The R Journal

Volume 14/4, December 2022

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

Contents

Editorial 4

Contributed Research Articles

knitrdata: A Tool for Creating Standalone Rmarkdown Source Documents. 6
netgwas: An R Package for Network-Based Genome Wide Association Studies 18
robslopes: Efficient Computation of the (Repeated) Median Slope 38
Generalized Mosaic Plots in the **ggplot2** Framework 50
A Study in Reproducibility: The Congruent Matching Cells Algorithm and cmcR Package. 79
Bootstrapping Clustered Data in R using lmeresampler 103
DGLMExtPois: Advances in Dealing with Over and Under-dispersion in a Double GLM Framework. 121
Making Provenance Work for You 141
remap: Regionalized Models with Spatially Smooth Predictions 160
HostSwitch: An R Package to Simulate the Extent of Host-Switching by a Consumer . 179
OTrecod: An R Package for Data Fusion using Optimal Transportation Theory 195
populR: a Package for Population Downscaling in R 223
dycdtools: an R Package for Assisting Calibration and Visualising Outputs of an Aquatic Ecosystem Model 235
SurvMetrics: An R package for Predictive Evaluation Metrics in Survival Analysis . . 252
Limitations in Detecting Multicollinearity due to Scaling Issues in the mcvis Package . 264
ppseq: An R Package for Sequential Predictive Probability Monitoring 280
pCODE: Estimating Parameters of ODE Models 291
TreeSearch: Morphological Phylogenetic Analysis in R 305
The openVA Toolkit for Verbal Autopsies 316
BayesPPD: An R Package for Bayesian Sample Size Determination Using the Power and Normalized Power Prior for Generalized Linear Models. 335

News and Notes

Bioconductor Notes 352

Changes on CRAN 356
News from the Forwards Taskforce 358
Changes in R 361
R Foundation News 365

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:
Catherine Hurley, Maynooth University, Ireland

Executive editors:

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 Catherine Hurley

On behalf of the editorial board, I am pleased to present Volume 14 Issue 4 of the R Journal. In response to our increased number of publications (now about 80 per year), this is the first year the R Journal moved to four issues. In addition, over the last year our software supporting the publication process has substantially improved. As a result of these developments, accepted articles are now published in a more timely fashion.

We have also added some R Journal reporting summaries to our webpage, and will expand on this in future. For published articles, the time from first submission to accept averages at under a year. Currently, just a very small number of 2021 submissions await a final decision, and even for 2022 submission, the majority of articles have received an accept/reject decision.

This is my last issue as Editor-in-Chief. Simon Urbanek takes over as Editor-in-Chief for 2023, having served as an Executive Editor since 2020. Simon is a huge contributor to the R community as a long-standing member of the R-core team. During his term, he plans to build further infrastructure to further streamline the R Journal submission and review process.

Dianne Cook recently finished her Editorial board term. She has been a hugely positive influence on the R Journal over the last number of years, and has been the driving force behind new software for journal operations and article production. Personally, I was very grateful for the guidance she provided as I took on the daunting role of EIC.

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

The R Journal Editors recently held our first meeting with our Editorial Advisory Board. The purpose of the board is to assist with continuity across editors and to provide independent advice. We thank them for their time and guidance.

Behind the scenes, several people assist with the journal operations. Mitchell O'Hara-Wild continues to work on infrastructure, H. Sherry Zhang continues to develop the **rjtools** package under the direction of Professor Dianne Cook. In addition, articles in this issue have been carefully copy edited by Hannah Comiskey.

In this issue

News from CRAN, Rcore, Bioconductor, RFoundation and RForwards are included in this issue.

This issue features 20 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

Reproducible Research

- knitrdata: A Tool for Creating Standalone Rmarkdown Documents
- Making Provenance Work for You
- A Study in Reproducibility: The Congruent Matching Cells Algorithm and cmcR package
- The openVA Toolkit for Verbal Autopsies

Multivariate Statistics, Visualisation

- Bootstrapping Clustered Data in R using lmeresampler
- Generalized Mosaic Plots in the ggplot2 Framework
- robslopes: Efficient Computation of the (Repeated) Median Slope

Econometrics

- DGLMExtPois: Advances in Dealing with Over and Underdispersion in a Double GLM Framework
- Limitations of the R mcvis package

Spatial Analysis

- remap: Regionalized Models with Spatially Smooth Predictions
- populR: A Package for Population Down-Scaling in R

Ecological and Environmental analysis

- HostSwitch: An R Package to Simulate the Extent of Host-Switching by a Consumer
- dycdtools: an R Package for Assisting Calibration and Visualising Output

Statistical Genetics

- SurvMetrics: An R package for Predictive Evaluation Metrics in Survival Analysis
- netgwas: An R Package for Network-Based Genome Wide Association Studies

Bayesian inference

- BayesPPD: An R Package for Bayesian Sample Size Determination Using the Power and Normalized Power Prior for Generalized Linear Models
- ppseq: An R Package for Sequential Predictive Probability Monitoring

Other

- pCODE: Estimating Parameters of ODE Models
- TreeSearch: Morphological Phylogenetic Analysis
- OTrecod: An R Package for Data Fusion using Optimal Transportation Theory

Catherine Hurley
Maynooth University

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

knitrdata: A Tool for Creating Standalone Rmarkdown Source Documents

by David M. Kaplan

Abstract Though Rmarkdown is a powerful tool for integrating text with code for analyses in a single source document exportable to a variety of output formats, until now there has been no simple way to integrate the data behind analyses into Rmarkdown source documents. The `knitrdata` package makes it possible for arbitrary text and binary data to be integrated directly into Rmarkdown source documents via implementation of a new `data` chunk type. The package includes command-line and graphical tools that facilitate creating and inserting `data` chunks into Rmarkdown documents, and the treatment of `data` chunks is highly configurable via chunk options. These tools allow one to easily create fully standalone Rmarkdown source documents integrating data, ancillary formatting files, analysis code and text in a single file. Used properly, the package can facilitate open distribution of source documents that demonstrate computational reproducibility of scientific results.

Introduction

The basic principles of open science are that the data, research methodologies and analysis tools (e.g., the specific computational tools) used for scientific research should be made publicly available so that others can confirm and validate scientific analyses. Open science is particularly important for studies and disciplines for which true experimental replication is often difficult or impossible due to spatial, temporal or individual specificity [e.g., we cannot replicate Earth; Powers and Hampton (2019)]. In these cases, computational reproducibility, i.e., the ability to reproduce analytic results given the original data and analysis code, can still be achieved and can provide significant credibility to results (Powers and Hampton 2019). Though scientists, governments and journals often place great emphasis on access to raw data (Cassey and Blackburn 2006; Lowndes et al. 2017), it is important to remember that computational reproducibility can only be assured if data, methods, computational tools and the relationships between these are *all* openly accessible. Even when data are made publicly available, there are often significant gaps between the Methods section of a publication and the raw data that complicate reproducibility without access to the detailed code used to generate results. It is, therefore, essential for computational reproducibility that the code used to generate results be distributed along with the data and the publication itself. Though there are a number of potential ways to distribute all these elements together, probably the most common current approach is to place the data in a publicly accessible data store (e.g., Dryad) and to associate the code with the publication via the supplementary material and/or by including it in the data store. Though this is a perfectly viable approach that can greatly enhance transparency of research, it physically separates data from analysis code and interpretation of results, potentially leading to confusion and/or loss of information regarding how these different element interrelate. At times, it would be more convenient, transparent and/or effective to join all the elements into a single document. The R package presented here, **knitrdata** (Kaplan 2020a), provides tools for doing just that - integrating data directly into Rmarkdown source documents so that data, code for analyses and text interpreting results are all available in a single file.

Rmarkdown (Allaire, Xie, McPherson, et al. 2022) has become an increasingly popular tool for generating rich scientific documents while maintaining a source document that makes explicit the relationship between text and specific analyses used to produce results (Xie 2014; Lowndes et al. 2017). In a nutshell, Rmarkdown source documents are text documents comprised of two major elements: structured text that make up the headings and paragraphs of the document, and blocks of code (typically, but not exclusively, R code) for doing analyses and generating figures and tables. Rmarkdown source documents can be processed into a variety of final output formats, including PDF documents formatted for scientific publication. During this processing, the blocks of code in the source document are executed and used to augment the final output document with figures, tables and analytic results. In addition to providing a single source document that includes both written text and code for carrying out analyses, Rmarkdown has other benefits for open science, such as requiring the user to provide fully functioning code that runs from start to end without errors and facilitating reuse and updating of documents when new data arrives.

Until now, however, it has been difficult to integrate the raw data itself that are the bases for analyses directly into Rmarkdown source documents. Typically, data are placed in external files that are accessed via R code contained in the Rmarkdown source document that is executed during the knitting. As previously mentioned, this has the disadvantage of physically separating data from analysis code and text contained in the Rmarkdown source document, potentially leading to confusion

and/or information loss. Furthermore, on a practical level, it often can be extremely convenient to merge all pertinent information into a single source document (e.g., to facilitate collaboration on an Rmarkdown source document). `knitrdata` provides a simple mechanism for integrating arbitrary text or binary data directly into Rmarkdown source documents, thereby allowing one to create standalone source documents that include all the elements necessary for conducting analyses. This integration is done with minimal additional formatting of the data (e.g., allowing one to insert comma-separated value (CSV) data without escaping quotation marks directly into Rmarkdown documents) and in a way that clearly visually separates data from R code, thereby facilitating comprehension of the different elements that contribute to analyses. `knitrdata` also facilitates encryption of data integrated into Rmarkdown source documents, thereby allowing one to merge data with analysis code and text even in cases where industrial or ethical privacy constraints restrict data access to a specific group of individuals.

Below, I briefly provide a conceptual overview of how `knitrdata` works, presenting some simple examples of its use and the tools available to facilitate integrating data into Rmarkdown source documents. I then discuss typical use cases and workflows for development of Rmarkdown source documents with `knitrdata`, as well as a number of potential caveats for its use. I conclude by reflecting on the value of `knitrdata` for achieving computational reproducibility and its place within the growing pantheon of tools that make Rmarkdown an increasingly essential tool for research.

knitrdata installation and usage

The `knitrdata` package is available on CRAN, though the latest version can be installed from github using the **remotes** (Csárdi et al. 2021) package:

```
remotes::install_github("dmkaplan2000/knitrdata",
                        build_vignettes=TRUE)
```

Once the package has been installed, all that is needed to use the functionality provided by the package in a Rmarkdown source document is to load the library at the start of the document, typically in the `setup` chunk:

```
library(knitrdata)
```

Conceptual overview of `knitrdata`

To understand how `knitrdata` works and the functionality it provides, one must first understand some of the terminology and functioning of Rmarkdown itself. As previously mentioned, Rmarkdown documents are a combination of text written in *markdown*, a simple, structured text format that can be translated into a large number of final output formats, and code for doing analyses that can augment the final output document with analytic results, tables and figures. The code is contained in specially delimited blocks, referred to as *chunks*. For example, adding the following to an Rmarkdown document:

````
```{r}
plot(-5:5,(-5:5)^2,type="l")
```
````

would add a plot of a parabola to the final output document. The process of translating a Rmarkdown document into a final output document is known as *knitting*, and this process is carried out using (often implicitly via RStudio) the **knitr** package (Xie 2015).

Though code chunks typically contain R code, `knitr` supports a large number of other *language engines*, allowing one to integrate analyses in a number of other computer languages, including C, Python and SQL. For example, one could use the SQL language engine to import the contents of a database table into the R environment by including the following chunk in a Rmarkdown source document:

````
```{sql connection="dbcon",output.var="d"}
SELECT * FROM "MyTable";
```
````

During knitting, this will create in the R environment a variable `d` containing the contents of the table `MyTable` accessible via the (previously created) active R database connection `dbcon`. Note that the name of the database connection and the name of the output variable are supplied in the chunk header via what are known as *chunk options*. Though this database table could be imported into the R environment without the SQL language engine using R code:

````
```{r}
d = dbGetQuery(dccon,"SELECT * FROM \"MyTable\";")
```
````

the use of the SQL language engine has both practical and conceptual advantages. On the practical side, it avoids the need to escape quotation marks and allows text editors to recognize and highlight the code as SQL, both of which becoming increasingly valuable as the length and complexity of SQL queries increase. On the conceptual side, using the SQL engine visually separates database queries from R code and text, thereby better communicating the structure and functioning of analyses in Rmarkdown documents.

The `knitrdata` package works in many ways analogously to the SQL language engine, adding a new `data` language engine to the list of language engines known to `knitr` that is specifically designed to import raw "data" into the R environment and/or export it to external files. Here the term "data" is used in a very wide sense, including not only standard data tables (e.g., CSV text files) or binary data (e.g., RDS files, NetCDF files, images), but also text and formatted text (e.g., XML files, BibTeX files). For example, placing the following chunk in a Rmarkdown source document will, during the knitting process, create in the R environment a data frame `d` containing the contents of the comma-separated values (CSV) data in the chunk (provided that the `knitrdata` package has been previously loaded as described above):

````
```{data output.var="d",loader.function=read.csv}
name,score
David M. Kaplan,1.2
The man formerly known as "Prince",3.4
Peter O'Toole,5.6
```
````

As with the SQL language engine, the name of the output variable for the chunk is supplied with a chunk option and in this example a `loader.function` option instructs `knitrdata` how to translate the contents of the chunk into a usable R object (in this example the R function `read.csv` is used to translate the CSV data into a data frame).

There are of course a number of other ways that such a simple data table could be imported into the R environment, including via an external data file or directly in R code, one approach to which might be:

````
```{r}
d = read.csv(textConnection(
"name,score
David M. Kaplan,1.2
The man formerly known as \"Prince\",3.4
Peter O'Toole,5.6
"))
```
````

However, using the `data` language engine has much the same practical and conceptual advantages as the SQL data language engine, avoiding the need for escaping certain characters and visually separating data from code, both of which become increasingly valuable as dataset size increases.

Incorporating binary data into Rmarkdown source documents is a bit more complicated as the data must first be *encoded* as ASCII text (see the Section below on Binary data chunks for details), but the basic principles are the same - encoded binary data is incorporated into a `data` chunk and chunk options are used to tell `knitrdata` how to decode the data and load it into the R environment during knitting (see Table 1 for a full list of `data` chunk options). There is also the possibility of saving `data` chunk contents out to external files using the `output.file` chunk option. This option is particularly useful for integrating into Rmarkdown source documents ancillary text files used in the final formatting of the output of the knitting process, such as BibTeX files with references, LaTeX style files for PDF output and CSS style files for HTML output. For example, the following chunk would export a BibTeX reference to a file named `refs.bib`, taking care not to overwrite an existing file with

the same name [though note that similar functionality can also be achieved with the `cat` language engine; Xie, Dervieux, and Riederer (2020)]:

````
```{data output.file = "refs.bib", eval=!file.exists("refs.bib")}

@book{allaireRmarkdownDynamicDocuments2020,
  title = {Rmarkdown: {{Dynamic}} Documents for r},
 author = {Allaire, JJ and Xie, Yihui and McPherson, Jonathan and Luraschi, Javier and Ushey, Kevin and Atkins, Ar
  year = {2020}
}
```
````

As code chunks are processed during knitting before generating the final output document, these files can be generated at any point during the knitting process using `data` chunks (in particular, it is often most practical to place this information at the end of a Rmarkdown document).

Table 1: Full list of knitrdata chunk options.

Chunk option	Description
`decoding.ops`	A list with additional arguments for `data_decode`. Currently only useful for passing the `verify` argument to gpg::`gpg_decrypt` (Ooms 2022) for gpg encrypted chunks.
`echo`	A boolean indicating whether or not to include chunk contents in Rmarkdown output. Defaults to `FALSE`.
`encoding`	One of `'asis'`, `'base64'` or `'gpg'`. Defaults to `'asis'` for `format='text'` and `'base64'` for `format='binary'`.
`eval`	A boolean indicating whether or not to process the chunk. Defaults to `TRUE`.
`external.file`	A character string with the name of a file whose text contents will be used as if they were the contents of the data chunk.
`format`	One of `'text'` or `'binary'`. Defaults to `'text'`.
`line.sep`	Only used when `encoding='asis'`. In this cases, specifies the character string that will be used to join the lines of the data chunk before export to an external file, further processing or returning the data. Defaults to `knitrdata::platform.newline()`.
`loader.function`	A function that will be passed (as the first argument) the name of a file containing the (potentially decoded) contents of the data chunk.
`loader.ops`	A list of additional arguments to be passed to `loader.function`.
`max.echo`	An integer specifying the maximum number of lines of data to echo in the final output document. Defaults to 20. If the data exceeds this length, only the first 20 lines will be shown and a final line indicating the number of ommitted lines will be added.
`md5sum`	A character string giving the correct md5sum of the *decoded* chunk data. If supplied, the md5sum of the decoded data will be calculated and compared to the supplied value, returning an error if the two do not match.
`output.file`	A character string with the filename to which the chunk output will be written. At least one of `output.var` or `output.file` must always be supplied.
`output.var`	A character string with the variable name to which the chunk output will be assigned. At least one of `output.var` or `output.file` must always be supplied.

Using `data` chunks, just about any data or information that would typically be stored in external files can be integrated directly into Rmarkdown source documents. In particular, this permits creating standalone Rmarkdown source documents that can be knitted without need for external data files, thereby uniting text, code and data in a single source document.

Note that this is different from the `self_contained` YAML header option permitted by some Rmarkdown output formats, notably HTML output formats. This option attempts to create a single *output* file that contains everything needed to visualize the final output document (e.g., in the case of HTML documents, the output HTML file will contain any CSS styles, javascript libraries and/or images used by the document), but it says nothing about whether or not external files are needed to

knit the Rmarkdown source document (i.e., it is relevant to the output side of knitting, not the input side). In fact, a source document can be standalone in that all data and formatting files needed for knitting are incorporated within it using `data` chunks, but the final output (HTML) document may not be self contained because it relies on external files or libraries for visualization, and vice-versa (i.e., standalone source documents and standalone output documents are two separate and independent concerns).

Under the hood, the way `knitrdata` works is by adding (using the `knitr::knit_engines$set()` function) to the list of language engines that `knitr` maintains internally a `data` entry that points to a function inside the `knitrdata` package that processes `data` chunks (specifically the `eng_data` function, though users would typically not interact directly with this function). When knitting a Rmarkdown document, `knitr` will call this function each time a `data` chunk is encountered, passing it both the textual contents of the chunk and any chunk options. The function then uses this information to process the chunk, decoding it if necessary (via the `format` and `encoding` chunk options) and returning it as either a variable in the R environment (`output.var` chunk option) and/or an external file (`output.file` chunk option) after any additional processing has been carried out (via, e.g., the `loader.function` chunk option).

Text data chunks

Though a basic example of a `data` chunk containing CSV tabular data has been presented in the previous section, it is useful to develop that example a bit more to better understand the functioning of `knitrdata`. The simplest `data` chunks contain plain text that is read, but not processed by `knitrdata`. For example, omitting the `loader.function` chunk option from the previously presented `data` chunk with CSV data produces a different outcome:

````
```{data output.var="txt"}
site,density
a,1.2
b,3.4
c,5.6
```
````

During the knitting process, this will place the text contents of the chunk into a R variable named `txt`, but no further processing of the text will be carried out (i.e., the variable `txt` will contain the literal text contents of the chunk, excluding the header with the chunk options and the tail). One could later convert the text into a R `data.frame` using the `read.csv` command in a R chunk placed after the `data` chunk:

````
```{r}
d <- read.csv(text=txt)
```
````

The `loader.function` chunk option used in the initial `data` chunk example above causes `knitrdata` to combine into one process the two steps of (1) reading in the chunk contents and (2) converting them into a usable R data object. Whereas the first of these steps, reading the chunk contents, is carried out for all `data` chunks, the second only occurs if the `loader.function` chunk option is given. `loader.function` should be a function that takes in the name of a file containing the chunk contents and returns the processed contents. Though `read.csv` is likely to be a common choice, there are many other possibilities including `readRDS`, `read.csv2`, `scan`, `png::readPNG` and custom, user-defined functions. One can also supply a list of additional input arguments to the loader function using the `loader.ops` chunk option (e.g., one could change the expected separator in CSV data using `loader.ops=list(sep="|")`).

Binary data chunks

Though text data chunks can integrate into Rmarkdown source documents many small- to medium-sized tabular data sets, binary data formats, such as RDS files, are more convenient for more complicated and/or larger data sets. Incorporating binary data into Rmarkdown documents requires additional steps relative to text data: encoding the binary data as text and telling the `data` chunk how to decode the encoded data. `knitrdata` provides tools for simplifying these two steps that currently support encoding and decoding of two widely-used encodings: `base64` and `gpg`. `base64` is a standard format for encoding binary data as ASCII text based on translation of 6 bits of information into one

of 64 alphanumeric and symbolic ASCII characters. Base64 encoded data looks like a somewhat intimidating jumble of characters, but the format is extremely widely used behind the scenes in many common web applications, such as email attachments and embedding images in HTML pages. In particular, `base64` is widely supported by a number of software packages and programming languages, including R, Python, Matlab and Julia, so `base64` encoded data is highly readable and likely to remain so for a very long time.

`gpg`, standing for GNU Privacy Guard, is a standard protocol for encrypting information so that only those with specific decryption keys can have access. This format can be used to ensure that only specific individuals can actually read and utilize the data contained in a Rmarkdown source document, as might be necessary when dealing with confidential (e.g., medical or trade-secret) data. Here, I focus primarily on `base64` encoding as this is the simplest and likely most common format for binary `data` chunks, and a full description of the configuration and use of GPG is beyond the scope of this document. The detailed use of `gpg` is, however, described in the package vignette.

Though `knitrdata` users rarely need to encode data by hand as the package provides graphical tools for this, it is instructive to have a basic understanding of the underlying functions for encoding and decoding data: `data_encode` and `data_decode`. `data_encode` takes the name of the file containing the data and the name of the encoding format, and it returns the encoded data that one would incorporate into a `data` chunk, either to the R command line or to a file. For example, if one saves the data frame `d` created in the previous section to a binary `RDS` file:

```
saveRDS(d,"data.RDS")
```

then one can encode this data as `base64` using:

```
b64 = knitrdata::data_encode("data.RDS","base64")
cat(b64)
```

```
#> H4sIAAAAAAAAA12OvQ+CMBDFKx+DGNTExPk2J1lc3HQwLsbIgInrBUokQmsKkbj5
#> PzsrXrEMekn72l/f693JY4zZzLEsZrt0ZO4x2s6XxCZ0sWiNvwbWJx1t8JYlsA9g
#> h9cchcEQnTkUKCCVquAqv8NFyFoAlhCqTMTc+PyQV1zBYRZJmXMCQ/336rloaz0w
#> Ok3bYjQVvfdM2BVY8NIMZBnoaNgZylgq/p+Kcyy7VAe9BCsMUqWzv/a+knXQNfJ1
#> owdtTdO8SN4fPb8RnS0BAAA=
```

This jumble of characters starting with `"H4sI"` is the base64 encoded contents of the binary file that one would place in a `data` chunk. For large files, it is often more practical to output the encoded data to a file by supplying the `output` argument:

```
knitrdata::data_encode("data.RDS","base64",
                       output="data.RDS.base64")
```

GPG encoding works similarly to base64 encoding, but one must change the format from `"base64"` to `"gpg"` and specify the encryption key (i.e., the receiver ID) to be used to encrypt the data.

Once one has the encoded data, one can use it in a `data` chunk by supplying the `format="binary"` chunk option and, optionally, an appropriate `loader.function` to convert the data into a R object:

````
```{data format="binary",output.var="d",loader.function=readRDS}
H4sIAAAAAAAAA12OvQvCMBDFz48OKn6A4Hybk11c3HQQFxE7VHA9aopimkgiFjf/
Z2etl5oOesO93I/3crdvA0ADmnXuAT8h2MWryYynIQ9MYfA1QIu1v6Tb6YCbENd0
kaQ8xvgoMCOFqTaZMPKOZ6VzhWQxMieVCO/rRuIqDG7HsdZSMOi5v+fPaVmLjtdR
WhaUV0HNhwNFmbD+oLqHTQcrg020Ef+pRJKtUhVsH+hKYWpc9tfeMjoPq0Vdt+jB
rSiKF8v7A6bdy9EtAQAA
```
````

During knitting, this chunk will be processed, decoding the encoded binary RDS file and loading it into the variable `d` using the `readRDS` function. `knitrdata` will by default assume that the encoding is base64 when `format="binary"`, but one can also specify the chunk option `encoding="base64"` for increased clarity. For GPG encoded data, one would use `encoding="gpg"`. As with text `data` chunks, one can alternatively output the decoded contents of the chunk to a file (`output.file` option) or return it to the R session as a `raw` binary vector (by not supplying a `loader.function`).

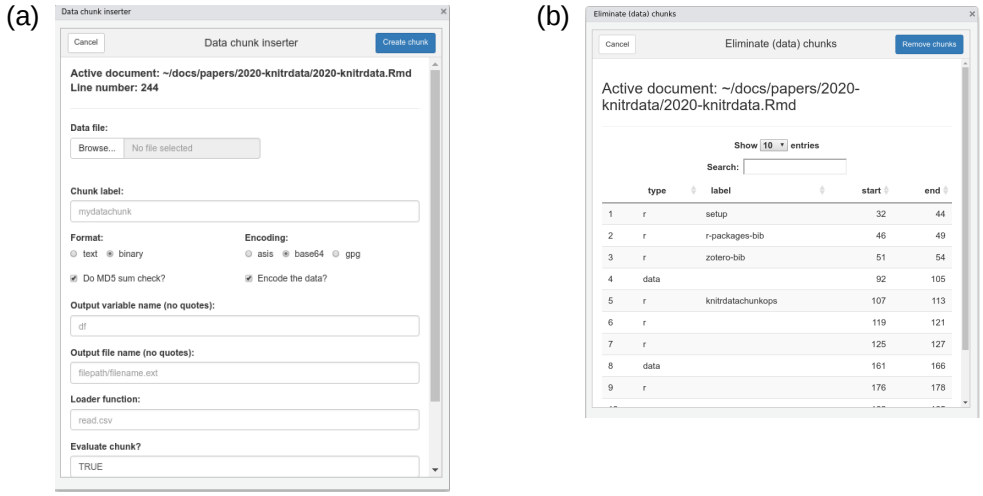


Figure 1: The (a) 'Insert filled data chunk' and (b) 'Remove chunks' RStudio add-ins included with knitrdata. The dialogues will open when selected from the 'Addins' menu of RStudio. They allow one to (a) insert a data chunk containing the contents of an existing external data file into an open Rmarkdown document, and (b) delete one or more chunks from an open Rmarkdown document.

