it very easy to distinguish extreme values. Some palettes also have a pronounced peak of chroma somewhere in the middle, which makes it easier to distinguish moderate values from extreme values (e.g., "Reds 3", "Blues 3", etc.).

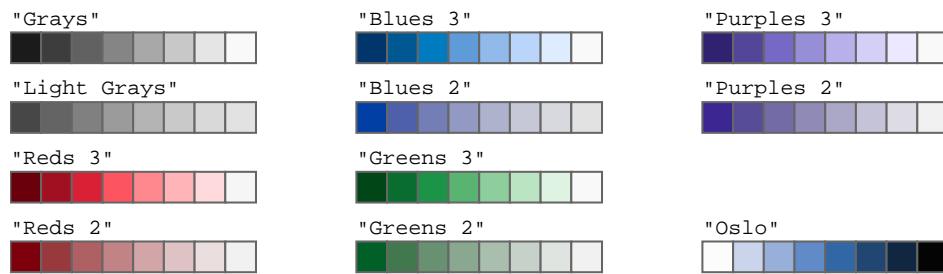


Figure 9: The single-hue sequential palettes that are available with the `hcl.colors()` function.

All palettes in this group, except the last one, are inspired by the ColorBrewer.org palettes with the same base name, but are restricted to a single hue only. They are intended for a white/light background. The last palette, "Oslo", is taken from Cramer's scientific color maps and is intended for a black/dark background and hence the order is reversed starting from a very light blue (almost white).

When only a few colors are needed (e.g., for coding an ordinal categorical variable with few levels) then a lower luminance contrast may suffice (e.g., "Light Grays", "Reds 2", "Blues 2", etc.).

Multi-hue sequential palettes

Multi-hue sequential palettes not only vary luminance, from light to dark (typically along with chroma), but also vary hue. In order to not only bring out extreme colors in a sequential palette but also better distinguish middle colors, it is a common strategy to employ a sequence of hues. This leads to a large range of possible palettes. Figure 6 shows examples of the hue, chroma, and luminance trajectories from multi-hue palettes.

Note that the palettes in this section differ substantially in the amount of chroma and luminance contrasts. For example, many palettes go from a dark high-chroma color to a neutral low-chroma color (e.g., "Reds", "Purples", "Greens", "Blues") or even light gray (e.g., "Purple-Blue"). But some palettes also employ relatively high chroma throughout the palette (e.g., emulations of Viridis and CARTOCColor palettes). The former strategy is suitable to emphasize extreme values, while the latter works better if all values along the sequence should receive the same perceptual weight.

Palettes that involve a significant variation in hue, e.g., "YlGnBu", can be more effective when we need to match specific colors to a legend (e.g., the bottom row of Figure 5) or across several small-multiples, as in faceted plots.

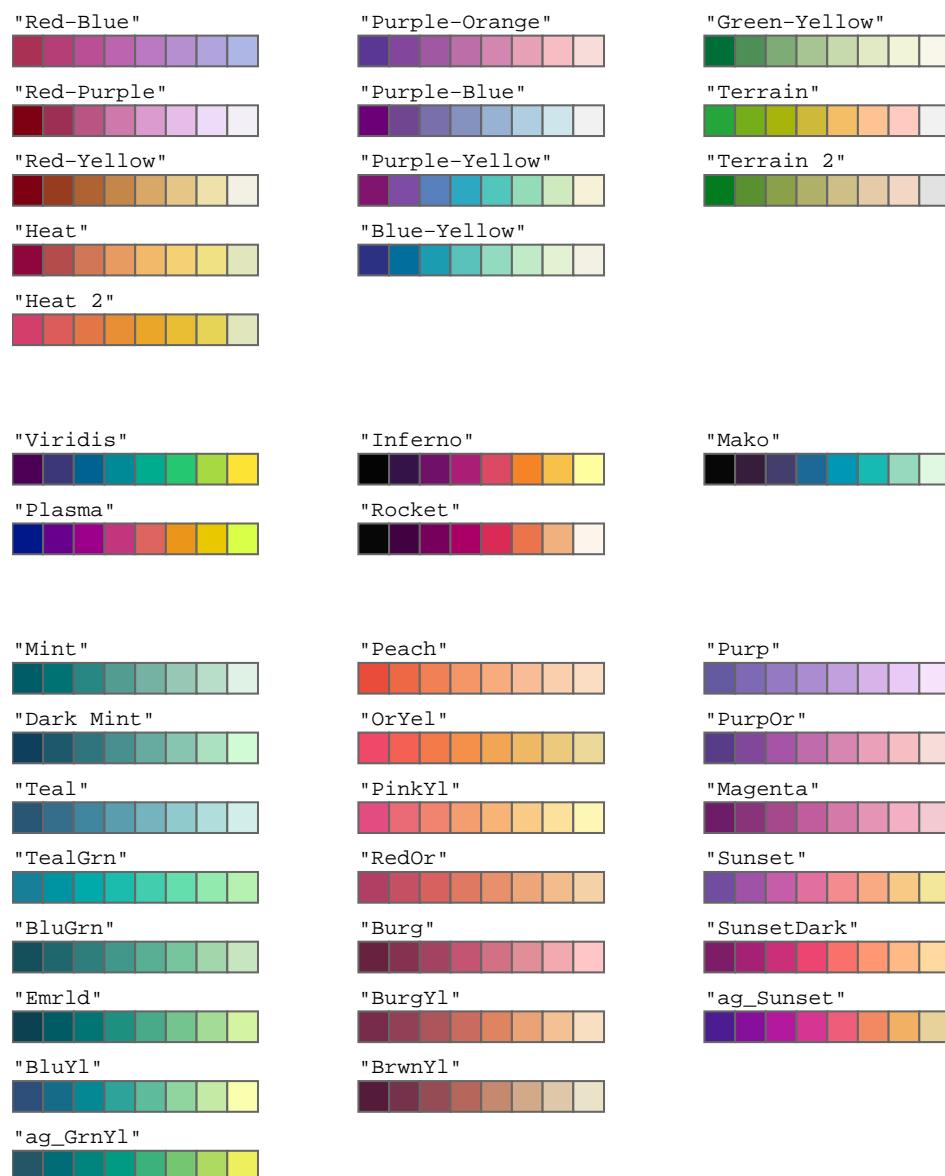


Figure 10: Some of the multi-hue sequential palettes that are available with the `hcl.colors()` function.

Of the palettes shown in Figure 10, "Red-Blue" to "Terrain 2" are palettes created during the development of the `colorspace` package.

The next collection of palettes, "Viridis" to "Mako", emulate popular palettes within the Python

community. The "Viridis", "Plasma", and "Inferno" palettes come from the matplotlib Python library and work well for identifying features of interest in false-color images. This means that they should also work well for heatmaps. The large range of hues means that these palettes can also serve as qualitative palettes, which makes them robust default palettes. However, this versatility means that a palette that is purely sequential or purely qualitative may serve better for a specific purpose.

The "Mako" and "Rocket" palettes are from the seaborn Python library with an emphasis on high chroma and a wide range of luminance. This makes these palettes a good choice for heatmaps.

The remaining palettes in Figure 10, from "Mint" to "Sunset" closely match the corresponding CARTOCOLORS palettes. These palettes tend to span a much narrower range of hues, chroma, and luminance, so can be useful if we just need to represent a small number of ordered values. The resulting colors from these palettes will have, for example, more similar hues than a palette generated from "Viridis", with its wide range of hues.



Figure 11: Some of the multi-hue sequential palettes that are available with the `hcl.colors()` function.

Figure 11 shows the remaining multi-hue sequential palettes that are available in `hcl.colors()`. Most of the top group of palettes, starting with "Reds", "Greens", and "Blues", closely match ColorBrewer.org palettes of the same name. The "YlGnBu" palette is of particular note as it uses essentially the same hues as the "Viridis" palette (Figure 10), but it is more useful as a sequential palette because chroma decreases for the high-luminance colors (see also Figure 6).

The next group of palettes, "Lajolla" to "Batlow" closely match the palettes of the same name from Crameri's scientific color maps. These palettes are constructed with a luminance scale so that there is a clear visual ordering of the palette. They are also designed to be readable by colorblind users, to work for grayscale printing, and to provide perceptual balance, so that no color has a greater visual emphasis than any other. While "Lajolla" and "Turku" are intended for use with a black/dark background, "Hawaii" and "Batlow" are for use with a white/light background. Moreover, the latter two span a particularly large range of hues, thus yielding a kind of "scientific rainbow".

Diverging palettes

The diverging palettes offer a range of underlying hues for either extreme, with either light gray or yellow as the central "neutral" value. The palettes with yellow at the centre provide less of a change in colorfulness, so the "neutral" value is more of a turning point rather than a local minimum.

Figure 13 shows the selection of diverging palettes for use with `hcl.colors()`.

All of these palettes are "balanced" in the sense that chroma and luminance vary in the same way as we move from the central neutral color towards either end of the palette. Figure 12 (left) shows this idea of balance for the "Purple-Green" palette.

When choosing a particular palette for a display similar considerations apply as for the sequential

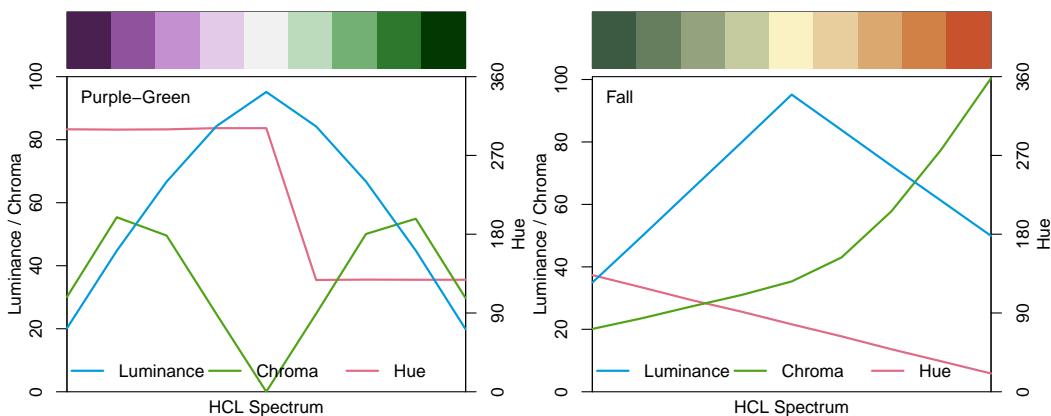


Figure 12: Hue, chroma, and luminance paths for the "Purple-Green" (left) and "Fall" (right) palettes. The plots are created by the `colorspace::specplot()` function. We can see that the "Purple-Green" palette is "balanced" with luminance and chroma varying symmetrically about the central neutral color for both hues. In contrast, the "Fall" palette is "unbalanced" with the left arm of the palette having somewhat darker colors with far less chroma than the right arm. Hue changes gradually from green through yellow to red, yielding a warmer palette compared to "Purple-Green".

palettes. For example, large luminance differences are important when many colors are used while smaller luminance contrasts may suffice for palettes with fewer colors.



Figure 13: The balanced diverging palettes that are available with the `hcl.colors()` function.

Almost all of the palettes in the first two groups, those involving simple color pairs like "Blue-Red" or "Cyan-Magenta", were developed as part of the `colorspace` package, taking inspiration from various other palettes, including more balanced and simplified versions of several ColorBrewer.org palettes. The exception is the "Tropic" palette, which closely matches the palette of the same name from CARTOCOLORS.

The palettes "Broc" to "Vik" and "Berlin" to "Tofino" closely match Crameri's scientific color maps of the same name, where the first three are intended for a white/light background and the other three for a black/dark background.

Flexible diverging palettes

Figure 14 shows a set of more flexible diverging palettes. These do not impose any restrictions that the two "arms" of the palette need to be balanced and also may go through a non-gray neutral color (typically light yellow). Consequently, these palettes may be used to provide a larger set of distinguishable colors compared to the diverging palettes from the previous section. The price of this

flexibility is that the chroma/luminance within these palettes can be rather unbalanced. For example, Figure 12 (right) demonstrates this feature of the "Fall" palette.

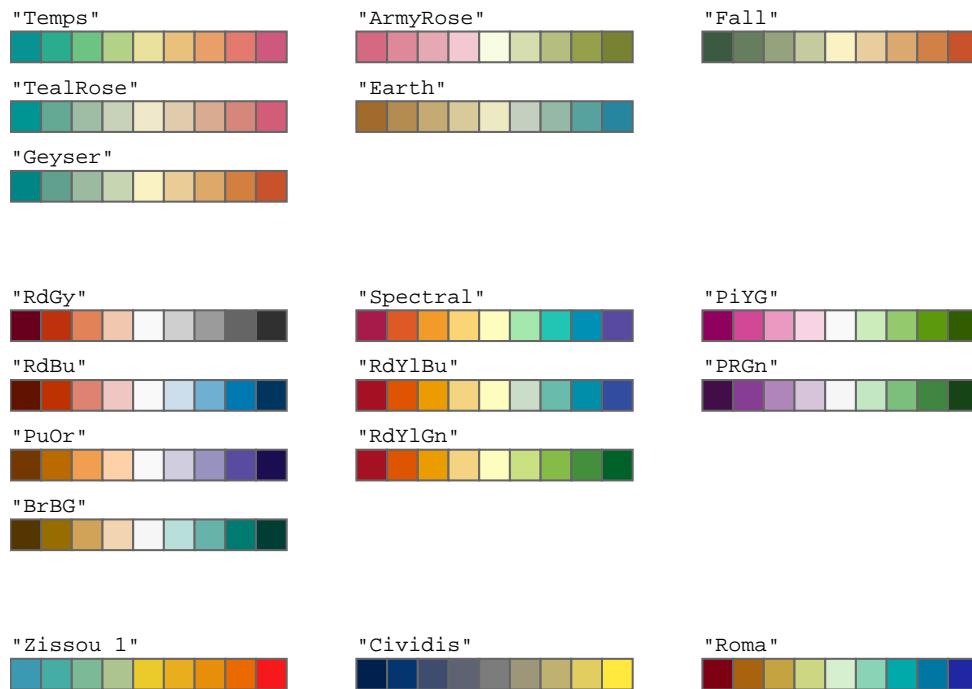


Figure 14: The flexible diverging palettes that are available with the `hcl.colors()` function.

The first group of palettes, including "ArmyRose" and "Temps" closely match the palettes of the same name from CARTOCOLORS.

The next group, based on two or three hues, like "PuOr" and "RdYlGn" closely match the palettes of the same name from ColorBrewer.org.

The final group contains "Zissou 1", which closely matches the palette of the same name from the `wesanderson` package (Ram and Wickham 2023), "Cividis", which is an even more colorblind-safe version of "Viridis" (Nuñez, Anderton, and Renslow 2018) and "Roma", which closely matches the palette of the same name from Crameri's scientific color maps.

5 New defaults in graphical functions

The new default color palette will be most visible in the output from functions in the `grDevices` and `graphics` packages. Several functions from these packages now have slightly different default output, namely when they are using integer color specifications such as 2 or 3. The resulting colors will still be similar to the old output, e.g., still a red or a green, but just a different shade.

Moreover, a couple of functions explicitly have new defaults: `image()` and `filled.contour()`, now use the sequential "YlOrRd" palette (from ColorBrewer) which uses similar hues as the old `heat.colors()`. See the left panel in Figure 15.

Finally, the `hist()` and `boxplot()` functions (and therefore formula-based calls of the form `plot(num ~ factor, ...)`, also have a new default color: light gray which makes it easier to compare the shaded areas (see the middle and right panels in Figure 15).

```
image(volcano)
boxplot(weight ~ feed, data = chickwts)
hist(chickwts$weight)
```

Package authors may also benefit from the new palettes available in R; the new functions `palette.colors()` and `hcl.colors()` allow good default palettes to be set without requiring additional package dependencies. For example, the `lattice` package has already changed its default colors to use the "Okabe-Ito" and "YlGnBu" palettes (for categorical and numerical data, respectively).

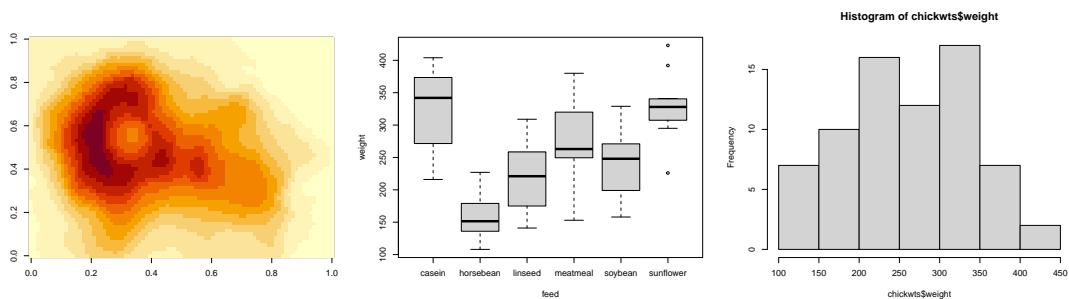


Figure 15: Examples of the new default color palettes that are used in the base graphics functions `image()`, `boxplot()`, and `hist()`.

6 Summary

The default color palette in R has been improved since R version 4.0.0. The functions `palette.colors()` and `hcl.colors()`, from the `grDevices` package, also provide a wide range of predefined palettes based on a number of widely used graphics systems. There are qualitative palettes for use with categorical data and sequential and diverging palettes for use with ordinal or continuous data. The table below summarizes the main types of palettes and provides suggestions for good default palettes for each type. We encourage package authors to make use of these palettes when providing default colors for functions that produce plots.

Table 1: An overview of the new palette functionality: For each main type of palette, the *Purpose* row describes what sort of data the type of palette is appropriate for, the *Generate* row gives the functions that can be used to generate palettes of that type, the *List* row names the functions that can be used to list available palettes, and the *Robust* row identifies two or three good default palettes of that type.

	Qualitative	Sequential	Diverging
<i>Purpose</i>	Categorical data	Ordered or numeric data (high → low)	Ordered or numeric data with a central value (high ← neutral → low)
<i>Generate</i>	<code>palette.colors()</code> , <code>hcl.colors()</code>	<code>hcl.colors()</code>	<code>hcl.colors()</code>
<i>List</i>	<code>palette.pals()</code> , <code>hcl.pals("qualitative")</code>	<code>hcl.pals("sequential")</code>	<code>hcl.pals("diverging")</code> , <code>hcl.pals("divergingx")</code>
<i>Robust</i>	"Okabe-Ito", "R4"	"Blues 3", "YlGnBu", "Viridis"	"Purple-Green", "Blue-Red 3"

References

- Bartram, Lyn, Abhisekh Patra, and Maureen Stone. 2017. “Affective Color in Visualization.” In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 1364–74. New York: Association for Computing Machinery. <https://doi.org/10.1145/3025453.3026041>.
- CARTO. 2023. “CARTOCOLORS: Data-Driven Color Schemes.” <https://carto.com/carto-colors/>.
- Cleveland, William S., and Robert McGill. 1984. “Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods.” *Journal of the American Statistical Association* 79 (387): 531–54. <https://doi.org/10.1080/01621459.1984.10478080>.
- Coombes, Kevin R., Guy Brock, Zachary B. Abrams, and Lynne V. Abruzzo. 2019. “Polychrome: Creating and Assessing Qualitative Palettes with Many Colors.” *Journal of Statistical Software, Code Snippets* 90 (1): 1–23. <https://doi.org/10.18637/jss.v090.c01>.
- Cramer, Fabio, Grace E. Shephard, and Philip J. Heron. 2020. “The Misuse of Colour in Science Communication.” *Nature Communications* 11 (1): 5444. <https://doi.org/10.1038/s41467-020-19160-7>.
- Etchebehere, Sergio, and Elena Fedorovskaya. 2017. “On the Role of Color in Visual Saliency.” *Electronic Imaging* 2017 (14): 58–63. <https://doi.org/10.2352/ISSN.2470-1173.2017.14.HVEI-119>.
- Garnier, Simon. 2023. *viridis: Default Color Maps from matplotlib*. <https://CRAN.R-project.org/package=viridis>.

- Harrower, Mark A., and Cynthia A. Brewer. 2003. "ColorBrewer.org: An Online Tool for Selecting Color Schemes for Maps." *The Cartographic Journal* 40: 27–37. <https://ColorBrewer.org/>.
- Ihaka, Ross. 2003. "Colour for Presentation Graphics." In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing, Vienna, Austria*, edited by Kurt Hornik, Friedrich Leisch, and Achim Zeileis. <https://www.R-project.org/conferences/DSC-2003/Proceedings/>.
- Ihaka, Ross, Paul Murrell, Kurt Hornik, Jason C. Fisher, Reto Stauffer, Claus O. Wilke, Claire D. McWhite, and Achim Zeileis. 2023. *colorspace: A Toolbox for Manipulating and Assessing Colors and Palettes*. <https://CRAN.R-project.org/package=colorspace>.
- Lonsdale, Maria Dos Santos, and David Lonsdale. 2019. "Design2Inform: Information Visualisation." Report. The Office of the Chief Scientific Advisor, Gov UK. <https://eprints.whiterose.ac.uk/141931/>.
- Machado, Gustavo M., Manuel M. Oliveira, and Leandro A. F. Fernandes. 2009. "A Physiologically-Based Model for Simulation of Color Vision Deficiency." *IEEE Transactions on Visualization and Computer Graphics* 15 (6): 1291–98. <https://doi.org/10.1109/TVCG.2009.113>.
- Neuwirth, Erich. 2022. *RColorBrewer: ColorBrewer Palettes*. <https://CRAN.R-project.org/package=RColorBrewer>.
- Nowosad, Jakub. 2023a. *colorblindcheck: Check Color Palettes for Problems with Color Vision Deficiency*. <https://CRAN.R-project.org/package=colorblindcheck>.
- . 2023b. *rkartocolor: CARTOCOLORS Palettes*. <https://CRAN.R-project.org/package=rkartocolor>.
- Nuñez, Jamie R., Christopher R. Anderton, and Ryan S. Renslow. 2018. "Optimizing Colormaps with Consideration for Color Cision Deficiency to Enable Accurate Interpretation of Scientific Data." *PLOS ONE* 13 (7): 1–14. <https://doi.org/10.1371/journal.pone.0199239>.
- Okabe, Masataka, and Kei Ito. 2008. "Color Universal Design (CUD): How to Make Figures and Presentations That Are Friendly to Colorblind People." <https://jfly.uni-koeln.de/color/>.
- Pedersen, Thomas Lin, and Fabio Crameri. 2023. *scico: Colour Palettes Based on the Scientific Colour-Maps*. <https://CRAN.R-project.org/package=scico>.
- Ram, Karthik, and Hadley Wickham. 2023. *wesanderson: A Wes Anderson Palette Generator*. <https://CRAN.R-project.org/package=wesanderson>.
- Sarkar, Deepayan. 2008. *lattice: Multivariate Data Visualization with R*. New York: Springer-Verlag. <https://lmdvr.R-Forge.R-project.org/>.
- Smith, Nathaniel, and Stéfan Van der Walt. 2015. "A Better Default Colormap for matplotlib." In *SciPy 2015 – Scientific Computing with Python*. Austin. <https://www.youtube.com/watch?v=xAoljeRJ3lU>.
- Stone, Maureen. 2016. "How We Designed the New Color Palettes in Tableau 10." <https://www.tableau.com/about/blog/2016/7/colors-upgrade-tableau-10-56782>.
- Tableau Software, LLC. 2021. "Tableau: Business Intelligence and Analytics Software." <https://www.tableau.com/>.
- Tennekes, Martijn. 2023. *cols4all: Colors for All*. <https://CRAN.R-project.org/package=cols4all>.
- Ware, Colin. 2012. *Information Visualization: Perception for Design*. 3rd ed. Morgan Kaufmann. <https://doi.org/10.1016/C2009-0-62432-6>.
- Wickham, Hadley. 2016. *ggplot2: Elegant Graphics for Data Analysis*. 2nd ed. Springer-Verlag. <https://ggplot2.tidyverse.org/>.
- Wikipedia. 2023. "HCL Color Space — Wikipedia, the Free Encyclopedia." URL https://en.wikipedia.org/wiki/HCL_color_space, accessed 2023-11-20.
- Zeileis, Achim, Jason C. Fisher, Kurt Hornik, Ross Ihaka, Claire D. McWhite, Paul Murrell, Reto Stauffer, and Claus O. Wilke. 2020. "colorspace: A Toolbox for Manipulating and Assessing Colors and Palettes." *Journal of Statistical Software* 96 (1): 1–49. <https://doi.org/10.18637/jss.v096.i01>.
- Zeileis, Achim, Kurt Hornik, and Paul Murrell. 2009. "Escaping RGBland: Selecting Colors for Statistical Graphics." *Computational Statistics & Data Analysis* 53: 3259–70. <https://doi.org/10.1016/j.csda.2008.11.033>.
- Zeileis, Achim, and Paul Murrell. 2019. "HCL-Based Color Palettes in grDevices." <https://blog.R-project.org/2019/04/01/hcl-based-color-palettes-in-grdevices/>.
- Zeileis, Achim, Paul Murrell, Martin Maechler, and Deepayan Sarkar. 2019. "A New palette() for R." <https://blog.R-project.org/2019/11/21/a-new-palette-for-r/>.
- Zeileis, Achim, and Reto Stauffer. 2019. "Was Trump Confused by Rainbow Color Map?" https://www.zeileis.org/news/dorian_rainbow/.

Achim Zeileis
Universität Innsbruck
Department of Statistics
<https://www.zeileis.org/>
ORCID: 0000-0003-0918-3766
Achim.Zeileis@R-project.org

Paul Murrell
University of Auckland
Department of Statistics
<https://www.stat.auckland.ac.nz/~paul/>
ORCID: 0000-0002-3224-8858
paul@stat.auckland.ac.nz

Updates to the R Graphics Engine: One Person's Chart Junk is Another's Chart Treasure

by Paul Murrell

Abstract Starting from R version 4.1.0, the R graphics engine has gained support for gradient fills, pattern fills, clipping paths, masks, compositing operators, and stroked and filled paths. This document provides a basic introduction to each of these new features and demonstrates how to use the new features in R.

1 Introduction

Figure 1 shows a data visualization with a subtle feature on each circle. ¹ The border of each circle in the visualization is semi-transparent and the border of each circle becomes more transparent from left to right. We could argue about whether circles are the best representation of these values and whether the semi-transparent gradient is particularly helpful, but for the purposes of this paper, this image represents a specific example of a general problem: does R graphics allow us to produce the exact final image that we desire, or do the limitations of the R graphics engine force us to use external, manual software like Adobe Illustrator to perform additional modifications? Over the past few years, a number of capabilities have been added to the R graphics engine with the aim of being able to produce the exact final image that we desire entirely within R code.

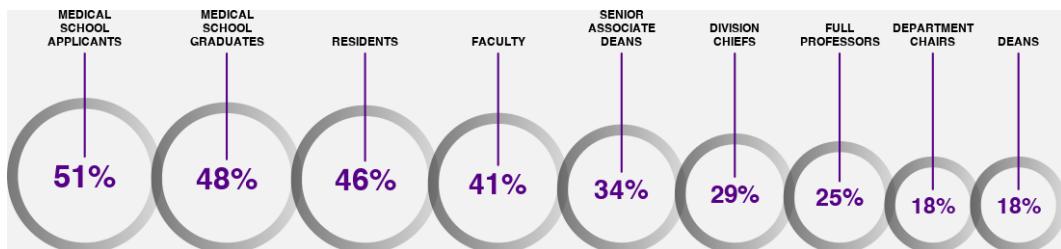


Figure 1: A diagram with a semi-transparent linear gradient on each circle border.

For example, it is now possible to define a linear gradient in R. The following code defines a horizontal linear gradient from a light gray on the left to a very light gray on the right.

```
library(grid)
linearbg <- linearGradient(grey(c(.8, .99)),
                           x1=0, x2=1, y1=.5, y2=.5)
```

The following code uses the `linearbg` gradient defined above to add a subtle linear gradient background to a `ggplot2` plot (Wickham 2016) by setting the `fill` in the `plot.background` (see Figure 2).

```
library(ggplot2)
ggplot(mtcars) +
  geom_point(aes(disp, mpg)) +
  theme(plot.background=element_rect(fill=linearbg, colour=NA),
        plot.margin=unit(c(1, 1, .25, .25), "cm"),
        panel.background=element_rect(fill=NA, colour="black"),
        panel.grid=element_blank())
```

Only some of the new graphics features are available at this stage via high-level interfaces like `ggplot2`, as in the example above, so Section 2 will introduce the new features via a number of lower-level examples using the `grid` package. Section 2 will serve two purposes: to explain what the new features are and to show how they can be used (at least with `grid`). Section 3 will provide some examples of how to integrate the lower-level `grid` usage with higher-level packages like `ggplot2`, `lattice`,

¹This image was inspired by a post on stack overflow.

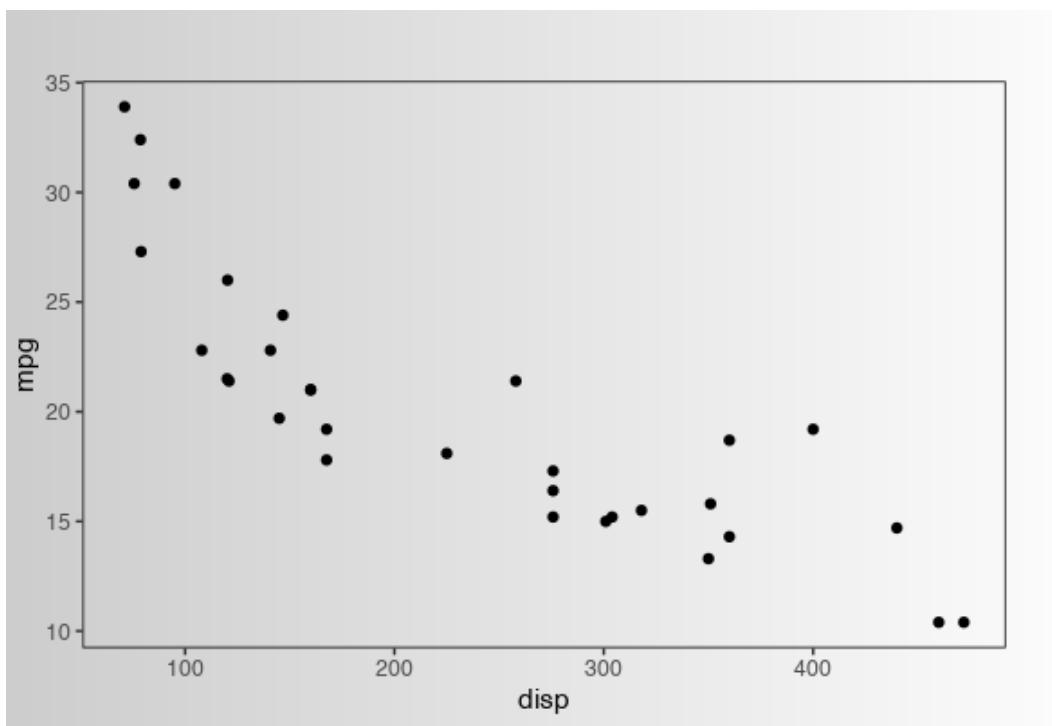


Figure 2: A ggplot2 plot with a subtle linear gradient background.

and base **graphics** plots. Section 4 provides a discussion of important limitations of the new features, how the new features relate to the capabilities of existing R packages, and pointers to further readings. For readers unfamiliar with the organization of R graphics systems, for example, the relationship between **grid** and **ggplot2**, the Appendix provides a brief overview.

2 The new graphics features

In this section, we introduce the new graphics features one by one. The aim is to explain each feature, show what it can do, and show how to use the feature with the **grid** package. Since **grid** will not be familiar for many R users, the examples will be kept as simple as possible and will involve only drawing basic shapes; Section 3 will provide some more complicated examples involving complete plots.

2.1 Gradient fills

There are new functions in the **grid** package for defining gradient fills. For example, we can use the `linearGradient()` function to define a linear gradient based on a start point, an end point, and a set of colors to transition between (along the straight line between the start and end point).

In the following code, we use the `rgb()` function to define two R colors, both blue, but one almost opaque and one almost completely transparent.

```
blues <- rgb(0, 0, 1, alpha=c(.8, .1))
blues
#> [1] "#0000FFCC" "#0000FF1A"
```

Next we define a linear gradient that transitions horizontally, left-to-right, between these two colors. In **grid**, we can specify positions relative to a variety of coordinate systems, but the default is a “normalized” system where 0 means left/bottom and 1 means right/top (and .5 means the center).

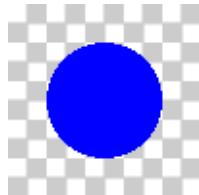
```
gradient <- linearGradient(blues, x1=0, x2=1, y1=.5, y2=.5)
```

The code above does not draw anything; it just defines a linear gradient. In order to draw the gradient, we provide this object as the “fill color” to a function that does draw something. For example,

as we saw in the introduction, we can pass that linear gradient object as the `fill` argument to the `element_rect()` function so that `ggplot2` will fill a rectangle with the gradient.

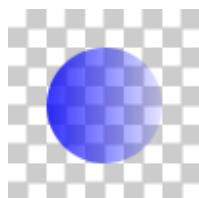
In `grid`, we can draw a shape with a fill color by calling a function like `grid.circle()` and specifying the `fill` argument to the `gpar()` function. For example, the following code draws a circle with a "blue" fill. By default, the circle is drawn in the center of the image and `r=.3` means that the radius of the circle is 30% of the width of the image. All of the images in this section will have a light gray checkerboard pattern in the background so that we can properly see any semi-transparency in what we draw.

```
grid.circle(r=.3, gp=gpar(col=NA, fill="blue"))
```



The following code draws a circle and uses the linear gradient that we defined at the start of this section as the `fill`. Notice that the location `x1=0` in the linear gradient definition refers to the "left" of the circle and `x2=1` refers to the "right" of the circle. In other words, the gradient is relative to the bounding box of the shape that is being filled. There are many other possibilities, including using the `unit()` function to specify the locations of the start and end points of the gradient in absolute units like inches (though still relative to the shape's bounding box) and we can also set the `fill` on a grid viewport rather than on individual shapes. The [Further reading](#) Section provides links to several technical reports that demonstrate a wider range of examples.

```
grid.circle(r=.3, gp=gpar(col=NA, fill=gradient))
```

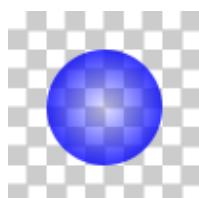


There is also a `radialGradient()` function for defining a gradient that is based on a start circle and an end circle and a set of colors to transition between. For example, the following code defines a radial gradient that starts with radius `r1=.5` and ends with radius `r2=0`, transitioning from an almost opaque blue to almost fully transparent blue.

```
radial <- radialGradient(blues, r1=.5, r2=0)
```

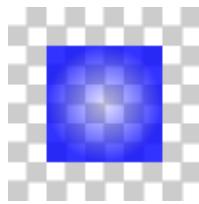
In the following code, the radial gradient is used to fill a circle. The radii of the start and end circles of the gradient are relative to the bounding box of the circle that is being filled, so `r1=.5` corresponds to the circumference of the circle being filled and `r2=0` corresponds to the center of the circle being filled.

```
grid.circle(r=.3, gp=gpar(col=NA, fill=radial))
```



Any shape can be filled with either a linear or radial gradient. For example, the following code fills a rectangle with the radial gradient defined above.

```
grid.rect(width=.6, height=.6, gp=gpar(col=NA, fill=radial))
```



2.2 Pattern fills

It is now also possible to fill a shape with a pattern fill with the help of the `pattern()` function from the `grid` package. The first argument to this function describes what to draw for the pattern and subsequent arguments describe the size of the pattern - the extent of the pattern *tile* - along with what happens outside of the pattern tile. As an example, we will look at the construction of the checkerboard pattern that is being used as the background for the images in this section.

The first step is to describe the shape for the pattern. In this case, we use a call to `rectGrob()` to describe two rectangles, both .1 inches square and filled with light gray, with one rectangle just above and to the right of center and one just below and to the left of center. The `grid` function `unit()` is used to associate a value with a coordinate system. In the "npc" coordinate system, 0 means left/bottom, 1 means right/top, and .5 means the center, and in the "in" coordinate system, values are in inches. The image below the code shows what this shape looks like drawn on its own.

```
bgRect <- rectGrob(x=unit(.5, "npc") + unit(c(0, -.1), "in"),
                     y=unit(.5, "npc") + unit(c(0, -.1), "in"),
                     just=c("left", "bottom"),
                     width=.1, height=.1, default.units="in",
                     gp=gpar(col="gray80", fill="gray80"))
```



The following code creates a pattern based on this shape. The size of the tile is .2" square and the tile will be repeated to fill a region (`extend="repeat"`). The image below the code shows the pattern tile (without repetition); the extent of the tile is indicated by a red rectangle.

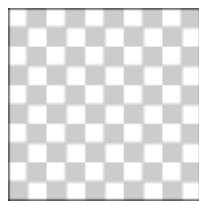
```
bg <- pattern(bgRect,
               width=unit(.2, "in"),
               height=unit(.2, "in"),
               extend="repeat")
```



As with the `linearGradient()` and `radialGradient()` functions, the result of a call to `pattern()` is just an object that describes a pattern; nothing is drawn. We supply this pattern object as the `fill` for a shape that is to be drawn.

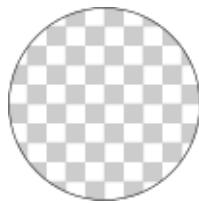
For example, the object that we created with a call to `pattern()` can be used as the `fill` graphical parameter in `grid` functions. The following code draws a rectangle that is the size of the entire image and fills it with the `bg` pattern. The pattern tile is drawn in the center of the rectangle and then repeated to fill the entire rectangle.

```
grid.rect(gp=gpar(fill=bg))
```



As with gradient fills, we can use the pattern to fill any shape. For example, the following code fills a circle with the checkerboard pattern.

```
grid.circle(gp=gpar(fill=bg))
```



2.3 Clipping paths

A clipping region describes an area within which drawing will be visible. By default, everything that we draw within the limits of an image is visible, but we can set a clipping region to limit what gets drawn. *Rectangular* clipping regions have been available in R graphics for a very long time. For example, in scatter plots, the data symbols are usually clipped to the (rectangular) plot region. The code below produces a `ggplot2` plot with the x-axis scale set to exactly the range of the `disp` variable so that the left-most and right-most data symbols are centered on the left and right edge of the plot region. The `ggplot2` package clips the drawing of the data symbols to the plot region so only half of the left-most and right-most data symbols are drawn (see Figure 3).

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg), size=3) +
  scale_x_continuous(expand=c(0, 0)) +
  theme(plot.margin=unit(c(1, 1, .25, .25), "cm"),
        panel.background=element_rect(fill=NA, colour="black"),
        panel.grid=element_blank())
```

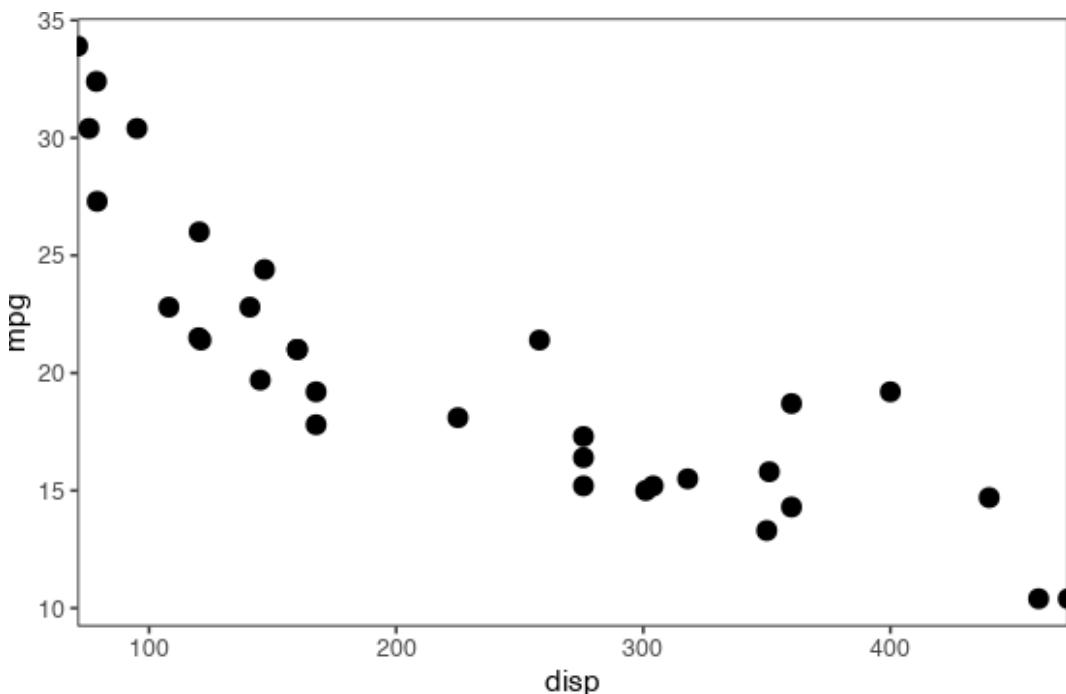
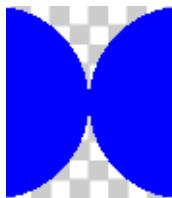


Figure 3: A `ggplot2` plot with the left-most and right-most data symbols clipped to the edges of the plot region.

In `grid`, a clipping region can be defined when we push a *viewport*. A viewport is a rectangular region that defines a temporary coordinate system for drawing, plus some other parameters like the clipping region. For example, the plot region on a `ggplot2` plot is a `grid` viewport that provides coordinates for drawing data symbols, with clipping turned on so that nothing is drawn outside the plot region.

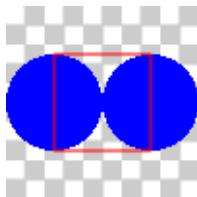
The following code provides a simple demonstration of a viewport. First, we describe two circles, one centered on the left edge of the image and one centered on the right edge of the image. The `circleGrob()` function is similar to the `grid.circle()` function, except that it just creates a description of a circle and does not draw anything. The `grid.draw()` function draws the circles. Because the circles are centered on the edges of the image, we can only see half of each circle.