RStudio add-ins for creating data chunks

As manually encoding data and creating `data` chunks can be complicated, particularly for large data files, `knitrdata` includes graphical RStudio add-ins that do all the hard work of incorporating data in Rmarkdown documents. The principal add-in is called `Insert filled data chunk` (Fig. 1a). Though its use is meant to be largely self-explanatory, an instructional video is available on YouTube (Kaplan 2020b). The basic idea is that one opens a Rmarkdown source document in the RStudio editor, places the cursor at the location one wants to insert a `data` chunk and then activates the add-in. The add-in prompts for the name of the data file to be incorporated, as well as values for various `data` chunk output and processing options. Based on the type of data file selected, the add-in will attempt to select or suggest various appropriate options. For example, if a RDS file is chosen, then format will be set to `binary`, encoding will be set to `base64` and the loader function will be set to `readRDS`. These defaults can be manually modified if not appropriate. The add-in also greatly facilitates and encourages the use of MD5 sum data integrity checks. After all options have been set, one clicks on `Create chunk` and an appropriately-formatted `data` chunk will be inserted in the active Rmarkdown source document at the cursor location.

`knitrdata` also provides a `Remove chunks` add-in that allows ones one to quickly delete unwanted (`data` and non-`data`) chunks (Fig. 1b), as well as a set of functions for command-line examination, creation and removal of chunks from Rmarkdown documents (e.g., `create_chunk`, `insert_chunk`, `list_rmd_chunks`, `remove_chunks`).

If one is not using RStudio to edit and knit Rmarkdown documents, then one can invoke the Skiny dialog to create `data` chunks directly from the command line using the `create_data_chunk_dialog` function contained in the `knitrdata` package. In this case, chunk contents will be (silently) returned on the command line for later insertion in a Rmarkdown document.

Use cases

There are a number of use cases for the functionality provided by `knitrdata`, primary among them providing a single source for public diffusion of all information related to a publication or report, and/or making collaboration on Rmarkdown source documents simpler by eliminating or reducing

the need for external files. A simple example of the prior is the Rmarkdown source document used to generate this publication, which includes text `data` chunks for the tabular data in Table 1, as well as the ancillary formatting files associated with the document (BibTeX and LaTeX style files), and encoded binary `data` chunks for the PNG images in Figs. 1 & 2.

A more complicated example is the Rmarkdown source document for Wain et al. (2021), publicly available on github. In this case, we wished to provide a permanent public record of the methods used in the paper and ensure that results could be verified, while at the same time respecting confidentiality agreements with respect to fine scale fishing activity data used in the paper. To achieve this we integrated the fine scale data in the Rmarkdown source document as an encrypted GPG `data` chunk. This approach may have value for a wide number of other studies using sensitive economic, social or medical data. To provide a complete record of the paper in a single document, we also integrated the Rmarkdown source document for the online supplementary materials into a `data` chunk within the Rmarkdown source document for the paper itself. As this supplementary materials document contains Rmarkdown chunks that would otherwise confuse the knitting process if integrated as raw text inside a `data` chunk, we base64 encoded this source document before including it in a `data` chunk. The document also contains `data` chunks for small data tables and for exporting to external files the ancillary formatting files required for knitting the document (BibTeX references, the LaTeX style file, the CSL citation style file, etc.). Finally, during the knitting process, the document also generates a lightweight version of itself that does not include the main data chunk, using the functionality of the `knitrdata` package to remove large data chunks. Overall, `knitrdata` provided a convenient way of generating a single document that contained all the necessary information for generating the final publication, thereby demonstrating computational reproducibility for the publication.

The uses of `data` chunks tend to fall into one of four general, not mutually-exclusive use cases:

1) Integration of ancillary formatting files into the Rmarkdown source document, thereby reducing the number of external files needed to knit a document
2) Inclusion of small- to medium-sized tabular data used in analyses and/or for tables
3) Inclusion of larger data sets using encoded binary data
4) Inclusion of confidential data using GPG-encrypted `data` chunks

Though the first of these use cases, integration of ancillary formatting files, can also be achieved with the `cat` language engine that is included with the `knitr` package (Xie, Dervieux, and Riederer 2020), `knitrdata` provides functionality that make this task easier and more secure. First, `knitrdata` allows for integrity checks on chunk contents that can control for unintentional modification of chunk contents (see the section on data integrity below). Second, RStudio add-ins provided by the `knitrdata` package facilitate the integration of data into Rmarkdown source documents and the use of integrity checks. Finally, encoding of text documents permits integrating files that contain Rmarkdown chunks or other formatting that would otherwise be problematic within a `cat` chunk.

The second of these use cases, tabular data, can also in principle be achieved using other tools in R, such as a `textConnection` as shown above or via functionality in the **tibble** package (Müller and Wickham 2022). Nevertheless, the use of `data` chunks is generally more ergonomic and flexible for anything but the smallest data tables as it allows the user to format data exactly as it would be in an external CSV file, without additional markup or the need to escape quotation marks. As an example, the information contained in Table 1 was implemented in the source document for this paper as a `data` chunk as it contains lots of quotations and formatting that would have been tedious to include using other approaches.

The third and fourth use cases for `data` chunks, involving encoded binary data, are unique to `knitrdata` and allow for integration of complex data sets that would otherwise be very difficult to include in a Rmarkdown source document.

Workflow

When and in what ways to use the functionality provided by `knitrdata` during the development of a Rmarkdown source document requires some thought and depends to some degree on the project goals. If the goal is to create a final Rmarkdown source document that demonstrates computational reproducibility of a set of results, then it may not be necessary or practical to use `data` chunks during the development stages of the project as the use of `data` chunks necessarily weighs down a Rmarkdown source document with information (e.g., binary data) that may not be immediately useful to authors during development. In this case, it may be best to work initially as one has always done, relying on external files for data and formatting. External data and formatting files can be incorporated in `data` chunks at the end of development when it is time to generate a final archival/public version of a Rmarkdown source document.

On the other hand, if the objective of using `knitrdata` also includes reducing the complexity of collaborating on a Rmarkdown source document by reducing the number of external files necessary for knitting a document, then certain types of `data` chunks can be incorporated in a Rmarkdown source document during the initial phases of development with little impact on authors. Small- to medium-sized tabular data sets can be incorporated and this can have the benefit of making the tabular data visually available during the development process. Similarly, most ancillary formatting files can be placed at the very end of the Rmarkdown source document as these are only used after all chunks have been processed and, therefore, will not encumber the development process. Larger data sets, and in particular binary data sets, are a bit more problematic as they necessarily appear in the Rmarkdown source document before the data is used for analyses and will introduce significant amounts of text that are not human readable into the Rmarkdown source document. For this reason it may be best to leave incorporation of these data until the final stages of development, though see the sections below on file size and readability for workarounds to these issues.

This latter workflow involving incorporation of `data` chunks in two distinct stages of development is what was used when creating the source document for Wain et al. (2021). Small data tables and formatting files were incorporated directly into the document from the start, but the larger data set that was the basis for statistical analyses and the Rmarkdown source document for the supplementary materials were only incorporated at the end of development to provide an archival source document for the paper capable of demonstrating computational reproducibility.

Caveats and concerns

There are a number caveats and concerns with respect to the use of `knitrdata`, all of which have some validity, but for which a number of simple approaches exist to limit their impact. Below, I discuss four of them: file size, document readability, data integrity and security.

Won't this create huge Rmarkdown files?

Incorporation of large data sets into `data` chunks will significantly increase the file size of Rmarkdown source documents, potentially making them more difficult to work with. Though it is unlikely to be practical to place extremely large data sets in Rmarkdown source documents, there are many contexts where data sets are sufficiently small so as to be included directly in a Rmarkdown source document. For example, the 8 years of fine scale fishing data used in Wain et al. (2021) added about 2 MB to the size of the Rmarkdown document when incorporated as a (compressed) RDS file, a size that is manageable and well within the limits of typical email attachments. RStudio currently will not open Rmarkdown documents larger than 5 MB in size, effectively limiting the amount of data that can be placed in a document unless one is willing to forgo graphical editing tools (larger documents can be rendered from the command line using the `rmarkdown::render` function, but not the more convenient and common "Knit" button of RStudio). Despite being in the era of big data, many scientific studies use primary data sets that are smaller than this size limit. As for Wain et al. (2021), many experimental or field studies may rely on data sets that are relatively small, and building a standalone Rmarkdown source document for these studies is an effective approach to documenting all quantitative information needed to reproduce results.

Won't Rmarkdown source documents become unreadable?

Large amounts of encoded binary data are undoubtedly not pretty to look at, but readability is not necessarily the primary benefit of using Rmarkdown. Rather, completeness and the articulation of text and analyses are the strengths of Rmarkdown, benefits that `data` chunks enhance as opposed to diminish. Many Rmarkdown documents are already a complex mix of text and code that is difficult to read and manage without the tools RStudio and other editors provide to navigate the document, such as the ability to jump between sections and chunks. `data` chunks are no different in this sense, and use of informative chunk labels can greatly facilitate document navigation. Furthermore, RStudio includes the possibility of hiding chunk contents with a single click (Fig. 2), which can be quite practical when dealing with large `data` chunks. Once hidden, `data` chunk contents can be ignored, allowing one to edit the document unhindered.

Won't a misplaced keystroke mess up my data?

It is possible to unintentionally corrupt a `data` chunk due to a misplaced keystroke, particularly if the data is encoded and not readily human readable. However, the use of navigation tools and hiding

Figure 2: Images demonstrating before (top) and after (bottom) a data chunk has been hidden from view in the RStudio editor. The top image shows two base64-encoded data chunks, one of which is hidden, whereas the other is visible. In the bottom image, both chunks have been hidden. The control for hiding chunk contents (top) and an indicator of a hidden chunk (bottom) are highlighted with red boxes.

of `data` chunk contents as described above (Fig. 2) can drastically reduce interaction with chunk contents, thereby limiting the possibility for error. Furthermore, there are a number of methods to validate chunk contents, the simplest of which is to do a MD5 sum check using functionality included in `knitrdata`. A MD5 sum is a very large number (typically encoded in hexadecimal) derived from a file's contents that has a vanishingly small probability of being equal to that of another file if the files are not identical. `data` chunks can include a `md5sum` chunk option that specifies a MD5 sum that will be checked against that of the decoded chunk contents, generating an error if the two do not match. In this way, data corruption can be swiftly identified and corrected. The `Insert filled data chunk` RStudio add-in will by default calculate and include a MD5 sum check when inserting binary `data` chunks (and such a check can be optionally requested for text `data` chunks) so that users can easily benefit from these checks without having to worry about the details.

Are there security concerns when using `knitrdata`?

Any time one runs code from a third party, there are security risks. Typically, code can write files to disk, potentially modifying essential files or installing malicious software. Rmarkdown source documents using the functionality of `knitrdata` are no different in this sense, though the practical risks may be more important as `knitrdata` may encourage users to knit entire documents to gain access to raw data and arbitrary data may be encoded in formats that are not human readable. Reducing these risks involves responsibilities for both the authors and the users of Rmarkdown source documents. For authors, the primary responsibilities are to assure that source documents cannot be modified by third parties between the author and the user, and to use best practices when carrying out file input and output during the knitting process. Integrity of source documents can be protected by using reputable websites with established security protocols for publishing Rmarkdown source documents, including, but not limited to, github, Zenodo and the Dryad. Authors can also publish MD5 sums for Rmarkdown source documents so users can verify the integrity of those documents, though the security of those MD5 sums is only as strong as the websites on which MD5 sums and Rmarkdown source documents are published. Best practices for file input and output include using temporary files and/or relative paths entirely within the base directory containing the Rmarkdown source document when writing files to disk, using file names that are unique (e.g., avoiding generic names like `data.csv`)

and performing checks for the existence of files with the same name before writing information to disk. The `Insert filled data chunk` RStudio add-in provided by `knitrdata` encourages the use of file existence checks in the `eval` chunk option controlling whether or not to process `data` chunks that write data to disk using the `output.file` chunk option, thereby avoiding overwriting existing files.

For users of Rmarkdown source documents, there are a number of simple steps one can take to avoid the most severe security risks. Knitting Rmarkdown source documents from an unprivileged user account and placing Rmarkdown source documents in new, empty directories can reduce the risks of the most malicious attacks. Users should also familiarize themselves with the workings of Rmarkdown source documents before knitting them and check for potentially problematic actions, such as use of absolute file paths and/or communication with external internet resources. This includes examination of the chunk options associated with `data` chunks (in particular, the `output.file` and `loader.function`). If one is primarily interested in just the raw data contained in `data` chunks within a Rmarkdown source document, then RStudio permits manual execution of individual chunks. This includes execution of `data` chunks, which can be processed individually using the `Run current chunk` button of RStudio once the `knitrdata` library has been loaded.

Conclusion

`knitrdata` provides a simple, but effective, tool for integrating arbitrary data into Rmarkdown source documents. If used appropriately, this can help assure computational reproducibility of many scientific documents by allowing one to integrate all relevant external files and data directly into a single Rmarkdown source document. Anyone who has attempted to validate the results in a publication by requesting the associated data has potentially encountered, if they managed to get the data, a set of one or more data tables with limited metadata and only the publication itself as documentation of the methods. Validating publication results under these conditions is often difficult and time consuming. By encouraging the integration of data, code for carrying out analyses, and text interpreting results in standalone Rmarkdown source documents, Rmarkdown with `knitrdata` can make it much easier to understand, reproduce and validate the details of scientific analyses. This combination can be particularly powerful when combined with other enhancements to Rmarkdown that make it possible to produce a wide variety of sophisticated scientific documents entirely within the confines of Rmarkdown, such as **bookdown** (Xie 2022), **rticles** (Allaire, Xie, Dervieux, et al. 2022) and **starticles** (Kaplan 2022) for producing books and scientific publications with Rmarkdown, **citr** (Aust 2019) for bibliographic citations, and **kableExtra** (Zhu 2021) for producing sophisticated data tables.

Online supporting information

The Rmarkdown source documents for this publication and Wain et al. (2021) are available online at https://github.com/dmkaplan2000/knitrdata_examples. Additional examples and the package vignette are available in the `knitrdata` package itself.

Acknowledgements

I would like to thank my colleagues at the MARBEC laboratory in Sète, France for numerous conversations that encouraged me to develop the `knitrdata` package. I would also like to thank Yihui Xie for advice and encouragement regarding the development of the package. The handling editor and an anonymous reviewer provided valuable feedback that significantly improved the manuscript.

References

Allaire, JJ, Yihui Xie, Christophe Dervieux, R Foundation, Hadley Wickham, Journal of Statistical Software, Ramnath Vaidyanathan, et al. 2022. *Rticles: Article Formats for r Markdown*. https://github.com/rstudio/rticles.

Allaire, JJ, Yihui Xie, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, Hadley Wickham, Joe Cheng, Winston Chang, and Richard Iannone. 2022. *Rmarkdown: Dynamic Documents for r*. https://CRAN.R-project.org/package=rmarkdown.

Aust, Frederik. 2019. *Citr: RStudio Add-in to Insert Markdown Citations*. https://github.com/crsh/citr.

Cassey, Phillip, and Tim M. Blackburn. 2006. "Reproducibility and Repeatability in Ecology." *BioScience* 56 (12): 958–59. https://doi.org/10.1641/0006-3568(2006)56%5B958:RARIE%5D2.0.CO;2.

Csárdi, Gábor, Jim Hester, Hadley Wickham, Winston Chang, Martin Morgan, and Dan Tenenbaum. 2021. *Remotes: R Package Installation from Remote Repositories, Including GitHub*. https://CRAN.R-project.org/package=remotes.

Kaplan, David M. 2020a. *Knitrdata: Data Language Engine for Knitr / Rmarkdown*. https://github.com/dmkaplan2000/knitrdata.

———. 2020b. "Using Knitrdata to Create Standalone Rmarkdown Documents in Rstudio." https://www.youtube.com/watch?v

———. 2022. *Starticles: A Generic, Publisher-Independent Template for Writing Scientific Documents in Rmarkdown*. https://github.com/dmkaplan2000/starticles.

Lowndes, Julia S. Stewart, Benjamin D. Best, Courtney Scarborough, Jamie C. Afflerbach, Melanie R. Frazier, Casey C. O'Hara, Ning Jiang, and Benjamin S. Halpern. 2017. "Our Path to Better Science in Less Time Using Open Data Science Tools." *Nature Ecology & Evolution* 1 (6): 1–7. https://doi.org/10.1038/s41559-017-0160.

Müller, Kirill, and Hadley Wickham. 2022. *Tibble: Simple Data Frames*. https://CRAN.R-project.org/package=tibble.

Ooms, Jeroen. 2022. *Gpg: GNU Privacy Guard for r*. https://github.com/jeroen/gpg.

Powers, Stephen M., and Stephanie E. Hampton. 2019. "Open Science, Reproducibility, and Transparency in Ecology." *Ecological Applications* 29 (1): e01822. https://doi.org/10.1002/eap.1822.

Wain, Gwenaëlle, Loreleï Guéry, David Michael Kaplan, and Daniel Gaertner. 2021. "Quantifying the Increase in Fishing Efficiency Due to the Use of Drifting FADs Equipped with Echosounders in Tropical Tuna Purse Seine Fisheries." *ICES Journal of Marine Science* 78 (1): 235–45. https://doi.org/10.1093/icesjms/fsaa216.

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. http://www.crcpress.com/product/isbn/9781466561595.

———. 2015. *Dynamic Documents with R and Knitr*. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. https://yihui.org/knitr/.

———. 2022. *Bookdown: Authoring Books and Technical Documents with r Markdown*. https://CRAN.R-project.org/package=bookdown.

Xie, Yihui, Christophe Dervieux, and Emily Riederer. 2020. *R Markdown Cookbook*. 1st edition. The R Series. Boca Raton, Florida: CRC Press.

Zhu, Hao. 2021. *kableExtra: Construct Complex Table with Kable and Pipe Syntax*. https://CRAN.R-project.org/package=kableExtra.

David M. Kaplan
MARBEC
Univ Montepllier, CNRS, Ifremer, IRD
Sète, France
Institute de Recherche pour le Developpement (IRD)
UMR MARBEC
av. Jean Monnet
CS 30171
34203 Sète cedex, France
https://www.davidmkaplan.fr
ORCiD: 0000-0001-6087-359X
david.kaplan@ird.fr

netgwas: An R Package for Network-Based Genome Wide Association Studies

by Pariya Behrouzi, Danny Arends and Ernst C. Wit

Abstract Graphical models are a powerful tool in modelling and analysing complex biological associations in high-dimensional data. The R-package **netgwas** implements the recent methodological development on copula graphical models to (i) construct linkage maps, (ii) infer linkage disequilibrium networks from genotype data, and (iii) detect high-dimensional genotype-phenotype networks. The **netgwas** learns the structure of networks from ordinal data and mixed ordinal-and-continuous data. Here, we apply the functionality in **netgwas** to various multivariate example datasets taken from the literature to demonstrate the kind of insight that can be obtained from the package. We show that our package offers a more realistic association analysis than the classical approaches, as it discriminates between direct and induced correlations by adjusting for the effect of all other variables while performing pairwise associations. This feature controls for spurious interactions between variables that can arise from conventional approaches in a biological sense. The **netgwas** package uses a parallelization strategy on multi-core processors to speed-up computations.

1 Introduction

Graphical models are commonly used in statistics and machine learning to model complex dependency structures in multivariate data (Lauritzen, 1996; Hartemink et al., 2000; Lauritzen and Sheehan, 2003; Jordan, 2004; Friedman, 2004; Dobra et al., 2004; Edwards et al., 2010; Behrouzi et al., 2018; Vinciotti et al., 2022) where each node in the graph represents a random variable and edges represent conditional dependence relationships between pairs of variables. Therefore, the absence of an edge between two nodes indicates that the two variables are conditionally independent. The **netgwas** package contains an implementation of undirected graphical models to address the three key and interrelated goals in genetics association studies: (i) building linkage maps, (ii) reconstructing linkage disequilibrium networks, and (iii) detecting genotype-phenotype networks (see Fig 1). Below we provide a brief introduction for each section of **netgwas**.

A linkage map describes the linear order of genetic markers within linkage groups (chromosomes). It is the first requirement for estimating the genetic background of phenotypic traits in quantitative trait loci (QTL) studies and are commonly used in QTL studies to link phenotypic traits to the underlying genetics of the population. In practice, many software packages for performing QTL analysis require linkage maps (Lander et al., 1987; Broman et al., 2003; Yang et al., 2008; Taylor et al., 2011; Huang et al., 2012; Broman et al., 2019). Most organisms are categorized as diploid or polyploid by comparing the copy number of each chromosome. Diploids have two copies of each chromosome (like humans). Polyploid organisms have more than two copies of each chromosome (like most crops). Polyploidy is common in plants and in different crops such as apple, potato, and wheat, which contain three (triploid), four (tetraploid), and six (hexaploid) copies from each of their chromosomes, respectively. Despite the importance of polyploids, statistical tools for their map construction are underdeveloped (Grandke et al., 2017; Bourke et al., 2018). Most software packages such as `R/qtl` (Broman et al., 2003), `OneMap` (Margarido et al., 2007), `Pheno2Geno` (Zych et al., 2015), and `MSTMAP` (Wu et al., 2008; Taylor and Butler, 2017) contain functionality to only construct linkage maps for diploid species. Packages such as `MAPMAKER` (Lander et al., 1987), `TetraploidSNPMap` (Hackett et al., 2017), and `polymapR` (Bourke et al., 2018) have the functionality to construct polyploid linkage maps but focus mainly on a specific type of polyploid species (e.g. tetraploids). All the aforementioned packages contain methods that use pairwise estimation of recombination frequencies and LOD (logarithm of the odds ratio) scores (Wang et al., 2016) that often require manual interaction. This often leads to an underpowered approach and confounding of correlated genotypes by failing to correct for intermediate markers (Behrouzi and Wit, 2019b). In contrast, the **netgwas** R package uses a multivariate approach to construct linkage maps for diploid and polyploid species in a unified way. This is achieved by utilising the pairwise Markov property between any two genetic markers and constructing the linkage map by simultaneously assessing the complete set of pairwise comparisons. This often leads to an improved marker order over more conventional methods.

The linkage map, which can be constructed using the first key function of **netgwas** in Fig 1, provides the genetic basis for the second key function which detects the patterns of linkage disequilibrium and segregation distortion in a population. Segregation distortion (SD) refers to any deviation from expected segregation ratios based on Mendelian rules of inheritance. And linkage disequilibrium (LD) refers to non-random relations between loci (locations) on the same or different chromosomes.

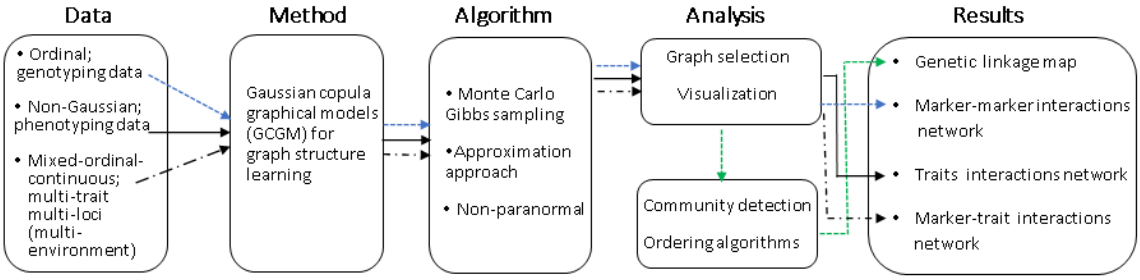


Figure 1: Configuration of the **netgwas** package. Depending on the data type different functions can be used. Each color represents one main function of the package. The three main functions are: i) `netmap()` (in green) constructs linkage maps for bi-parental species with any ploidy level, ii) `netsnp()` (in blue) detects the conditional dependent short- and long-range linkage disequilibrium structure of genomes and iii) `netphenogeno()` (in black) reconstructs association networks between phenotypes and genetic markers.

Revealing the structures of LD is important for association mapping study as well as for studying the genomic architecture of a genome. Various methods have been published in the literature for measuring statistical associations between alleles at different loci, for instance see Hedrick (1987); Mangin et al. (2012); Clarke et al. (2011); Bush and Moore (2012); Kaler et al. (2020). Most of these measures are based on an exhaustive genome scan for pairs of loci and the r^2 measure, the square of the loci correlation. The drawback of such approaches is that association testing in the genome–scale is underpowered, so that weak long–range LD will go undetected. Furthermore, they do not simultaneously take the information of more than two loci into account to make full and efficient use of modern multi–loci data. The **netgwas** package contains functionality to estimate pairwise interactions between different loci in a genome while adjusting for the effect of other loci to efficiently detect short- and long–range LD patterns in diploid and polyploid species (Behrouzi and Wit, 2019a). Technically, this requires estimating a sparse adjacency matrix from multi-loci genotype data, which usually contains a large number of markers (loci), where the number of markers can far exceed the number of individuals. The non-zero patterns of the adjacency matrix created by the functions in **netgwas** shows the structures of short– and long–range LD of the genome. The strength of associations between distant loci can be calculated using partial correlations. Furthermore, the methods implemented in **netgwas** already account for the correlation between markers, while associating them to each other and thereby avoids the problem of population structure (that is physically unlinked markers are correlated).

A major problem in genetics is the association between genetic markers and the status of a disease (trait or phenotype). Genome-wide association studies are among the most important approaches for further understanding of genetics underlying complex traits (Welter et al., 2013; Kruijer et al., 2020). However, in genome-wide association methods genetic markers are often tested individually for association with the phenotype. Since genome-wide scans analyze thousands or even millions of markers, the issue of multiple testing is usually addressed by using a stringent significance threshold (Panagiotou et al., 2011). Such methods work only if the associations are strong enough to pass the stringent threshold. However, even if that is the case, this type of analysis has several limitations, which have been discussed extensively in the literature (Hoggart et al., 2008; He and Lin, 2010; Rakitsch et al., 2012; Buzdugan et al., 2016). Particularly, the main issue of this type of analysis is when we test the association of the phenotype to each genetic marker individually, and ignore the effects of all other genetic markers. This leads to failures in the identification of causal loci. If we consider two correlated loci, of which only one is causal for the phenotype, both may show a marginal association, but only the causal locus will be detected by our method. The methods implemented in the **netgwas** package tackle this issue by using Gaussian copula graphical model (Klaassen and Wellner, 1997; Hoff, 2007), which accounts for the correlation between markers, while associating them to the phenotypes. As it is shown in Klasen et al. (2016), this key feature avoids the need to correct for population structure or any genetic background, as the method implemented in the **netgwas** package simultaneously associates all markers to the phenotype. In contrast, most GWAS methods rely on population structure correction to avoid false genotype-phenotype associations due to their single-loci approach (Yu et al., 2006; Kang et al., 2008, 2010; Lippert et al., 2011).

2 Technical details

Graphical models combine graph theory and probability theory to create networks that model complex probabilistic relationships. Undirected graphical models represent the joint probability distribution of a set of variables via a graph $G = (V, E)$, where $V = \{1, 2, \ldots, p\}$ specifies the set of random variables and $E \subset V \times V$ represents undirected edges $(i, j) \in E \Leftrightarrow (j, i) \in E$. The pattern of edges in the graph represents the conditional dependencies between the variables; the absence of an edge between two nodes indicates that any statistical dependency between these two variables is mediated via some other variable or set of variables. The conditional dependencies between variables, which are represented by edges between nodes, are specified via parameterized conditional distributions. We refer to the pattern of edges as the structure of the graph. In this paper, the goal is to learn the graph structure from ordinal data and mixed ordinal-and-continuous data.

Sparse latent graphical model. A p-dimensional copula $\mathcal{C}$ is a multivariate distribution with uniform margins on $[0, 1]$. Any joint distribution function can be written in terms of its marginals and a copula which encodes the dependence structure. Here we consider a subclass of joint distributions encoded by the Gaussian copula $F(y_1, \ldots, y_p) = \Phi_p\Big(\Phi^{-1}(F_1(y_1)), \ldots, \Phi^{-1}(F_p(y_p)) \mid \Omega\Big)$ where $\Phi_p(. \mid \Omega)$ is the cumulative distribution function (CDF) of p-variate Gaussian distribution with correlation matrix Ω; Φ is the univariate standard normal CDF; and F_j is the CDF of j-th variable, Y_j for $j = 1, \ldots, p$. Note that y_j and $y_{j'}$ are independent if and only if $\Omega_{jj'} = 0$.

A Gaussian copula can be written in terms of latent variables Z: Let $F_j^{-1}(y) = \inf\{y : F_j(x) \geq y, x \in \mathcal{R}\}$ be the pseudo-inverse of the marginals. Then a Gaussian copula is defined as $Y_{ij} = F_j^{-1}(\Phi(Z_{ij}))$ where $Z \sim \mathcal{N}_p(0, \Omega)$ and $Y = (Y_1, \ldots, Y_p)$ and $Z = (Z_1, \ldots, Z_p)$ represent the non-Gaussian observed variables and Gaussian latent variables, respectively. Without lose of generality, we assume that Z_j's have unit variances of $\sigma_{jj} = 1$ for $j = 1, \ldots, p$. Thus, Z_j's marginally follow standard Gaussian distribution. Each observed variable Y_j is discretized from its latent counterpart Z_j. For the j-th latent variable ($j = 1, \ldots, p$), we assume that the range $(-\infty, \infty)$ splits into K_j disjointed intervals by a set of thresholds $-\infty = t_j^{(0)} < t_j^{(1)} < \ldots < t_j^{(K_j - 1)} < t_j^{(K_j)} = \infty$ such that $Y_j = k$ if and only if Z_j falls in the interval $(t_j^{(K-1)}, t_j^{(K)})$. Let the parameter $\mathcal{D} = \{t_j^{(k)} : j = 1, \ldots, p; k = 1, \ldots, K_j\}$ holds the boundaries for the truncation points and $z^{(1:i)} = [z^{(1)}, \ldots, z^{(n)}]$ where $z^{(i)} = (z_1^{(i)}, \ldots, z_p^{(i)})$. In order to learn the graph structures, our objective is to estimate the precision matrix $\Theta = \Omega^{-1}$ from n independent observations $y^{(1:i)} = [y^{(1)}, \ldots, y^{(n)}]$, where $y^{(i)} = (y_1^{(i)}, \ldots, y_p^{(i)})$. The conditional

Algorithm 1 Monte Carlo Gibbs sampling for estimating $\bar{R}$ in (1)

Input: A data set containing the variables $Y^{(i)}$ for $i = 1 \ldots, n$.

Output: Mean of the conditional expectation, $\bar{R} = \frac{1}{n} \sum_{i=1}^{n} E(Z^{(i)} Z^{(i)^T} | y^{(i)}; \mathcal{D}, \Theta^{(m-1)})$ in (1).

1: For each $j = \{1, \ldots, p\}$ generate the latent data from $Y_j = F_j^{-1}(\Phi(Z_j))$, where F_j and Φ define the empirical marginals and the CDF of standard normal distribution, respectively for $i = 1 \ldots, n$ and $j = 1, \ldots, p$;
2: Estimate $\mathcal{D}$, vectors of lower and upper truncation points, whose boundaries come from $Y^{(i)}$ for $i = 1 \ldots, n$;
3: **for** $i = 1, \ldots, n$ **do**
4: Sample from a p-variate truncated normal distribution with the boundaries in Line 2 above;
5: **for** N iterations **do**
6: Estimate $R^{(i),N} = E(Z_\star^{(i)} Z_\star^{(i)T} | y^{(i)}, \widehat{\Theta}^{(m)})$, where $Z_\star^{(i)} = \begin{pmatrix} Z_\star^{(i)1} \\ \vdots \\ Z_\star^{(i)N} \end{pmatrix} \in \mathbb{R}^{N \times p}$;
7: **end for**
8: Update $\widehat{R}^{(i)} = \frac{1}{N} Z_\star^{(i)} Z_\star^{(i)T}$;
9: **end for**
10: Calculate $\widehat{\bar{R}} = \frac{1}{n} \sum_{i=1}^{n} \widehat{R}^{(i)}$.

independence between two variables given other variables is equivalent to the corresponding element in the precision matrix being zero, i.e. $\theta_{ij} = 0$; or put another way, a missing edge between two variables in a graph G represents conditional independence between the two variables given all other variables.

In the classical low-dimensional setting of p smaller than n, it is natural to implement a maximum likelihood approach to obtain the inverse of the sample covariance matrix. However, in modern applications the dimension p is routinely far larger than n, meaning that the inverse sample covariance matrix does not exist. Motivated by the sparseness assumption of the graph we tackle the high-dimensional inference problem for discrete Y's by a penalized expectation maximization (EM) algorithm as

$$\mathrm{Q}(\Theta \mid \widehat{\Theta}^{(m-1)}) = \frac{n}{2}\Big[\log\det(\Theta) - \mathrm{tr}(\bar{R}\Theta)\Big] \tag{1}$$

and

$$\widehat{\Theta}^{(m)} = \arg\max_{\Theta} \tilde{\mathrm{Q}}(\Theta|\hat{\Theta}^{(m-1)}) - \sum_{j\neq j'}^{p} \mathrm{P}_{\rho}(|\theta_{jj'}|), \tag{2}$$

where m is the iteration number within the EM algorithm. The last term in equation (2) represents different penalty functions. Here we impose the sparsity by means of L_1 penalty, on the jj'-th element of the precision matrix.

We compute the conditional expectation $\bar{R} = \frac{1}{n}\sum_{i=1}^{n} E\Big(Z^{(i)}Z^{(i)^T} \mid y^{(i)}; \widehat{\mathcal{D}}, \widehat{\Theta}^{(m-1)}\Big)$ in equation (1) using two different approaches: numerically through a Monte Carlo (MC) sampling method as explained in algorithm 1, and through a first order approximation based on algorithm 2. The most flexible and generally applicable approach for obtaining a sample in each iteration of an MCEM algorithm is through a Markov chain Monte Carlo (MCMC) routine like Gibbs and Metropolis – Hastings samplers (Metropolis et al., 1953; Hastings, 1970; Geman and Geman, 1984). Alternatively, the conditional expectation in equation (1) can be computed by using an efficient approximation approach which calculates elements of the empirical covariance matrix using the first and second moments of a truncated normal distribution with mean $\mu_{ij} = \widehat{\Omega}_{j,-j}\widehat{\Omega}^{(-1)}_{-j,-j}z^{(i)^T}_{-j}$ and variance $\sigma^2_{i,j} = 1 - \widehat{\Sigma}_{j,-j}\widehat{\Sigma}^{-1}_{-j,-j}\widehat{\Sigma}_{-j,-j}$ (see Behrouzi and Wit (2019a) for details). The two proposed approaches are practical when some observations are missing. For example, if genotype information for genotype j is missing, it is still possible to draw Gibbs samples for Z_j or approximate the empirical covariance matrix, as the corresponding conditional distribution is Gaussian.

The optimization problem in (2) can be solved efficiently in various ways by using glasso or CLIME approaches (Friedman et al., 2008; Hsieh et al., 2011). Convergence of the EM algorithm for penalized likelihood problems has been proved in Green (1990). Our experimental study shows that the algorithm usually converges after several iterations (< 10). Note that the sparsity of the estimated precision matrix in Equation (2) is controlled by a vector of penalty parameter ρ. We follow Foygel and Drton (2010) in using the extended Bayesian information criterion (eBIC) to select a suitable regularization parameter ρ^* to produce a sparse graph with a sparsity pattern corresponding to $\widehat{\Theta}_{\rho^*}$. Alternatively, instead of using the EM algorithm, a nonparanormal skeptic approach can be used to estimate graph structure through Spearman's rho and Kendall's tau statistics; details can be found in Liu et al. (2012).

Extension to linkage map construction

Here we convert the estimated network to a one-dimensional map using two different approaches. Depending on the type of (experimental) population (i.e. inbred or outbred), we order markers based on dimensionality reduction or based on bandwidth reduction, which both result in an one-dimensional map.

In inbred populations, loci in the genome of the progenies can be assigned to their parental homologues, resulting in a simpler conditional independence relationship between neighboring markers. Here, we use multidimensional scaling (MDS) to project markers in a p-dimensional space onto a one-dimensional map while attempting to maintain pairwise distances. Let $G(V^{(d)}, E^{(d)})$ be a sub–graph on the set of unordered d markers, where $V^{(d)} = \{1, \ldots, d\}$, $d \leq p$ and the edge set $E^{(d)}$ represents all the links among d markers. We calculate the distance matrix D as follows

$$D_{ij} = \begin{cases} -\log(r_{ij}) & \text{if } i \neq j \\ 0 & \text{if } i = j, \end{cases} \tag{3}$$

$$r_{ij} = -\frac{\theta_{ij}}{\sqrt{\theta_{ii}\theta_{jj}}}, \tag{4}$$

where θ_{ij} is the ij-th element of the precision matrix $\widehat{\Theta}_{\rho^*}$. We aim to construct a configuration of d data points in a one–dimensional Euclidean space by using information about the distances between the d nodes. In this regards, we define a sequential ordering L of d elements such that the distance $\widehat{D}$ between them is similar to $\mathcal{D}$. We consider a metric multi-dimensional scaling

$$\widehat{E} = \arg\min_{L} \sum_{i=1}^{d}\sum_{j=1}^{d}(D_{ij} - \widehat{D}_{ij})^2, \tag{5}$$

that minimizes the so called mapping error $\widehat{E}$ across all sequential orderings (Sammon, 1969).

We propose a different ordering algorithm for outbred populations. In these populations, mating of two non-homozygous parents result in markers in the genome of progenies that cannot easily be mapped into their parental homologues. To order markers in outbred populations, we use the Cuthill-McKee (RCM) algorithm (Cuthill and McKee, 1969) to permute the sparse matrix $\widehat{\Theta}_{\rho}^{(d)}$ that has a symmetric sparsity pattern into a band matrix form with a small bandwidth. The bandwidth of the associated adjacency matrix A is defined as $\beta = \max_{A_{ij}\neq 0}|i-j|$. The algorithm produces a permutation matrix P such that PAP^T has a smaller bandwidth than matrix A does. The bandwidth decreases by moving the non-zero elements of the matrix A closer to the main diagonal. The way to move the non-zero elements is determined by relabeling the nodes in graph $G(V_d, E_d)$ in consecutive order. Moreover, all of the nonzero elements are clustered near the main diagonal.

3 Package design and functionality

The **netgwas** R package implements the Gaussian copula graphical models (Behrouzi and Wit, 2019a) for (i) constructing linkage maps in diploid and polyploid species and learning (ii) linkage disequilibrium networks and (iii) genotype-phenotype networks. Below, we illustrate the three main functions using a diploid A.thaliana population, a tetraploid potato, and maize NAM populations. Given that the computational cost for the usual size of GWAS data ($> 10^5$) is expensive, we use small data sets to explain the functionality of the package. All the results can be replicated using the functions in the **netgwas** package (see Supplementary Materials).

Algorithm 2 Approximation of the conditional expectation in (1)

Input: A $(n \times p)$ data matrix Y, where $\mathrm{Y}_j^{(i)} = F_j^{-1}(\Phi(Z_j^{(i)}))$ the F_j and Φ define the empirical marginals and the CDF of standard normal, respectively for $i = 1\ldots, n$ and $j = 1, \ldots, p$;

Output: The conditional expectation $R = \frac{1}{n}\sum_{i=1}^{n} E(Z^{(i)}Z^{(i)^T}|y^{(i)}; \widehat{\mathcal{D}}, \boldsymbol{\Theta})$;

1: Initialize $E(z_j^{(i)} \mid y^{(i)}; \widehat{\mathcal{D}}, \widehat{\boldsymbol{\Theta}}) \approx E(z_j^{(i)} \mid y_j^{(i)}; \widehat{\mathcal{D}})$, $E((z_j{}^{(i)})^2 \mid y^{(i)}; \widehat{\mathcal{D}}, \widehat{\boldsymbol{\Theta}}) \approx E((z_j{}^{(i)})^2 \mid y_j^{(i)}; \widehat{\mathcal{D}})$, and $E(z_j^{(i)} z_{j'}^{(i)} \mid y^{(i)}; \widehat{\mathcal{D}}, \widehat{\boldsymbol{\Theta}}) \approx E(z_j^{(i)} \mid y_j^{(i)}; \widehat{\mathcal{D}}) E(z_{j'}^{(i)} \mid y_{j'}^{(i)}; \widehat{\mathcal{D}})$ for $i = 1, \ldots, n$ and $j, j' = 1, \ldots, p$;

2: Initialize $r_{j,j'}$ for $1 \leq j, j' \leq p$ using the Line 1 above, then estimate $\widehat{\boldsymbol{\Theta}}$ by maximizing (2);

3: **for** $i = 1, \ldots, n$ **do**

4: **if** $j = j'$ **then**

5: Calculate $E((z_j{}^{(i)})^2 \mid y_j^{(i)}; \widehat{\mathcal{D}}, \widehat{\boldsymbol{\Theta}})$ for $j = 1, \ldots, p$;

6: **else**

7: Calculate $E(z_j^{(i)} \mid y^{(i)}; \widehat{\mathcal{D}}, \widehat{\boldsymbol{\Theta}})$ and then $E(z_j^{(i)} z_{j'}^{(i)} \mid y^{(i)}; \widehat{\mathcal{D}}, \widehat{\boldsymbol{\Theta}}) = E(z_j^{(i)} \mid y^{(i)}; \widehat{\mathcal{D}}, \widehat{\boldsymbol{\Theta}}) E(z_{j'}^{(i)} \mid y^{(i)}; \widehat{\mathcal{D}}, \widehat{\boldsymbol{\Theta}})$ for $i = 1, \ldots, n$ and $j = 1, \ldots, p$;

8: **end if**

9: **end for**

10: Calculate $r_{j,j'} = \frac{1}{n}\sum_{i=1}^{n} E(z_j^{(i)} z_j^{(i)t} \mid y^{(i)}; \widehat{\mathcal{D}}, \widehat{\boldsymbol{\Theta}})$ for $1 \leq j = j' \leq p$.

Linkage map construction

This module reconstructs linkage maps for diploid and polyploid organisms. Diploid organisms contain two copies of each chromosome, one from each parent, whereas polyploids contain more than two copies of each chromosome. In polyploids the number of chromosome sets reflects their level of ploidy: triploids have three copies, tetraploids have four, pentaploids have five, and so forth.

Typically, mating is between two parental lines that have recent common biological ancestors; this is called inbreeding. If they have no common ancestors up to roughly 4-6 generations, then this is called outcrossing. In both cases the genomes of the derived progenies are random mosaics of the genome of the parents. However, in the case of inbreeding parental alleles are distinguishable in the genome of the progeny, in outcrossing this does not hold.

Some *inbreeding* designs, such as Doubled haploid (DH), lead to a homozygous population where the derived genotype data include only homozygous genotypes of the parents namely AA and aa (conveniently coded as 0 and 1) for diploid species. Other inbreeding designs, such as F2, lead to a heterozygous population where the derived genotype data contain heterozygous genotypes as well as homozygous ones, namely aa, Aa, and AA for diploid species, conveniently coded as 0, 1 and 2 which correspond to the dosage of the reference allele A. We remark that the Gaussian copula graphical models help us to keep heterozygous markers in the linkage map construction, rather than turn them to missing values as most other methods do in map construction for recombinant inbred line (RIL) populations.

Outcrossing or outbred experimental designs, such as full-sib families, derive from two non-homozygous parents. Thus, the genome of the progenies includes a mixed set of many different marker types containing fully informative markers and partially informative markers . Markers are called fully informative when all of the resulting gamete types can be phenotypically distinguished on the basis of their genotypes; markers are called partially informative when they have identical phenotypes (Wu et al., 2002).

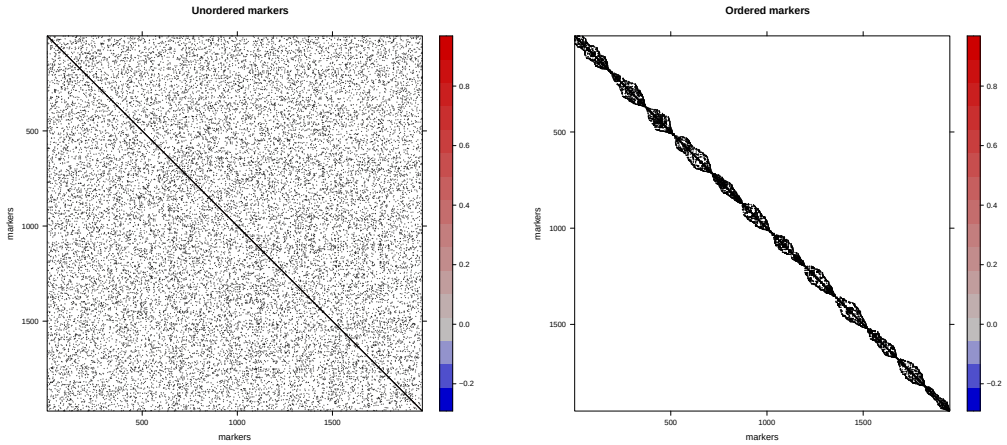
`netmap()`

The `netmap()` function handles various inbred and outbred mapping populations, including recombinant inbred lines (RILs), F2, backcross, doubled haploid, and full-sib families, among others. Not all existing methods for linkage mapping support all inbreeding and outbreeding experimental designs. However, our proposed algorithm constructs a linkage map for any type of experimental design of biparental inbreeding and outbreeding. Also, it covers a wide range of possible population types. Argument `cross` in the map function must be specified to choose an ordering method. In inbred populations, markers in the genome of the progenies can be assigned to their parental homologous, resulting in a simpler conditional independence pattern between neighboring markers. In the case of inbreeding, we use multidimensional scaling (MDS). A metric MDS is a classical approach that maps the original high-dimensional space to a lower dimensional space, while attempting to maintain pairwise distances. An outbred population derived from mating two non-homozygous parents results in markers in the genome of progenies that cannot be easily assigned to their parental homologues. Neighboring markers that vary only on different haploids will appear as independent, therefore requiring a different ordering algorithm. In that case, we use the reverse Cuthill-McKee (RCM) algorithm (Cuthill and McKee, 1969) to order markers.

The function can be called with the following arguments

```
netmap(data, method = "npn", cross = NULL, rho = NULL, n.rho = NULL, rho.ratio = NULL,
       min.m = NULL, use.comu = FALSE, ncores = "all", verbose = TRUE)
```

The main task of this function is to construct a linkage map based on conditional (in)dependence relationships between markers, which can be estimated using the methods, "`gibbs`", "`approx`", and "`npn`". The estimation procedure relies on maximum penalized log-likelihood, where the argument `rho`, a decreasing sequence of non-negative numbers, controls the sparsity levels, which corresponds to the last term in Equation (2). Leaving the input as `rho = NULL`, the program automatically computes a sequence of `rho` based on `n.rho` and `rho.ratio`. The argument `n.rho` specifies the number of regularization parameters (the default is 6) and `rho.ratio` determines the ratio between the consecutive elements of `rho`. Depending on the population type, `inbred` or `outbred`, different algorithms are applied to order markers in the genome. If it is known, the user can specify an expected minimum number of markers in a linkage group (LG) via the argument `min.m`. Furthermore, linkage groups can be identified either using the fast greedy community detection algorithm (Newman, 2004) or simply each disconnect sub-networks can form a linkage group. The `ncores = "all"` automatically detects number of available cores and runs the computations in parallel on (available cores -1).



(a) (b)

Figure 2: Linkage map construction in tetraploid potato. A total of 156 F1 plants were genotyped across 1972 genetic markers. (a) The estimated partial correlation matrix for the genotyping data. (b) The estimated partial correlation matrix after ordering the genetic markers, where all the 12 potato chromosomes are detected correctly.

The `netmap()` function returns an object of the `S3` class type `netgwasmap` and `plot.netgwasmap` and `print.netgwasmap` are summary method functions for this object class. The `netgwasmap` mainly holds the estimated linkage map (in object `map`) and a list containing all output results (in object `res`) of the regularization path `rho`.

`buildMap()`

The function `buildMap()` allows users to interact with the map construction procedure and to build the linkage map on the manually selected penalty term. Whereas the function `netmap()` selects the optimal penalty term $\rho^\star$ using the eBIC method.

The function can be called via

```
buildMap(res, opt.index, min.m = NULL, use.comu = FALSE).
```

The argument `opt.index` can be chosen manually which is a number between one and the number of penalty parameter `n.rho` in `netmap()`. In the default setting, the `n.rho` is 6. So, the `opt.index` can get a value between 1 and 6. Like function `netmap()`, the argument `min.m` is an optional argument in `buildMap()` function, where it keeps the clusters of markers that at least have a size of `min.m` member of markers. The default value for this argument is 2. The `use.comu` argument is an alternative approach to find linkage groups. The `use.comu` argument is an alternative approach to find linkage groups.

Detecting linkage disequilibrium networks

The function netsnp() reconstructs a high-dimensional linkage disequilibrium interactions network for diploid and polyploid (GWAS) genotype data. Genetic viability can be considered as a phenotype. This function detects the conditional dependent short- and long-range linkage disequilibrium structure of genomes and thus reveals aberrant marker-marker associations that are due to epistatic selection. In other words, this function detects intra– and inter–chromosomal conditional interactions networks and can be called via

```
netsnp(data, method = "gibbs", rho = NULL, n.rho = NULL, rho.ratio = NULL,
       ncores = "all", verbose = TRUE)
```

for any bi-parental genotype data containing at least two genotype states and possibly missing values. The input data can be either an (n $\times$ p) genotype data matrix, an object of class `netgwasmap`, which is an output of functions `netmap()` and `buildMap()`, or a simulated data from the function `simgeno()`. Depending on the dimension of the input data a suitable '`method`' and its related arguments can be specified. The argument `ncores` determines the number of cores to use for the calculations. Using `ncores = "all"` automatically detects number of available cores and runs the computations in parallel.

This function returns an *S*3 object of class "netgwas", which holds mainly the following objects: (i) `Theta` a list of estimated $(p \times p)$ precision (inverse of variance-covariance) matrices that infer the conditional independence relationships patterns among genetic loci, (ii) `path` which is a list of estimated $(p \times p)$ adjacency matrices. This is the graph path corresponding to `Theta`, (iii) `rho` which is a vector with `n.rho` dimension containing the penalties, and (iv) `loglik` contains the maximum log-likelihood values along the graph path. To select an optimal graph the function `selectnet()` can be used.