```
gradient2 <- linearGradient(blues, x1=0, x2=1, y1=.5, y2=.5,
                             group=FALSE)
circles <- circleGrob(x=0:1, r=.5, gp=gpar(col=NA, fill="blue"))
grid.draw(circles)
```



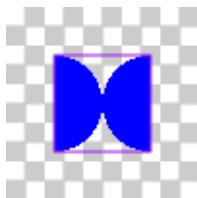
In the following code, the `viewport()` function describes a viewport that is only 50% of the width and 50% of the height of the image. The `pushViewport()` function creates that viewport on the image. We then call the `grid.draw()` function to draw the circles again, but now we are within the viewport, so the circles are drawn centered on the left and right edges of the *viewport*. Furthermore, the radius of the circles (.5) now means 50% of the width of the *viewport*, so the circles are also drawn smaller than before. Notice that the circles extend beyond the edges of the viewport because, by default, there is no clipping region. The `popViewport()` function removes the viewport from the image (so further drawing would be at full size). In the image below, a red rectangle has been drawn just to show the extent of the viewport, but normally nothing is drawn when a viewport is created.

```
pushViewport(viewport(width=.5, height=.5))
grid.draw(circles)
popViewport()
```



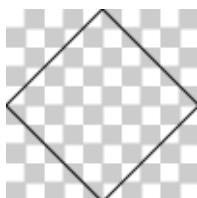
The following code demonstrates the idea of clipping to a viewport. We push a viewport again, but this time the viewport has clipping turned on (`clip=TRUE`). A purple rectangle shows the extent of this viewport. When we draw the circles within this viewport, drawing is limited to the extent of the viewport, so half of the left circle and half of the right circle are clipped.

```
pushViewport(viewport(width=.5, height=.5, clip=TRUE))
grid.draw(circles)
popViewport()
```



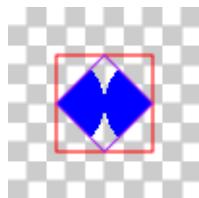
This idea of a clipping region has now been extended to a clipping *path* so that we can limit drawing to a region that is any shape at all, not just a rectangle. For example, the following code describes a diamond shape that we will use as a clipping path. This code does not draw anything, but the shape is shown below the code anyway just so that we know what it looks like.

```
diamondPath <- polygonGrob(c(0, .5, 1, .5), c(.5, 1, .5, 0))
```



The following code pushes a viewport again, but this time sets the clipping region for the viewport to be the diamond shape that we defined above. Now, when we draw the two circles within this viewport, only the parts of the circles that lie within the diamond-shaped clipping path are visible. Just as a guide, a red rectangle has been drawn to show the extent of the viewport that we have pushed, and a purple diamond has been drawn to show the extent of the clipping path. Notice that the location and size of the diamond path is relative to the viewport, so 0 is the left/bottom of the *viewport* and 1 is the right/top of the *viewport*.

```
pushViewport(viewport(width=.5, height=.5, clip=diamondPath))
grid.draw(circles)
popViewport()
```

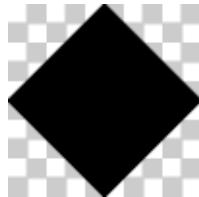


2.4 Masks

Masks are similar to clipping paths in that they constrain what parts of a shape are visible. However, a mask provides greater flexibility because it allows intermediate levels of semi-transparency; a clipping path effectively only allows either fully opaque or fully semi-transparent masking.

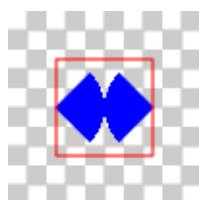
A mask, like a clipping path, can be based on any shape. For example, the following code describes a diamond shape (that is filled black). This code does not draw anything, but the diamond is shown below the code anyway so that we know what it looks like.

```
diamondMask <- polygonGrob(c(0, .5, 1, .5), c(.5, 1, .5, 0),
                             gp=gpar(fill="black"))
```



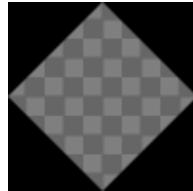
Also like clipping paths, we can enforce a mask when we push a viewport. The following code pushes a viewport that is half the width and half the height of the total image and enforces the mask defined above and then draws the two blue circles. The mask works by transferring its opacity to the circles. Where the mask is opaque, the circles are drawn normally and where the mask is transparent, the circles are not drawn at all. The effect of this mask is exactly the same as the clipping path example from the previous section.

```
pushViewport(viewport(width=.5, height=.5, mask=diamondMask))
grid.draw(circles)
popViewport()
```



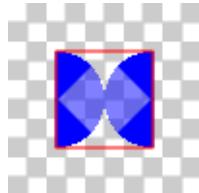
The additional flexibility that is available with masks comes from the fact that the mask may have fully opaque areas and fully transparent areas *and* areas of *semi-transparency*. For example, the following code defines a more complex shape. This shape is a square with opaque corners and a semitransparent diamond in the middle (again, the shape is shown below the code just for reference; the code does no actual drawing). In this code, the `grobTree()` function is used to describe a mask that is made up from two shapes: a `polygonGrob()` that describes the semitransparent diamond and a `pathGrob()` that describes the opaque corners.

```
alphaMask <- grobTree(polygonGrob(c(0, .5, 1, .5), c(.5, 1, .5, 0),
                                    gp=gpar(fill=rgb(0,0,0,.5))),
                      pathGrob(c(0, 0, 1, 1, 0, .5, 1, .5),
                               c(0, 1, 1, 0, .5, 1, .5, 0),
                               id=rep(1:2, each=4),
                               rule="evenodd",
                               gp=gpar(fill="black")))
```



When we push a viewport with this mask, and draw the two circles, the result is the original circles where the mask is opaque and a semitransparent version of the circles where the mask is semitransparent.

```
pushViewport(viewport(width=.5, height=.5, mask=alphaMask))
grid.draw(circles)
popViewport()
```



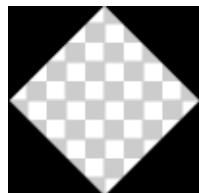
The masks used above are examples of *alpha* masks. They work by transferring the “alpha channel”, or opacity, of the mask to the shape that is being drawn. It is also possible to define *luminance* masks. With a luminance mask, areas of the mask that are white mean that drawing will occur normally, areas that are black mean that no drawing will occur, and areas that are gray produce semi-transparent drawing.

2.5 Stroked and filled paths

Another new feature in R graphics is the ability to stroke and/or fill a *path*, where the path is defined by one or more other shapes. As an example, we will look at producing the “square with opaque corners” from the previous section on masks.

The code below is similar to the code used in the previous section (and the shape that it draws is shown below the code). This code uses the `grid.path()` function and it is relatively complex because it has explicit x and y locations to describe the corners of a square followed by the corners of a diamond and an `id` argument to divide the x and y locations into separate shapes (square and diamond). The `rule="evenodd"` argument means that the interior of the path (that is filled with black) is inside the square, but not inside the diamond.

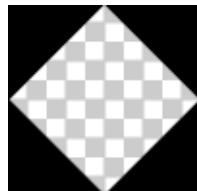
```
grid.path(c(0, 0, 1, 1, 0, .5, 1, .5),
          c(0, 1, 1, 0, .5, 1, .5, 0),
          id=rep(1:2, each=4),
          rule="evenodd",
          gp=gpar(fill="black"))
```



The idea with stroked and filled paths is that we can describe more complex shapes like this by combining simpler shapes instead of having to specify explicit x and y locations. For example, the

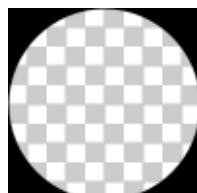
following code draws the same shape using `grid.fillStroke()`, which allows us to describe the shape as a combination of a simple rectangle, with `rectGrob()`, and the `diamondPath` that we defined earlier.

```
grid.fillStroke(grobTree(rectGrob(), diamondPath),
  rule="evenodd",
  gp=gpar(fill="black"))
```



The following code shows a similar example, this time creating a square with rounded corners by combining a rectangle with a circle. This shape would be much harder to describe using `pathGrob()` and explicit x and y locations.

```
grid.fillStroke(grobTree(rectGrob(), circleGrob()),
  rule="evenodd",
  gp=gpar(fill="black"))
```



We can also use stroked and filled paths to stroke the outline of text. Normally, we can only draw filled text in R, but the new `grid.stroke()` function treats its first argument as just a path to draw the outline of. For example, the following code strokes the outline of the text "path".

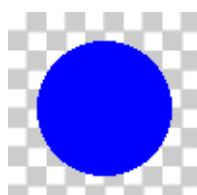
```
grid.stroke(textGrob("path", gp=gpar(cex=2.5)))
```



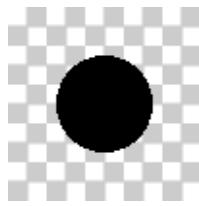
2.6 Compositing operators

Normally when we draw two shapes in R, the second shape is drawn on top of the first shape; if the two shapes overlap, the second shape will obscure the first shape. For example, the following code defines two concentric circles, a larger one with a blue fill and a smaller one with a black fill.

```
circle1 <- circleGrob(r=.35, gp=gpar(col=NA, fill="blue"))
```

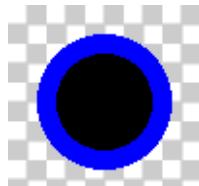


```
circle2 <- circleGrob(r=.25, gp=gpar(col=NA, fill="black"))
```



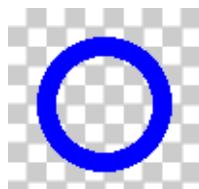
If we draw these circles, one after the other, the black circle partially obscures the blue circle. In technical terms, the black circle is *composited* with the blue circle using an “over” operator.

```
grid.draw(circle1)
grid.draw(circle2)
```



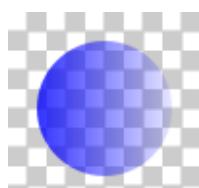
The `grid.group()` function in R allows us to select a different compositing operator. For example, the following code draws the two circles again, but this time using a “`dest.out`” operator, which means that, instead of drawing the black circle on top of the blue circle, the blue circle is removed where it is overlapped by the black circle (and the black circle is not drawn at all).

```
grid.group(circle2, "dest.out", circle1)
```

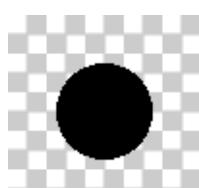


The ability to control the compositing operator provides us with one way to draw a circle with a semi-transparent gradient border that we saw in the very first example in the Introduction. First, we define two circles, one with a semi-transparent gradient fill and one with a black fill.

```
dst <- circleGrob(r=.35, gp=gpar(col=NA, fill=gradient))
```

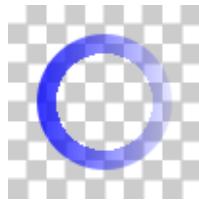


```
src <- circleGrob(r=.25, gp=gpar(col=NA, fill="black"))
```



Then we composite the black circle with the semi-transparent circle, using “`dest.out`”, which creates a hole in the semi-transparent circle. The result is a circle with a semi-transparent gradient border.

```
grid.group(src, "dest.out", dst)
```



The full set of Porter-Duff compositing operators (Porter and Duff 1984) are available along with a set of “blend modes” from the PDF language specification (Adobe Systems Incorporated 2001).

3 Integrating with higher-level interfaces

In this section we look at several more complex examples to show how the new graphics features might be used as part of a high-level plot.

Some features are already accessible via high-level interfaces like `ggplot2` and its extension packages. We saw one example of adding a background linear gradient to a `ggplot2` plot in Figure 2. Furthermore, the following code demonstrates that some `ggplot2` extension packages also already make use of the new graphics features. In this case, we are using the `stat_gradientinterval()` function from the `ggdist` package (Kay 2023) to draw linear gradients that represent within-group variability (see Figure 4).² In theory, any package that extends `ggplot2` can potentially make use of the new features because extending `ggplot2` involves writing `grid` code.

```
library(ggdist)
library(distributional)
ggplot(TGsummary) +
  stat_gradientinterval(aes(x=dose, ydist=dist_normal(len.mean, len.sd))) +
  facet_wrap("supp")
```

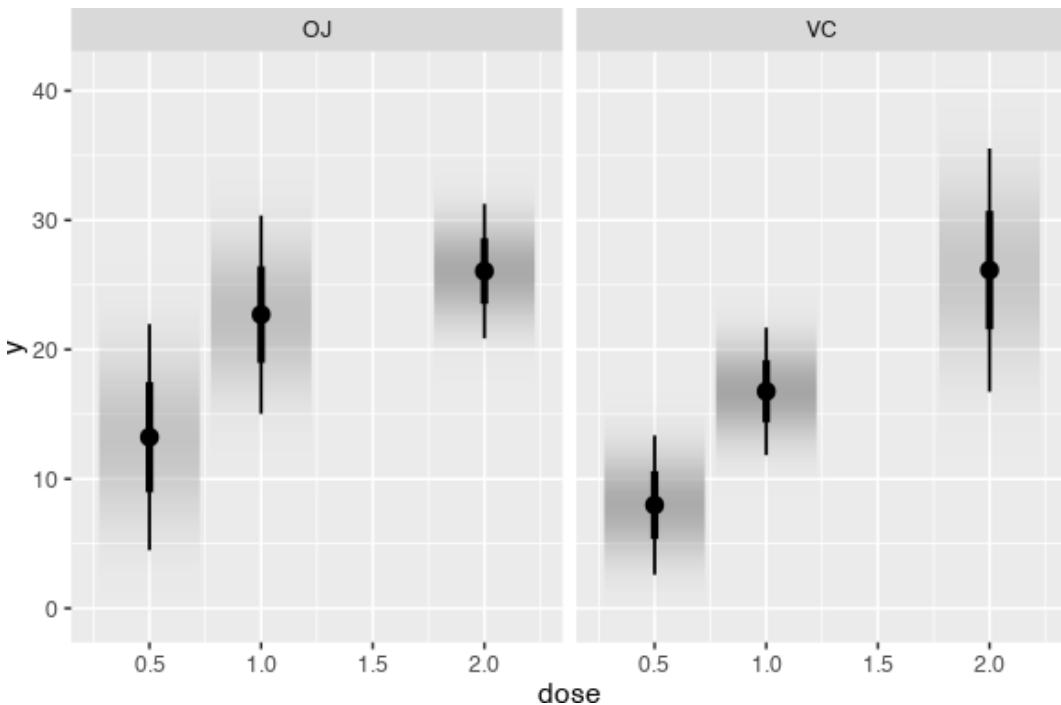


Figure 4: A `ggdist` plot that uses linear gradients to represent variability within groups.

3.1 Integrating with `ggplot2`

Where there is no interface yet for new graphics features in `ggplot2` or its extension packages, it is possible to access the new graphics features using the `gggrid` package (Murrell 2022a). For example,

²Code that generates the data, in this case `TGsummary`, will not be shown in this section, but it is available as part of the supplementary materials for this article.

the following example uses `gggrid` to access the new feature for drawing stroked paths in order to draw outlined text within a `ggplot2` plot.

The first step is to write a function that generates a description of something to draw. The following code first creates a `textGrob()` and then creates a `strokeGrob()` that will draw the outline of the text with a thick white line. The function returns a `grobTree()` that will draw the original text on top of the thick white outline.

```
library(gggrid)
outlineLabel <- function(data, coords) {
  text <- textGrob(paste0(data$percent[2], "%"), gp=gpar(cex=3))
  grobTree(strokeGrob(text, gp=gpar(col="white", lwd=10)),
           text)
}
```

The following code draws a faceted `ggplot2` plot, where each panel consists of an array of yellow and gray tiles.³ The call to `grid_panel()` means that the `outlineLabel()` function that we defined above is called for each panel of the plot, which adds the outlined text to each panel (see Figure 5).

```
ggplot(judges) +
  geom_tile(aes(x, y, fill=race), color="white", lwd=.7) +
  grid_panel(outlineLabel, aes(percent=Freq), data=judgeCounts) +
  facet_wrap("president") +
  scale_fill_manual(values=c(white="#F8C216", other="grey80")) +
  theme_void() +
  theme(aspect.ratio=1,
        legend.position="none")
```

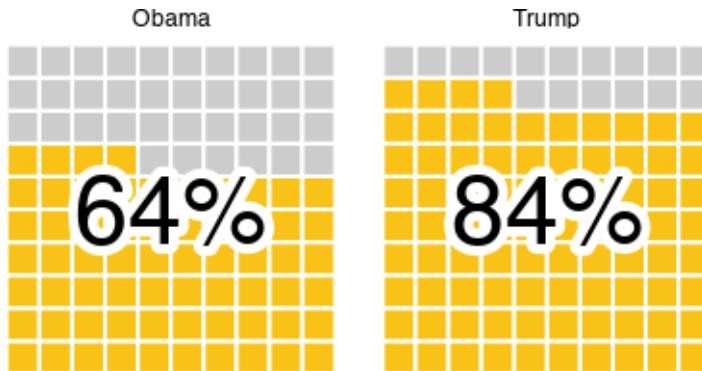


Figure 5: Outlined text on a `ggplot2` plot produced with the `gggrid` package and stroked paths.

This particular result (outlined text) can also be produced using the `shadowtext` package (Yu 2022). However, that package currently produces outlined text by drawing 17 different copies of the text (slightly offset from each other) to get the final result. This is an example where an existing package may be able to gain some efficiencies by adopting some of the new graphics features.

3.2 Integrating with lattice

The `lattice` package (Sarkar 2008) provides an alternative high-level plotting system. For example, the following code uses `lattice` to draw a line plot representing two mathematical functions. The plot has a transparent background and heavy grid lines, which creates a problem where the legend overlaps with the grid lines (see Figure 6).

```
library(lattice)
key <- list(text=list(expression(y1 == x^2, y2 == x^3)),
            lines=list(col=2:3, lwd=3),
            padding.text=3,
            border=TRUE,
            x=.5, y=.75, corner=c(.5, .5))
```

³This example is based on a [stackoverflow post](#), which itself is based on a [post on fivethirtyeight.com](#)

```
xyplot(y1 + y2 ~ x, type="l", lwd=3, col=2:3,
  panel=function(...) {
    panel.grid(h=-1, v=-1, col="black")
    panel.xyplot(...),
  },
  key=key)
```

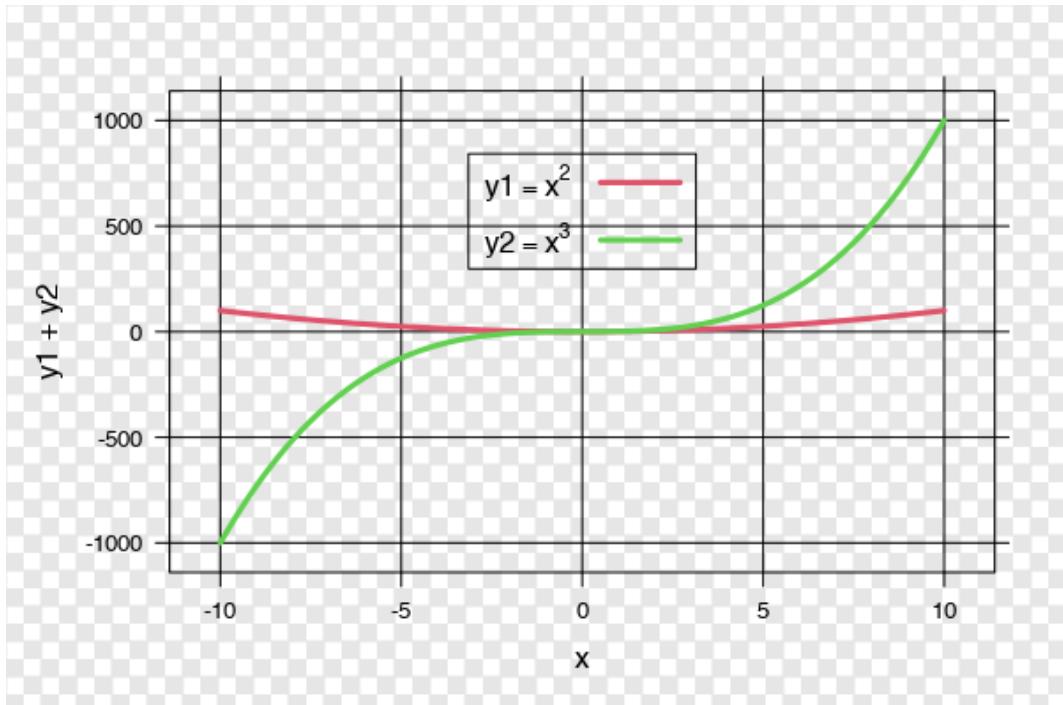


Figure 6: A plot of two mathematical functions drawn with the lattice package. The checkerboard pattern is there to show that the plot itself has a transparent background.

The `lattice` package, like `ggplot2`, is based on `grid`, and we can easily access the `grid` interface to new graphics features by calling `grid` functions from within the `panel` function in the `xyplot()` call. We will use this to access the new masking feature so that the drawing of the grid lines does not overlap with the legend.

The first step is to use the `draw.key()` function from `lattice`, which gives us a `grid` object that describes the legend that `lattice` will draw.

```
keygrob <- draw.key(key=key, vp=viewport(x=.5, y=.75))
```

The following code defines a mask, based on that `keygrob`, that is a black rectangle with a hole where the legend will be drawn. This uses a filled path that is based on a much larger rectangle with the `keygrob` inside it to create the hole.