Reconstructing genotype-phenotype networks

Complex genetic traits are influenced by multiple interacting loci, each with a possibly small effect. Our approach reduces the number of candidate genes from hundreds to much fewer genes. It is of great interest to geneticists and biologist to discover a set of most effective genes that directly affect a complex trait in GWAS. To overcome the limitations of traditional analysis, such as single-locus association analysis (looking for main effects of single marker loci), multiple testing and QTL analysis, we use the proposed mixed graphical model to study the simultaneous associations between phenotypes and SNPs. Our method allows for a more accurate interpretation of findings, because it adjusts for the effects of remaining variables –SNPs and phenotypes– while measuring the pairwise associations, whereas the traditional methods use marginal associations to often analyze SNPs and phenotypes one at a time .

Graphical modeling is a powerful tool for describing complex interaction patterns among variables in high-dimensional data used frequently in microarray analysis (Butte et al., 2000). In our modelling framework, a genotype–phenotype network is a complex network made up of interactions among: (i) genetic markers, (ii) phenotypes (e.g. disease), and (iii) between genetic markers and phenotypes. The first problem in analyzing genotype-phenotype data is the mixed variable-types, where markers are ordinal (counting the number of a major allele), and phenotypes (disease) can be measured in continuous or discrete scales. We deal with mixed discrete-and-continuous variables by means of copula. A second issue relates to the high-dimensional setting of the data, where thousands of genetic markers are measured across a few samples; we deal with inferring potentially large networks with only few biological samples. Fortunately, genotype-phenotype networks are intrinsically sparse, in the sense that only few elements interact with each other. This sparsity assumption is incorporated into our algorithm based on penalized graphical models. The proposed method is implemented in the netphenogeno() function, where the input data can be an $(n \times p)$ `matrix` or a `data.frame` where n is the sample size and p is the dimension that includes marker data and phenotype measurements. One may consider including more columns, like environmental variables.

The function is defined by

```
netphenogeno(data, method = "npn", rho = NULL, n.rho = NULL, rho.ratio = NULL,
               ncores = "all", em.iter = 5, em.tol = .001, verbose = TRUE)
```

and reconstructs genotype-phenotype interactions network for an input data of a genotype-phenotype data matrix or a data.frame. Detecting interactions network among genotypes, phenotypes and environmental factors is also possible using this function. Depending on the size of the input data, the user may choose "gibbs", "approx", or "npn" method for learning the networks. For a medium ($\sim$500) and a large number of variables we recommend to choose "gibbs" and "approx", respectively. Choosing "npn" for a very large number of variables (>2000) is computationally efficient. The default method is set to "npn". Like the function `netsnp()`, the `netphenogeno()` function returns an object of class **netgwas**.

For objects of type 'netgwas' there are plot, print and summary methods available. The plotting function `plot.netgwas()` provides a visualization plot to monitor the path of estimated networks for a range of penalty terms. The functions `plot.netgwasmap(), plot.select() and plot.simgeno()` visualize the corresponding network, the optimal graph and the results of model-based simulated data, respectively.

To speed up computations in all the three key functions of the **netgwas** package, we use the **parallel** package on the Comprehensive `R` Archive Network (CRAN) at `http://CRAN.R-project.org` to support parallel computing on a multi-core machine to deal with large inference problems. For the optimizing the memory usage, we use the **Matrix** package (Bates et al., 2022) for sparse matrix output when estimating and storing full regularization paths for large datasets. The use of these libraries significantly improves the computational speed of the functions within the package.

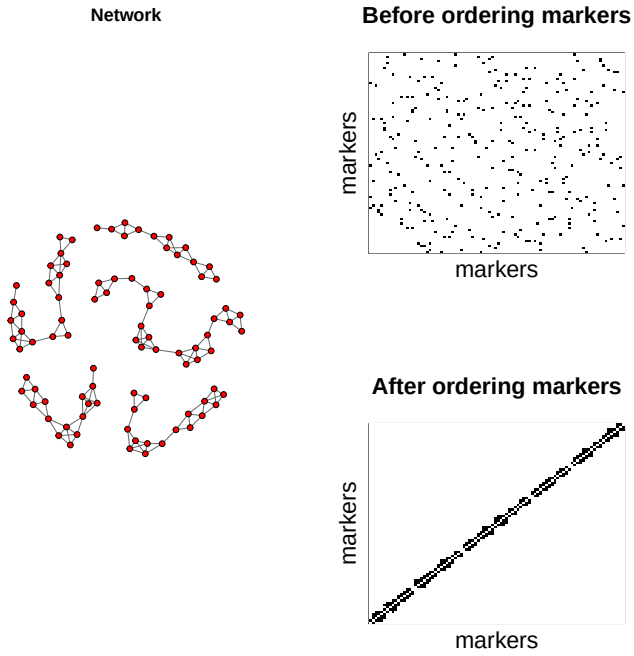


Figure 3: Linkage map construction in *A.thaliana* using RILs (recombinant inbred lines) population of 367 individuals and 90 genetic markers. The inferred network detects the five chromosomes of A.thaliana and the structure of associations among SNPs (single nucleotide polymorphisms) within chromosomes. The left figures show the partial correlation matrix before and after ordering the SNPs.

4 Application to real datasets

Linkage map construction

In Fig 2 and Fig 3, we provide two examples output of building linkage map in outbred tetraploid *potato* and inbred diploid *A.thaliana* datasets. The map construction was computed in about 7 minutes for tetraploid potato data and in 0.6 seconds for A.thaliana data on an Intel i7 laptop with 16 GB RAM.

Linkage map construction in potato. For the sake of illustration, below we show the steps to construct a linkage map for `TetraPotato` in **netgwas**. The tetraploid *potato* data are derived from a cross between "Jacqueline Lee" and "MSG227-2", where 156 F1 plants were genotyped across 1972 genetic markers (Massa et al., 2015). Five allele dosages are possible in this full-sib autotetraploid mapping population (AAAA, AAAB, AABB, ABBB, BBBB), where the genotypes are coded as $\{0, 1, 2, 3, 4\}$. This dataset includes 0.07% missing observations.

```
data(tetraPotato)
# Shuffle the order of markers
dat <- tetraPotato[ , sample(ncol(tetraPotato))]
potato.map <- netmap(dat, cross = "outbred")
potato.map.ordered <- buildMap(potato.map, opt.index = 3)
potato.map.ordered
```

```
Number of linkage groups:  12
Number of markers per linkage group: 165 157 129 153 183 196 173 148 152 161 187 146
Total number of markers in the linkage map: 1950.(22  markers removed from the input genotype data)
Number of sample size: n = 156
Number of categories in dataset: 5  ( 0 1 2 3 4 )
The estimated linkage map is inserted in <OUTPUT NAME>$map
To visualize the network consider plot(<OUTPUT NAME>)
-----------------------
To visualize the other associated networks consider plot(<OUTPUT NAME>$allres)
```

```
plot(potato.map.ordered, vis = "unordered markers")
plot(potato.map.ordered, vis = "ordered markers")
map <- potato.map.ordered$map
```

The argument `vis` in the above `plot` function can be fixed to `"interactive"`, which it gives a better network resolution particularly for a large number of nodes. Fig 2 visualizes a summary of its mapping process, where Fig 2a shows the conditional dependence pattern between unordered SNP markers in the Jacqueline Lee $\times$ MSG227-2 population. Fig 2b shows the structure of the selected graph after ordering markers. All 12 potato chromosomes were detected correctly. The tetraploid potato map construction was computed in about 7 minutes on an Intel i7 laptop with 16 GB RAM.

Linkage map construction in *A.thaliana*. In this example, we construct a linkage map for the *Arabadopsis thaliana* data which are derived from a RIL cross between Columbia-0 (Col-0) and Cape Verde Island (Cvi-0), where 367 individual plants were genotyped across 90 genetic markers (Simon et al., 2008). The dataset `CviCol` contains 0.2% missing values and three possible genotype states, where A and B denote parental homozygous loci, coded as 0 and 2, respectively and H denotes heterozygous loci which coded as 1.

```
data(CviCol)
set.seed(1)
cvicol <- CviCol[ ,sample(ncol(CviCol))]
out <- netmap(cvicol, cross= "inbred", ncores= 1)
out$opt.index
[1] 6
```

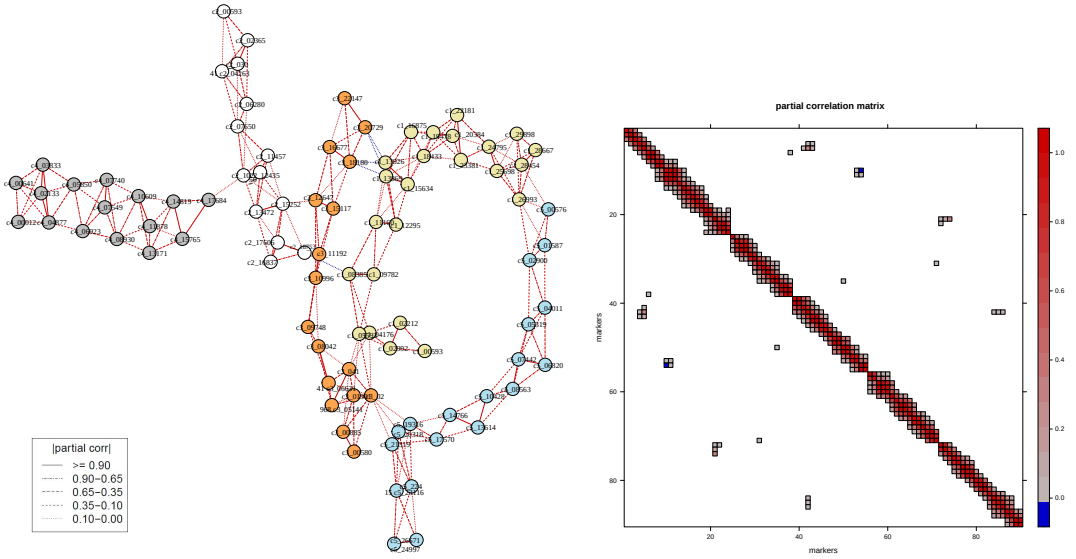
In the above code, the `out$opt.index` shows the index of the selected penalty term using the eBIC method. If one is interested in building linkage map, for instance, on the 4th estimated network then the `buildMap()` function can be used as follow

```
bm.thaliana <- buildMap(out, opt.index= 4)
bm.thaliana

Number of linkage groups:  5
Number of markers per linkage group:  24 14 17 16 19
Total number of markers in the linkage map: 90.
(0 markers removed from the input genotype data)
Number of sample size: n = 367
Number of categories in dataset: 3  ( 0 1 2 )
The estimated linkage map is inserted in <OUTPUT NAME>$map
To visualize the network consider plot(<OUTPUT NAME>)
-----------------------
To visualize the other associated networks consider plot(<OUTPUT NAME>$allres)
To build a linkage map for your desired network consider buildMap() function

thalianaMap <- bm.thaliana$map
plot(bm.thaliana, vis= "summary")
```

The estimated linkage map in Fig 3 is consistent with the existing linkage map in *A.thaliana* (Simon et al., 2008; Behrouzi and Wit, 2019b).

If required, `detect.err()` function detects genotyping errors. This function calculates the error LOD score for each individual at each marker using Lincoln and Lander (1992) approach; large scores show likely genotyping errors. Here, the **qtl** package (Broman et al., 2003) is used for identification of genotyping errors, where the output gives a list of genotypes that might be in error, when the error LOD scores are smaller than 4 they can probably be ignored (Broman, 2009). This function supports doubled haploid (DH), backcross (BC), non-advanced recombinant inbred line population with n generations of selfing (RILn) and advanced RIL (ARIL) population types.

The `cal.pos()` function calculates the genetic distance for diploid populations. It uses the **qtl** package to calculate genetic distance using different distance functions. The `netgwas2cross()` function converts the map object to a `cross` object from **qtl** package, and vice versa using the function `cross2netgwas()`. These two functions make **netgwas** flexible with respect to further genetic investigation using **qtl** package. Furthermore, `cross` objects from the **qtl** package can also be analyzed using **netgwas** package.



(a) (b)

Figure 4: Short- and long-range linkage disequilibrium networks between 90 markers across the A.*thaliana* genome. (a) Each color corresponds to a different chromosome: yellow, white, orange, gray, and blue represent chromosomes 1 to 5, respectively. Different edge colors show positive ▬ and negative ▬ values of partial correlations. (b) Plots simultaneous marker-marker interactions across the genome. Values represent partial correlations.

Genome wide association studies

Linkage disequilibrium networks in *A.thaliana*. We use the dataset `CviCol` to learn conditionally dependent short- and long-range LD structure in *A.thaliana* genome. The aim here is to identify associations between distant markers that are due to epistatic selection rather than gametic linkage.

```
data(CviCol)
set.seed(2)
out <- netsnp(CviCol)
sel <- selectnet(out)
# Steps to visualize the selected network
cl <- c(rep("palegoldenrod", 24), rep("white",14), rep("tan1",17),
rep("gray",16), rep("lightblue2",19))
plot(sel, vis= "parcor.network", sign.edg = TRUE, layout = NULL, vertex.color = cl)
plot(sel, vis= "image.parcorMatrix", xlab="markers", ylab="markers")
```

In Fig 4, our method finds that in *Cvi* × *Col* population some trans-chromosomal regions conditionally interact. In particular, the bottom of chromosome 1 and the top of chromosome 5 do not segregate independently of each other. Besides this, interactions between the tops of chromosomes 1 and 3 involve pairs of loci that also do not segregate independently. Bikard et al. (2009) studied this genotype data extensively in their lab. They reported that the first interaction (between chr 1 and 5) that our method finds causes arrested embryo development, resulting in seed abortion, and the latter interaction (between chr 1 and 3) causes root growth impairment. In addition to these two regions, we have discovered a few other trans-chromosomal interactions in the A.*thaliana* genome. In particular, two adjacent markers, c1-13869 and c1-13926 in the middle of the chromosome 1, interact epistatically with the adjacent markers, c3-18180 and c3-20729, at the bottom of chromosome 3. The sign of their conditional correlation score is negative, indicating strong negative epistatic selection in F_2 population. These markers therefore seem evolutionarily favored to come from the two different F_0 grandparents. This suggests some positive effect of the interbreeding of the two parental lines: it could be that the paternal-maternal combination at these two loci protects against some underlying disorder, or that it actively enhances the fitness of the resulting progeny. Regarding the computational time, this example was run in 4 minutes on an Intel i7 laptop with 16 GB RAM.

Genotype-phenotype networks in *A.thaliana*. We apply our algorithm to the model plant *Arabidopsis thaliana* dataset, where the accession Kend-L (Kendalville-Lehle; Lehle-WT-16-03) is crossed with

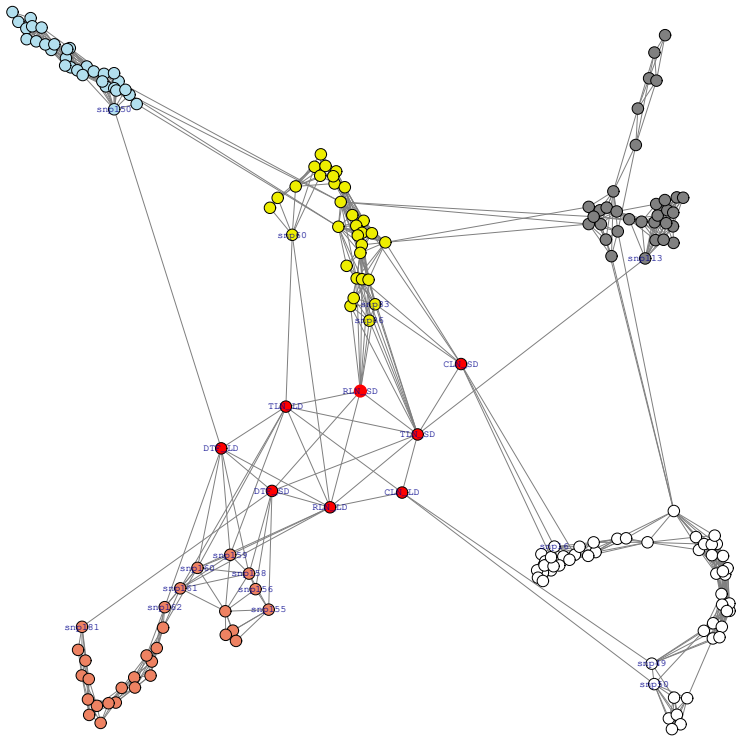


Figure 5: Genotype–phenotype association networks of A.*thaliana*. This shows an example of multi-loci multi-trait genome-wide association analysis. Red nodes show phenotypes; white, yellow, gray, blue, and brown colors stand for chromosomes 1 to 5, respectively. Phenotypes measured in long days (TLN-LD, RLN-LD, DTF-LD) conditionally dependent on a region on top of chromosome 5. Phenotypes measured in short days (CLN-SD, RLN-SD, DTF-SD) are linked mostly to chromosomes 1, 2, and 5.

the common lab strain Col (Columbia) (Balasubramanian et al., 2009). The resulting lines were taken through six rounds of selfing without any intentional selection. The resulting 282 KendC (Kend-L × Col) lines were genotyped at 181 markers. Flowering time was measured for 197 lines of this population both in long days, which promote rapid flowering in many A. *thaliana* strains, and in short days. Flowering time was measured using days to flowering (DTF) as well as the total number of leaves (TLN), partitioned into rosette and cauline leaves. In total, eight phenotypes were measured, namely days to flowering (DTF), cauline leaf number (CLN), rosette leaf number (RLN), and total leaf number (TLN) in long days (LD), and DTF, CLN, RLN, and TLN in short days (SD). Thus, the final dataset consists of 197 observations for 189 variables (8 phenotypes and 181 genotypes - SNP markers).

```
data(thaliana)
head(thaliana, n = 3)
     DTF_LD CLN_LD ... DTF_SD CLN_SD RLN_SD TLN_SD snp1  ... snp181
[1,] 17.58   3.42  ...  56.92  12.42  50.92  63.33    2  ...    2
[2,] 17.00   2.58  ...  53.33   8.42  41.58  50.00    0  ...    2
[3,] 27.50   8.08  ...  69.17  15.17  66.92  82.08    2  ...    0

set.seed(12)
out <- netphenogeno(thaliana)
sel <- selectnet(out)

# Steps to visualize the network
cl <- c(rep("red", 8), rep("white",56), rep("yellow2",31),
          rep("gray",33), rep("lightblue2",31), rep("salmon2",30))

id <- c("DTF_LD","CLN_LD","RLN_LD","TLN_LD","DTF_SD","CLN_SD",
        "RLN_SD", "TLN_SD","snp16", "snp49","snp50", "snp60","snp83",
         "snp86", "snp113","snp150", "snp155","snp159","snp156",
         "snp161","snp158", "snp160","snp162", "snp181")
```

```
plot(sel, vis= "interactive", n.mem= c(8,56,31,33,31,30),
       vertex.color= cl, label.vertex= "some", sel.nod.label= id,
       edge.color= "gray", w.btw= 200, w.within= 20, tk.width = 900,
       tk.height = 900)
```

The *A.thaliana* genotype-phenotype network in Fig 5 reveals those SNP markers that are directly affect flowering phenotypes. For example, markers $snp158$, $snp159$, $snp160$, and $snp162$ on chromosome 5 with the assay IDs 44607857, 44606159, 44607242, and 44607209 regulate the phenotype days to flowering (DTF-LD). For the same phenotype, Balasubramanian et al. (2009) have reported a wider range of markers (from $snp158$ to $snp162$ with the assay ID 44607857 to 44607209) that associate with DTF-LD. Our obtained smaller markers set is the result of controlling for all possible effects. In particular, the proposed method finds that $snp161$ does not show any association with DTF-LD after adjustments, but $snp159$, $snp160$ and $snp162$ on chromosome 5 do show an association with DTF-LD, even after taking into account the effect of all other SNPs and phenotypes. Therefore, the `netphenogeno()` function reduces the number of candidate SNPs and gives a small set of much more plausible ones. Moreover, Balasubramanian et al. (2009) have reported that the TLN-SD phenotype is associated with a region in chromosome 5, whereas our proposed method do not find any direct effect between TLN-SD and the region in chromosome 5, only through the DTF-SD phenotype. Furthermore, associations between phenotype CLN-LD and markers $snp49$ and $snp50$ have remained undetected in the previous studies of this population. This example was run in about 4 minutes on an Intel i7 laptop with 16 GB RAM.

In short, unlike traditional QTL analysis, the proposed method goes beyond the bivariate testing of individual SNPs, which only look at marginal association, instead it uses a multivariate approach which includes all the SNPs and phenotypes simultaneously.

Genotype-phenotype networks in maize. The high-dimensional genotypic and phenotypic maize data used in this paper were downloaded from `www.panzea.org`. The data comprised three datasets: a genotype data, and two phenotype datasets from the flowering time (Buckler et al., 2009) and the leaf architecture (Tian et al., 2011). The SNP data included 1106 genetic markers for 194 diverse maize recombinant inbred lines, which were derived from a cross between B73 and B97 from the maize Nested Association Mapping (NAM) populations. The 194 maize lines were scored for their flowering time using days to silking (DS), days to anthesis (DA), and the anthesis-silking interval (ASI) phenotypes. The leaf related traits such as upper leaf angle (ULA), leaf length (LL) and leaf width (LW) were also measured for all 194 maize lines.

Fig 6 reconstructs genotype–phenotype networks between the 6 phenotypes and 1106 SNPs. Five SNPs on chromosome 1 (from i140 until i144) directly affect both DA and DS traits (related to flowering time) after removing the effect of other variables. Moreover, a few SNPs on chromosome 1 (from i60 until i64) and on the beginning of chromosome 2 (i188 until i191) regulate DS. Two SNP markers (i762 and i763) on chromosome 7 affect DA, and chromosome 8 (i877 until i883) regulates ASI phenotype, after adjustments. The two leaf related traits, ULA and LL, are linked together, but not to the LW. Three SNPs i1064, i1062, and i1080 are yet conditionally associated to both LL and LW traits after adjustments. Chromosomes 4 and 6 do not have any role in the studied flowering time and leaf architecture traits.

Simulations and computational timing

The package generates simulated data in two ways

1. `simgeno()` function simulates genotype data based on a Gaussian copula graphical model. An inbred genotype data can be generated for `p` number of SNP markers, for `n` number of individuals, for `k` genotype states in a q-ploid species where q represents chromosome copy number (or ploidy level of chromosomes). The simulated data mimic a genome-like graph structure: First, there are `g` linkage groups (each of which represents a chromosome); then within each linkage group adjacent markers, `adjacent`, are linked via an edge as a result of genetic linkage. Also, with probability `alpha` a pair of non-adjacent markers in the same chromosome are given an edge. Inter-chromosomal edges are simulated with probability `beta`. These links represent long-range linkage disequilibriums. The corresponding positive definite precision matrix Θ has a zero pattern corresponding to the non-present edges. The underlying variable vector Z is simulated from either a multivariate normal distribution, $N_p(0, \Theta^{-1})$, or a multivariate t-distribution with degrees of freedom `d` and covariance matrix Θ^{-1}. We generate the genotype marginals using random cutoff-points from a uniform distribution, and partition the latent space into `k` states. The function can be called with the following arguments

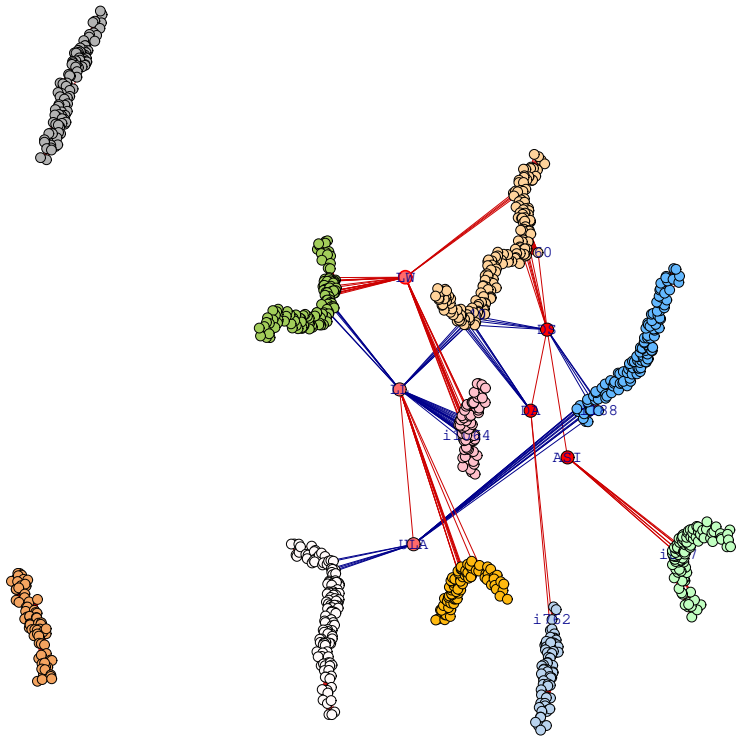


Figure 6: Genotype–phenotype networks for 1106 SNP markers and 6 phenotypes in mazie NAM population, where flowering traits (DS, DA, ASI) are shown in ● and leaf traits (LW, LL, ULA) are in ●, respectively. SNPs are shown on chromosome 1 (snp1 - snp175) as ○, chromosome 2 (snp176 - snp302) as ●, chromosome 3 (snp303 - snp432) as ●, chromosome 4 (snp433 - snp543) as ●, chromosome 5 (snp544 - snp682) as ○, chromosome 6 (snp683 - snp760) as ●, chromosome 7 (snp761 - snp838) as ○, chromosome 8 (snp839 - snp944) as ○, chromosome 9 (snp945 - snp1029) as ●, and chromosome 10 (snp1030 - snp1106) as ○. Blue edges show negative and red positive partial correlations.

```
set.seed(2)
sim <- simgeno(p = 90, n = 200, k = 3, g = 5, adjacent = 3, alpha = 0.1,
beta = 0.02, con.dist = "Mnorm", d = NULL, vis = TRUE)
```

The output of the example is shown in Figure 7.

2. `simRIL()` function generates diploid recombinant inbred lines (RILs) using recombination fraction and a CentiMorgan position of markers across the chromosomes. The function can be called with the following arguments

```
set.seed(2)
ril <- simRIL(g = 5, d = 25, n = 200, cM = 100, selfing = 2)
ril$data[1:3, ]
      M1.1 M2.1 M3.1 M4.1 M5.1 M6.1 M7.1 M8.1  ... M24.5 M25.5
ind1    0    0    0    0    0    0    0    0    ...   0     0
ind2    2    1    1    1    1    1    2    2    ...   1     1
ind3    1    2    2    2    2    2    2    1    ...   0     0

ril$map
     chr  marker  cM
1     1   M1.1   0.000000
2     1   M2.1   4.166667
.
.
.
124   5   M24.5  95.833333
125   5   M25.5  100.00000
```

where `g` and `d` represent the number of chromosomes and the number of markers in each chromosome, respectively. The number of sample size can be specified by `n`. The arguments `cM` and `selfing` show the length of chromosome based on centiMorgan position and the number of selfing in the RIL population, respectively.

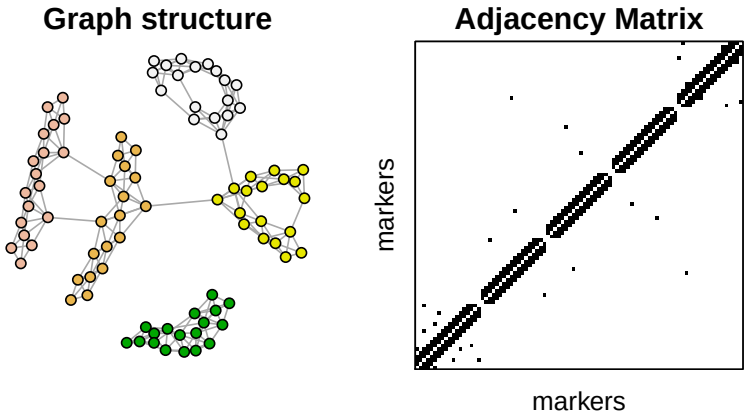


Figure 7: An example simulated (genotype) data using the function `simgeno()`. (left) Each node presents an SNP marker and the colors correspond to a given number of linkage groups, (right) the correspondent adjacency matrix. The connectivity between the pairs of non-adjacent markers in a same linkage group can be controlled via the argument `alpha` and the inter-chromosomal edges with `beta`.

Computational timing. Fig 8 shows computational timing of **netgwas** for different number of variables p and different sample sizes n. In this figure, we report computational timing in minutes for the genetic map construction, which includes the graph estimation procedure and the ordering algorithm. Note that the other two functions netsnp() and nethenogeno() include only the graph estimation, so we have only considered netmap() function to cover the computational aspect of the **netgwas** package. For the simulated data, we generated $p = 1000, 2000, 3500, 5000$ markers using simRIL function, which evenly are distributed across 10 chromosomes, for different individuals $n = 100, 200, 300$. Fig 8 shows that computational time is not affected by sample size n and is roughly proportional to p^3, as long as $p \times \max\{n, p\}$ elements can be stored in memory. The reported timing is based on the result from a computer with an Intel Core i7–6700 CPU and 32GB RAM.

5 Conclusion and future directions

The **netgwas** package implements the methods developed by Behrouzi and Wit (2019b) and Behrouzi and Wit (2019a) to (i) construct linkage maps for bi-parental species with any ploidy level, namely diploid (2 sets), triploid (3 sets), tetraploid (4 sets) and so on; (ii) explore high–dimensional short– and long–range linkage disequilibrium (LD) networks among pairs of SNP markers while controlling for the effect of other SNPs. The inferred LD networks reveal epistatic interactions across a genome when viability of the particular genetic recombination of the parental lines is considered as phenotype; (iii) infer genotype-phenotype networks from multi–loci multi–trait data, where it measures the pairwise associations with adjusting for the effect of other markers and phenotypes. Moreover, it detects markers that directly are responsible for that phenotype (disease), and reports the strength of their associations in terms of partial correlations. In addition, the package is able to reconstruct conditional dependence networks among SNPs, phenotypes, and environmental variables.

The implemented method is based on copula graphical models that enables us to infer conditional independence networks from incomplete non-Gaussian data, ordinal data, and mixed ordinal-and-continuous data. The package uses a parallelization strategy on multi-core processors to speed-up computations for large datasets. In addition, the code is memory-optimized, using the sparse matrix data structure when estimating and storing full regularization paths for large data sets. The **netgwas** package contains several functions for simulation and interactive network visualization. We note that reproducibility of our results and all the example data used to illustrate the package is supported by the open-source R package **netgwas**.

The **netgwas** and **qtl2** (Broman et al., 2019) software are for high-dimensional genotype and phenotype data. The **qtl2** performs QTL analysis in multi-parental populations solely by genome scans with single-QTL models. The function `netphnogeno` in **netgwas** uses a multivariate approach to detect conditional interactions networks between genotypes and phenotypes. It uses network models to detect multiple causal SNPs in a QTL region in bi-parental populations, while adjusting for the effect of remaining QTLs. One of the primary directions for the future work is to extend our methodology for multi-parental map construction and perform their QTL analysis. This would require the calculation of genotype probabilities using hidden Markov models (Broman and Sen, 2009; Zheng et al., 2018) before implementing the proposed Gaussian copula graphical model.

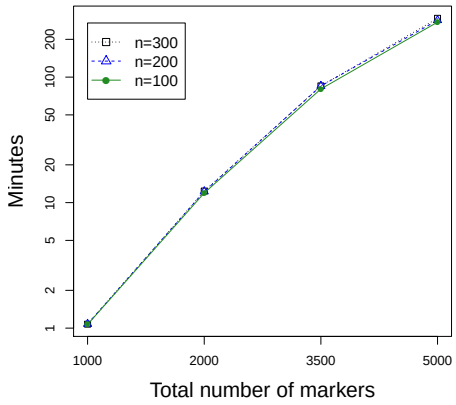


Figure 8: The computational time of linkage map construction in **netgwas** for various simulated data with different combinations of individuals n and the number of markers p, where they were distributed evenly across 10 linkage groups. This shows that the computational time is roughly proportional to p^3, as long as $p \times \max\{n, p\}$ elements can be stored in memory.

We will maintain and develop the package further. In the future we will include time component into our model where the interest is to infer dynamic networks for longitudinal (phenotyping) data and to learn changes of networks over time. Implementation of such model is desirable in many fields, particularly in plant breeding where the main goal is to optimize yield using high-throughput phenotypic data.

Acknowledgment

The authors are grateful to the associated editor and reviewers for their valuable comments that improved the manuscript and the R package.

Bibliography

S. Balasubramanian, C. Schwartz, A. Singh, N. Warthmann, M. C. Kim, J. N. Maloof, O. Loudet, G. T. Trainer, T. Dabi, J. O. Borevitz, et al. Qtl mapping in new arabidopsis thaliana advanced intercross-recombinant inbred lines. *PLoS One*, 4(2):e4318, 2009. [p29, 30]

D. Bates, M. Maechler, and M. Jagan. *Matrix: Sparse and Dense Matrix Classes and Methods*, 2022. URL https://CRAN.R-project.org/package=Matrix. R package version 1.4-1. [p25]

P. Behrouzi and E. C. Wit. Detecting epistatic selection with partially observed genotype data by using copula graphical models. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 68(1): 141–160, 2019a. [p19, 21, 22, 32]

P. Behrouzi and E. C. Wit. De novo construction of polyploid linkage maps using discrete graphical models. *Bioinformatics*, 35(7):1083–1093, 2019b. [p18, 27, 32]

P. Behrouzi, F. Abegaz, and E. C. Wit. Dynamic chain graph models for ordinal time series data. *ArXiv preprint ArXiv:1805.09840*, 2018. [p18]

D. Bikard, D. Patel, C. Le Mette, V. Giorgi, C. Camilleri, M. J. Bennett, and O. Loudet. Divergent evolution of duplicate genes leads to genetic incompatibilities within a. thaliana. *Science*, 323(5914): 623–626, 2009. [p28]

P. M. Bourke, G. van Geest, R. E. Voorrips, J. Jansen, T. Kranenburg, A. Shahin, R. G. Visser, P. Arens, M. J. Smulders, and C. Maliepaard. polymapr–linkage analysis and genetic map construction from f1 populations of outcrossing polyploids. *Bioinformatics*, 1:7, 2018. [p18]

K. W. Broman. A brief tour of r/qtl. *Disponivel http://www. rqtl. org/tutorials/rqtltour. pdf*, 2009. [p27]

K. W. Broman and S. Sen. *A Guide to QTL Mapping with R/qtl*, volume 46. Springer, 2009. [p32]

K. W. Broman, H. Wu, S. Sen, and G. A. Churchill. R/qtl: Qtl mapping in experimental crosses. *Bioinformatics*, 19(7):889–890, 2003. [p18, 27]

K. W. Broman, D. M. Gatti, P. Simecek, N. A. Furlotte, P. Prins, Ś. Sen, B. S. Yandell, and G. A. Churchill. R/qtl2: software for mapping quantitative trait loci with high-dimensional data and multiparent populations. *Genetics*, 211(2):495–502, 2019. [p18, 32]

E. S. Buckler, J. B. Holland, P. J. Bradbury, C. B. Acharya, P. J. Brown, C. Browne, E. Ersoz, S. Flint-Garcia, A. Garcia, J. C. Glaubitz, et al. The genetic architecture of maize flowering time. *Science*, 325 (5941):714–718, 2009. [p30]

W. S. Bush and J. H. Moore. Chapter 11: Genome-wide association studies. *PLoS Computational Biology*, 8(12):e1002822, 2012. [p19]

A. J. Butte, P. Tamayo, D. Slonim, T. R. Golub, and I. S. Kohane. Discovering functional relationships between rna expression and chemotherapeutic susceptibility using relevance networks. *Proceedings of the National Academy of Sciences*, 97(22):12182–12186, 2000. [p25]

L. Buzdugan, M. Kalisch, A. Navarro, D. Schunk, E. Fehr, and P. Bühlmann. Assessing statistical significance in multivariable genome wide association analysis. *Bioinformatics*, 32(13):1990–2000, 2016. [p19]

G. M. Clarke, C. A. Anderson, F. H. Pettersson, L. R. Cardon, A. P. Morris, and K. T. Zondervan. Basic statistical analysis in genetic case-control studies. *Nature Protocols*, 6(2):121–133, 2011. [p19]

E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *proceedings of the 1969 24th national conference*, pages 157–172. ACM, 1969. [p22, 23]

A. Dobra, C. Hans, B. Jones, J. R. Nevins, G. Yao, and M. West. Sparse graphical models for exploring gene expression data. *Journal of Multivariate Analysis*, 90(1):196–212, 2004. [p18]

D. Edwards, G. C. De Abreu, and R. Labouriau. Selecting high-dimensional mixed graphical models using minimal aic or bic forests. *BMC Bioinformatics*, 11(1):1–13, 2010. [p18]

R. Foygel and M. Drton. Extended bayesian information criteria for gaussian graphical models. *Advances in neural information processing systems*, 23, 2010. [p21]

J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008. [p21]

N. Friedman. Inferring cellular networks using probabilistic graphical models. *Science*, 303(5659): 799–805, 2004. [p18]

S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):721–741, 1984. [p21]

F. Grandke, S. Ranganathan, N. van Bers, J. R. de Haan, and D. Metzler. Pergola: Fast and deterministic linkage mapping of polyploids. *BMC Bioinformatics*, 18(1):12, 2017. [p18]

P. J. Green. On use of the em for penalized likelihood estimation. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 443–452, 1990. [p21]

C. A. Hackett, B. Boskamp, A. Vogogias, K. F. Preedy, and I. Milne. Tetraploidsnpmap: software for linkage analysis and qtl mapping in autotetraploid populations using snp dosage data. *Journal of Heredity*, 108(4):438–442, 2017. [p18]

A. J. Hartemink, D. K. Gifford, T. S. Jaakkola, and R. A. Young. Using graphical models and genomic expression data to statistically validate models of genetic regulatory networks. In *Biocomputing 2001*, pages 422–433. World Scientific, 2000. [p18]

W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. 1970. [p21]

Q. He and D.-Y. Lin. A variable selection method for genome-wide association studies. *Bioinformatics*, 27(1):1–8, 2010. [p19]

P. W. Hedrick. Gametic disequilibrium measures: proceed with caution. *Genetics*, 117(2):331–341, 1987. [p19]

P. D. Hoff. Extending the rank likelihood for semiparametric copula estimation. *The Annals of Applied Statistics*, 1(1):265–283, 2007. [p19]

C. J. Hoggart, J. C. Whittaker, M. De Iorio, and D. J. Balding. Simultaneous analysis of all snps in genome-wide and re-sequencing association studies. *PLoS Genetics*, 4(7):e1000130, 2008. [p19]

C.-J. Hsieh, I. S. Dhillon, P. K. Ravikumar, and M. A. Sustik. Sparse inverse covariance matrix estimation using quadratic approximation. In *Advances in Neural Information Processing Systems*, pages 2330–2338, 2011. [p21]

B. E. Huang, R. Shah, A. W. George, et al. dlmap: An r package for mixed model qtl and association analysis. *Journal of Statistical Software*, 50(6):1–22, 2012. [p18]

M. I. Jordan. Graphical models. *Statistical Science*, 19(1):140–155, 2004. [p18]

A. S. Kaler, J. D. Gillman, T. Beissinger, and L. C. Purcell. Comparing different statistical models and multiple testing corrections for association mapping in soybean and maize. *Frontiers in Plant Science*, 10:1794, 2020. [p19]

H. M. Kang, N. A. Zaitlen, C. M. Wade, A. Kirby, D. Heckerman, M. J. Daly, and E. Eskin. Efficient control of population structure in model organism association mapping. *Genetics*, 178(3):1709–1723, 2008. [p19]

H. M. Kang, J. H. Sul, S. K. Service, N. A. Zaitlen, S.-y. Kong, N. B. Freimer, C. Sabatti, and E. Eskin. Variance component model to account for sample structure in genome-wide association studies. *Nature Genetics*, 42(4):348–354, 2010. [p19]

C. A. Klaassen and J. A. Wellner. Efficient estimation in the bivariate normal copula model: normal margins are least favourable. *Bernoulli*, pages 55–77, 1997. [p19]

J. R. Klasen, E. Barbez, L. Meier, N. Meinshausen, P. Bühlmann, M. Koornneef, W. Busch, and K. Schneeberger. A multi-marker association method for genome-wide association studies without the need for population structure correction. *Nature Communications*, 7:13299, 2016. [p19]

W. Kruijer, P. Behrouzi, D. Bustos-Korts, M. X. Rodríguez-Álvarez, S. M. Mahmoudi, B. Yandell, E. Wit, and F. A. van Eeuwijk. Reconstruction of networks with direct and indirect genetic effects. *Genetics*, 214(4):781–807, 2020. [p19]

E. S. Lander, P. Green, J. Abrahamson, A. Barlow, M. J. Daly, S. E. Lincoln, and L. Newburg. Mapmaker: an interactive computer package for constructing primary genetic linkage maps of experimental and natural populations. *Genomics*, 1(2):174–181, 1987. [p18]

S. Lauritzen. *Graphical Models*, volume 17. Oxford University Press, USA, 1996. [p18]

S. L. Lauritzen and N. A. Sheehan. Graphical models for genetic analyses. *Statistical Science*, pages 489–514, 2003. [p18]

S. E. Lincoln and E. S. Lander. Systematic detection of errors in genetic linkage data. *Genomics*, 14(3): 604–610, 1992. [p27]

C. Lippert, J. Listgarten, Y. Liu, C. M. Kadie, R. I. Davidson, and D. Heckerman. Fast linear mixed models for genome-wide association studies. *Nature Methods*, 8(10):833–835, 2011. [p19]

H. Liu, F. Han, M. Yuan, J. Lafferty, L. Wasserman, et al. High-dimensional semiparametric gaussian copula graphical models. *The Annals of Statistics*, 40(4):2293–2326, 2012. [p21]

B. Mangin, A. Siberchicot, S. Nicolas, A. Doligez, P. This, and C. Cierco-Ayrolles. Novel measures of linkage disequilibrium that correct the bias due to population structure and relatedness. *Heredity*, 108(3):285, 2012. [p19]

G. Margarido, A. Souza, and A. Garcia. Onemap: Software for genetic mapping in outcrossing species. *Hereditas*, 144(3):78–79, 2007. [p18]

A. N. Massa, N. C. Manrique-Carpintero, J. J. Coombs, D. G. Zarka, A. E. Boone, W. W. Kirk, C. A. Hackett, G. J. Bryan, and D. S. Douches. Genetic linkage mapping of economically important traits in cultivated tetraploid potato (solanum tuberosum l.). *G3: Genes, Genomes, Genetics*, 5(11):2357–2364, 2015. [p26]

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953. [p21]

M. E. Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6): 066133, 2004. [p23]

O. A. Panagiotou, J. P. Ioannidis, and G.-W. S. Project. What should the genome-wide significance threshold be? empirical replication of borderline genetic associations. *International Journal of Epidemiology*, 41(1):273–286, 2011. [p19]

B. Rakitsch, C. Lippert, O. Stegle, and K. Borgwardt. A lasso multi-marker mixed model for association mapping with population structure correction. *Bioinformatics*, 29(2):206–214, 2012. [p19]

J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on computers*, 100(5): 401–409, 1969. [p22]

M. Simon, O. Loudet, S. Durand, A. Berard, D. Brunel, F. Sennesal, M. Durand-Tardif, G. Pelletier, and C. Camilleri. Qtl mapping in five new large ril populations of arabidopsis thaliana genotyped with consensus snp markers. *Genetics*, 178:2253–2264, 2008. [p27]

J. Taylor and D. Butler. R package asmap: Efficient genetic linkage map construction and diagnosis. *Journal of Statistical Software*, 79(6), 2017. [p18]

J. Taylor, A. Verbyla, et al. R package wgaim: Qtl analysis in bi-parental populations using linear mixed models. *Journal of Statistical Software*, 40(7):1–18, 2011. [p18]

F. Tian, P. J. Bradbury, P. J. Brown, H. Hung, Q. Sun, S. Flint-Garcia, T. R. Rocheford, M. D. McMullen, J. B. Holland, and E. S. Buckler. Genome-wide association study of leaf architecture in the maize nested association mapping population. *Nature Genetics*, 43(2):159, 2011. [p30]

V. Vinciotti, P. Behrouzi, and R. Mohammadi. Bayesian structural learning of microbiota systems from count metagenomic data. *arXiv preprint arXiv:2203.10118*, 2022. [p18]

H. Wang, F. A. van Eeuwijk, and J. Jansen. The potential of probabilistic graphical models in linkage map construction. *Theoretical and Applied Genetics*, pages 1–12, 2016. [p18]

D. Welter, J. MacArthur, J. Morales, T. Burdett, P. Hall, H. Junkins, A. Klemm, P. Flicek, T. Manolio, L. Hindorff, et al. The nhgri gwas catalog, a curated resource of snp-trait associations. *Nucleic Acids Research*, 42(D1):D1001–D1006, 2013. [p19]

R. Wu, C.-X. Ma, I. Painter, and Z.-B. Zeng. Simultaneous maximum likelihood estimation of linkage and linkage phases in outcrossing species. *Theoretical Population Biology*, 61(3):349–363, 2002. [p23]

Y. Wu, P. R. Bhat, T. J. Close, and S. Lonardi. Efficient and accurate construction of genetic linkage maps from the minimum spanning tree of a graph. *PLoS genetics*, 4(10):e1000212, 2008. [p18]

J. Yang, C. Hu, H. Hu, R. Yu, Z. Xia, X. Ye, and J. Zhu. Qtlnetwork: mapping and visualizing genetic architecture of complex traits in experimental populations. *Bioinformatics*, 24(5):721–723, 2008. [p18]

J. Yu, G. Pressoir, W. H. Briggs, I. V. Bi, M. Yamasaki, J. F. Doebley, M. D. McMullen, B. S. Gaut, D. M. Nielsen, J. B. Holland, et al. A unified mixed-model method for association mapping that accounts for multiple levels of relatedness. *Nature Genetics*, 38(2):203–208, 2006. [p19]

C. Zheng, M. P. Boer, and F. A. van Eeuwijk. Accurate genotype imputation in multiparental populations from low-coverage sequence. *Genetics*, 210(1):71–82, 2018. [p32]

K. Zych, Y. Li, J. K. van der Velde, R. V. Joosen, W. Ligterink, R. C. Jansen, and D. Arends. Pheno2geno-high-throughput generation of genetic markers and maps from molecular phenotypes for crosses between inbred strains. *BMC Bioinformatics*, 16(1):51, 2015. [p18]

Pariya Behrouzi
Applied Mathematics and Statistics - Biometris
Wageningen University and Research
6708PB, Wageningen
The Netherlands
(https://orcid.org/0000-0001-6762-5433)
pariya.behrouzi@wur.nl

Danny Arends
Albrecht Daniel Thaer-Institut für Agrar und Gartenbauwissenschaften
Humboldt-Universität zu Berlin

Invalidenstraße 42, 10115 Berlin
Germany
(https://orcid.org/0000-0001-8738-0162)
Danny.Arends@Gmail.com

Ernst C. Wit
Faculty of Science & Informatics
Universita della Svizzera Italiana (USI)
Via Buffi 13
6900 Lugano
Switzerland
(https://orcid.org/0000-0002-3671-9610)
wite@usi.ch

robslopes: Efficient Computation of the (Repeated) Median Slope

by Jakob Raymaekers

Abstract Modern use of slope estimation often involves the (repeated) estimation of a large number of slopes on a large number of data points. Some of the most popular non-parametric and robust alternatives to the least squares estimator are the Theil-Sen and Siegel's repeated median slope estimators. The **robslopes** package contains fast algorithms for these slope estimators. The implemented randomized algorithms run in $\mathcal{O}(n\log(n))$ and $\mathcal{O}(n\log^2(n))$ expected time respectively and use $\mathcal{O}(n)$ space. They achieve speedups up to a factor 10^3 compared with existing implementations for common sample sizes, as illustrated in a benchmark study, and they allow for the possibility of estimating the slopes on samples of size 10^5 and larger thanks to the limited space usage. Finally, the original algorithms are adjusted in order to properly handle duplicate values in the data set.

1 Introduction

The Theil-Sen estimator (Theil, 1950; Sen, 1968) is arguably the most popular non-parametric and robust alternative to the least squares estimator for estimating the slope in simple linear regression. A variation on this estimator, Siegel's repeated median slope (Siegel, 1982), was the first slope estimator to attain the maximal breakdown value of 50 %, which roughly means that it can withstand up to 50% of outliers in the data. Since their introduction, these estimators have seen widespread use in a variety of applications including signal extraction (Davies et al., 2004; Gather et al., 2006), filtering (Fried et al., 2006; Bernholt et al., 2006; Fried et al., 2007; Gelper et al., 2010), computer vision Meer et al. (1991), climatology (Zhang et al., 2000; Zhai et al., 01 Apr. 2005; Kosaka and Xie, 2013) and very recently differentially private estimation (Alabi et al., 2022; Fu et al., 2019).

Several implementations of these estimators exist, most notably in the R packages **deming** (Therneau, 2018), **zyp** (Bronaugh and for the Pacific Climate Impacts Consortium, 2019), **mblm** (Komsta, 2019), and **RobustLinearReg** (Hurtado, 2020), all publicly available on CRAN. Despite their popularity and importance, all of these publicly available implementations are based on a brute-force computation of the (repeated) median slope. Given a sample of n observations, this approach requires a computational cost of $\mathcal{O}(n^2)$ as well as $\mathcal{O}(n^2)$ space. This makes the currently available implementations unsuitable for modern applications involving larger data sets as they are rather slow and potentially use prohibitively large amounts of storage space.

Nevertheless, there exist several algorithms for the Theil-Sen and repeated median estimators which run in quasilinear time and require only $\mathcal{O}(n)$ space. For the Theil-Sen estimator, Cole et al. (1989), Katz and Sharir (1993), and Brönnimann and Chazelle (1998) proposed algorithms running in $\mathcal{O}(n\log(n))$ deterministic time. A randomized algorithm requiring $\mathcal{O}(n\log(n))$-expected time was proposed in Matoušek (1991), Dillencourt et al. (1992) and Shafer and Steiger (1993). For the repeated median slope, Stein and Werman (1992), Matoušek et al. (1993) and Matoušek et al. (1998) proposed algorithms running in $\mathcal{O}(n\log(n))$ and $\mathcal{O}(n\log(n)^2)$ (-expected) time. A potential explanation for these algorithms not being available in R may be twofold. Firstly, they are all more involved than brute-force computation of the median slopes and sample implementations in other programming languages are scarce, if available at all. Secondly, most of them make efficient use of features not (readily) available in R, such as pointers and in-place assignment, as well as complex data structures.

In this article we discuss the R package **robslopes** (Raymaekers, 2022) which contains implementations of the randomized algorithms by Matoušek (1991) and Matoušek et al. (1998) for the Theil-Sen and repeated median estimators respectively. The original algorithms have been adjusted in order to properly deal with potential duplicates in the data. We start by briefly reviewing the problem setting and the estimators. Next, we present a rough outline of the key ideas underlying the algorithms and a brief description of the usage of the functions in the package. We end with a benchmarking study comparing the implementation with existing implementations in publicly available R packages and concluding remarks.