```
keyMask <- fillGrob(as.path(grobTree(rectGrob(width=2, height=2),
  keygrob),
  rule="evenodd",
  gp=gpar(fill="black")))
```

Now we can draw the `lattice` plot again, but this time we push a `grid` viewport that enforces the `keyMask` and draw the grid lines within that viewport. This means that the grid lines are not drawn where they overlap with the legend (see Figure 7).

```
xyplot(y1 + y2 ~ x, type="l", lwd=3, col=2:3,
  panel=function(...) {
    limits <- current.panel.limits()
    pushViewport(viewport(mask=keyMask,
      xscale=limits$xlim,
```

```

            yscale=limits$ylim))
panel.grid(h=-1, v=-1, col="black")
popViewport()
panel.xyplot(...)

},
key=key)

```

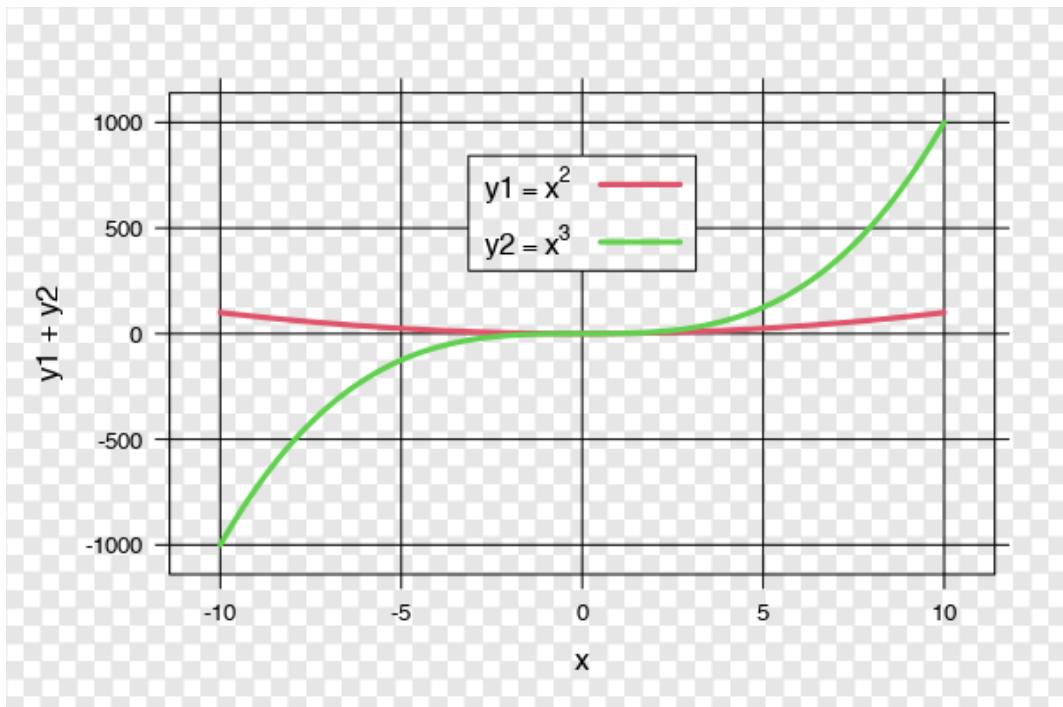


Figure 7: A lattice plot with a mask used to only draw the grid lines where they do not overlap with the legend. The checkerboard pattern is there to show that the plot itself has a transparent background.

3.3 Integrating with base graphics

There is no interface to the new graphics features for the base graphics system (the **graphics** package) or packages that build on it. However, the **gridGraphics** package (Murrell and Wen 2020) can be used to convert base graphics to **grid** and then the **grid** interface can be used to access the new graphics features.

For example, the following code uses base graphics to draw a map with a set of contour regions overlaid (see the map on the left in Figure 8).

```

library(maps)
par(mar=rep(2, 4))
map("nz")
invisible(mapply(function(c, col, fill) {
  polygon(c$x, c$y, default.units="native",
          border=col, col=fill)
},
contours, colours, fills))

```

The following code redraws the map, but uses the new clipping paths feature to clip the contour regions with the map boundary as the clipping path. The first step is to draw the map as before, but this time we also record the coordinates of the map outline and use them to create a clipping path. Next, we use the **grid.echo()** function from the **gridGraphics** package to convert the map into a **grid** drawing, which includes a set of **grid** viewports. We navigate to the viewport where the map was drawn ("graphics-window-1-0A") so that we have the correct coordinate system and then we push a viewport with the clipping path. Finally, we draw the contour regions, which are clipped to the outline of the map (see the map on the right of Figure 8).

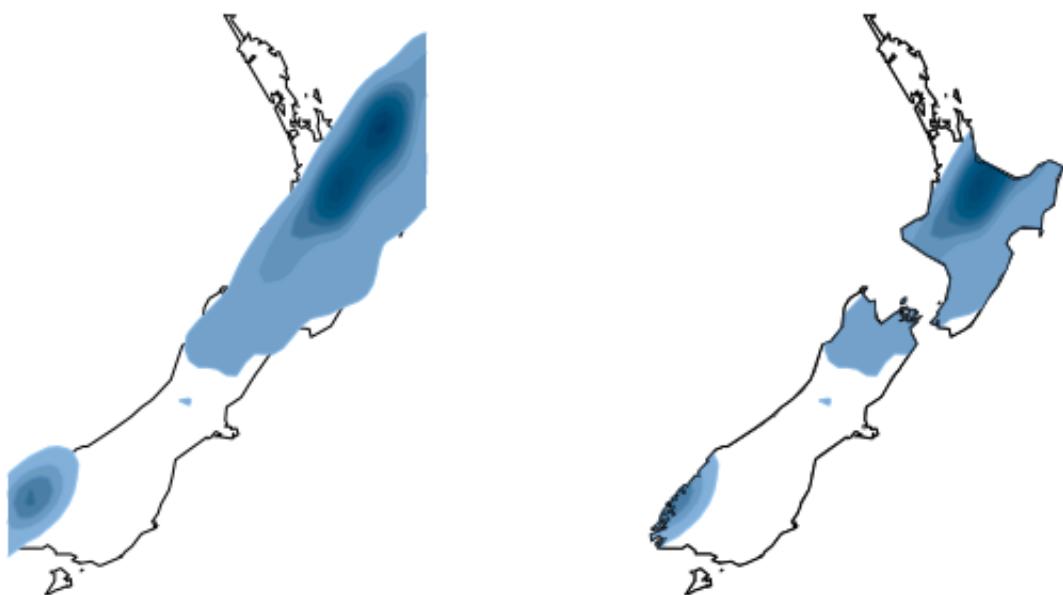


Figure 8: Left: A set of contours drawn on top of a map using base graphics; Right: The same map, converted to grid graphics using the `gridGraphics` package, with the contours clipped by using the boundary of the map as the clipping path.

```

par(mar=rep(2, 4))
outline <- map("nz")
clipPath <- polylineGrob(outline$x, outline$y, default.units="native")
library(gridGraphics)
grid.echo()
downViewport("graphics-window-1-0A")
cvp <- current.viewport()
pushViewport(viewport(xscale=cvp$xscale, yscale=cvp$yscale,
                      clip=clipPath))
invisible(mapply(function(c, col, fill) {
    grid.polygon(c$x, c$y, default.units="native",
                  gp=gpar(col=col, fill=fill))
},
                contours, colours, fills))
grid.draw(clipPath)
popViewport()

```

3.4 Drawing with grid

In some cases, it can be easier to produce a complete plot just using `grid`, rather than using a high-level system like `lattice` or `ggplot2`. With this approach, all of the new graphics features are available. For example, the following code draws the original example from Figure 1 (reproduced again in Figure 9).

First, we define a linear gradient (gray that becomes more transparent from left to right) and then we define a function that draws a circle with that linear gradient as its border (as described in the section on compositing operators).

```

library(grid)
grad <- linearGradient(rgb(0, 0, 0, c(.5, .1)),
                        y1=.5, y2=.5)
gradientCircle <- function(y, r) {
  circle1 <- circleGrob(y=unit(y, "native"),
                         r=unit(r, "native") + unit(1.5, "mm"),
                         gp=gpar(col=NA, fill=grad))
  circle2 <- circleGrob(y=unit(y, "native"),
                         r=unit(r, "native") - unit(1.5, "mm"),
                         gp=gpar(fill="white"))
  grid.group(circle2, "dest.out", circle1)
}

```

The main plot is drawn by dividing up the image into columns, using a `grid.layout()`, with each column the correct width for one of the circles. We then run a loop and, for each circle, we push a viewport into one of the layout columns, call the `gradientCircle()` function to draw a circle, add a text label at the center of the circle, add another label above the circle, and draw a line segment that starts just above the first label and ends just below the second label (see Figure 9).

```
grid.rect(gp=gpar(col=NA, fill="grey95"))
pushViewport(viewport(width=unit(1, "npc") - unit(3, "mm"),
                      y=unit(1.5, "mm"), just="bottom",
                      layout=grid.layout(1, numCircles,
                                         widths=circleWidths, heights=h,
                                         respect=TRUE)))
for (i in 1:numCircles) {
  pushViewport(viewport(layout.pos.col=i, yscale=c(0, h)))
  gradientCircle(circleRadii[i], circleRadii[i])
  amountGrob <- textGrob(paste0(amount[i], "%"),
                         unit(.5, "npc"),
                         unit(circleRadii[i], "native"),
                         gp=gpar(col=purple,
                                  fontface="bold",
                                  cex=1 + amount[i]/max(amount)))
  labelGrob <- textGrob(toupper(label[i]),
                        .5, .7,
                        just="bottom",
                        gp=gpar(fontface="bold", cex=.7, lineheight=1))
  grid.draw(amountGrob)
  grid.draw(labelGrob)
  grid.segments(.5,
                grobY(amountGrob, 90) + unit(2, "mm"),
                .5,
                grobY(labelGrob, 270) - unit(2, "mm"),
                gp=gpar(col=purple, lwd=2))
  popViewport()
}
```

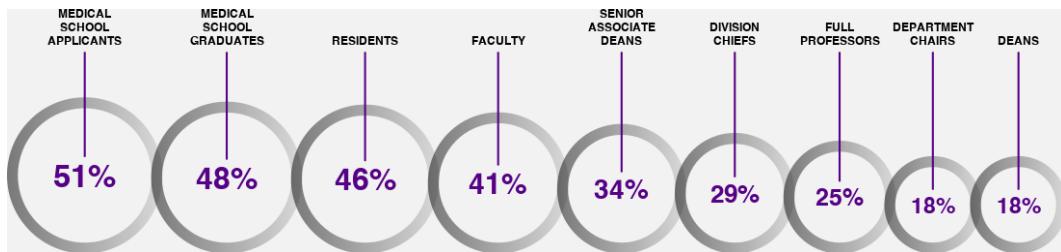


Figure 9: A diagram with a semi-transparent linear gradient on each circle border. This is a reproduction of the original image from the Introduction.

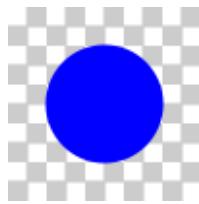
4 Discussion

Since R 4.1.0, support for a number of new graphics features has been added to the R graphics engine. These features are available via functions in the `grid` package, with some higher-level interfaces also appearing, for example in the `ggplot2` package and its extensions.

A small number of examples have been provided to show how the new graphics features might be useful within the context of a statistical plot, but it is hopefully clear that a very large range of other graphical images could be generated with gradients, patterns, clipping paths, masks, compositing operators, and stroked/filled paths.

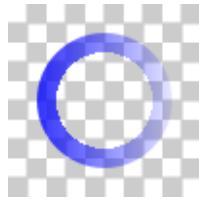
The trick to solving a particular graphical problem is being able to express the problem in terms of the graphical tools at our disposal. If we just want to draw a blue circle, we only have to know how to draw a circle and how to specify the color blue.

```
grid.circle(r=.3, gp=gpar(col="blue", fill="blue"))
```



However, if we want to draw a circle with a thick semi-transparent gradient border, we cannot express a solution in terms of circles and the color blue. We need additional tools, like knowing how to describe a linear gradient and knowing what the "dest.out" compositing operator does, and how to define a group with a specific compositing operator.

```
grid.group(src, "dest.out", dst)
```



By adding more graphical tools, we increase our ability to express problems in terms of our graphical tools and we give ourselves a better chance of finding solutions to graphical problems. As argued at the very beginning of this document, the ability to produce more complex and fine-detailed images, entirely in code, means that we can avoid costly and inefficient manual fine-tuning of our final images.

4.1 Limitations

It is important to note that not all of the new features in the R graphics engine are supported on all graphics devices. Most features are fully supported on Cairo-based graphics devices, e.g., `png(type="cairo")` and `svg()`, though these devices do not support luminance masks. Most features are also supported on the `pdf()` device, including luminance masks, but the Porter-Duff compositing operators are not supported. Most features are supported by the `quartz()` device (on macOS) from R version 4.3.0, though not alpha masks and not paths based on text. The `grDevices::dev.capabilities()` function can be used to report on the level of support in the current graphics device.

Several R packages that provide graphics devices have also added at least some support for these features. For example, the `ragg` package (Pedersen and Shemanarev 2022) and the `svglite` package (Wickham et al. 2022) have support for gradients, patterns, clipping paths, and masks at least.

It is also important to note that some PDF viewers do not render features like masks correctly, so it may appear that the `pdf()` device is not working when in fact it is a problem with the viewing software.

4.2 Related and prior work

Although support for gradients, patterns, clipping paths, and masks has only been added to the R graphics engine (and supported in some graphics devices, with a user interface in `grid`) since R version 4.1.0, at least some of these capabilities have been available in some form via CRAN packages, in some cases for many years. This section, acknowledges some of these packages and discusses the advantages and disadvantages of having the capabilities available in the R graphics engine.

The `gridSVG` package (Murrell and Potter 2022) has provided an interface to many graphical features of the SVG language (including gradients, patterns, clipping paths, and masks) since 2011. Some advantages of this package are that it provides access to features that are still not available in the R graphics engine, including filter effects (such as "blur") and animation. The downsides to this package include the fact that it can only generate SVG output and it can only work with graphics drawn by `grid`, although the `gridGraphics` package can help with that.

The `gridpattern` and the `ggpattern` packages both provide ways to fill regions with a pattern. These packages make use of the new features in the R graphics engine, e.g., to limit a pattern fill to a region by setting a clipping path or mask. In other words, they build upon the new features that are available in R itself. Some of the advantages of these packages include: a set of predefined patterns and, in the case of `ggpattern`, a higher-level interface to pattern fills that is compatible with `ggplot2`.

The `ggfx` package (Pedersen 2022) provides filter effects at both the `grid` and `ggplot2` level. On one hand, this package complements the new features in the R graphics engine by adding filter effects. However, there is also a considerable overlap because several filters provide results similar or identical to the graphics engine features. For example, `ggfx::with_mask()` provides masking and `ggfx::with_blend()` provides compositing operators. The advantages of this package include filter effects that are not available in R itself (such as “blur”) and a high-level interface that is integrated with `ggplot2`. On the other hand, this package works only with raster images, whereas the graphics engine is vector-based so it can produce better quality scalable images, at least on some devices.

The `magick` package (Ooms 2021) is not entirely unlike `ggfx` in that it provides access to raster-based image operations that include, for example, `magick::image_flatten()` for compositing operators, `magick::image_composite()`, which can perform masking, and `magick::image_blur()` for a blur filter effect. A major difference is that `magick` works on entire images whereas `ggfx` can isolate its effects to only specific elements of an image. A combination of `magick` and the `rasterize` package (Murrell 2019) could be used to limit the effects to just elements of an image, but `ggfx` provides a more convenient and higher-level interface.

The `gridGeometry` package (Murrell and Wong 2022) overlaps with some of the new masking, compositing, and path-drawing features of the new graphics engine because it can combine several paths to produce a single new path. For example, `gridGeometry::grid.polyclip()` can be used to punch a hole in one shape using another shape. The advantage of this package is that, for the features that it supports, it works entirely with vector images, including returning vector results.

4.3 Further reading

The graphical features introduced in this document are explained in much greater detail in the series of technical reports listed below. Maintainers of R packages that provide third-party graphics devices will find information about the graphics device interface to the new graphics features in these reports.

- “Catching Up with R Graphics: Gradients, Patterns, Clipping Paths, and Masks” (Murrell 2020)
- “Becoming an R Graphics Groupie: Groups, Compositing Operators, and Affine Transformations in R Graphics” (Murrell 2021a)
- “The Path to Path Enlightenment: Stroking and Filling Paths in R Graphics” (Murrell 2021c)
- “Luminance Masks in R Graphics” (Murrell 2021b)
- “Vectorised Pattern Fills in R Graphics” (Murrell 2022b)
- “Porter-Duff Compositing Operators in R Graphics” (Murrell, Pedersen, and Skintzos 2023)

5 Appendix: The structure of R graphics

This appendix gives a brief introduction to how the (static, 2D) graphics systems in R are organized (see Figure 10).

R has two low-level graphics systems, which are provided by the `graphics` (“base graphics”) and `grid` packages. These provide functions for drawing basic shapes and text. They both rely on a “graphics engine”, roughly corresponding to the `grDevices` package, which provides facilities for specifying things like colors and fonts.

The `graphics` package can also draw some high-level plots, such as scatter plots and bar plots, and many other packages build on it to offer a much wider range of high-level plots. Similarly, several packages build on top of `grid` to provide high-level plots, notably, `ggplot2` and `lattice`.

Graphical output is generated by a set of “graphics devices”, some of which are provided with R, with others provided by packages like `ragg`.

Typically, an R user will call functions from a package like `ggplot2` or `graphics` to draw a high-level plot.

Those packages will call lower-level functions, in `graphics` or `grid`, which in turn call functions in the graphics engine. Finally, the graphics engine send instructions to a graphics device to do the actual drawing.

The new graphics features described in this document have mostly involved changes to the graphics engine and `grid` and some graphics devices, which means that users may have to call `grid` functions to access the new features and the new features will only work on some graphics devices (as indicated by the gray nodes in Figure 10). However, some of the new features are accessible from `ggplot2` and its extensions and some packages that provide graphics devices have added support (as indicated by the nodes with gray gradients in Figure 10).

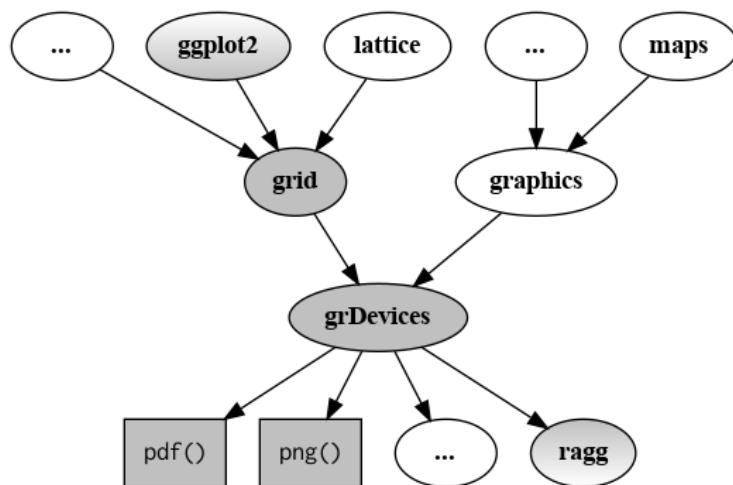


Figure 10: The structure of R graphics. The features described in this paper involved changes to the grid and grDevices packages, plus some of the graphics devices that are provided with R. Some of the features are becoming available via higher-level packages like ggplot2 and some of the features are supported in external graphics devices like those provided by the ragg package.

References

- Adobe Systems Incorporated. 2001. *PDF Reference: Adobe Portable Document Format Version 1.4*. 3rd ed. https://www.adobe.com/content/dam/acom/en/devnet/pdf/pdfs/pdf_reference_archives/PDFReference.pdf.
- Kay, Matthew. 2023. *ggdist: Visualizations of Distributions and Uncertainty*. <https://doi.org/10.5281/zenodo.3879620>.
- Murrell, Paul. 2019. *rasterize: Rasterize Graphical Output*. <https://CRAN.R-project.org/package=rasterize>.
- . 2020. “Catching up with R Graphics.” 2020-04. Department of Statistics, The University of Auckland. <https://doi.org/10.17608/k6.auckland.12649751>.
- . 2021a. “Groups, Compositing Operators, and Affine Transformations in R Graphics.” 2021-02. Department of Statistics, The University of Auckland. <https://doi.org/10.17608/k6.auckland.17009120>.
- . 2021b. “Luminance Masks in R Graphics.” 2021-04. Department of Statistics, The University of Auckland. <https://doi.org/10.17608/k6.auckland.17102657>.
- . 2021c. “ stroking and Filling Paths in R Graphics.” 2021-03. Department of Statistics, The University of Auckland. <https://doi.org/10.17608/k6.auckland.17019272>.
- . 2022a. *gggrid: Draw with 'grid' in 'ggplot2'*. <https://CRAN.R-project.org/package=gggrid>.
- . 2022b. “Vectorised Pattern Fills in R Graphics.” 2022-01. Department of Statistics, The University of Auckland. <https://doi.org/10.17608/k6.auckland.19945787>.
- Murrell, Paul, Thomas Lin Pedersen, and Panagiotis Skintzos. 2023. “Porter-Duff Compositing Operators in R Graphics.” 2023-02. Department of Statistics, The University of Auckland. <https://doi.org/10.17608/k6.auckland.23034641>.
- Murrell, Paul, and Simon Potter. 2022. *gridSVG: Export 'grid' Graphics as SVG*. <https://r-forge.r-project.org/projects/gridsvg/>.
- Murrell, Paul, and Zhijian Wen. 2020. *gridGraphics: Redraw Base Graphics Using 'grid' Graphics*. <https://CRAN.R-project.org/package=gridGraphics>.
- Murrell, Paul, and Jack Wong. 2022. *gridGeometry: Polygon Geometry in 'grid'*. <https://CRAN.R-project.org/package=gridGeometry>.
- Ooms, Jeroen. 2021. *magick: Advanced Graphics and Image-Processing in R*. <https://CRAN.R-project.org/package=magick>.
- Pedersen, Thomas Lin. 2022. *ggfx: Pixel Filters for 'ggplot2' and 'grid'*. <https://CRAN.R-project.org/package=ggfx>.
- Pedersen, Thomas Lin, and Maxim Shemanarev. 2022. *ragg: Graphic Devices Based on AGG*. <https://CRAN.R-project.org/package=ragg>.
- Porter, Thomas, and Tom Duff. 1984. “Compositing Digital Images.” *SIGGRAPH Comput. Graph.* 18 (3): 253–59. <https://doi.org/10.1145/964965.808606>.
- Sarkar, Deepayan. 2008. *Lattice: Multivariate Data Visualization with R*. New York: Springer. <http://lmdvr.r-forge.r-project.org>.
- Wickham, Hadley. 2016. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York.

<https://ggplot2.tidyverse.org>.
Wickham, Hadley, Lionel Henry, Thomas Lin Pedersen, T Jake Luciani, Matthieu Decorde, and Vaudor Lise. 2022. *svglite: An SVG Graphics Device*. <https://CRAN.R-project.org/package=svglite>.
Yu, Guangchuang. 2022. *shadowtext: Shadow Text Grob and Layer*. <https://CRAN.R-project.org/package=shadowtext>.

Paul Murrell
The University of Auckland
Department of Statistics
38 Princes Street
Auckland
New Zealand
<https://www.stat.auckland.ac.nz/~paul/>
ORCID: 0000-0002-3224-8858
paul@stat.auckland.ac.nz

mathml: Translate R Expressions to MathML and LaTeX

by Matthias Gondan and Irene Alfarone

Abstract This R package translates R objects to suitable elements in MathML or LaTeX, thereby allowing for a pretty mathematical representation of R objects and functions in data analyses, scientific reports and interactive web content. In the R Markdown document rendering language, R code and mathematical content already exist side-by-side. The present package enables use of the same R objects for both data analysis and typesetting in documents or web content. This tightens the link between the statistical analysis and its verbal description or symbolic representation, which is another step towards reproducible science. User-defined hooks enable extension of the package by mapping specific variables or functions to new MathML and LaTeX entities. Throughout the paper, examples are given for the functions of the package, and a case study illustrates its use in a scientific report.

1 Introduction

The R extension of the markdown language (Xie, Allaire, and Grolemund 2018; Allaire et al. 2023) enables reproducible statistical reports with nice typesetting in HTML, Microsoft Word, and LaTeX. Moreover, since recently (R Core Team 2022, version 4.2), Rs manual pages include support for mathematical expressions (Sarkar and Hornik 2022; Viechtbauer 2022), which is already a big improvement. However, except for special cases such as regression models (Anderson, Heiss, and Sumners 2023) and R's own plotmath annotation, rules for the mapping of built-in language elements to their mathematical representation are still lacking. So far, R expressions such as `pbinom(k, N, p)` are printed as they are and pretty mathematical formulae such as $P_{Bi}(X \leq k; N, p)$ require explicit LaTeX commands like `P_{\mathit{Bi}}(X \leq k; N, p)`. Except for very basic use cases, these commands are tedious to type and their source code is hard to read.

The present R package defines a set of rules for the automatic translation of R expressions to mathematical output in R Markdown documents (Xie, Dervieux, and Riederer 2020) and Shiny Apps (Chang et al. 2022). The translation is done by an embedded Prolog interpreter that maps nested expressions recursively to MathML and LaTeX/MathJax, respectively. User-defined hooks enable extension of the set of rules, for example, to represent specific R elements by custom mathematical signs.

The main feature of the package is that the same R expressions and equations can be used for both mathematical typesetting and calculations. This saves time and potentially reduces mistakes, as will be illustrated below. Readers should have basic knowledge of knitr and R Markdown to be able to follow this article (Xie 2023; Allaire et al. 2023), while to extend and customize the package, some basic knowledge of Prolog is needed.

The paper is organized as follows. We start with a description of the technical background of the package, including the two main classes of rules for translating R objects to mathematical expressions. The next section illustrates the main features of the `mathml` package, potential issues and their workarounds using examples from the day-to-day perspective of a user. A case study follows with a scientific report written with the help of the package. The last section concludes with a discussion and ideas for further development.

2 Background

Similar to other high-level programming languages, R is homoiconic, that is, R commands (i.e., R “calls”) are, themselves, symbolic data structures that can be created, parsed and modified. Because the default response of the R interpreter is to evaluate a call and return its result, this property is not transparent to the general user. There exists, however, a number of built-in R functions (e.g., `quote()`, `call()` etc.) that allow the user to create R calls which can be stored in regular variables and then, for example, evaluated at a later stage or in a specific environment (Wickham 2019). The present package includes a set of rules that translate such calls to a mathematical representation in MathML and LaTeX. For a first illustration of the `mathml` package, we consider the binomial probability.