2 The Theil-Sen and repeated median estimators

The typical problem setting of slope estimation is that of the simple linear model. Suppose we have n observations (x_i, y_i) which follow the model

$$y_i = \alpha + \beta x_i + e_i$$

where α and β are the (unknown) true intercept and slope parameters and e_i represents the noise. The most popular estimator for the regression coefficients is the ordinary least squares (OLS) estimator, i.e. the $(\hat{\alpha}, \hat{\beta})$ minimizing $\sum_{i=1}^{n} (y_i - \hat{\alpha} - \hat{\beta} x_i)^2$. The OLS estimator possesses several attractive properties, including ease of computation and interpretation and optimal performance when the errors are i.i.d. and follow a normal distribution. Despite these properties, it is well known that the OLS estimator is very sensitive to outliers and can have a quickly deteriorating performance as the error term deviates from normality. For these reasons, many alternatives have been suggested, of which the Theil-Sen estimator and Siegel's repeated median slope are two popular and intuitively attractive examples.

The Theil-Sen (TS) estimator (Theil, 1950; Sen, 1968) of β is defined as the median of the slopes of all the lines determined by two different observations (x_i, y_i) and (x_j, y_j):

$$\hat{\beta}_{\text{TS}}(\boldsymbol{x}, \boldsymbol{y}) = \text{med}_{i \neq j} \frac{y_i - y_j}{x_i - x_j},$$

where the median avoids $i = j$ as the slope through one point is undefined and where $\boldsymbol{x} = x_1, \ldots, x_n$ and $\boldsymbol{y} = y_1, \ldots, y_n$. In case there are duplicate values in $x_1, \ldots, x_n$, the proposal of Sen (1968) is to only include the slopes which are constructed using two observations with different x-values and we adopt this approach here.

The TS estimator has been analyzed extensively from a theoretical point of view. Sen (1968) showed its asymptotic normality and equivariance properties. In particular, we have

1. scale equivariance: $\hat{\beta}_{\text{TS}}(\boldsymbol{x}, c\boldsymbol{y}) = c\hat{\beta}_{\text{TS}}(\boldsymbol{x}, \boldsymbol{y})$ for all $c \in \mathbb{R}$
2. regression equivariance $\hat{\beta}_{\text{TS}}(\boldsymbol{x}, \boldsymbol{y} + a\boldsymbol{x}) = \hat{\beta}_{\text{TS}}(\boldsymbol{x}, \boldsymbol{y}) + a$ for all $a \in \mathbb{R}$,

but the estimator is not equivariant under affine transformations of both predictor and response variables. Wang and Yu (2005) give precise conditions for the unbiasedness of the TS estimator and (Wilcox, 1998) studied its behavior in the case of heteroscedastic errors. From a robust statistics perspective, it is known that the TS estimator is much more robust against outliers than the OLS estimator (see Rousseeuw and Leroy (2005)). In particular, it has a bounded influence function (Hampel et al., 1986) and its breakdown value is $1 - \sqrt{\frac{1}{2}} \approx 0.293\%$, where the latter can be roughly interpreted as the maximum percentage of contaminated samples that the estimator can handle (see Donoho and Huber (1983) for an exact definition). Finally, its maximum bias properties under contamination have been studied by Adrover and Zamar (2004).

The search for a slope estimator with a breakdown value higher than 30% prompted Siegel to propose the repeated median (RM) slope (Siegel, 1982). It is the first slope estimator attaining the maximal breakdown value of 50%. The repeated median estimator of β is computed by first calculating the median slope per observation x_i, yielding n values, and then taking the median of these values:

$$\hat{\beta}_{\text{RM}}(\boldsymbol{x}, \boldsymbol{y}) = \text{med}_i \, \text{med}_{j \neq i} \frac{y_j - y_i}{x_j - x_i},$$

where now the inner median avoids $i = j$. We handle duplicate values in $x_1, \ldots, x_n$ by skipping them in the computation of the inner median, in the same spirit as duplicate handling for the TS estimator. The consistency, unbiasedness and efficiency of the RM estimator were discussed by Siegel (1982), whereas the asymptotic normality was analyzed in Hossjer et al. (1994). The estimator was designed to have a 50 % asymptotic breakdown value, and like the TS estimator, its influence function is also bounded (Rousseeuw et al., 1993, 1995; Rousseeuw and Leroy, 2005). Finally, it possesses the same equivariance properties as the TS estimator, i.e. it is both scale and regression equivariant but not affine equivariant, and its maximum bias properties are also discussed in Adrover and Zamar (2004).

It is clear that both $\hat{\beta}_{\text{TS}}$ and $\hat{\beta}_{\text{RM}}$ can be computed by calculating all $\frac{n(n-1)}{2}$ pairwise slopes, and selecting the median or repeated median of these slopes. Clearly, this brute-force approach has a $\mathcal{O}(n^2)$ computational cost, and the available implementations of this approach also require storing the $\mathcal{O}(n^2)$ slopes. In the next section, we describe the more efficient algorithms implemented in the **robslopes** package.

While not the main focus of this article, it is worth mentioning that the intercept can be estimated in several ways. We opt for the most straight-forward approach of taking the median of the residuals

as the estimator for α:

$$\hat{\alpha} = \text{med}_i \left(y_i - \hat{\beta} x_i \right),$$

where $\hat{\beta}$ can be $\hat{\beta}_{\text{TS}}$ or $\hat{\beta}_{\text{RM}}$.

3 Randomized algorithms for slope selection

The R-package **robslopes** contains an implementation of the randomized algorithm by Matoušek (1991) for the Theil-Sen estimator and the algorithm of Matoušek et al. (1993, 1998) for the repeated median estimator. Unlike in their original proposals, we haven taken into account the case of possible duplicate values in the x_i and adjusted the algorithms to work properly in that case as well. We now briefly outline these algorithms without going too much into detail. For specific details and a complete description we refer to the original references and our code.

Suppose we are given n data points (x_i, y_i) with $i = 1, \ldots, n$ and we are interested in the median (or more generally, any order statistic) of the slopes formed by connecting two data points. The main idea behind the algorithms is to consider each observed data point (x_i, y_i) in dual space by associating it with the line

$$v = x_i u - y_i.$$

We will denote the coordinates in dual space with (u, v) in the following. It can be verified that the u-coordinate of the intersection of two lines in dual space (also called "intersection abscissa" (IA)) is equal to the slope of the line passing through the two points in the original space. Therefore, the problem of finding order statistics of slopes can be translated into the problem of finding order statistics of intersection abscissas in dual space. To find these order statistics in dual space, the algorithms use the following idea. Given an interval $(u_{\text{low}}, u_{\text{high}}]$ in dual space, the number of IAs in that interval can be computed by counting the number of inversions in a certain permutation. For a permutation π on $1, \ldots, n$, an inversion is a pair of elements $(\pi(i), \pi(j))$ for which $i < j$ and $\pi(i) > \pi(j)$. Figure 1 shows three lines in dual space and illustrates the connection between IAs and inversions which works as follows. Suppose we have two lines associated with the points (x_i, y_i) and (x_j, y_j). The v-coordinates of the intersection of these lines with u_{low} in dual space are given by $x_i u_{\text{low}} - y_i$ and $x_j u_{\text{low}} - y_j$ respectively. Suppose without loss of generality that $x_i u_{\text{low}} - y_i < x_j u_{\text{low}} - y_j$. Now, if lines i and j intersect in the interval $(u_{\text{low}}, u_{\text{high}}]$, we must have that the v-coordinate of the intersection of these lines with u_{high} have switched order: $x_i u_{\text{high}} - y_i > x_j u_{\text{high}} - y_j$. Therefore, there is a bijection between the IAs in $(u_{\text{low}}, u_{\text{high}}]$ and the permutation obtained by going from the order of the intersections of the lines with $u = u_{\text{low}}$ to the order of the intersections with $u = u_{\text{high}}$. Counting the number of inversions in a permutation can be done in $\mathcal{O}(n \log(n))$ time using an adaptation of merge sort Knuth (1998), and constructing the permutation itself has the same computational complexity.

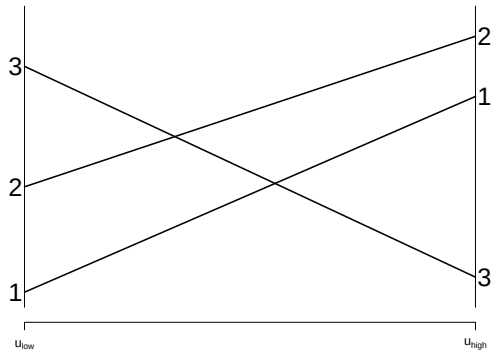


Figure 1: Three lines in dual space, restricted to the interval $(u_{\text{low}}, u_{\text{high}}]$. If two lines intersect in this interval, the ordering of their values of the v-coordinate at u_{low} and u_{high} must have turned around. Therefore, the number of IAs in the interval $(u_{\text{low}}, u_{\text{high}}]$ is given by the number of inversions in the permutation (3 1 2).

For the Theil-Sen estimator, we obtain $n(n-1)$ IAs in dual space, where parallel lines meet at $+\infty$ by convention. Suppose we want to find the k-th smallest IA (e.g., k could be $\lfloor (n(n-1)+1)/2 \rfloor$ for the lower median). The idea is to maintain a half-open interval $(u_{\text{low}}, u_{\text{high}}]$, which is initialized at $(-\infty, \infty]$ and always contains the IA we are looking for. We also keep a count L and C of the total number of IAs to the left and within the interval respectively. If the number of IAs in the interval is of order $\mathcal{O}(n)$, we can enumerate them in $\mathcal{O}(n \log(n))$ time and select the desired element (we call this "brute-force computation"). If there are more IAs left in the interval, we use a contraction

strategy which makes the interval $(u_{\text{low}}, u_{\text{high}}]$ progressively smaller while still containing the target k-th smallest slope, until it is small enough for brute-force computation.

In order to contract the interval, we randomly sample a $\mathcal{O}(n)$ number of IAs. These are then used to estimate a new interval $(u'_{\text{low}}, u'_{\text{high}}] \subset (u_{\text{low}}, u_{\text{high}}]$ which contains the target IA with high probability. To check whether the contracted interval contains the target, we count the number of IAs in the intervals $(u_{\text{low}}, u'_{\text{low}}]$ and $(u'_{\text{low}}, u'_{\text{high}}]$. If the current count L added to the number of IAs in the former interval exceeds k, we know that the target IA is in $(u_{\text{low}}, u'_{\text{low}}]$. If not, we check whether additionally adding the number of IAs in $(u'_{\text{low}}, u'_{\text{high}}]$ gives a total count exceeding k, in which case the target IA is in $(u'_{\text{low}}, u'_{\text{high}}]$. If neither of those cases hold, we know the target IA is in $(u'_{\text{high}}, u_{\text{high}}]$. After the contraction, we update L and C and this process is repeated until C, the number of IAs in the current interval, is of the order $\mathcal{O}(n)$. To execute this strategy, we need a method to randomly sample IAs. It turns out that this can be achieved in $\mathcal{O}(n \log(n))$ time, again using an adaptation of merge sort. For the Theil-Sen estimator there is an expected $\mathcal{O}(1)$ number of iterations required for convergence, leading to a total expected complexity of $\mathcal{O}(n \log(n))$.

For the repeated median estimator, the algorithm is similar to the previously described algorithm in that it also works with an interval-contraction strategy. In contrast with the previous algorithm however, we now keep track of the lines for which the median IA falls within the interval, in addition to the number of abscissas on the left and within the current interval for each individual line. In each contraction step, we first sample a $\mathcal{O}(\sqrt{n})$ number of lines for which we know that their median IA lies within the current interval. For each of these lines, a $\mathcal{O}(\sqrt{n})$ number of IAs are now sampled which allow for the estimation of a new interval $(u'_{\text{low}}, u'_{\text{high}}] \subset (u_{\text{low}}, u_{\text{high}}]$ containing the target slope with high probability. The interval $(u_{\text{low}}, u_{\text{high}}]$ can then be contracted by counting the number of IAs for each line on the left and within $(u'_{\text{low}}, u'_{\text{high}}]$. As before, the algorithm switches to brute-force computation once the number of IAs in $(u_{\text{low}}, u_{\text{high}}]$ is of the order $\mathcal{O}(n)$. For this algorithm, there is an expected $\mathcal{O}(\log(n))$ number of contraction steps required, resulting in a total complexity of $\mathcal{O}(n \log^2(n))$.

The original algorithms were only described for the case where the x_i values are all distinct. In case of duplicate values however, some of the slopes are not defined and the natural way of handling this is by ignoring the undefined slopes and computing the median (or any other order statistic) on the remaining slopes. Fortunately, we can incorporate this into our algorithms by realizing that these duplicate x_i values correspond with parallel lines in dual space. Using the convention that parallel lines in dual space meet at $+\infty$, the (repeated) median slope can be found by appropriately adjusting the value of the order statistic of the slope we want to find. For the Theil-Sen estimator, this means we need to find the order statistic corresponding with $\frac{n(n-1)}{2} - \sum_{i=1}^{n} \frac{d_i - 1}{2}$ values, where d_i denotes the number of times the value x_i occurs in the predictor variable. For the repeated median estimator, we only have to adjust the inner median, but the adjustment is dependent on the individual line i. More specifically, the order statistic for the inner median needs to be computed on $n - d_i$ values. Note that we can count the number of duplicates easily in $\mathcal{O}(n)$ time and so the overall complexity of the algorithm is not affected by this change. There is one additional complication, namely that of duplicate pairs (x_i, y_i). If such pairs are present, one should be careful with computing the permutation of intersections in dual space given an interval $(u_{\text{low}}, u_{\text{high}}]$. Obtaining such a permutation involves sorting (and ranking) a vector of IAs. However, in case of duplicate pairs, this sorting needs to be done using stable sort. If not, duplicate pairs may randomly produce inversions, yielding sampled IAs at ∞ even if the current interval has $u_{\text{high}} < \infty$. In the original algorithm, this scenario was not considered and thus not explicitly accounted for. These changes have been incorporated in the implementation provided in the **robslopes** package.

4 Implementation and usage

We briefly describe the implementation and usage of the functions in the **robslopes** package. The package revolves around 2 main functions, the `TheilSen` function and the `RepeatedMedian` function, both returning a list with the self-explanatory elements `intercept` and `slope`. Both functions start with input checking and duplicate counting, which are done in R. The duplicate counts are then used to set the correct target order statistics for the main part of the algorithm. This information is then passed on to the randomized algorithms which are implemented in C++, making use of the **rcpparmadillo** package (Eddelbuettel and François; Eddelbuettel and Sanderson, 2014).

The `TheilSen` function has the layout `TheilSen(x,y,alpha = NULL,verbose = TRUE)`, where the `x` and `y` arguments are the input vectors for the predictor and response variable respectively. The `verbose` option allows for switching off the printing of the computation progress. Finally, the `alpha` argument is a value between 0 and 1 which determines the order statistic corresponding with the target slope. When `alpha = NULL`, the default, the upper median of the m slopes is computed, which corresponds with the $\lfloor (m+2)/2 \rfloor$-th order statistic. For any other value of $0 \leq$`alpha`≤ 1, the function

computes the $\lfloor$alpha $m\rfloor$-th order statistic of the slopes, where $\lfloor\cdot\rfloor$ is the rounding operator.

The RepeatedMedian function has the layout RepeatedMedian(x,y,alpha = NULL,beta = NULL,verbose = TRUE), where the x, y and verbose arguments play the same role as for the TheilSen function. The arguments alpha and beta determine the order statistics of the inner and outer "median". When NULL, the default, they again correspond to the upper median. If they contain values between zero and one, the order statistics corresponding with the inner and outer "median" are given by $\lfloor$alpha $m\rfloor$ and $\lfloor$beta $m\rfloor$ respectively.

For convenience, the TheilSen and RepeatedMedian functions have been wrapped in the user-friendly functions robslope and robslope.fit. These mimic the structure of common regression functions (e.g., lm and lm.fit). In particular, robslope takes the standard arguments formula, data, subset, weights and na.action, in addition to the type argument selecting which type of slope to compute, as well as the optional alpha, beta and verbose arguments of the TheilSen and RepeatedMedian functions.

As an example we analyze the flights data of the **nycflights13** package (Wickham, 2019). It contains on-time data for all flights that departed in New York City (i.e. JFK, LGA or EWR airports) in 2013. We consider the distance traveled in miles as the predictor variable, and take the air time in minutes as the response. After removing all NA values, we end up with a data set of 327,346 observations. In contrast to the previous example, this one is too large to compute the TS or RM slope with a brute-force algorithm (on most computers). We now calculate the TS and RM slopes three times each, where we change the inner order statistic to the first, second and third quartiles. With the exception of graphical parameters, the code executed is shown below.

```
library("robslopes")
library("nycflights13")
data("flights")

ts.out.25 <- robslope(formula = air_time~distance, data = data, alpha = 0.25)
ts.out.50 <- robslope(formula = air_time~distance, data = data, alpha = 0.50)
ts.out.75 <- robslope(formula = air_time~distance, data = data, alpha = 0.75)
plot(data$distance, data$air_time)
abline(coef(ts.out.25), col = "red", lwd = 3)
abline(coef(ts.out.50), col = "green", lwd = 3)
abline(coef(ts.out.75), col = "blue", lwd = 3)

rm.out.25 <- robslope(formula = air_time~distance, data = data, beta = 0.25,
                      type = "RepeatedMedian")
rm.out.50 <- robslope(formula = air_time~distance, data = data, beta = 0.50,
                      type = "RepeatedMedian")
rm.out.75 <- robslope(formula = air_time~distance, data = data, beta = 0.75,
                      type = "RepeatedMedian")
plot(data$distance, data$air_time)
abline(coef(rm.out.25), col = "red", lwd = 3)
abline(coef(rm.out.50), col = "green", lwd = 3)
abline(coef(rm.out.75), col = "blue", lwd = 3)
```

The resulting figure is shown in Figure 2. In this example, the slopes calculated by the TS and RM estimators are virtually identical due to the very limited number of outliers and the large number of data points. Note that there are many duplicates in this data set, which are handled appropriately by the proposed implementation.

5 Benchmarking study

We now benchmark the implemented algorithm against the existing implementations in the **deming**, **zyp**, **mblm**, and **RobustLinearReg** packages. Note that the **deming** and **zyp** packages only contain the TS estimator, the others contain both the TS and RM estimators. All simulations were done using the R-package **microbenchmark** (Mersmann, 2019) and were run on an Intel® Core™ i7-10750H @2.60GHz processor. The setup is as follows. For each value of $n = \{10, 10^2, 10^3, 10^4\}$, we have generated 100 samples (x_i, y_i) from the bivariate standard normal distribution. Afterward, each of the available implementations was used to estimate the regression parameters and the execution time was measured (in nanoseconds).

The following code snippets show the code used for benchmarking the TS and RM estimator for a sample of size $n = 10^3$, the same code was used for the other sample sizes, with exception of $n = 10^4$,

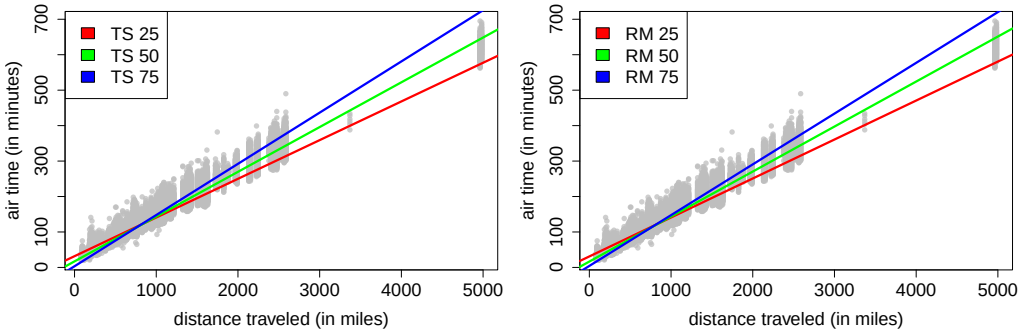


Figure 2: The TS (left) and RM (right) slopes fitted on the flights data. The first, second and third quartiles as the (inner) order statistic are shown in red, green and blue respectively. The slopes are virtually identical for both estimators in this example, as there are not many outliers in the data. The implemented algorithm appropriately deals with the many duplicate x-values in this data set.

in which case the implementation of the **mblm** package was left out due to its computational burden (it takes several hours to compute it once).

```
n   <- 10^3
mbm <- microbenchmark("deming" = deming::theilsen(y~x, data = data),
                      "zyp" = zyp::zyp.sen(y~x, dataframe = data),
                      "mblm" = mblm::mblm(y~x, dataframe = data, repeated = FALSE),
                      "RobustLinearReg" = RobustLinearReg::theil_sen_regression(y~x,
                      data = data),
                      "robslopes" = robslopes::TheilSen(x, y),
                      setup = {x = rnorm(n); y = rnorm(n);
                      data = as.data.frame(cbind(x, y))}, times = 100)
autoplot(mbm)

n   <- 10^3
mbm <- microbenchmark("mblm" = mblm::mblm(y~x, dataframe = data, repeated = TRUE),
                      "RobustLinearReg" = RobustLinearReg::siegel_regression(y~x,
                      data = data),
                      "robslopes" = robslopes::RepeatedMedian(x, y),
                      setup = {x = rnorm(n); y = rnorm(n);
                      data = as.data.frame(cbind(x, y))}, times = 100)
autoplot(mbm)
```

Figure 3 shows the results for the Theil-Sen estimator. We see that for the smallest sample size, $n = 10$, the absolute computation times are extremely low and while there are visible differences, they are probably not of practical relevance. For $n = 100$, the **robslopes** package is roughly five times faster than the fastest competitor, and the **mblm** implementation starts to run away from the other competitors. For $n = 10^3$, the differences become much clearer. The **robslopes** implementation is now between one and two orders of magnitude faster than the best competitor. Finally, for $n = 10^4$ the gap widens further to a difference of over two orders of magnitude. Note that for $n = 10^4$, we have left out the **mblm** package in the comparison as a single run takes several hours and the resulting plot would hinder a clear comparison.

Figure 4 shows the results for the RM estimator. As for the TS estimator, the difference in computation time is already visible for small sample sizes of $n = 10$ and $n = 100$, where the **robslopes** implementation is roughly a factor 20 faster than the competition. However, the absolute computation times for these sample sizes is by no means prohibitive for any of the implementations. Starting at $n = 1000$ however, we start to see a larger difference, especially between the **robslopes** and **mblm** implementations. The former is now roughly 800 times faster than the latter, and 60 times faster than the **RobustLinearReg** implementation. For sample size $n = 10^4$, we did not include the **mblm** implementation as it now takes roughly half an hour to compute, in contrast with 16 seconds for the **RobustLinearReg** implementation and 0.034 seconds (on average) for the **robslopes** implementation. Finally note that interestingly, the repeated median of the **mblm** package is much faster than the Theil-Sen estimator of the same package. This is somewhat counter-intuitive, but turns out to be caused by a much higher number of calls to the `c()` function (concatenate) in the implementation of the TS estimator.

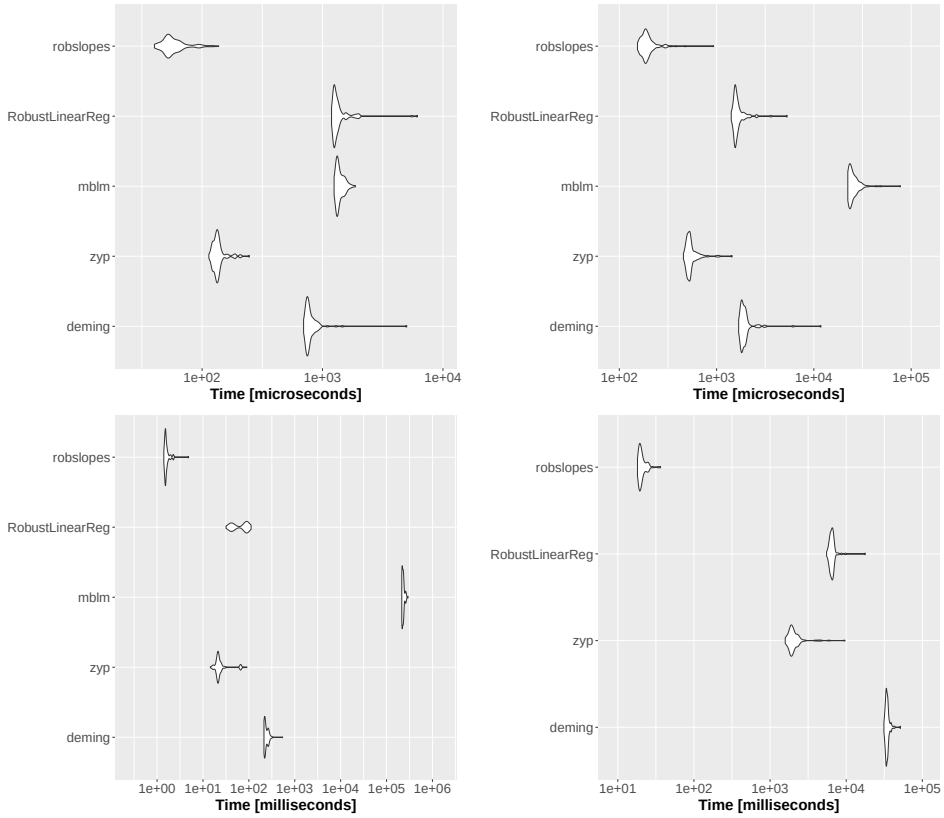


Figure 3: Computation times of the different implementations of the Theil-Sen slope estimator for the sample size n equal to 10, 10^2, 10^3 and 10^4 in the top left, top right, bottom left and bottom right panels respectively. The **robslopes** implementation is consistently faster than the competition by orders of magnitude. The difference scales close to linearly in n, gaining almost one order of magnitude advantage for a similar increase in the sample size, which is what we expect based on the theoretical computational complexities. The **mblm** estimator was left out for sample size $n = 10^4$ due to it requiring several hours to compute once.