```
term <- quote(pbinom(k, N, p))
```

The term is quoted to avoid its immediate evaluation (which would raise an error anyway since

the variables k , N , p have not yet been defined). Experienced R users will recognize that the expression is a short form for

```
term <- call("pbinaryom", as.name("k"), as.name("N"), as.name("p"))
term

#> pbinaryom(k, N, p)
```

As can be seen from the output, to the variable `term` is not assigned the result of the calculation, but instead an R call (see, e.g., Wickham 2019, for details on “non-standard evaluation”), which can eventually be evaluated with `eval()`,

```
k <- 10
N <- 22
p <- 0.4
eval(term)

#> [1] 0.77195
```

The R package `mathml` can now be used to render the call in MathML or in MathJax/LaTeX. MathML is the dialect for mathematical elements on HTML webpages, whereas LaTeX is typically used for typesetting printed documents, as shown below.

```
library(mathml)
substr(mathml(term), 1, 70)

#> [1] "<math><mrow><msub><mi>P</mi><mtext>Bi</mtext></msub><mo>&af; </mo><mrow>

mathjax(term)

#> [1] "${P}_{\mathrm{Bi}}\left(X\leq k;N,p\right)$"
```

Some of the curly braces are not really needed in the LaTeX output, but are necessary in edge cases. The package also includes a function `mathout()` that wraps a call to `mathml()` for HTML output and `mathjax()` for LaTeX output. Moreover, the function `math(x)` adds the class “math” to its argument, such that a special knitr printing function is invoked (see the vignette on custom print methods in Xie 2023). An R Markdown code chunk with `mathout(term)` thus produces:

$$P_{\mathrm{Bi}}(X \leq k; N, p)$$

Similarly, `inline()` produces inline output, r `inline(term)` yields $P_{\mathrm{Bi}}(X \leq k; N, p)$.

3 Package `mathml` in practice

The currently supported R objects are listed below, roughly following the order proposed by Murrell and Ihaka (2000).

3.1 Basic elements

`mathml` handles the basic elements of everyday mathematical expressions, such as integers, floating-point numbers, Latin and Greek letters, multi-letter identifiers, accents, subscripts, and superscripts.

```
term <- quote(1 + -2L + a + abc + "a" + phi + Phi + varphi + roof(b)[i, j]^2L)
math(term)

1.00+-2+a+abc+a+φ+Φ+φ+ $\hat{b}_{ij}^2$ 

term <- quote(round(3.1415, 3L) + NaN + NA + TRUE + FALSE + Inf + (-Inf))
math(term)

3.142+nan+na+T+F+∞+(−∞)
```

An expression such as `1 + -2` may be considered aesthetically unsatisfactory. It is correct R syntax, though, and is reproduced accordingly, without the parentheses. Parentheses around negative numbers or symbols can be added as shown above for `+ (-Inf)`.

To avoid name clashes with package `stats`, `roof()` is used to put a hat on a symbol (see next section for further decorations). Note that an R function `roof()` does not exist in base R, it is provided by the package for convenience and points to the identity function.

3.2 Decorations

The package offers some support for different fonts as well as accents and boxes etc. Internally, these decorations are implemented as identity functions, so that they can be introduced into R expressions without side-effects.

```
term <- quote(bold(b[x, 5L]) + bold(b[italic(x)]) + italic(ab) + italic(42L))
math(term)
```

$$\mathbf{b}_{x5}+\mathbf{b}_x+ab+42$$

```
term <- quote(tilde(a) + mean(X) + box(c) + cancel(d) + phantom(e) + prime(f))
math(term)
```

$$\tilde{a}+\overline{X}+\boxed{c}+\cancel{d}+ +f'$$

Note that the font styles only affect the display of identifiers, whereas numbers, character strings etc. are left untouched.

3.3 Operators and parentheses

Arithmetic operators and parentheses are translated as they are, as illustrated below.

```
term <- quote(a - ((b + c)) - d*e + f*(g + h) + i/j + k^(l + m) + (n)o^{p + q})
math(term)
```

$$a - [(b+c)] - de + f \cdot (g+h) + i/j + k^{(l+m)} + (no)^{p+q}$$

```
term <- quote(dot(a, b) + frac(1L, nodot(c, d + e)) + dfrac(1L, times(g, h)))
math(term)
```

$$a \cdot b + \frac{1}{c(d+e)} + \frac{1}{g \times h}$$

For multiplications involving only numbers and symbols, the multiplication sign is omitted. This heuristic does not always produce the desired result; therefore, `mathml` defines alternative R functions `dot()`, `nodot()`, and `times()`. These functions calculate a product and produce the respective multiplication signs. Similarly, `frac()` and `dfrac()` can be used for small and large fractions.

For standard operators with known precedence, `mathml` is generally able to detect if parentheses are needed; for example, parentheses are automatically placed around `d + e` in the `nodot`-example. However, we note unnecessary parentheses around `l + m` above. These parentheses are a consequence of `quote(a^(b + c))` actually producing a nested R call of the form `^(a, (b + c))` instead of `^(a, b + c)`:

```
term <- quote(a^(b + c))
paste(term)

#> [1] "a"      "(b + c)"
```

For the present purpose, this R feature is unfortunate because the extra parentheses around `b + c` are not needed. The preferred result is obtained by the functional form `quote(`(k, l + m))` of the power, or curly braces as a workaround (see `p + q` above).

3.4 Custom operators

Whereas in standard infix operators, the parentheses typically follow the rules for precedence, undesirable results may be obtained in custom operators.

```
term <- quote(mean(X) %+-% 1.96 * s / sqrt(N))
math(term)
```

$$(\bar{X} \pm 1.96) \cdot s / \sqrt{N}$$

```
term <- quote('%+-%'(mean(X), 1.96 * s / sqrt(N))) # functional form of '%+-%'
term <- quote(mean(X) %+-% {1.96 * s / sqrt(N)}) # the same
math(term)
```

$$\bar{X} \pm 1.96s / \sqrt{N}$$

The example is a reminder that it is not possible to define the precedence of custom operators in R, and that expressions with such operators are evaluated strictly from left to right. Again, the problem can be worked around by the functional form of the operator or a curly brace to hide the parenthesis, and, at the same time, enforce the correct operator precedence.

More operators are shown in Table 1, including the suggestions by Murrell and Ihaka (2000) for graphical annotations and arrows in R figures.

Table 1: Custom operators in mathml

Operator	Output	Operator	Output	Operator	Arrow
A %*% B	$A \times B$	A != B	$A \neq B$	A %% B	$A \leftrightarrow B$
A %.% B	$A \cdot B$	A ~ B	$A \sim B$	A %>% B	$A \rightarrow B$
A %x% B	$A \otimes B$	A %~~% B	$A \approx B$	A %<% B	$A \leftarrow B$
A %/% B	$\lfloor A/B \rfloor$	A %==% B	$A \equiv B$	A %up% B	$A \uparrow B$
A %% B	$mod(A,B)$	A %~=% B	$A \cong B$	A %down% B	$A \downarrow B$
A & B	$A \wedge B$	A %prop% B	$A \bowtie B$	A %<=% B	$A \iff B$
A B	$A \vee B$	A %in% B	$A \in B$	A %=>% B	$A \Rightarrow B$
xor(A, B)	$A \vee B$	intersect(A, B)	$A \cap B$	A %<=% B	$A \Leftarrow B$
!A	$\neg A$	union(A, B)	$A \cup B$	A %dblup% B	$A \upuparrows B$
A == B	$A = B$	crossprod(A, B)	$A^T \times B$	A %dbldown% B	$A \downdownarrows B$
A <- B	$A = B$	is.null(A)	$A = \emptyset$		

3.5 Builtin functions

There is support for most functions from package base, with adequate use and omission of parentheses.

```
term <- quote(sin(x) + sin(x)^2L + cos(pi/2L) + tan(2L*pi) * expm1(x))
math(term)
```

$$\sin x + (\sin x)^2 + \cos(\pi/2) + \tan(2\pi) \cdot (\exp x - 1)$$

```
term <- quote(choose(N, k) + abs(x) + sqrt(x) + floor(x) + exp(frac(x, y)))
math(term)
```

$${N \choose k} + |x| + \sqrt{x} + \lfloor x \rfloor + \exp\left(\frac{x}{y}\right)$$

A few more examples are shown in Table 2, including functions from stats.

Table 2: R functions from base and stats

Function	Output	Function	Output
sin(x)	$\sin x$	dbinom(k, N, pi)	$P_{Bi}(X=k; N, \pi)$
cosh(x)	$\cosh x$	pbinom(k, N, pi)	$P_{Bi}(X \leq k; N, \pi)$
tanpi(alpha)	$\tan(\alpha\pi)$	qbinom(p, N, pi)	$\arg \min_k [P_{Bi}(X \leq k; N, \pi) > p]$
asinh(x)	$\sinh^{-1} x$	dpois(k, lambda)	$P_{Po}(X=k; \lambda)$

Function	Output	Function	Output
<code>log(p)</code>	$\log p$	<code>ppois(k, lambda)</code>	$P_{\text{Po}}(X \leq k; \lambda)$
<code>log1p(x)</code>	$\log(1+x)$	<code>qpois(p, lambda)</code>	$\arg \max_k [P_{\text{Po}}(X \leq k; \lambda) > p]$
<code>logb(x, e)</code>	$\log_e x$	<code>dexp(x, lambda)</code>	$f_{\text{Exp}}(x; \lambda)$
<code>exp(x)</code>	$\exp x$	<code>pexp(x, lambda)</code>	$F_{\text{Exp}}(x; \lambda)$
<code>expm1(x)</code>	$\exp x - 1$	<code>qexp(p, lambda)</code>	$F_{\text{Exp}}^{-1}(p; \lambda)$
<code>choose(n, k)</code>	$\binom{n}{k}$	<code>dnorm(x, mu, sigma)</code>	$\phi(x; \mu, \sigma)$
<code>lchoose(n, k)</code>	$\log \binom{n}{k}$	<code>pnorm(x, mu, sigma)</code>	$\Phi(x; \mu, \sigma)$
<code>factorial(n)</code>	$n!$	<code>qnorm(alpha/2L)</code>	$\Phi^{-1}(\alpha/2)$
<code>lfactorial(n)</code>	$\log n!$	<code>1L - pchisq(x, 1L)</code>	$1 - F_{\chi^2(1 \text{ df})}(x)$
<code>sqrt(x)</code>	\sqrt{x}	<code>qchisq(1L - alpha, 1L)</code>	$F_{\chi^2(1 \text{ df})}^{-1}(1 - \alpha)$
<code>mean(X)</code>	\bar{X}	<code>pt(t, N - 1L)</code>	$P(T \leq t; N - 1 \text{ df})$
<code>abs(x)</code>	$ x $	<code>qt(alpha/2L, N - 1L)</code>	$T_{\alpha/2}(N - 1 \text{ df})$

3.6 Custom functions

For self-written functions, the matter is somewhat more complicated. For a function such as `g <- function(...)` ..., the name `g` is not transparent to R, because only the function body is represented. We can still display functions in the form `head(x) = body` if we embed the object to be shown into a call "`<-`"(`head`, `body`).

```
sgn <- function(x)
{
  if(x == 0L) return(0L)
  if(x < 0L) return(-1L)
  if(x > 0L) return(1L)
}

math(sgn)


$$\begin{cases} 0, & \text{if } x=0 \\ -1, & \text{if } x<0 \\ 1, & \text{if } x>0 \end{cases}$$


math(call("<-", quote(sgn(x)), sgn))


$$\text{sgn } x = \begin{cases} 0, & \text{if } x=0 \\ -1, & \text{if } x<0 \\ 1, & \text{if } x>0 \end{cases}$$

```

The function body is generally a nested R call of the form `{(L)}`, with `L` being a list of commands (the semicolon, not necessary in R, is translated to a newline). The example also illustrates that `mathml` provides limited support for control structures such as `if` that is internally represented as `if(condition, action)`.

3.7 Indices and powers

Indices in square brackets are rendered as subscripts, powers are rendered as superscript. Moreover, `mathml` defines the functions `sum_over(x, from, to)`, and `prod_over(x, from, to)` that simply return their first argument. The other two arguments serve as decorations (`to` is optional), for example, for summation and product signs.

```
term <- quote(S[Y]^2L <- frac(1L, N) * sum(Y[i] - mean(Y))^2L)
math(term)
```

$$S_Y^2 = \frac{1}{N} \cdot \sum (Y_i - \bar{Y})^2$$

```
term <- quote(log(prod_over(L[i], i==1L, N)) <- sum_over(log(L[i]), i==1L, N))
math(term)
```

$$\log \prod_{i=1}^N L_i = \sum_{i=1}^N \log L_i$$

3.8 Ringing back to R

Rs integrate function takes a number of arguments, the most important ones being the function to integrate, and the lower and the upper bound of the integration.

```
term <- quote(integrate(sin, 0L, 2L*pi))
math(term)


$$\int_0^{2\pi} \sin x \, dx$$


eval(term)

#> 2.221482e-16 with absolute error < 4.4e-14
```

For mathematical typesetting in the form of $\int f(x) \, dx$, mathml needs to find out the name of the integration variable. For that purpose, the underlying Prolog bridge provides a predicate `r_eval/2` that calls R from Prolog. This predicate is used to evaluate `formalArgs(args(sin))` and returns the names of the arguments of `sin()`, namely, `x`.

Note that in the example above, the quoted term is an abbreviation for `call("integrate", quote(sin), ...)`, with `sin` being an R symbol, not a function. While the R function `integrate()` can handle both symbols and functions, mathml needs the symbol because it is unable to determine the function name of custom functions.

3.9 Names and order of arguments

One of R's great features is the possibility to refer to function arguments by their names, not only by their position in the list of arguments. At the other end, the Prolog handlers for R calls are rather rigid, for example, `integrate/3` accepts exactly three arguments in a particular order and without names, that is, `integrate(lower=0L, upper=2L*pi, sin)`, would not print the desired result.

To "canonicalize" function calls with named arguments and arguments in unusual order, mathml provides an auxiliary R function `canonical(f, drop)` that reorders the argument list of calls to known R functions and, if `drop=TRUE` (which is the default), also removes the names of the arguments.

```
term <- quote(integrate(lower=0L, upper=2L*pi, sin))
canonical(term)

#> integrate(sin, 0L, 2L * pi)

math(canonical(term))
```

$$\int_0^{2\pi} \sin x \, dx$$

This function can be used to feed mixtures of partially named and positional arguments into the renderer. For details, see the R function `match.call()`.

3.10 Matrices and Vectors

Of course, mathml also supports matrices and vectors.

```
v <- 1:3
math(call("t", v))

(123)T

A <- matrix(data=11:16, nrow=2, ncol=3)
B <- matrix(data=21:26, nrow=2, ncol=3)
term <- call("+", A, B)
math(term)
```

$$\begin{pmatrix} 11 & 13 & 15 \\ 12 & 14 & 16 \end{pmatrix} + \begin{pmatrix} 21 & 23 & 25 \\ 22 & 24 & 26 \end{pmatrix}$$

Note that the seemingly more convenient `term <- quote(A + B)` yields $A + B$ in the output—instead of the desired matrix representation. This behavior is expected because quotation of R calls also quote the components of the call (here, `A` and `B`).

3.11 Short mathematical names for R symbols

In typical R functions, variable names are typically longer than just single letters, which may yield unsatisfactory results in the mathematical output.

```
term <- quote(pbinom(successes, Ntotal, prob))
math(term)
```

$$P_{Bi}(X \leq successes; Ntotal, prob)$$

```
hook(successes, k)
hook(quote(Ntotal), quote(N), quote=FALSE)
hook(prob, pi)
math(term)
```

$$P_{Bi}(X \leq k; N, \pi)$$

To improve the situation, mathml provides a simple hook that can be used to replace elements (e.g., verbose variable names) of the code by concise mathematical symbols, as illustrated in the example. To simplify notation, hook() uses non-standard evaluation of its arguments. If the quote flag of hook() is set to FALSE, the user has to provide the quoted expressions. Care should be taken to avoid recursive hooks such as hook(s, s["A"]) that endlessly replace the s from s_A as in $s_{A_{A\dots}}$.

The hooks can also be used for more complex elements such as R calls, with dotted symbols representing Prolog variables.