As explained before, the brute-force implementations of the (repeated) median slope require $\mathcal{O}(n^2)$ storage as they typically compute all slopes and store them in a $n \times n$ matrix. Therefore, while a sample size of $n = 10^4$ is by no means extremely large, we cannot go much higher than that. A sample size of $n = 10^5$ for example, would require a RAM memory of about 75GB, which is rather uncommon on most computers and laptops. The implementations in the **robslopes** package however, require only $\mathcal{O}(n)$ storage space, and we can thus easily compute the estimators on much larger samples. To illustrate this, we have continued the benchmarking study with only the **robslopes** implementation, but this time for the sample sizes $n = 10^5, 10^6$ and 10^7. This was done using the following code snippet, where the saving of the results in the for loop is omitted:

```
for (n in 10^(5:7)) {
        mbm <- microbenchmark("robslopes" = robslopes::TheilSen(x, y),
                    setup = {x = rnorm(n); y = rnorm(n);
                    data = as.data.frame(cbind(x, y))}, times = 100)
}

for (n in 10^(5:7)) {
        mbm <- microbenchmark("robslopes" = robslopes::RepeatedMedian(x, y),
                    setup = {x = rnorm(n); y = rnorm(n);
                    data = as.data.frame(cbind(x, y))}, times = 100)
}
```

Figure 5 shows the mean computation times of the TS estimator and the RM estimator for sample sizes up to $n = 10^7$. As an example, for a sample size of $n = 10^6$, the TS estimator requires roughly 4 seconds of computation time, whereas the RM estimator requires roughly 6 seconds. The blue shade on the figure indicates the maximum and minimum computation time over the 100 replications, showing that the computation times have a fairly small variance around their mean. The red line

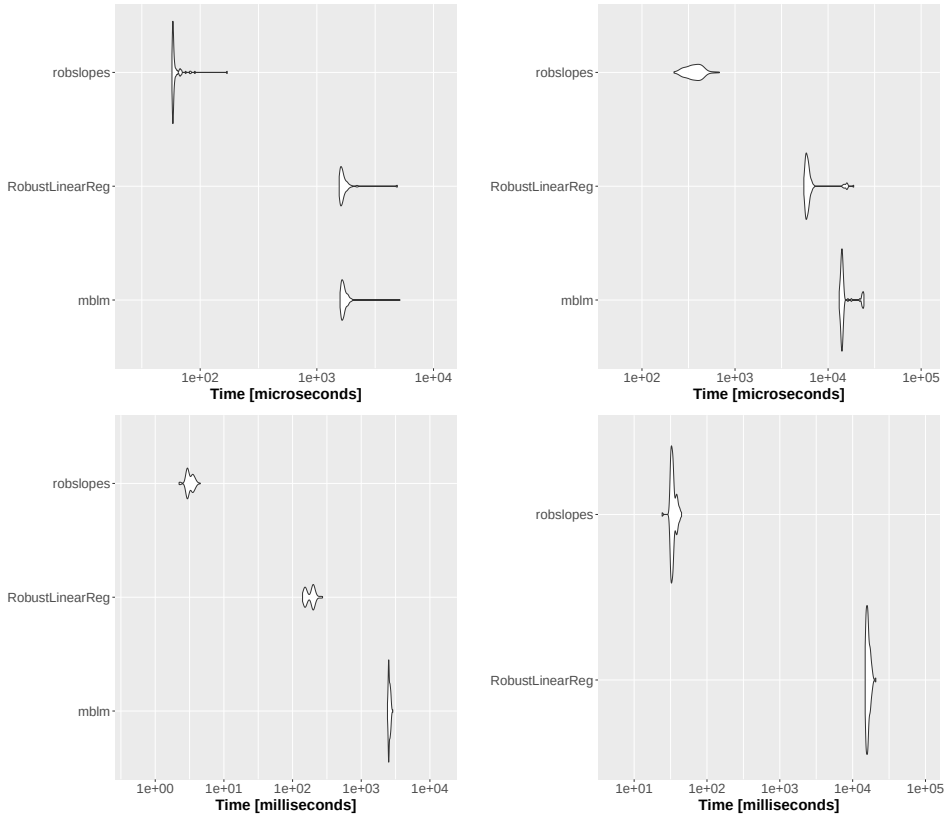


Figure 4: Computation times of the different implementations of the repeated median slope estimator for the sample size n equal to 10, 10^2, 10^3 and 10^4 in the top left, top right, bottom left and bottom right panels respectively. The **robslopes** implementation is consistently faster than the competition by orders of magnitude. The difference scales close to linearly in n, which is what we expect based on the theoretical computational complexities. Note that the **mblm** estimator was left out for sample size $n = 10^4$ due to it requiring more than 10^6 milliseconds (approx. 30 minutes) to compute.

shows a (robustly) estimated fit of the theoretical computation times to the observed ones (i.e. of the functions $f(n) = \beta n \log(n)$ and $f(n) = \beta n \log^2(n)$ for a $\beta \in \mathbb{R}$), indicating that for $n \geq 10^3$, the observed computational cost grows according to the theoretical complexity.

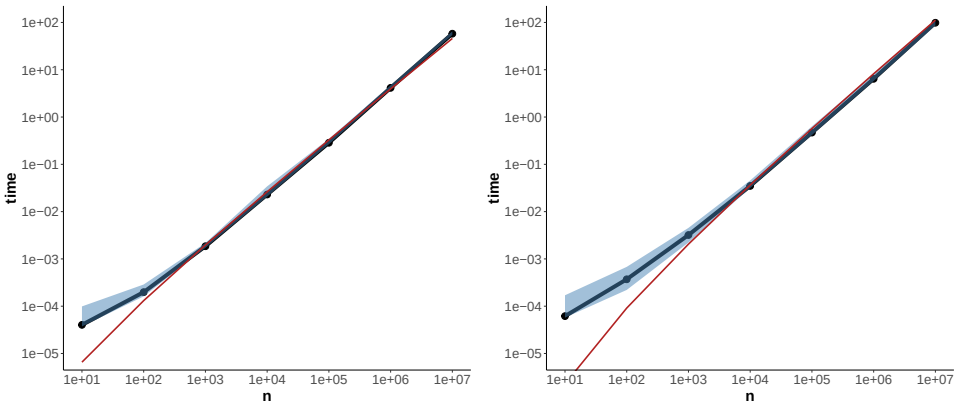


Figure 5: Mean computation times of the Theil-Sen (left) and repeated median (right) estimators as implemented in **robslopes** for increasing sample size n. The red line is an estimate of the $\sim n\log(n)$ (for TS) and $\sim n\log^2(n)$ (for RM) expected computational cost. The blue shade indicates the minimum and maximum computation times. Other than the deviations for very small n which are due to computational overhead, the computational cost scales precisely as the theory predicts. Furthermore, the variance of the computation times around their mean is negligible.

6 Summary

We have introduced the **robslopes** package which contains fast implementations of the popular Theil-Sen and repeated median slope estimators. The implemented algorithms are randomized algorithms running in quasilinear expected time and use linear space. In contrast, the currently available implementations in different R packages on CRAN require quadratic time and space. A benchmark study comparing the common implementations of the slope estimator with the newly introduced one illustrates speedups up to a factor 10^3 compared with the next best alternative implementation for common sample sizes. Additionally, due to the linear space requirements of the algorithms, the slope estimators can be computed on much larger sample sizes than the current maximum. Finally, the original algorithms were adjusted to properly deal with potential duplicates in the predictor variable.

The fast implementation of these popular slope estimators unlocks new possibilities for their use in modern applications where the slope has to be estimated repeatedly and on a large number of data points. Evidently, inferential procedures based on bootstrapping are also highly facilitated by these fast algorithms. Finally, the underlying C++ implementation may serve as a useful reference for implementations of these algorithms in other programming languages, which also seem to be scarce.

Bibliography

J. Adrover and R. H. Zamar. Bias robustness of three median-based regression estimates. *Journal of statistical planning and inference*, 122(1-2):203–227, 2004. URL https://doi.org/10.1016/j.jspi.2003.06.001. [p39]

D. Alabi, A. McMillan, J. Sarathy, A. Smith, and S. Vadhan. Differentially private simple linear regression. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2022(2):184–204, 2022. URL https://doi.org/10.2478/popets-2022-0041. [p38]

T. Bernholt, R. Fried, U. Gather, and I. Wegener. Modified repeated median filters. *Statistics and Computing*, 16(2):177–192, 2006. URL https://doi.org/10.1007/s11222-006-8449-1. [p38]

D. Bronaugh and A. W. for the Pacific Climate Impacts Consortium. *zyp: Zhang + Yue-Pilon Trends Package*, 2019. URL https://CRAN.R-project.org/package=zyp. R package version 0.10-1.1. [p38]

H. Brönnimann and B. Chazelle. Optimal slope selection via cuttings. *Computational Geometry*, 10(1):23–29, 1998. URL https://doi.org/10.1016/S0925-7721(97)00025-4. [p38]

R. Cole, J. S. Salowe, W. L. Steiger, and E. Szemerédi. An optimal-time algorithm for slope selection. *SIAM Journal on Computing*, 18(4):792–810, 1989. URL https://doi.org/10.1137/0218055. [p38]

P. L. Davies, R. Fried, and U. Gather. Robust signal extraction for on-line monitoring data. *Journal of Statistical Planning and Inference*, 122(1-2):65–78, 2004. URL https://doi.org/10.1016/j.jspi.2003.06.012. [p38]

M. B. Dillencourt, D. M. Mount, and N. S. Netanyahu. A randomized algorithm for slope selection. *International Journal of Computational Geometry & Applications*, 2(01):1–27, 1992. URL https://doi.org/10.1142/S0218195992000020. [p38]

D. Donoho and P. Huber. The notion of breakdown point. In P. Bickel, K. Doksum, and J. Hodges, editors, *A Festschrift for Erich Lehmann*, pages 157–184, Belmont, 1983. Wadsworth. [p39]

D. Eddelbuettel and R. François. Rcpp: Seamless r and c++ integration. URL https://doi.org/10.18637/jss.v040.i08. [p41]

D. Eddelbuettel and C. Sanderson. Rcpparmadillo: Accelerating r with high-performance c++ linear algebra. *Computational Statistics and Data Analysis*, 71:1054–1063, March 2014. URL http://dx.doi.org/10.1016/j.csda.2013.02.005. [p41]

R. Fried, T. Bernholt, and U. Gather. Repeated median and hybrid filters. *Computational statistics & data analysis*, 50(9):2313–2338, 2006. URL https://doi.org/10.1016/j.csda.2004.12.013. [p38]

R. Fried, J. Einbeck, and U. Gather. Weighted repeated median smoothing and filtering. *Journal of the American Statistical Association*, 102(480):1300–1308, 2007. URL https://doi.org/10.1198/016214507000001166. [p38]

S. Fu, C. Xie, B. Li, and Q. Chen. Attack-resistant federated learning with residual-based reweighting, 2019. URL https://doi.org/10.48550/ARXIV.1912.11464. [p38]

U. Gather, K. Schettlinger, and R. Fried. Online signal extraction by robust linear regression. *Computational Statistics*, 21(1):33–51, 2006. URL https://doi.org/10.1007/s00180-006-0249-8. [p38]

S. Gelper, R. Fried, and C. Croux. Robust forecasting with exponential and holt–winters smoothing. *Journal of forecasting*, 29(3):285–300, 2010. URL https://doi.org/10.1002/for.1125. [p38]

F. Hampel, E. Ronchetti, P. Rousseeuw, and W. Stahel. *Robust Statistics: The Approach Based on Influence Functions*. Wiley, New York, 1986. [p39]

O. Hossjer, P. J. Rousseeuw, and C. Croux. Asymptotics of the repeated median slope estimator. *The Annals of Statistics*, pages 1478–1501, 1994. URL https://doi.org/10.1214/aos/1176325638. [p39]

S. I. Hurtado. *RobustLinearReg: Robust Linear Regressions*, 2020. URL https://CRAN.R-project.org/package=RobustLinearReg. R package version 1.2.0. [p38]

M. J. Katz and M. Sharir. Optimal slope selection via expanders. *Information Processing Letters*, 47(3): 115–122, 1993. URL https://doi.org/10.1016/0020-0190(93)90234-Z. [p38]

D. E. Knuth. *The art of computer programming: Volume 3: Sorting and Searching*. Addison-Wesley Professional, 1998. [p40]

L. Komsta. *mblm: Median-Based Linear Models*, 2019. URL https://CRAN.R-project.org/package=mblm. R package version 0.12.1. [p38]

Y. Kosaka and S.-P. Xie. Recent global-warming hiatus tied to equatorial pacific surface cooling. *Nature*, 501(7467):403–407, 2013. URL https://doi.org/10.1038/nature12534. [p38]

J. Matoušek. Randomized optimal algorithm for slope selection. *Information processing letters*, 39(4): 183–187, 1991. URL https://doi.org/10.1016/0020-0190(91)90177-J. [p38, 40]

J. Matoušek, D. M. Mount, and N. S. Netanyahu. Efficient randomized algorithms for the repeated median line estimator. In *Proceedings of the Fourth ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 74–82, 1993. [p38, 40]

J. Matoušek, D. M. Mount, and N. S. Netanyahu. Efficient randomized algorithms for the repeated median line estimator. *Algorithmica*, 20(2):136–150, 1998. URL https://doi.org/10.1007/PL00009190. [p38, 40]

P. Meer, D. Mintz, A. Rosenfeld, and D. Y. Kim. Robust regression methods for computer vision: A review. *International journal of computer vision*, 6(1):59–70, 1991. URL https://doi.org/10.1007/BF00127126. [p38]

O. Mersmann. *microbenchmark: Accurate Timing Functions*, 2019. URL https://CRAN.R-project.org/package=microbenchmark. R package version 1.4-7. [p42]

J. Raymaekers. *robslopes: Fast Algorithms for Robust Slopes*, 2022. R package version 1.1.2. [p38]

P. Rousseeuw, C. Croux, and O. Hössjer. Sensitivity functions and numerical analysis of the repeated median slope. *Computational Statistics*, 10(1):71–90, 1995. [p39]

P. J. Rousseeuw and A. M. Leroy. *Robust regression and outlier detection*, volume 589. John wiley & sons, 2005. [p39]

P. J. Rousseeuw, N. S. Netanyahu, and D. M. Mount. New statistical and computational results on the repeated median line. In S. Morgenthaler, E. Ronchetti, and W. A. Stahel, editors, *New Directions in Statistical Data Analysis and Robustnes*, pages 177–194. Birkh¨auser-Verlag, Basel, 1993. [p39]

P. K. Sen. Estimates of the regression coefficient based on kendall's tau. *Journal of the American statistical association*, 63(324):1379–1389, 1968. URL https://doi.org/10.1080/01621459.1968.10480934. [p38, 39]

L. Shafer and W. Steiger. Randomizing optimal geometric algorithms. In *CCCG*, 1993. [p38]

A. F. Siegel. Robust regression using repeated medians. *Biometrika*, 69(1):242–244, 1982. URL https://doi.org/10.2307/2335877. [p38, 39]

A. Stein and M. Werman. Finding the repeated median regression line. In *SODA '92*, 1992. [p38]

H. Theil. A rank-invariant method of linear and polynomial regression analysis. i, ii, iii. *Nederl. Akad. Wetensch., Proc.*, 53:386—392, 521–525, 1397–141, 1950. [p38, 39]

T. Therneau. *deming: Deming, Theil-Sen, Passing-Bablock and Total Least Squares Regression*, 2018. URL https://CRAN.R-project.org/package=deming. R package version 1.4. [p38]

X. Wang and Q. Yu. Unbiasedness of the theil–sen estimator. *Journal of Nonparametric Statistics*, 17(6):685–695, 2005. doi: 10.1080/10485250500039452. URL https://doi.org/10.1080/10485250500039452. [p39]

H. Wickham. *nycflights13: Flights that Departed NYC in 2013*, 2019. URL https://CRAN.R-project.org/package=nycflights13. R package version 1.0.1. [p42]

R. Wilcox. A note on the theil-sen regression estimator when the regressor is random and the error term is heteroscedastic. *Biometrical Journal: Journal of Mathematical Methods in Biosciences*, 40(3):261–268, 1998. URL https://doi.org/10.1002/(SICI)1521-4036(199807)40:3<261::AID-BIMJ261>3.0.CO;2-V. [p39]

P. Zhai, X. Zhang, H. Wan, and X. Pan. Trends in total precipitation and frequency of daily precipitation extremes over china. *Journal of Climate*, 18(7):1096 – 1108, 01 Apr. 2005. doi: 10.1175/JCLI-3318.1. URL https://doi.org/10.1175/JCLI-3318.1. [p38]

X. Zhang, L. A. Vincent, W. Hogg, and A. Niitsoo. Temperature and precipitation trends in canada during the 20th century. *Atmosphere-ocean*, 38(3):395–429, 2000. URL https://doi.org/10.1080/07055900.2000.9649654. [p38]

Jakob Raymaekers
Maastricht University
Department of Quantitative Economics
6200 MD Maastricht
The Netherlands
j.raymaekers@maastrichtuniversity.nl

KU Leuven
Department of Mathematics
3001 Leuven
Belgium
jakob.raymaekers@kuleuven.be

Generalized Mosaic Plots in the ggplot2 Framework

by Haley Jeppson and Heike Hofmann

Abstract Graphical methods for categorical variables are not well developed when compared with visualizations for numeric data. One method available for multidimensional categorical data visualizations is mosaic plots. Mosaic plots are an easy and powerful option for identifying relationships between multiple categorical variables. Although various packages have implemented mosaic plots, no implementation within the grammar of graphics supports mosaic plots. We present a new implementation of mosaic plots in R, **ggmosaic**, that implements a custom **ggplot2** geom designed for generalized mosaic plots. Equipped with the functionality and flexibility of **ggplot2**, **ggmosaic** introduces new features not previously available for mosaic plots, including a novel method of incorporating a rendering of the underlying density via jittering. This paper provides an overview of the implementation and examples that highlight the versatility and ease of use of **ggmosaic** while demonstrating the practicality of mosaic plots.

1 Introduction

Graphical methods for categorical variables are not as thoroughly developed when compared with what is available for numeric variables. Categorical variables in scientific publications often appear in the form of bar charts (first published in the Commercial and Political Atlas by William Playfair in 1789, re-edited by Playfair, Wainer, and Spence (2005)) and spine plots (Hummel 1996).

Beyond those visualizations, categorical variables with a low number of levels are often incorporated in visualizations in the form of aesthetics such as color and shape. However, these indirect methods of visualizing categorical information are better suited as supplemental information and not as the primary source of information due to the associated loss of accuracy in retrieving the corresponding information (Cleveland and McGill 1984).

Mosaic plots provide a direct method of visualizing multidimensional categorical data; they are similar to a stacked bar chart but include the marginal distribution in addition to the conditional distribution, can be extended beyond two variables, and provide additional flexibility. Modern mosaic plots are usually attributed to Hartigan and Kleiner (Hartigan and Kleiner 1981, 1984; Kleiner and Hartigan 1981), but historical versions of mosaic plots can be found as far back as the late 1800s. The first mosaic plot is often attributed to Georg von Mayr (Friendly 2002); however, the mosaic plots in the Statistical Atlas of the 1870 Decennial Census (United States Census office. 9th census, 1870 and Walker 1874) pre-dates Mayr's by a few years. Mosaic plots may also be identified as Marimekko charts (mainly in the InfoVis world) or Eikosograms (Oldford et al. 2018).

In R, mosaic plots have been implemented in a variety of packages. Base R is equipped with the function `mosaicplot()` from the graphics package based on code by Emerson (1998) adapted by Kurt Hornik. The base R generic function `plot()` contains a method for `table` objects and produces a mosaic plot from a contingency table using the `mosaicplot()` function. Similarly, the **yardstick** package (Kuhn, Vaughan, and Hvitfeldt 2022) provides a method for the **ggplot2** (Wickham et al. 2020) function `autoplot()` to visualize confusion matrices, with one of the options being a mosaic plot.

The **vcd** package (Meyer, Zeileis, and Hornik 2020), influenced by Michael Friendly's "Visualizing Categorical Data," provides the functions `mosaic()` and `strucplot()` which allow for expanded versions of the original mosaic plot (Hartigan and Kleiner 1981). The **eikosogram** package (Oldford et al. 2018) is another R package capable of producing mosaic plots. The function `tileplot()` in the **latticeExtra** package (Sarkar and Andrews 2016) provides functionality to create mosaic plots in the **lattice** framework (Sarkar 2020).

The **productplots** package (Wickham and Hofmann 2016) provides a wrapper of **ggplot2** functionality but does not provide universal access to all aspects of the **ggplot2** framework like a geom does. The left plot in Figure 1 is an example of a mosaic plot created with the **productplots** package. While the default plot can be annotated with additional aesthetics, labels, themes, and color schemes, we can not add layers to this plot, and faceting is unavailable. A preview of the relevant code follows the next paragraph.

We present a new implementation of mosaic plots in R, **ggmosaic**, that implements a **ggplot2** geom for mosaic plots. While several other packages are available for constructing mosaic plots, **ggmosaic** allows users to fully leverage the widely used **ggplot2** framework resulting in a flexible package for generalized mosaic plots. More noteworthy, **ggmosaic** introduces several new features not previously

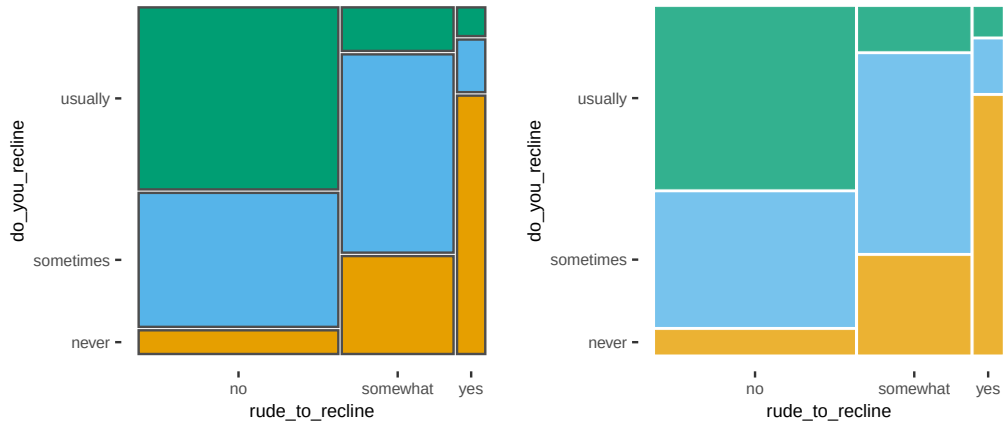


Figure 1: Example mosaic plots made with the productplots package (left) and the ggmosaic package (right). While the differences are subtle in this basic example, the differences between the two packages become more apparent as the customization of the mosaic plot increases.

available for mosaic plots, including a novel method of incorporating a rendering of the underlying density via jittering. **ggmosaic** includes a Shiny app to preview plots interactively and allow for model exploration. Many of the features we describe are possible because **ggmosaic** is implemented within the architecture defined by **ggplot2**. A preview of the syntax used in **ggmosaic** is shown below compared to that of the **productplots** package. The resulting plot is the right plot of Figure 1.

```
# productplots
productplots::prodplot(flights, ~do_you_recline + rude_to_recline, mosaic()) +
  aes(fill = do_you_recline)

## ggmosaic
ggplot(data = flights) +
   geom_mosaic(aes(x = product(do_you_recline, rude_to_recline),
                   fill = do_you_recline))
```

In this paper, we motivate the **ggmosaic** implementation and introduce it with examples demonstrating the versatility and ease of use of **ggmosaic**. Next, we describe new features available in the latest release of **ggmosaic**, version 0.3.3, and discuss how these features can enhance the data visualizations made with **ggmosaic**. Finally, we conclude with a preview of a **shiny** application (Chang et al. 2021) designed for an exploratory model framework of logistic regression and loglinear models. The Shiny application is included in the development version of **ggmosaic** available from https://github.com/hjeppson/ggmosaic.

2 ggmosaic: A ggplot2 implementation of mosaic plots

ggplot2 implements an adaptation of the grammar of graphics (Wilkinson 1999), a layered grammar (Wickham 2010). Because of its flexibility and ease of use, **ggplot2** has become one of the most popular plotting packages available in the R ecosystem. Version 2.0.0 of **ggplot2** introduced a method for other R packages to implement custom geometries, or "geoms", allowing for an expansion of the utility of **ggplot2**. In turn, the increasingly popular package has continued to appeal to a more extensive user base.

In creating a **ggplot2** geom for mosaic plots, we seek to appeal to this user base and leverage the robust framework of the grammar of graphics. Having mosaic plots in the **ggplot2** framework makes creating mosaic plots more accessible as it is more straightforward for the novice user to create mosaic plots for data exploration purposes. For many users, having mosaic plots function as a true **ggplot2** geom reduces the amount of syntax required, eliminates the need for prior calculations, and provides complete access to additional benefits of **ggplot2**, allowing for highly customized generalized mosaic plots.

With the R package **ggmosaic**, a custom **ggplot2** geom designed for generalized mosaic plots is implemented. **ggmosaic** makes mosaic plots compatible with **ggplot2** and creates a flexible method

to generate a wide variety of categorical data visualizations. The remainder of the paper presents a thorough description of the **ggmosaic** package featuring examples that go beyond how to use **ggmosaic** and demonstrate how to make more informed decisions about how to use mosaic plots to answer various questions.