```
hook(pbinom(.K, .N, .P), sum_over(dbinom(i, .N, .P), i=0L, .K))
math(term)
```

$$\sum_{i=0}^k P_{Bi}(X=i; N, \pi)$$

Further customization requires the assertion of new Prolog rules `math/2`, `m1/3`, `jax/3`, as shown in the Appendix.

3.12 Abbreviations

We consider the t -statistic for independent samples with equal variance. To avoid clutter in the equation, the pooled variance s_{pool}^2 is abbreviated, and a comment is given with the expression for s_{pool}^2 . For this purpose, mathml provides a function `denote(abbr, expr, info)`, with `expr` actually being evaluated, `abbr` being rendered, plus a comment of the form “with `expr` denoting `info`”.

```
hook(m_A, mean(X)[["A"]]) ; hook(s2_A, s[["A"]]^2L) ;
hook(n_A, n[["A"]])
hook(m_B, mean(X)[["B"]]) ; hook(s2_B, s[["B"]]^2L)
hook(n_B, n[["B"]]) ; hook(s2_p, s[["pool"]]^2L)

term <- quote(t <- dfrac(m_A - m_B,
  sqrt(denote(s2_p, frac((n_A - 1L)*s2_A + (n_B - 1L)*s2_B, n_A + n_B - 2L),
    "the pooled variance.") * (frac(1L, n_A) + frac(1L, n_B))))))
math(term)
```

$$t = \frac{\bar{X}_A - \bar{X}_B}{\sqrt{s_{\text{pool}}^2 \cdot \left(\frac{1}{n_A} + \frac{1}{n_B}\right)}}, \text{ with } s_{\text{pool}}^2 = \frac{(n_A - 1) \cdot s_A^2 + (n_B - 1) \cdot s_B^2}{n_A + n_B - 2} \text{ denoting the pooled variance.}$$

The term is evaluated below. `print()` is needed because the return value of an assignment of the form `t <- dfrac(...)` is not visible in R.

```
m_A <- 1.5; s2_A <- 2.4^2; n_A <- 27; m_B <- 3.9; s2_B <- 2.8^2; n_B <- 20
print(eval(term))
```

```
#> [1] -3.157427
```

3.13 Context-dependent rendering

Consider an educational scenario in which we want to highlight a certain element of a term, for example, that a student has forgotten to subtract the null hypothesis in a t -ratio:

```
t <- quote(dfrac(omit_right(mean(D) - mu[0]), s / sqrt(N)))
math(t, flags=list(error="highlight"))
```

$$\frac{\overline{D} - \mu_0}{s/\sqrt{N}}$$

```
math(t, flags=list(error="fix"))
```

$$\frac{\overline{D} - \boxed{\mu_0}}{s/\sqrt{N}}$$

The R function `omit_right(a + b)` uses non-standard evaluation techniques (e.g., Wickham 2019) to return only the left part of an operation, and cancels the right part. This may not always be desired, for example, when illustrating how to fix the mistake.

For this purpose, the functions `mathml()`, `mathjax()`, `mathout()` and `math()` have an optional argument `flags` which is a list with named elements. In this example, we use this argument to tell `mathml` how to render such erroneous expressions using the flag `error` which can be “`asis`”, “`highlight`”, “`fix`”, or “`ignore`”. For more examples, see Table 3.

Table 3: Highlighting elements of a term

Operation	error = asis	highlight	fix	ignore
<code>omit_left(a + b)</code>	b	$\cancel{a+b}$	$\boxed{a+b}$	$a+b$
<code>omit_right(a + b)</code>	a	$a\cancel{+b}$	$a+\boxed{b}$	$a+b$
<code>list(quote(a), quote(omit(b)))</code>	a	$a\cancel{b}$	$a\boxed{b}$	ab
<code>add_left(a + b)</code>	$a+b$	$\boxed{a+b}$	$\cancel{a+b}$	b
<code>add_right(a + b)</code>	$a+b$	$a\boxed{+b}$	$a\cancel{+b}$	a
<code>list(quote(a), quote(add(b)))</code>	ab	$a\boxed{b}$	$a\cancel{b}$	a
<code>instead(a, b) + c</code>	$a+c$	$\underbrace{a}_{\text{instead of } b} + c$	$\boxed{b}+c$	$b+c$

4 A case study

This case study describes a model by Schwarz (1994) from mathematical psychology using the features of package `mathml`. Schwarz (1994) presents a new explanation of redundancy gains that occur when observers respond to stimuli of different sources, and the same information is presented on two or more channels. In Schwarz's (1994) model, decision-making builds on a process of noisy accumulation of information over time (e.g., Ratcliff et al. 2016). In redundant stimuli, the model assumes a superposition of channel-specific diffusion processes that eventually reach an absorbing barrier to elicit the response. For a detailed description the reader may refer to the original article.

Schwarz's (1994) model refers to two stimuli A and B, presented either alone or in combination (AB, redundant stimuli), with the redundant stimuli being presented either simultaneously or with onset asynchrony τ . The channel activation is described as a two-dimensional Wiener process with drifts μ_i , variances σ_i^2 , and initial conditions $X_i(t=0) = 0, i = A, B$. The buildup of channel-specific activation may be correlated with ρ_{AB} , but we assume $\rho_{AB} = 0$ for simplicity.

A response is elicited when the process reaches an absorbing barrier $c > 0$ for the first time. In single-target trials, the first passages of c are expected at

```
ED_single <- function(c, mu)
dfrac(c, mu)

# display as E(D; mu), c is a scaling parameter
hook(ED_single,.C, .Mu), E(`;(D, .Mu)))
math(call("=", quote(ED_single(c, mu)), ED_single))
```

$$E(D;\mu) = \frac{c}{\mu}$$

One would typically use chunk option echo=FALSE to suppress the R code.

In redundant stimuli, the activation from the channel-specific diffusion processes adds up, $X_{AB}(t) = X_A(t) + X_B(t)$, hence the name, superposition. $X_{AB}(t)$ is again a Wiener process with drift $\mu_A + \mu_B$ and variance $\sigma_A^2 + \sigma_B^2$. For the expected first-passage time, we have

```
hook(mu_A, mu["A"])
hook(mu_B, mu["B"])
hook(sigma_A, sigma["A"])
hook(sigma_B, sigma["B"])
hook(mu_M, mu["M"])
hook(M, overline(X))

math(call("=", quote(E(D["AB"])), quote(ED_single(c, mu_A + mu_B))))
```

$$E(D_{AB}) = E(D; \mu_A + \mu_B)$$

For asynchronous stimuli, Schwarz (1994) derived the expected first-passage time as a function of the stimulus onset asynchrony τ ,

```
ED_async <- function(tau, c, mu_A, sigma_A, mu_B)
{ dfrac(c, mu_A) + (dfrac(1L, mu_A) - dfrac(1L, mu_A + mu_B)) *
  ((mu_A*tau - c) * pnorm(dfrac(c - mu_A*tau, sqrt(sigma_A^2*L*tau)))
   - (mu_A*tau + c) * exp(dfrac(2L*c*mu_A, sigma_A^2*L))
   * pnorm(dfrac(-c - mu_A*tau, sqrt(sigma_A^2*L*tau))))
}

hook(ED_async(.Tau, .C, .MA, .SA, .MB), E(`;`(D[.Tau], `,-`(.MA, .SA, .MB))))
math(call("=", quote(E(D[tau])), ED_async))

E(D_tau) = 
$$\frac{c}{\mu_A} + \left( \frac{1}{\mu_A} - \frac{1}{\mu_A + \mu_B} \right) \cdot \left[ (\mu_A \tau - c) \cdot \Phi \left( \frac{c - \mu_A \tau}{\sqrt{\sigma_A^2 \tau}} \right) - (\mu_A \tau + c) \cdot \exp \left( \frac{2c\mu_A}{\sigma_A^2} \right) \cdot \Phi \left( \frac{-c - \mu_A \tau}{\sqrt{\sigma_A^2 \tau}} \right) \right]$$

```

For negative onset asynchrony (i.e., B before A), the parameters are simply switched.

```
ED <- function(tau, c, mu_A, sigma_A, mu_B, sigma_B)
{
  if(tau == Inf) return(ED_single(c, mu_A))
  if(tau == -Inf) return(ED_single(c, mu_B))
  if(tau == 0L) return(ED_single(c, mu_A + mu_B))
  if(tau > 0L) return(ED_async(tau, c, mu_A, sigma_A, mu_B))
  if(tau < 0L) return(ED_async(abs(tau), c, mu_B, sigma_B, mu_A))
}

hook(ED(.Tau, .C, .MA, .SA, .MB, .SB), E(`;`(D[.Tau], `,-`(.MA, .SA, .MB, .SB))))
math(call("=", quote(ED(tau, c, mu_A, sigma_A, mu_B, sigma_B)), ED))
```

$$E(D_\tau; \mu_A, \sigma_A, \mu_B, \sigma_B) = \begin{cases} E(D; \mu_A), & \text{if } \tau = \infty \\ E(D; \mu_B), & \text{if } \tau = -\infty \\ E(D; \mu_A + \mu_B), & \text{if } \tau = 0 \\ E(D_\tau; \mu_A, \sigma_A, \mu_B), & \text{if } \tau > 0 \\ E(D_{|\tau|}; \mu_B, \sigma_B, \mu_A), & \text{if } \tau < 0 \end{cases}$$

The observable response time is assumed to be the sum of D , the time employed to reach the threshold for the decision, and a residual M denoting other processes such as motor preparation and execution. Correspondingly, the expected response time amounts to

```
ET <- function(tau, c, mu_A, sigma_A, mu_B, sigma_B, mu_M)
  ED(tau, c, mu_A, sigma_A, mu_B, sigma_B) + mu_M

hook(ET(.Tau, .C, .MA, .SA, .MB, .SB, .MM),
     E(`;`(.Tau], `,-`(.MA, .SA, .MB, .SB, .MM)))
math(call("=", quote(E(T[tau])), ET))
```

$$E(T_\tau) = E(D_\tau; \mu_A, \sigma_A, \mu_B, \sigma_B) + \mu_M$$

Schwarz (1994) applied the model to data from a redundant signals task (Miller 1986) with 13 onset asynchronies $0, \pm 33, \pm 67, \pm 100, \pm 133, \pm 167, \pm \infty$ ms, where $\tau = 0$ refers to the synchronous condition, and $\pm \infty$ to the single-target presentations. Each condition was replicated 400 times. The observed mean response times and their standard deviations are given in Table 4.

Table 4: Miller (1986) data

τ	m	s	n
$-\infty$	231	56	400
-167	234	58	400
-133	230	40	400
-100	227	40	400
-67	228	32	400
-33	221	28	400
0	217	28	400
33	238	28	400
67	263	26	400
100	277	30	400
133	298	32	400
167	316	34	400
∞	348	92	400

Assuming that the model is correct, the observable mean reaction times follow an approximate Normal distribution around the model prediction $E(T_\tau)$ for each condition. We can, therefore, use a standard goodness-of-fit measure by z -standardization.

```

z <- function(m, s, n, tau, c, mu_A, sigma_A, mu_B, sigma_B, mu_M)
dfrac(m - denote(mu[tau], ET(tau, c, mu_A, sigma_A, mu_B, sigma_B, mu_M),
                 "the expected mean response time"),
      s / sqrt(n))

math(call("=", quote(z[tau]), z))


$$z_\tau = \frac{m - \mu_\tau}{s/\sqrt{n}}$$
, with  $\mu_\tau = E(T_\tau; \mu_A, \sigma_A, \mu_B, \sigma_B, \mu_M)$  denoting the expected mean response time

```

The overall goodness-of-fit is the sum of the squared z -statistics for each onset asynchrony. Assuming again that the architecture of the model is correct, but the parameters are adjusted to the data, it follows a $\chi^2(8 \text{ df})$ -distribution.

```

zv <- Vectorize(z, vectorize.args = c('m', 's', 'n', 'tau'))
hook(zv(.M, .S, .N, .Tau, .C, .MA, .SA, .MB, .SB, .MM), z[.Tau])

gof <- function(par, tau, m, s, n)
sum(zv(m, s, n, tau, c=100L, mu_A=par["mu_A"], sigma_A=par["sigma_A"],
       mu_B=par["mu_B"], sigma_B=par["sigma_B"], mu_M=par["mu_M"])^2L)

math(call("=", quote(X["8 df"]^2L), gof))

```

$$X_{\text{8df}}^2 = \sum z_\tau^2$$

with the degrees of freedom given by the difference between the number of observations (13) and the number of free model parameters $\theta = \langle \mu_A, \sigma_A, \mu_B, \sigma_B, \mu_M \rangle$; the barrier c is only a scaling parameter.

$$\hat{\theta} = \arg \min gof(\theta)$$

The best fitting parameter values and their confidence intervals are given in Table 5.

Table 5: Model fit

Parameter	Estimate	CI
μ_A	0.53	(0.51, 0.55)
σ_A	4.55	(3.95, 5.16)

Parameter	Estimate	CI
μ_B	1.36	(1.23,1.49)
σ_B	13.46	(7.80,19.11)
μ_M	161.09	(156.91,165.28)

The goodness-of-fit statistic indicates some lack of fit, $X^2(8 \text{ df}) = 28.34, p = 0.0004$. Given the large trial numbers in the original study, this is not an unexpected result. For more detail, especially on fitting the observed standard deviations, the reader is referred to the original paper (Schwarz 1994).

5 Conclusion

This package allows R to render its terms in pretty mathematical equations. It extends the current features of R and existing packages for displaying mathematical formulas in R (Murrell and Ihaka 2000), but most importantly, mathml bridges the gap between computational needs, presentation of results, and their reproducibility. The package supports both MathML and LaTeX/MathJax for use in R Markdown documents, presentations and Shiny App webpages.

Researchers or teachers can already use R Markdown to conduct analyses and show results, and mathml smoothes this process and allows for integrated calculations and output. As shown in the case study of the previous section, mathml can help to improve data analyses and statistical reports from an aesthetical perspective, as well as regarding reproducibility of research.

Furthermore, the package may also allow for a better detection of possible mistakes in R programs. Similar to most programming languages (Green 1977), R code is notoriously hard to read, and the poor legibility of the language is one of the main sources of mistakes. For illustration, we consider again Equation 10 in Schwarz (1994).

```
f1 <- function(tau)
{ dfrac(c, mu_A) + (dfrac(1L, mu_A) - dfrac(1L, mu_A + mu_B) *
  ((mu_A*tau - c) * pnorm(dfrac(c - mu_A*tau, sqrt(sigma_A^2L*tau)))
  - (mu_A*tau + c) * exp(dfrac(2L*mu_A*tau, sigma_A^2L)))
  * pnorm(dfrac(-c - mu_A*tau, sqrt(sigma_A^2L*tau)))) )
}

math(f1)


$$\frac{c}{\mu_A} + \left\{ \frac{1}{\mu_A} - \frac{1}{\mu_A + \mu_B} \cdot \left[ (\mu_A \tau - c) \cdot \Phi \left( \frac{c - \mu_A \tau}{\sqrt{\sigma_A^2 \tau}} \right) - (\mu_A \tau + c) \cdot \exp \left( \frac{2\mu_A \tau}{\sigma_A^2} \right) \cdot \Phi \left( \frac{-c - \mu_A \tau}{\sqrt{\sigma_A^2 \tau}} \right) \right] \right\}$$

```

The first version has a wrong parenthesis, which is barely visible in the code, whereas in the mathematical representation, the wrong curly brace is immediately obvious (the correct version is shown below for comparison).

```
f2 <- function(tau)
{ dfrac(c, mu_A) + (dfrac(1L, mu_A) - dfrac(1L, mu_A + mu_B)) *
  ((mu_A*tau - c) * pnorm(dfrac(c - mu_A*tau, sqrt(sigma_A^2L*tau)))
  - (mu_A*tau + c) * exp(dfrac(2L*mu_A*tau, sigma_A^2L)))
  * pnorm(dfrac(-c - mu_A*tau, sqrt(sigma_A^2L*tau)))) )
}

math(f2)
```

$$\frac{c}{\mu_A} + \left(\frac{1}{\mu_A} - \frac{1}{\mu_A + \mu_B} \right) \cdot \left[(\mu_A \tau - c) \cdot \Phi \left(\frac{c - \mu_A \tau}{\sqrt{\sigma_A^2 \tau}} \right) - (\mu_A \tau + c) \cdot \exp \left(\frac{2\mu_A \tau}{\sigma_A^2} \right) \cdot \Phi \left(\frac{-c - \mu_A \tau}{\sqrt{\sigma_A^2 \tau}} \right) \right]$$

As the reader may know from their own experience, missed parentheses are frequent causes of wrong results and errors that are hard to locate in programming code. This particular example shows that mathematical rendering can help to substantially reduce the amount of careless errors in programming.

One limitation of the package is the lack of a convenient way to insert line breaks. This is mostly due to lacking support by MathML and LaTeX renderers. For example, in its current stage, the LaTeX package breqn (Robertson et al. 2021) is mostly a proof of concept. Moreover, mathml only works in

one direction, that is, it is not possible to translate from LaTeX or HTML back to R (see Capretto 2023, for an example).

The package `mathml` is available for R version 4.2 and later, and can be easily installed using the usual `install.packages("mathml")`. At its present stage, it supports output in HTML, LaTeX, and Microsoft Word (via pandoc, MacFarlane 2022). The source code of the package is found at <https://github.com/mgondan/mathml>.

6 Appendix: Customizing the package

6.1 Implementation details

For convenience, the translation of the R expressions is achieved through a Prolog interpreter provided by another R package `rolog` (Gondan 2022). If a version of SWI-Prolog (Wielemaker et al. 2012) is found on the system, `rolog` connects to it. Alternatively, the SWI-Prolog runtime libraries can be conveniently accessed by installing the R package `rswipl` (Gondan 2023). Prolog is a classical logic programming language with many applications in expert systems, computer linguistics and symbolic artificial intelligence. The strength of Prolog lies in its concise representation of facts and rules for knowledge and grammar, as well as its efficient built-in search engine for closed world domains. Whereas Prolog is weak in statistical computation, but strong in symbolic manipulation, the converse may be said for the R language. `rolog` bridges this gap by providing an interface to a SWI-Prolog distribution (Wielemaker et al. 2012) in R. The communication between the two systems is mainly in the form of queries from R to Prolog, but two Prolog functions allow ring back and evaluation of terms in R.

The proper term for a Prolog “function” is predicate, and it is typically written with name and arity (i.e., number of arguments), separated by a forward slash. Thus, at the Prolog end, the predicate `math/2` translates the representation of the R call `pbinom(K, N, Pi)` into a more general representation of an R function `fn/2` with the name `P_Bi`, one argument `X =< K`, and the two parameters `N` and `Pi`, as shown below.

```
math(pbinom(K, N, Pi), M)
=> M = fn(subscript('P', "Bi"), (['X' =< K] ; [N, Pi])).
```

`math/2` operates like a macro that translates one mathematical element (here, `pbinom(K, N, Pi)`) to another mathematical element, namely `fn(Name, (Args ; Pars))`. The low-level predicate `ml/3` is used to convert these basic elements to MathML.

```
ml(fn(Name, (Args ; Pars)), M, Flags)
=> ml(Name, N, Flags),
   ml(paren(list(op(;), [list(op(',', Args), list(op(',', Pars))])), X, Flags),
   M = mrow([N, mo(&(af)), X]).
```

The relevant rule for `ml/3` builds the MathML entity `mrow([N, mo(&(af)), X])`, with `N` representing the name of the function and `X` its arguments and parameters enclosed in parentheses. A corresponding rule `jax/3` does the same for MathJax/LaTeX. A list of flags can be used for context-sensitive translation (see, e.g., the section on errors above).

Several ways exist for translating new R terms to their mathematical representation. We have already seen above how to use “hooks” to translate long variable names from R to compact mathematical signs, as well as functions such as cumulative probabilities $P(X \leq k)$ to different representations like $\sum_{i=0}^k P(X = i)$. Obviously, the hooks require that there already exists a rule to translate the target representation into MathML and MathJax.

In this appendix we describe a few more ways to extend the set of translations according to a user’s needs. As stated in the background section, the Prolog end provides two classes of rules for translation, macros `math/2, 3, 4` mirroring the R hooks mentioned above, and the low-level predicates `ml/3` and `jax/3` that create proper MathML and LaTeX terms.

6.2 Linear models

To render the model equation of a linear model such as `lm(EOT ~ T0 + Therapy, data=d)` in mathematical form (see also Anderson, Heiss, and Sumners 2023), it is sufficient to map the `Formula` in `lm(Formula, Data)` to its respective equation. This can be done in two ways, using either the hooks described above, or a new `math/2` macro at the Prolog end.

```
hook(lm(.Formula, .Data), .Formula)
```

The hook is simple, but is a bit limited because only R's tilde-form of linear models is shown, and it only works for a call with exactly two arguments.

Below is an example of how to build a linear equation of the form $Y = b_0 + b_1X_1 + \dots$ using the Prolog macros from mathml.

```
math_hook(LM, M) :-  
    compound(LM),  
    LM =.. [lm, ~(Y, Sum) | _Tail],  
    summands(Sum, Predictors),  
    findall(subscript(b, X) * X, member(X, Predictors), Terms),  
    summands(Model, Terms),  
    M = (Y == subscript(b, 0) + Model + epsilon).
```

The predicate `summands/2` unpacks an expression $A + B + C$ to a list $[C, B, A]$ and vice-versa (see the file `lm.pl` for details).

```
rolog::consult(system.file(file.path("pl", "lm.pl"), package="mathml"))

term <- quote(lm(EOT ~ T0 + Therapy, data=d, na.action=na.fail))
math(term)
```

$$EOT = b_0 + b_{T0}T0 + b_{Therapy}Therapy + \epsilon$$

6.3 n -th root

Base R does not provide a function like `cuberoot(x)` or `nthroot(x, n)`, and the present package does not support the respective representation. To obtain a cube root, a programmer would typically type $x^{(1/3)}$ or better $x^{\{1/3\}}$ (see the practice section why the curly brace is preferred in an exponent), resulting in $x^{1/3}$ which may still not match everyone's taste. Here we describe the steps needed to represent the n -th root as $\sqrt[n]{x}$.

We assume that `nthroot(x, n)` is available in the current namespace (manually defined, or from R package `pracma`, Borchers 2022), so that the names of the arguments and their order are accessible to `canonical()` if needed. As we can see below, `mathml` uses a default representation name(`arguments`) for such unknown functions.

```
nthroot <- function(x, n)
  x^{\{1L/n\}}
```

```
term <- canonical(quote(nthroot(n=3L, 2L)))
math(term)
```

$$\text{nthroot}(2,3)$$

A proper MathML term is obtained by `mlx/3` (the `x` in `mlx` indicates that it is an extension and is prioritized over the default `m1/3` rules). `mlx/3` recursively invokes `m1/3` for translating the function arguments `X` and `N`, and then constructs the correct MathML entity `<mroot>...</mroot>`.

```
mlx(nthroot(X, N), M, Flags) :-
  m1(X, X1, Flags),
  m1(N, N1, Flags),
  M = mroot([X1, N1]).
```

The explicit unification `M = ...` in the last line serves to avoid clutter in the head of `mlx/3`. The Prolog file `nthroot.pl` also includes the respective rule for LaTeX and can be consulted from the package folder via the underlying package `rolog`.

```
rolog::consult(system.file(file.path("pl", "nthroot.pl"), package="mathml"))

term <- quote(nthroot(a * (b + c), 3L)^2L)
math(term)
```

```


$$\left[ \sqrt[3]{a \cdot (b+c)} \right]^2$$

term <- quote(a^(1L/3L) + a^{1L/3L} + a^(1.0/3L))
math(term)

```

$$\sqrt[3]{a+a^{1/3}+a^{(1.00/3)}}$$

The file `nthroot.pl` includes three more statements `precx/3` and `parenx/3`, as well as a `math_hook/2` macro. The first sets the operator precedence of the cubic root above the power, thereby putting a parentheses around `nthroot` in $(\sqrt[3]{...})^2$. The second tells the system to increase the counter of the parentheses below the root, such that the outer parenthesis becomes a square bracket.

The last rule maps powers like $a^{(1L/3L)}$ to `nthroot/3`, as shown in the first summand. Of course, `mathml` is not a proper computer algebra system. As is illustrated by the other terms in the sum, such macros are limited to purely syntactical matching, and terms like $a^{(1L/3L)}$ with the curly brace or $a^{(1.0/3L)}$ with a floating point number in the numerator are not detected.

7 Acknowledgment

Supported by the Erasmus+ program of the European Commission (2019-1-EE01-KA203-051708).

References

- Allaire, J. J., Y. Xie, C. Dervieux, J. McPherson, J. Luraschi, K. Ushey, A. Atkins, et al. 2023. *Rmarkdown: Dynamic Documents for R*. <https://github.com/rstudio/rmarkdown>.
- Anderson, D., A. Heiss, and J. Sumners. 2023. *Equatiomatic: Transform Models into ‘LaTeX’ Equations*.
- Borchers, H. W. 2022. *pracma: Practical Numerical Math Functions*. <https://CRAN.R-project.org/package=pracma>.
- Capretto, T. 2023. *latex2r: Translate Latex Formulas to R Code*. <https://github.com/tomicapretto/latex2r>.
- Chang, W., J. Cheng, J. J. Allaire, C. Sievert, B. Schloerke, Y. Xie, J. Allen, J. McPherson, A. Dipert, and B. Borges. 2022. *Shiny: Web Application Framework for R*. <https://CRAN.R-project.org/package=shiny>.
- Gondan, M. 2022. *rolog: Query SWI-Prolog from R*. <https://github.com/mgondan/rolog>.
- . 2023. *rswipl: Embed SWI-Prolog*. <https://CRAN.R-project.org/package=rswipl>.
- Green, T. R. G. 1977. “Conditional Program Statements and Their Comprehensibility to Professional Programmers.” *Journal of Occupational Psychology* 50: 93–109.
- MacFarlane, J. 2022. *Pandoc: A Universal Document Converter*.
- Miller, J. O. 1986. “Timecourse of Coactivation in Bimodal Divided Attention.” *Perception & Psychophysics* 40: 331–43.
- Murrell, P., and R. Ihaka. 2000. “An Approach to Providing Mathematical Annotation in Plots.” *Journal of Computational and Graphical Statistics* 9: 582–99.
- R Core Team. 2022. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. <https://www.R-project.org/>.
- Ratcliff, R., P. L. Smith, S. D. Brown, and G. McKoon. 2016. “Diffusion Decision Model: Current Issues and History.” *Trends in Cognitive Sciences* 20: 260–81.
- Robertson, W., J. Wright, F. Mittelbach, and U. Fischer. 2021. *Breqn: Automatic Line Breaking of Displayed Equations*. <https://www.ctan.org/pkg/breqn>.
- Sarkar, D., and K. Hornik. 2022. *Enhancements to HTML Documentation*. <https://blog.r-project.org/2022/04/08/enhancements-to-html-documentation/index.html>.
- Schwarz, W. 1994. “Diffusion, Superposition, and the Redundant-Targets Effect.” *Journal of Mathematical Psychology* 38: 504–20.
- Viechtbauer, W. 2022. *mathjaxr: Using ‘Mathjax’ in Rd Files*. <https://CRAN.R-project.org/package=mathjaxr>.
- Wickham, H. 2019. *Advanced R*. Cambridge: Chapman; Hall/CRC.
- Wielemaker, J., T. Schrijvers, M. Triska, and T. Lager. 2012. “SWI-Prolog.” *Theory and Practice of Logic Programming* 12 (1-2): 67–96.

- Xie, Y. 2023. *knitr: A General-Purpose Package for Dynamic Report Generation in R*. <https://yihui.org/knitr/>.
- Xie, Y., J. J. Allaire, and G. Grolemund. 2018. *R Markdown: The Definitive Guide*. Boca Raton, Florida: Chapman; Hall/CRC. <https://bookdown.org/yihui/rmarkdown>.
- Xie, Y., C. Dervieux, and E. Riederer. 2020. *R Markdown Cookbook*. Cambridge: Chapman; Hall/CRC.

8 References

Matthias Gondan
Universität Innsbruck
Department of Psychology
Innsbruck, Austria
<https://www.uibk.ac.at/psychologie/mitarbeiter/gondan-rochon/index.html.en>
ORCID: 0000-0001-9974-0057
Matthias.Gondan-Rochon@uibk.ac.at

Irene Alfarone
Universität Innsbruck
Department of Psychology
Innsbruck, Austria
<https://www.uibk.ac.at/psychologie/mitarbeiter/alfarone/index.html.en>
ORCID: 0000-0002-8409-8900
Irene.Alfarone@uibk.ac.at

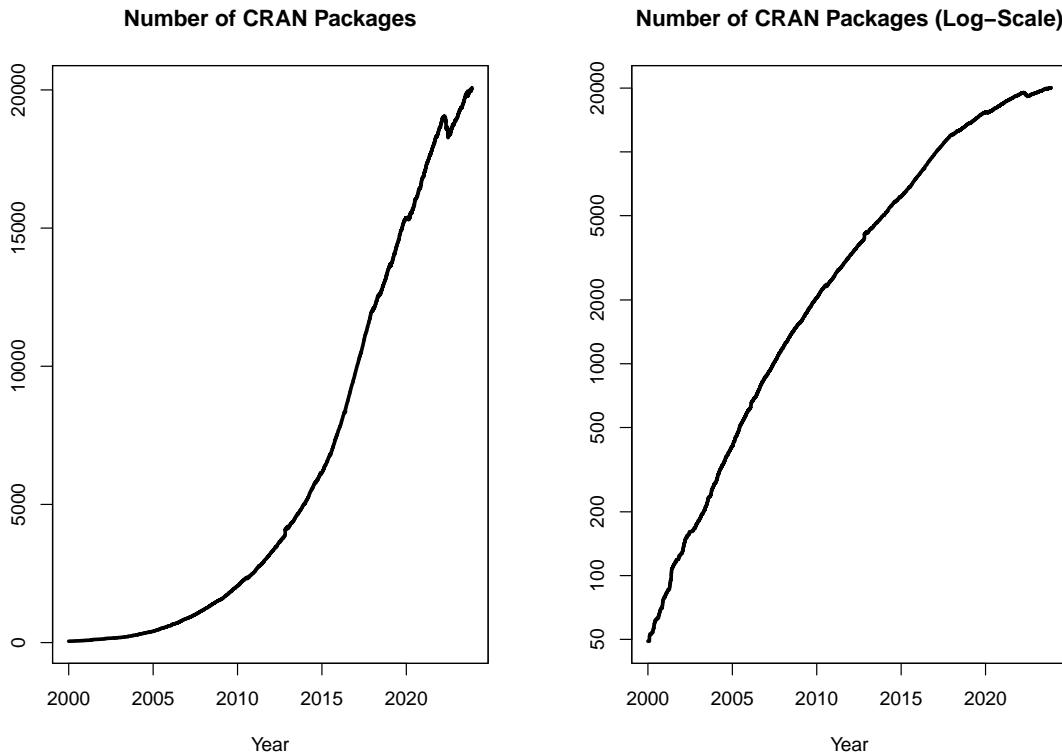
Changes on CRAN

2023-05-01 to 2023-09-30

by Kurt Hornik, Uwe Ligges, and Achim Zeileis

1 CRAN growth

In the past 5 months, 866 new packages were added to the CRAN package repository. 332 packages were unarchived, 701 were archived and 5 had to be removed. The following shows the growth of the number of active packages in the CRAN package repository:



On 2023-09-30, the number of active packages was around 19932.

2 Changes in the CRAN Repository Policy

The [Policy](#) now links to an accompanying document on [Using Rust in CRAN packages](#).

3 CRAN package submissions

From May 2023 to September 2023 CRAN received 12237 package submissions. For these, 20527 actions took place of which 14006 (68%) were auto processed actions and 6521 (32%) 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	3677	1889	2460	0	0	3734	1282	964
manual	2378	186	431	317	52	2423	599	135

These include the final decisions for the submissions which were

	archive	publish
auto	3573 (29.7%)	3246 (27%)
manual	2337 (19.4%)	2883 (23.9%)

where we only count those as *auto* processed whose publication or rejection happened automatically in all steps.

4 CRAN mirror security

Currently, there are 94 official CRAN mirrors, 76 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).

5 CRAN Task View Initiative

There is one new task view:

- [Actuarial Science](#): Maintained by Christophe Dutang, Vincent Goulet.

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 122, respectively. Overall, these task views cover 4496 CRAN packages, which is about 22% of all active CRAN packages.

Kurt Hornik
WU Wirtschaftsuniversität Wien
Austria
ORCID: 0000-0003-4198-9911
Kurt.Hornik@R-project.org

Uwe Ligges
TU Dortmund
Germany
ORCID: 0000-0001-5875-6167
Uwe.Ligges@R-project.org

Achim Zeileis
Universität Innsbruck
Austria
ORCID: 0000-0003-0918-3766
Achim.Zeileis@R-project.org

R Foundation News

by Torsten Hothorn

1 Donations and members

Membership fees and donations received between 2023-05-09 and 2023-10-20.

2 Donations

Daniel J. Duarte (Netherlands)
Spyridon Fortis (United States)
Dmitriy Grishin (United States)
Ken Ikeda (Japan)
Nickalus Redell (United States)
Axxos AG (Switzerland)

3 Supporting institutions

Digital Ecology Limited , Berkeley (United Kingdom)
NIFU Nordic Institute for Studies in Innovation, Research and Education, Oslo (Norway)
RIATE (CNRS), Paris (France)
Roseburg Forest Products, Springfield (United States)
University of Iowa, Iowa City (United States)

4 Supporting members

Vedo Alagic (Austria)
Kristoffer Winther Balling (Denmark)
Joaquín Baquer-Miravete (Spain)
Chris Billingham (United Kingdom)
Susan M Carlson (United States)
Robert Carnell (United States)
William Chiu (United States)
Rafael Costa (Brazil)
Charles Cowens (United States)
Terry Cox (United States)
Alistair Cullum (United States)
Gergely Daroczi (Hungary)
Ajit de Silva (United States)
Dubravko Dolic (Germany)
Serban Dragne (United Kingdom)
Laurent Drouet (Italy)
S Ellison (United Kingdom)
Mitch Eppley (United States)
Stephen Ewing (United States)
Gottfried Fischer (Austria)

David Freedman (United States)
Keita Fukasawa (Japan)
Chris Hanretty (United Kingdom)
James Harris (United States)
Takehiko Hayashi (Japan)
Malik Hebbat (Germany)
Alessandro Ielpi (Canada)
Brian Johnson (United States)
An Khuc (United States)
Miha Kosmac (United Kingdom)
Jan Herman Kuiper (United Kingdom)
Jindra Lacko (Czechia)
Bernardo Lares (Venezuela)
Rory Lawless (United States)
Seungdoe Lee (Korea, Republic of)
Mauro Lepore (United States)
Amanuel Medhanie (United States)
Harvey Minnigh (Puerto Rico)
Guido Möser (Germany)
Mark Niemann-Ross (United States)
George Ostrouchov (United States)
Jaesung James Park (Korea, Republic of)
Matt Parker (United States)
Bill Pikounis (United States)
Kelly Pisane (Netherlands)
Christian Seubert (Austria)
Jagat Sheth (United States)
Sindri Shtepani (Canada)
David Sides (United States)
Murray Sondergard (Canada)
Olga Starostecka (Germany)
Marco Steenbergen (Switzerland)
Berthold Stegemann (Germany)
Ricardo Torres-Jardón (Mexico)
Nicholas Turner (United States)
Philipp Upravitelev (Russian Federation)
Mark van der Loo (Netherlands)
Frans van Dunné (Costa Rica)
Vincent van Hees (Netherlands)
Yang Hu (New Zealand)

Torsten Hothorn
Universität Zürich
Switzerland
ORCID: 0000-0001-8301-0471
Torsten.Hothorn@R-project.org

News from Bioconductor

by Bioconductor Core Developers

Software

Bioconductor version 3.18 was released on Oct 25 2023. The system is compatible with R 4.3. See [the release announcement](#) for full details. Noteworthy additions since our last report to the R Journal include

- a [BSgenome](#) package for the telomere-to-telomere build of the human genome (Aganezov et al. (2022)),
- the [SparseArray](#) package for overcoming the limit on the number of non-zero elements allowable in a sparse Matrix instance,
- [BiocBook](#), a package to facilitate the creation of package-based, versioned online books authored in Quarto,
- a pair of Annotation packages with the AlphaMissense (Cheng et al. (2023)) pathogenicity scores for coding variants in the human genome for builds [hg19](#) and [hg38](#).

See the [release announcement](#) for full details. The growth of the package repertory is greatly aided by a group of committed and energetic reviewers. All reviews are conducted in github issues streams at [contributions.bioconductor.org/issues](#).

Infrastructure

National Science Foundation ACCESS Award BIR190004 provides significant compute resources in the [Jetstream2 academic cloud](#) along with storage provided by the [Open Storage Network](#).

These resources form the basis for the Galaxy/Kubernetes-backed [Bioconductor workshop platform](#) originally known as [Orchestra](#). Workshop submissions are now accepted through a [Shiny app](#) made available at the platform site. The “[BuildABiocWorkshop](#)” template has been updated with GitHub Actions, and now uses the GitHub Container Registry ([ghcr.io](#))

At present, Bioconductor 3.18 packages are tested regularly on Ubuntu 22.04, macOS 13.6 (arm64), macOS 12.7 (x86_64), and Windows Server 2022 Datacenter. Testing of packages in the devel branch includes an arm64 Linux platform (openEuler 22.03) thanks to efforts of Martin Grigorov and Yikun Jiang.

Docker container updates

An active effort to revamp the Bioconductor Docker stack is in progress, maintaining backwards compatibility, but featuring a number of new capabilities. Notably, all containers are now published both on [DockerHub](#) as well as the [GitHub Container Registry](#) (GHCR), so any container previously pulled as `bioconductor/bioconductor_docker` for example, can now also be pulled from GHCR as `ghcr.io/bioconductor/bioconductor_docker`.

Additionally, the traditional `rstudio`-based containers, previously published under the `bioconductor/bioconductor_docker` name, are now also available under the `bioconductor/bioconductor` name, eliminating the need to type the `_docker` suffix. Moreover, release tags can still be used as “`RELEASE_3_18`” as before, but the simpler “`3.18`” tag now suffices. The latest `rstudio`-based container can thus now be pulled as `docker pull bioconductor/bioconductor:3.18` and will be identical to `bioconductor/bioconductor_docker:RELEASE_3_18`.

Bioconductor now has containers built on top of different flavors of rocker such as `bioconductor/r-ver` container, a slimmer container with R but not RStudio, and `bioconductor/ml-verse`, featuring tidyverse and some GPU drivers pre-installed. These and more container flavors can be found at [DockerHub](#) and [GitHub](#).

Bioc2u alpha release

Ucar and Eddelbuettel (2021) discuss motivations and methods for distributing pre-compiled R packages via Linux system package managers. Bioconductor has recently

undergone an effort to make a full package repository of Bioconductor packages and their dependencies available via apt on Ubuntu systems. Building on top of previous work from the Debian [r-pkg-team](#) and the [r2u project](#), the Bioc2u repository currently offers 3708 packages for the Bioconductor 3.18 release for Ubuntu Jammy. More information can be found at <https://github.com/bioconductor/bioc2u>, and alpha testers are welcome to join the #bioc2u channel on the Bioconductor [Community Slack](#).

Developer support

Bioconductor is developing containers similar to the Bioconductor Linux build machines. [BBS containers](#) are configured like the build machines and aim to provide a comparable experience for developers to troubleshoot issues observed on the linux build machines. The 3.18 BBS container is available for testing with

```
docker pull ghcr.io/bioconductor/bioconductor_salt:jammy-bioc-3.18-r-4.3.2
```

Future work on the BBS containers will focus on testing the container's performance in comparison to the linux build machine, building the devel container, and incorporating the container in a GitHub Action Workflow.

User support

Thanks to support from the Chan-Zuckerberg Initiative Essential Open Source Software for Science program, the web site at bioconductor.org has been extensively revised.

Partnering with Outreachy

Bioconductor mentored three interns in the May - August 2023 Outreachy cohort. [Outreachy](#) partners with open source and open science organizations to create paid open source internships to individuals underrepresented in technology. The organization, which recently celebrated surpassing 1000 interns, funded the interns for three Bioconductor-mentored projects through their general fund. Interns are selected based on the contributions they make to projects as part of their final application.

Atrayee Samanta, an undergraduate student at IEST Shibpur, India, curated microbiome studies for [BugSigDB](#), a comprehensive database of published microbial signatures. Daena Rys, a computer science student from Cameroon, worked on issues within the [miaverse](#), an ecosystem based on (Tree)SummarizedExperiment for microbiome bioinformatics. Sonali Kumari, an IGDTUW student from New Dehli, India, converted Sweave vignettes in Bioconductor packages to R Markdown for [Sweave2Rmd](#). You can read more about their experiences on the Bioconductor blog at [Our Journey as Outreachy Interns with Bioconductor](#).

Outreachy will also fund three internships with Bioconductor for the December 2023 - March 2024 Outreachy cohort. Chioma Onyido, Ester Afuape, and Peace Sandy of Nigeria will curate microbiome studies for BugSigDB.

References

- Aganezov, Sergey, Stephanie M. Yan, Daniela C. Soto, Melanie Kirsche, Samantha Zarate, Pavel Avdeyev, Dylan J. Taylor, et al. 2022. "A Complete Reference Genome Improves Analysis of Human Genetic Variation." *Science* 376. <https://doi.org/10.1126/science.abl3533>.
- Cheng, Jun, Guido Novati, Joshua Pan, Clare Bycroft, Akvilė Žemgulytė, Taylor Applebaum, Alexander Pritzel, et al. 2023. "Accurate Proteome-Wide Missense Variant Effect Prediction with AlphaMissense." *Science* 381 (September). <https://doi.org/10.1126/science.adg7492>.
- Ucar, Iñaki, and Dirk Eddelbuettel. 2021. "Binary r Packages for Linux: Past, Present and Future." <https://arxiv.org/abs/2103.08069>.

Bioconductor Core Developers

Massachusetts, New York, Seattle
bioconductorcoreteam@gmail.com

R Project Sprint 2023

by Heather Turner and Gabriel Becker

Abstract R Project Sprint 2023 was a three-day event at the University of Warwick, UK, that brought together novice and experienced contributors to work alongside members of the R Core Team. 55 members of the R community participated, with external contributors selected to balance technical expertise and provide opportunities for members of historically under-represented groups. Participants worked collaboratively on contributions to base R and on infrastructure supporting contribution. Several small tasks were completed within the duration of the sprint, whilst significant steps were made on larger projects. The event provided a unique opportunity for external contributors to learn about the R development process and to develop their contribution skills.

1 Introduction

R Project Sprint 2023 was a three-day event hosted at the University of Warwick, UK. The aim of the event was to bring novice and experienced contributors together to work collaboratively with members of the R Core Team, who maintain and develop the code and documentation that forms the base distribution of R ("base R").

2 Participants

All members of the R Core Team were invited to the event and 11 were able to participate. Another 13 participants were invited/pre-selected - these included local organizers, representatives from sponsors, and experienced contributors. The remaining 31 participants - along with a few more who were ultimately unable to participate - were selected from a pool of 71 self-nominated applicants. Figure 1 shows group photos taken on Day 2 and Day 3 of the sprint, a full [list of participants](#) is on the sprint website. Participation was in-person by default, but exceptions were made in a few cases where travel was not possible, e.g., due to visa issues. The number online was higher than anticipated due to travel disruptions; in the end seven people participated online.

Members of demographic groups underrepresented within the contributor community were encouraged to apply for a place, by promoting the event to affinity groups (R-Ladies, MiR, RainbowR, AfricaR, ArabR, AsiaR, and LatinR) and by direct communication with potential participants. Figure 2 shows the geographical distribution of all 55 participants. There were 16 from Europe with 8 from the UK; 13 from North America with 12 from the USA; 7 from Asia with 5 from India; 6 from Latin America with 3 from Argentina; 5 from Africa with 2 from Nigeria; 4 from Oceania - all from New Zealand, and 3 from the Middle East.

We have further information from the nomination form, which was completed by 40 of the 44 invited/selected contributors. Over half (25/40) self-identified as belonging to one or more underrepresented groups. Figure 3 summarises the skills of these contributors as assessed by the selection committee, using data from the nomination forms. A "contributor level" was assigned based on self-ratings of familiarity with relevant concepts and processes, along with answers to free text questions about the applicant's experience and motivation. The committee deliberately selected participants to achieve the balance shown in the first plot of Figure 3: an equal number of advanced and novice contributors, with the remainder having intermediate expertise. The second plot summarises the potential for contribution to translations: 14 of the selected contributors expressed a specific interest in translation; 8 more were surmised to have potential based on country of residence. For the remainder (22) there was no evidence as we did not ask about this explicitly in the form.

3 Preparation

There were two sides to preparation for the sprint: gathering suitable tasks to work on and helping participants brush up their knowledge and skills.

We collected ideas for suitable tasks via the discussion forum on the R Project Sprint 2023 GitHub repository ([GitHub Discussions](#)). This provided a space for participating members of R Core to give feedback and for participating contributors to express an interest. Sprint participants could propose a project by adding a page to the [Projects](#) section of the sprint website. In the run up to the sprint, ideas and projects were transferred to [issues on the sprint GitHub repository](#) along with further last-minute



Figure 1: Photos of sprint participants on Day 2 (top) and Day 3 (bottom) including online participants on screen (not all participants photographed).

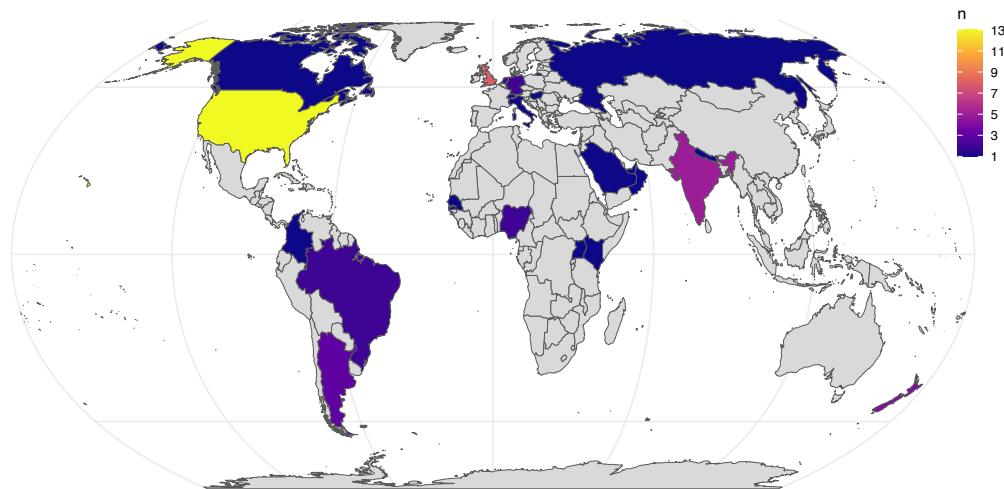


Figure 2: Choropleth showing the distribution of participants on the world map.

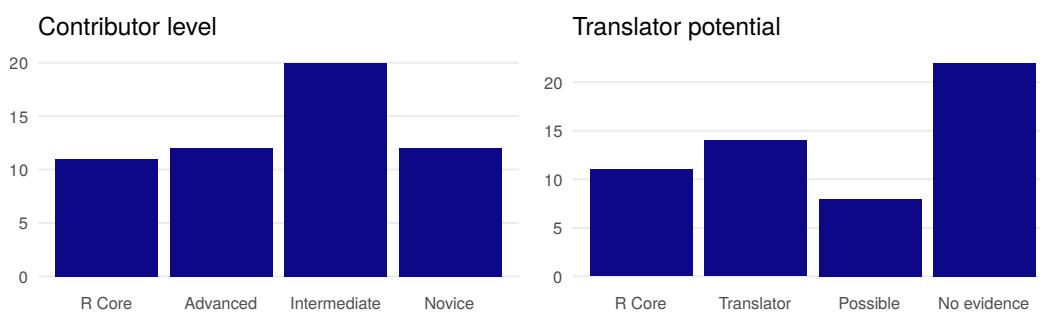


Figure 3: Skills of external contributors as judged by the selection committee. Left: level of expertise in R contribution. Right: potential as a translator of English to other languages.

ideas from core developers and members of the R Contribution Working Group (RCWG). This enabled participants to assign themselves to issues and provided a way to track tasks during the sprint.

In the lead-up to the sprint, participants were pointed to resources created by the R Core Team and the RCWG, including the R Blog post on [reviewing bugs](#) and the useR! 2021 tutorials on [analysing bugs/contributing patches](#) and [translating messages in R](#). In addition, participants were encouraged to engage with relevant events, in particular the [Debugging in R](#) tutorial run by Shannon Pileggi for R-Ladies Remote, and the [C Book Club for R Contributors](#) and [R Contributor Office Hours](#) run by the RCWG.

By the time of the sprint, participants were expected to be able to build R from source on the laptop they brought along. People new to this were pointed to the [R-admin manual](#), the [R Dev Guide](#) and the prototype [GitHub Codespace](#) which provides a virtual environment in which to build R - this was demonstrated in one of the contributor office hours.

4 Format

The sprint began with a hybrid evening welcome event where Martyn Plummer gave some opening remarks on contributing to the R Project, then participants split into small groups to chat with a member of R Core. An informal drinks reception then followed for in-person participants.

Each sprint day started with a kick-off session and ended with a report-back session, both hybrid to include our online participants. On the first day, R Core members gave short talks in these sessions, introducing themselves and their work for the R Project, and laying out their broad interests for work during the sprint. Beyond these introductions, participants used the daily kick-off sessions to collectively match people to tasks.

The remaining day time was spent working in small groups, sometimes arranging hybrid meetings to discuss specific issues.

On the second evening, in-person participants enjoyed a conference dinner, whilst on the final evening the sprint participants joined the Warwick R User Group for a hybrid meetup to present progress made thus far at the sprint, this was followed by a buffet dinner for in-person participants.

5 Translation

Translating English messages into other languages to enhance localization of R was a core activity at the sprint.

In 2022, Gergely Darócz set up a prototype Weblate instance <https://translate.rx.studio> to provide a modern, user-friendly interface for contributing translations to the R Project. Under the RCWG, he has developed and maintained this service, working with Michael Lawrence of R Core to incorporate translations submitted via Weblate into the R sources. Sprint participants created a new set of [guidelines for translators](#) and a new section in the R Dev Guide on [How to contribute new translations](#). Several new features were enabled on the Weblate instance, including translation memory, hyperlinking to the source string location and dedicated reviewers to approve translations. New components were added, so that the instance not only covers base R (messages, warnings, errors and the Windows GUI), but also the Mac GUI and recommended packages.

Figure 4 gives a summary of activity on Weblate during the sprint: around 2000 messages were changed over 14 languages. The vast majority of this activity can be attributed to the sprint directly or indirectly - the Hungarian translations were imported from earlier work in 2011 and the Turkish translations were made by external contributors after the Mac GUI component was added.

6 Code and Documentation

The remaining activity at the sprint related to code and documentation in base R. Code issues were split into topics to help organize work groups: accessibility, graphics, packages, statistics, translation and low-level. The translation issues here related to infrastructure maintained by the R Core Team, as opposed to Weblate. The low-level topic was a catch-all that covered utility functions and/or issues that required advanced technical expertise, e.g., in C.

Figure 5 shows the progress of issues at the end of the sprint and two months after. An issue is considered closed if a corresponding bug report on R's Bugzilla (<https://bugs.r-project.org>) was closed, if a corresponding patch was committed to base R, or if the issue was closed by an update to a CRAN package. By the end of the sprint, ten issues had been closed. Seven of these

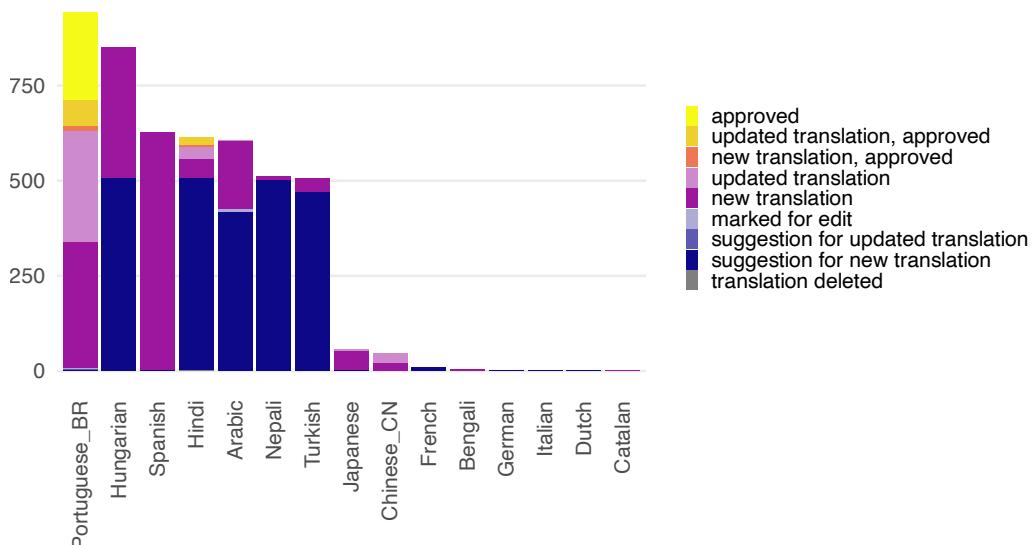


Figure 4: Changes in the R Project components on Weblate during the three days of the sprint

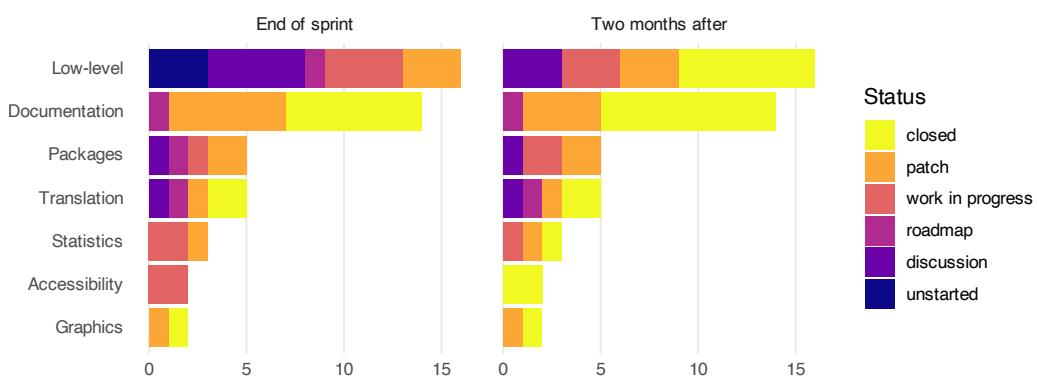


Figure 5: Status of issues at the end of the sprint and two months after

were documentation bugs, including one that was closed just before the sprint due to a participant reviewing issues in preparation. However, progress had been made on thirty-four other issues, ranging from discussing the issue, through defining a roadmap, to work in progress or proposing a patch. Two months after the sprint, another twelve issues had been closed and six more had progressed status (e.g., from roadmap to work in progress). These eighteen issues included three that were not started at the sprint, but worked on soon after as follow-up to a partial fix or due to participants reviewing the progress of sprint issues.

The low-level issues included new functionality, e.g., [supporting custom parallel backends](#); refactoring, e.g., [improving the speed of scalar random number generation](#); improving behaviour, e.g., [better formating of complex numbers](#), and bug fixes, e.g. [managing long names when creating tarballs](#).

Participants working on documentation began by triaging all open documentation bugs on Bugzilla to identify ones that could be closed without fixing, or ones that appeared straight-forward to fix, hence the high closure rate for these issues. Some closed bugs had been open for several years.

Package-related issues included adding [support for defining vignette order](#), [improving messages to CRAN maintainers](#), and [caching installed packages](#).

Translation-related issues included identifying [untranslated strings in the R source files](#), and creating a roadmap towards [internationalization of help pages](#). The R Consortium are funding a project by participants Elio Campitelli and Renata Hirota for the first step on this roadmap.

Statistics issues included [improving the behaviour of t.test.formula\(\)](#) and [wilcox.test.formula\(\)](#) for paired tests and enhancing [sample.int\(\)](#) for unequal probability sampling, for which a prototype package was developed after the sprint for testing.

Accessibility focused on two issues faced by screenreader users: logging base graphics and logging

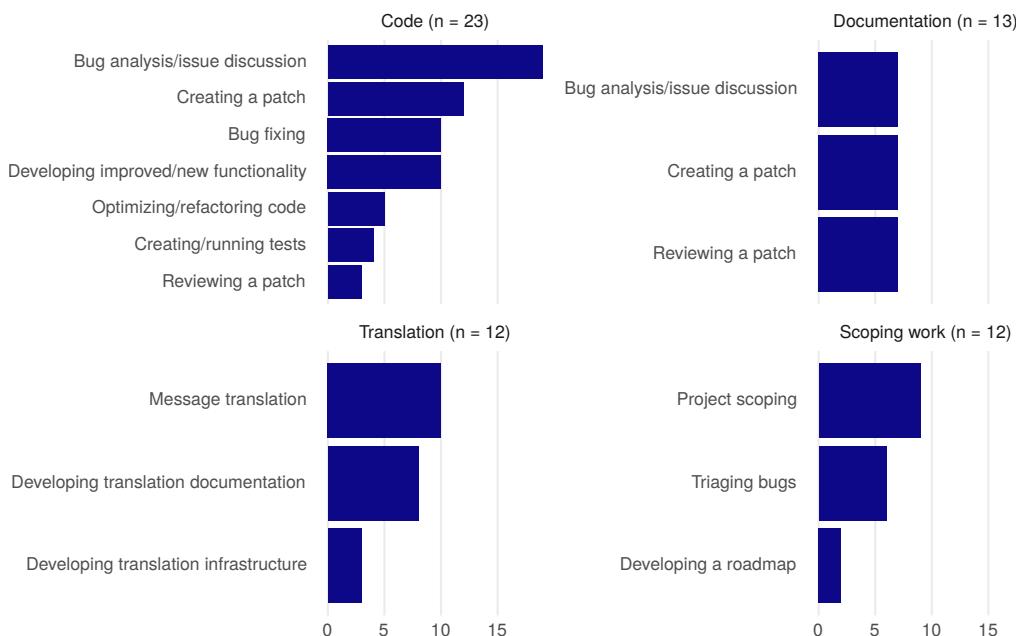


Figure 6: Activities of external contributors, based on 32 responses to the participant survey.

R sessions. Functions resulting from this work are now implemented in [BrailleR](#).

Finally there were two issues related to graphics, one fixed during the sprint implementing [3-digit hex colors](#) and one larger project on [adding alpha masks to the Quartz graphics device](#).

There were more issues prepared for the sprint than are summarised here, but they were not taken up at the sprint. In some cases there was insufficient support from R Core to pursue the idea, or it was considered out of scope for the sprint, or there were no available participants with relevant skills and/or bandwidth to take the idea forward. Often participants were interested in multiple issues and were encouraged to favour issues/topics where larger group discussions were taking place, to take advantage of everyone being together.

7 Participant experience

As well as aiming to make progress on contributions to the R Project, the sprint was intended to develop participants' knowledge and experience in contribution and motivate them to continue contributing after the sprint. We further looked to improve visibility and networking between the R Core Team and the R community (as represented by sprint participants).

Figure 6 summarises the activities engaged in at the sprint, for 32 out of 44 external contributors that responded to a post-sprint survey. Around two-thirds were involved in working on code issues and around a third worked on documentation and/or translation. Scoping work was also an important activity, in which a third of contributors engaged.

Figure 7 summarises the activities that contributors engaged in for the first time either during the sprint, or to prepare for or follow up on work done at the sprint. For around two-thirds of contributors it was the first time they had discussed a bug or issue with an R Core member, whether online or in person. About a third commented on Bugzilla for the first time and around a quarter posted their first patch. Around half the contributors built R from source for the first time, either on their own laptop or in the GitHub Codespace (or both) and for about half the contributors it was their first time working on R, C, or Rd (R documentation) files in base R.

8 Organizers and sponsors

The organization of the sprint was led by Heather Turner, as part of a research fellowship funded by the UK Engineering and Physical Sciences Research Council. This fellowship provided core funding and was supplemented by additional sponsorship:

- Platinum Sponsor (R Core travel): the R Foundation.

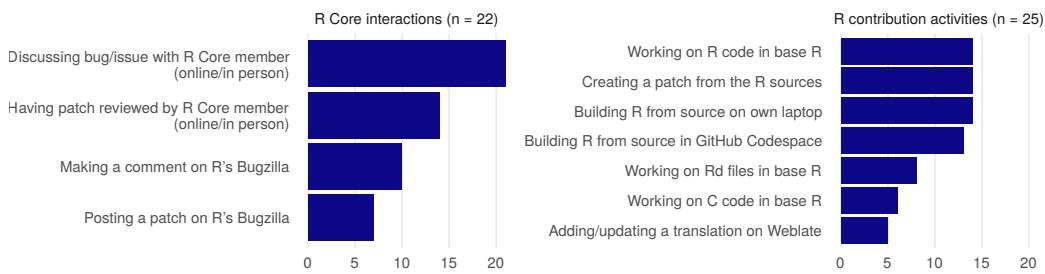


Figure 7: R core interactions and R contribution activities that external contributors engaged in for the first time, based on 32 responses to participant survey.

- Gold sponsors (evening events, participant travel): the R Consortium; the Centre for Research in Statistical Methodology, University of Warwick, and Posit.
- Silver Sponsors (participant travel): Seminar for Statistics, ETH Zurich; Rx Studio; The Prostate Cancer Clinical Trials Consortium, and Google.

This sponsorship funded travel, accommodation and subsistence for all participants.

Martyn Plummer and Ella Kaye completed the local organizer team. The selection committee was made up of Heather, Ella, and Gabe Becker of the RCWG.

Members of the RCWG helped with the planning, especially Gabe Becker who also helped gather issues in the run up to the sprint.

9 Summary

R Project Sprint 2023 was a very collaborative event, where external contributors had a unique opportunity to work closely with R Core members. Good progress was made across a broad range of issues with continued impact after the sprint. The feedback from both R Core and external participants was very positive, e.g.,

Thank you for organizing an incredible sprint and creating space for newcomers

There were many different parts that contributed so well to make it very productive, envigourating, and motivating

From arrival to departure, everything was seamless and I had a great time discovering what it takes to maintain R.

I'm exhausted but also super excited by all the work we did and that I take as homework.

Several participants - as well as R community members that could not attend this time - asked when we would hold a repeat event. Finding funding for ~50 people from around the world to attend a 3-day sprint is quite a challenge. So in the short term we plan to run 1-day events in collaboration with in-person conferences. Whilst this will limit the scope of tasks that can be tackled, we can benefit from people already travelling for the conference, with conference scholarship schemes helping to support inclusion.

10 Links

- Sprint website: <https://contributor.r-project.org/r-project-sprint-2023/>
- GitHub repository: <https://github.com/r-devel/r-project-sprint-2023>

Heather Turner

University of Warwick

Coventry, United Kingdom

<https://warwick.ac.uk/heatherturner>

ORCID: 0000-0002-1256-3375

h.turner.1@warwick.ac.uk

Gabriel Becker
Independent consultant
San Francisco, USA
gabembecker@gmail.com