We begin with an introduction to the package with examples of the flexible framework that **ggmosaic** offers. The following three sections illustrate how to use **ggmosaic**. First, we address the data structures required for constructing mosaic plots and which structures are compatible with **ggmosaic**. Next, we demonstrate how **ggmosaic** fits mosaic plots into the **ggplot2** framework and how to define the aesthetic mappings to construct the desired plot from the variables in the data. Lastly, we provide examples of mosaic plots customized with parameters unique to **ggmosaic**.

The remaining half of the paper presents the new features included in version 0.3.3 of **ggmosaic**. Frist, we introduce `geom_mosaic_text()`, designed to place text, or labels, in the center of each tile, followed by `geom_mosaic_jitter()`, a geom for jittered points designed to be superimposed on the mosaic plot and with multiple proposed applications. Next, we present `theme_mosaic()`, a theme for mosaic plots to remove items from the background of the plot. We conclude with an overview of an interactive Shiny app for exploratory data analysis (EDA) of high-dimensional categorical data using mosaic plots.

Many of the features we describe are possible because **ggmosaic** is implemented within the **ggplot2** framework. While other packages are available for constructing mosaic plots, **ggmosaic** allows users to leverage the widely used **ggplot2** infrastructure to create highly customized generalized mosaic plots. Most notably, **ggmosaic** introduces several new features not previously available for mosaic plots. In total, **ggmosaic** offers users a way to make mosaic plots with **ggplot2** to visualize and explore categorical data more effectively.

3 The ggmosaic package

The R package **ggmosaic** implements a custom **ggplot2** geom, `geom_mosaic()`, designed to offer a flexible framework for visualizing categorical data. The geom for mosaic plots was created using the **productplots** package and **ggplot2**'s custom object-oriented system and extension mechanism, ggproto. A stable version of **ggmosaic** is available on CRAN, and a development version is available at https://github.com/hjeppson/ggmosaic. This section provides a brief description of the package, the methods used in its implementation, and an example that showcases the flexibility that **ggmosaic** offers.

Designed to create visualizations of categorical data, `geom_mosaic()` has the capability to produce bar charts, stacked bar charts, mosaic plots, and double-decker plots and therefore offers a wide range of potential plots. Figures 2, 3, and 4 highlight the package's versatility. The code for these examples is provided below and can be summarized as consisting of the following components. First, the mosaic plot is created by adding a mosaic geom. The `x` aesthetic takes one or more variables wrapped in the `product()` function, and the `fill` aesthetic determines the fill color of the rectangles. An optional `divider` parameter determines the partitioning strategy for the rectangles. The code will be explained in greater detail in the subsequent sections.

```
# one-dimensional examples:
# spine plot
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline),
                  fill = do_you_recline))

# bar chart
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline),
                  fill = do_you_recline),
              divider = "hbar")


# two-dimensional examples:
# mosaic plot (2 variables)
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline, rude_to_recline),
                  fill = do_you_recline))

# stacked bar chart
```

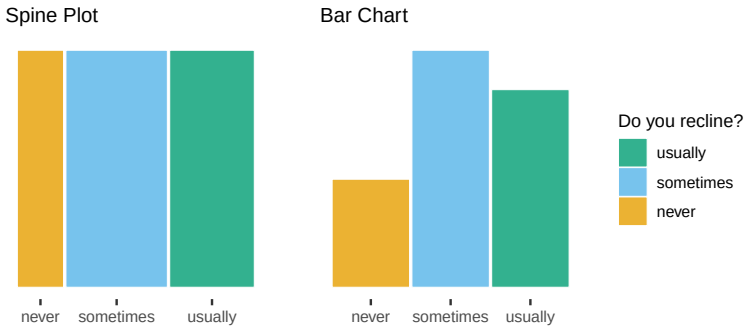


Figure 2: Both plots represent the distribution of `do_you_recline` and allow the same comparisons, though the spine plot does so with proportions and the bar chart with frequencies. The relative group sizes are more difficult to compare in the spine plot, but the `"sometimes"` group is still discernable as the largest and the `"never"` group as the smallest. The bar chart provides an easier comparisons of the relative group sizes.

```
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline, rude_to_recline),
                  fill = do_you_recline),
              divider = c("vspine", "hbar"))
```

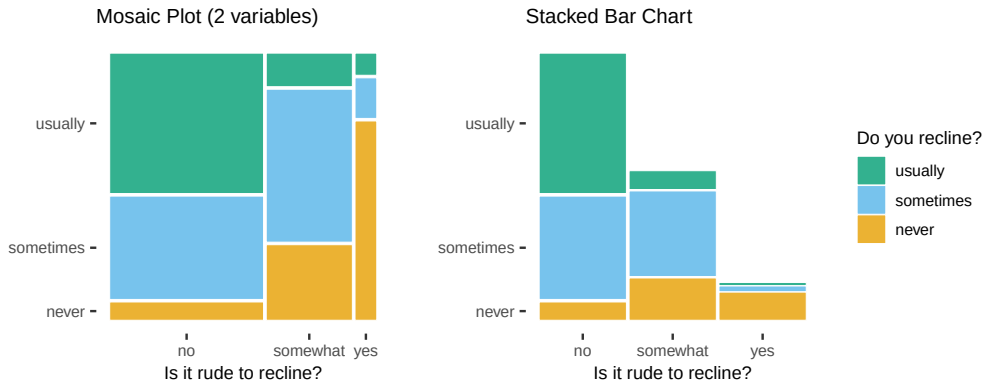


Figure 3: The conditional distribution of `do_you_recline` given `rude_to_recline` is represented by the heights of the bars in both the mosaic plot (left) and the stacked bar chart (right). The mosaic plot displays the joint and marginal distribution of `rude_to_recline` with the area and width of the tile, respectively. The stacked bar chart reveals a nearly equal number of never reclining responses across the categories of `rude_to_recline`. The conditional probabilities of `"never"` given `"somewhat"` and `"never"` given `"yes"` have a more significant impact in the mosaic plot, from which we understand that there is a positive correlation between a respondent considering reclining to be rude and electing never to recline their seat. There are, however, respondents that do not consider reclining to be rude and yet never recline, and perhaps more egregious, there are those that do regard reclining as rude and yet usually recline their seats, a group that is important in the mosaic plot but challenging to see in the stacked bar chart.

```
# three-dimensional examples:
# mosaic plot (3 variables)
ggplot(data = flights) +
  geom_mosaic(aes(x = product(eliminate_reclining, do_you_recline,
                              rude_to_recline),
                  fill = do_you_recline,
                  alpha = eliminate_reclining))

# double-decker plot
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline, eliminate_reclining,
                              rude_to_recline),
                  fill = do_you_recline,
                  alpha = eliminate_reclining),
              divider = ddecker())
```

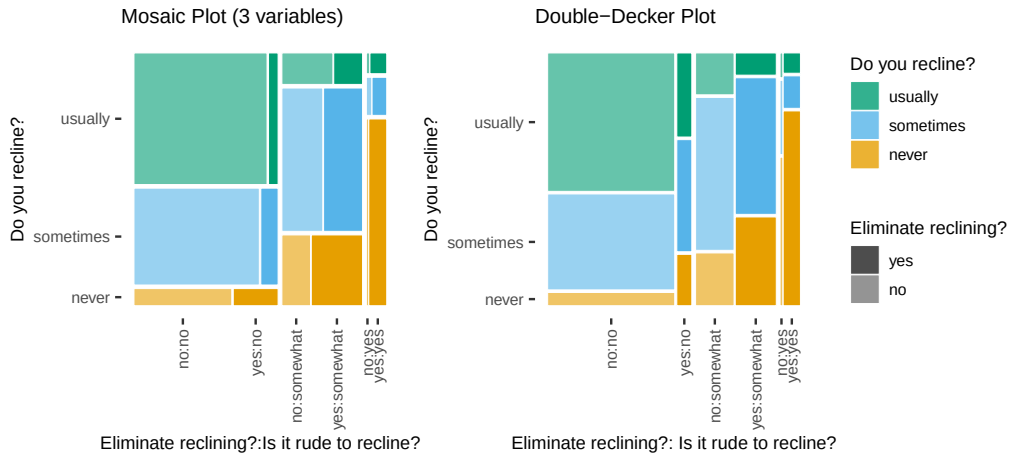


Figure 4: The mosaic plot (left) and double-decker plot (right) include the three variables (`do_you_recline`, `rude_to_recline`, `eliminate_reclining`), though the direction of the splits varies between the two plots. Both plots display a connection between a passenger's opinions of reclining and their tendency to recline. Those that consider reclining to be rude are less likely to recline and more likely to wish for reclining to be eliminated. The mosaic plot reveals an increasing desire to eliminate reclining when the respondents believe reclining to be rude across all categories of `do_you_recline`. There appears to be a slight positive correlation between reclining and not electing to eliminate reclining across all levels of `rude_to_recline`. The double-decker plot (a mosaic plot with a specific structure) best highlights the conditional distribution of `do_you_recline` and allows for easier comparisons across the categories of the other two variables. For example, the relationship between `do_you_recline` and `eliminate_reclining` seems consistent across categories of `rude_to_recline`, a conclusion that is less readily available from the mosaic plot.

Furthermore, **ggmosaic** allows various features to be customized:

- the **type of data structure**,
- the **order of the variables** (Figure 6),
- the **formula setup** of the plot (Figure 7),
- **faceting** (Figure 7),
- the **type of partition** (Figure 10 and Figure 11), and

- the **space between the categories** (Figure 12).

The following sections will discuss these features and provide examples of their use.

Data structure

The first step to creating a mosaic plot with **ggmosaic** is to ensure the data are in a supported format. There are three data structures to consider (Friendly 2016), and two of the three data structures are compatible with **ggmosaic**. The structure of the data impacts the use of `geom_mosaic()`, so the user must understand their data's structure. This section provides an example of each of the three data structures and highlights the essential differences.

The data set used for these examples, and all examples in this paper, is from a SurveyMonkey Audience poll conducted by FiveThirtyEight for two days in August of 2014. The survey asked twenty-six questions ranging from background information regarding the respondent to feelings on potentially aggravating behavior one might encounter on an airplane (Hickey 2014). The survey had 1,040 respondents (874 of whom had flown) aged 18-60+ from across the United States. The data set is available from FiveThirtyEight's data GitHub repository, and a cleaned version of it is one of the three data sets included in the **ggmosaic** package.

Here, we will use three variables `do_you_recline`, `rude_to_recline`, and `eliminate_reclining` from the `fly` data. For this paper, we made the following adjustments: remove all non-responses and collapse the variable `do_you_recline` from a factor with five levels (`always`, `usually`, `about half the time`, `once in a while`, and `never`) to a factor with three levels (`usually`, `sometimes`, `never`). Incidentally, by removing all non-responses, we also remove responses from those that have never flown. The code below completes the necessary modifications to the data.

```
# A few modifications to data
flights <- fly %>%
  select(do_you_recline, rude_to_recline, eliminate_reclining) %>%
  filter(!is.na(do_you_recline), !is.na(rude_to_recline)) %>%
  mutate(do_you_recline = do_you_recline %>%
           forcats::fct_collapse(
             usually = c("always", "usually"),
             sometimes = c("about half the time", "once in a while"),
             never = "never") %>%
           forcats::fct_relevel("never", "sometimes", "usually")
         )

# Summary of the modified data
flights %>% summary()

#>    do_you_recline rude_to_recline eliminate_reclining
#>  never    :170    no      :502    no :595
#>  sometimes:373    somewhat:281    yes:259
#>  usually  :311    yes     : 71
```

To simplify the examples of the three data structures, we will focus on the two variables `do_you_recline` and `rude_to_recline` from the `flights` data. The following code creates the desired subset:

```
flights_examp <- flights %>% select(do_you_recline, rude_to_recline)
names(flights_examp)

#> [1] "do_you_recline"  "rude_to_recline"
```

The example data, `flights_examp`, contains the individual observations from the survey. Each row accounts for one survey response to each of the questions, and there are two columns, one for `do_you_recline` and one for `rude_to_recline`. This is the first of the three types of data structure.

```
glimpse(flights_examp)

#> Rows: 854
#> Columns: 2
#> $ do_you_recline  <fct> sometimes, usually, usually, sometimes, usually, somet~
#> $ rude_to_recline <fct> somewhat, no, no, no, no, somewhat, no, no, yes, no, n~
```

The second data structure is a summary of the first data structure. This format consists of a data frame where each row is one of the possible combinations of levels of the categorical variables. In this structure, there is an additional column for the variable `freq` that supplies the number of observations in the first data structure with the row's particular combination of levels.

```
flights_examp %>%
  count(do_you_recline, rude_to_recline, name = "freq") %>%
  glimpse()
```

```
#> Rows: 9
#> Columns: 3
#> $ do_you_recline  <fct> never, never, never, sometimes, sometimes, sometimes, ~
#> $ rude_to_recline <fct> no, somewhat, yes, no, somewhat, yes, no, somewhat, yes
#> $ freq            <int> 35, 81, 54, 198, 164, 11, 269, 36, 6
```

The second data structure also occurs with weighted data. In which case, rather than a variable representing the counts or frequencies, a variable represents the weights. A typical example of weighted data is census data, where a weighting variable is used to compensate for a representation differential.

The final data structure summarizes the second data structure: it is the contingency table format. This data structure is not supported by **ggmosaic** and needs to be transformed into data structure one or two. Both `as.data.frame()` and `as_tibble()` provide a method for tables that converts this format into the summary format of data structure two.

```
flights_examp %>% table()
```

```
#>               rude_to_recline
#> do_you_recline  no somewhat yes
#>      never      35       81  54
#>      sometimes 198      164  11
#>      usually   269       36   6
```

While the contingency table data structure is incompatible with **ggmosaic**, or **ggplot2**, either the first or second structure is acceptable, and neither is preferable to the other. The data structure will affect the set of mappings constructed from the variables to the aesthetics, the topic of the next section.

Aesthetics

To fit **ggmosaic** within the **ggplot2** infrastructure, we must create the desired plot from the variables in the data. The first step, defining the aesthetic mappings, requires specifications of how variables in the data will be mapped to the visual properties of the plot. In **ggmosaic**, the aesthetic mappings are transformed into a model formula using the R formula notation with the ~ operator. The formula then determines what is represented in the mosaic plot, or how the joint distribution of the variables is broken down into the marginal distribution and conditional distribution(s). Understanding how the aesthetics translate into the model formula helps a user specify the correct aesthetics for the desired plot. This section describes each of the aesthetics used in `geom_mosaic()`, some of which are unique to **ggmosaic**, describes how the aesthetics are translated into the model formula, and provides examples of the impact changes in the aesthetic mappings have on the final plot.

Conflicting with the infrastructure provided by **ggplot2**, mosaic plots do not have a one-to-one mapping between a variable and the `x` or `y` axis. Instead, the grammar of graphics defines the coordinate system of mosaic plots as a system based on recursive partitioning that can integrate several variables (Wilkinson 1999). To accommodate the variable number of variables, the mapping to `x` is created by the `product()` function. For example, the variables `var1` and `var2` are read in as `x = product(var1, var2)`. However, if only one variable is to be included, it does not need to be wrapped in `product()`, and can be read in simply as `x = var1`. The `product()` function alludes to **ggmosaic**'s predecessor **productplots** and to the joint distribution as the product of the conditional and marginal distributions. The product function creates a special type of list that is evaluated internally and is what allows us to integrate multiple variables into the mosaic plot.

In order to include a variable in the mosaic plot, it must be specified as an aesthetic. In `geom_mosaic()`, the following aesthetics can be specified:

- `x`: select variables to add to the formula

  - declared as `x = product(var1, var2, ...)`

- `alpha`: add an alpha transparency to the rectangles of the selected variable
  - unless the variable is already part of the formula, it is added explicitly in the formula in the first position.
- `fill`: select a variable to determine the fill color of the rectangles
  - unless the variable is already part of the formula, it is added explicitly in the formula in the first position after the optional alpha variable.
- `conds`: select a variable to condition on
  - declared as `conds = product(cond1, cond2, ...)`
- `weight`: select a weighting variable

The specified aesthetics are then translated into the formula, `weight ~ alpha + fill + x | conds`, which determines what is to be represented in the mosaic plot and how to calculate the size of the corresponding tiles and determine their placement. Understanding the ordering of the specified aesthetics in the translation helps a user specify the correct aesthetics for the desired plot.

The minimal required aesthetics to create a mosaic plot is one variable mapped to the `x` aesthetic, wrapped in the `product()` function. Without a defined `weight` aesthetic, all observations are assumed to have equal weights, a weight of `1`. In this minimal scenario, the resulting formula is `1 ~ x`.

The `weight` aesthetic

A mosaic plot is constructed such that the area of each rectangle is proportional to the number of observations that the tile represents. Without weighting, each row of a data set represents one observation. Alternatively, the aesthetic `weight` is available to modify the interpretation of each row of the data. For example, the `weight` aesthetic will need to be used with data that contains a variable that supplies the number of observations for each row's particular combination of levels. In this case, if the `weight` aesthetic is left unused, the resulting mosaic plot will resemble the case of equal probabilities, as can be seen in Figure 5.

```
# unweighted plot
flights_examp %>%
  count(do_you_recline, rude_to_recline, name = "freq") %>%
  ggplot() +
  geom_mosaic(aes(x = product(rude_to_recline),
                  fill = do_you_recline))

# weighted plot
flights_examp %>%
  count(do_you_recline, rude_to_recline, name = "freq") %>%
  ggplot() +
  geom_mosaic(aes(weight = freq,
                  x = product(rude_to_recline),
                  fill = do_you_recline))
```

The `weight` aesthetic is also necessary with weighted data, such as weighted surveys. Otherwise, the resulting mosaic plot may not be a faithful representation of the relationship in the data. Here, weights serve to correct imbalances between the sample and the population.

The ordering of the variables

Because of a mosaic plot's hierarchical construction, the ordering of the variables in the formula is vital. While any order of the variables ultimately represents the same data, different variable orders may better support different comparisons. Ideally, the comparison of interest is positioned along a common scale, an easier perceptual task than comparing areas (Cleveland and McGill 1984). In the simple case of two variables, with one an explanatory variable and the other considered the response, the explanatory variable is best placed on the first split of the mosaic and the response variable as the last split, those that occur within each of the tiles created by the first split. The code below and corresponding mosaic plots in Figure 6 illustrate the change that occurs when the variables in the formula are swapped.

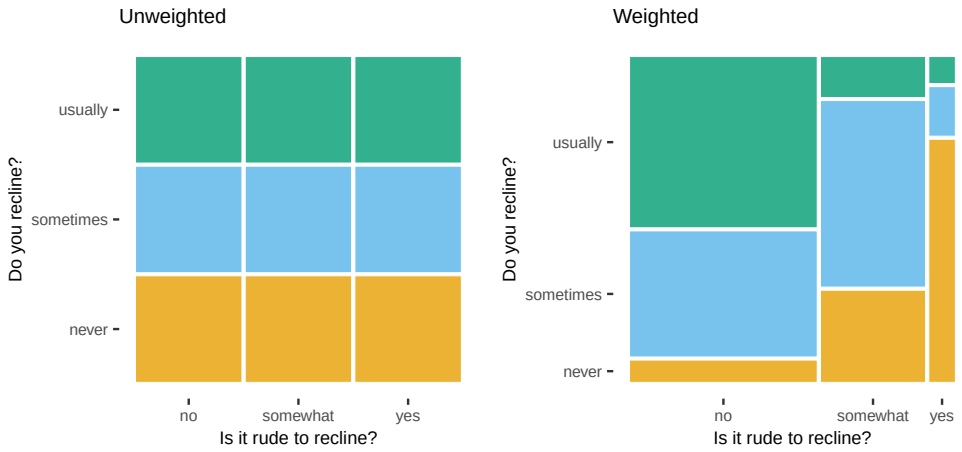


Figure 5: When using summarized data, a weighting aesthetic must be defined for the resulting mosaic plot to remain a faithful representation of the data. The mosaic plot on the left contains nine equally sized tiles for the nine combinations that result from the three levels of `rude_to_recline` crossed with the three levels of `do_you_recline`. The mosaic plot on the right considers the frequency of those nine combinations resulting in nine tiles proportional to the number of observations each tile represents.

```
# original order
ggplot(data = flights) +
geom_mosaic(aes(x = product(do_you_recline, rude_to_recline),
                fill = do_you_recline))

# order reversed
ggplot(data = flights) +
  geom_mosaic(aes(x = product(rude_to_recline, do_you_recline),
                  fill = do_you_recline))
```

The mosaic plots in Figure 6 represent the same joint distribution but are composed differently. The original order, the plot on the left, shows that most of those who believe it is rude to recline never recline their own seat. With the reversed order, we can see that of those who never recline their seats, a roughly equal proportion consider reclining rude as those that do not. The difference highlights how mosaic plots are more than a visualization of the joint distribution; mosaic plots are a visualization of the product of the conditional distribution and marginal distribution, which results in the joint distribution.

In the original order, `x = product(do_you_recline, rude_to_recline)`, the joint distribution is the product of the conditional distribution of `do_you_recline` given `rude_to_recline` and the marginal distribution of `rude_to_recline`. With this order, we can address questions regarding the marginal distribution of `rude_to_recline`, such as "what proportion of respondents believe it is rude to recline?", and questions regarding the conditional distribution of `do_you_recline` given each response to `rude_to_recline`, such as "Is there an association between a passenger's reclining tendencies and their opinion reclining.

In contrast, `x = product(rude_to_recline, do_you_recline)`, while resulting in the same joint distribution, is the product of the conditional distribution of `rude_to_recline` given `do_you_recline` and the marginal distribution of `do_you_recline`. Here, we can compare the responses to `rude_to_recline` within each response to `do_you_recline`, which reveals an interesting relationship between a passenger's perceived rudeness given their own inclination to recline. We can see that those that usually recline their seats are more likely to not find reclining to be a rude behavior. Those that never recline their seats, however, are evenly divided with their opinions on reclining. While it is easier to estimate the proportion that never reclines in the reversed order, it is more challenging to decipher what proportion of the respondents believe it is rude to recline, an artifact of the plot representing `rude_to_recline` only through its conditional distribution given the responses to `do_you_recline`.

Ultimately, the preferred order of the variables depends on the comparisons of interest. To (re)arrange the categories within a variable, the levels of the factor variable must be reordered before using **ggmosaic**.

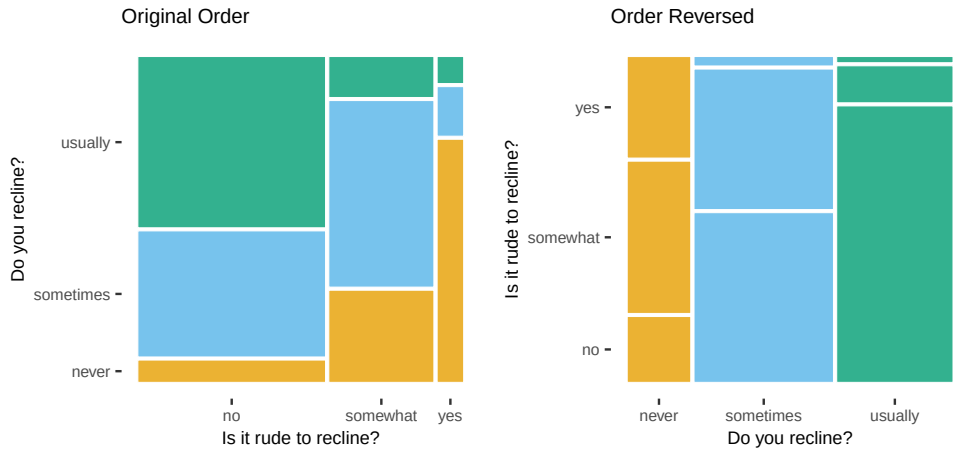


Figure 6: While the mosaics represent the same data, each supports different comparisons. The mosaic plot on the left emphasizes the effect `rude_to_recline` has on `do_you_recline`. The variable order was reversed in the construction of the mosaic plot on the right resulting in a plot that emphasizes the effect `do_you_recline` has on `rude_to_recline`.

The conds aesthetic

The formula setup of the plot can be further altered by electing to view a conditional distribution rather than the full joint distribution. Conditioning can be used to better focus on the comparison of interest either by removing relationships that are not of interest or positioning the comparison of interest along a common scale (Wickham and Hofmann 2011; Cleveland and McGill 1984). When a variable is mapped to the `conds` aesthetic, the mosaic plot is no longer represents the joint distribution, but instead maps the conditional distribution.

Faceting splits the data into subsets and generates a plot for each subset. Faceting therefore provides another method to accomplish conditioning. The result is the same as with the `conds` aesthetic, but the formatting is altered. The formatting difference will be discussed in more detail later. Figure 7 contains an example of each method of conditioning and is based on the code below.

```
# not conditioned
ggplot(data = flights) +
  geom_mosaic(aes(x = product(rude_to_recline),
                  fill = do_you_recline))

# conditioned with aesthetic mapping
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline),
                  fill = do_you_recline,
                  conds = product(rude_to_recline)))

# conditioned with facets
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline),
                  fill = do_you_recline)) +
  facet_grid(cols = vars(rude_to_recline))
```

For two variables, as in Figure 7, the difference between conditioning and not conditioning amounts to the difference between a mosaic plot and a stacked bar chart with a standardized height of 1. That is, when conditioning on `rude_to_recline`, the plot no longer represents the marginal distribution of `rude_to_recline`. Instead, as in a stacked bar chart, the plot only represents the conditional distribution of `do_you_recline` given the responses to `rude_to_recline`.

In the case of three variables, relationships can be removed via conditioning to focus on one relationship in particular and position that relationship along a common scale (Wickham and Hofmann 2011). Figure 8 provides an example of conditioning used to focus on the conditional distribution of `eliminate_reclining` by removing the joint distribution of `do_you_recline` and `rude_to_recline`. The example is based on the code below.

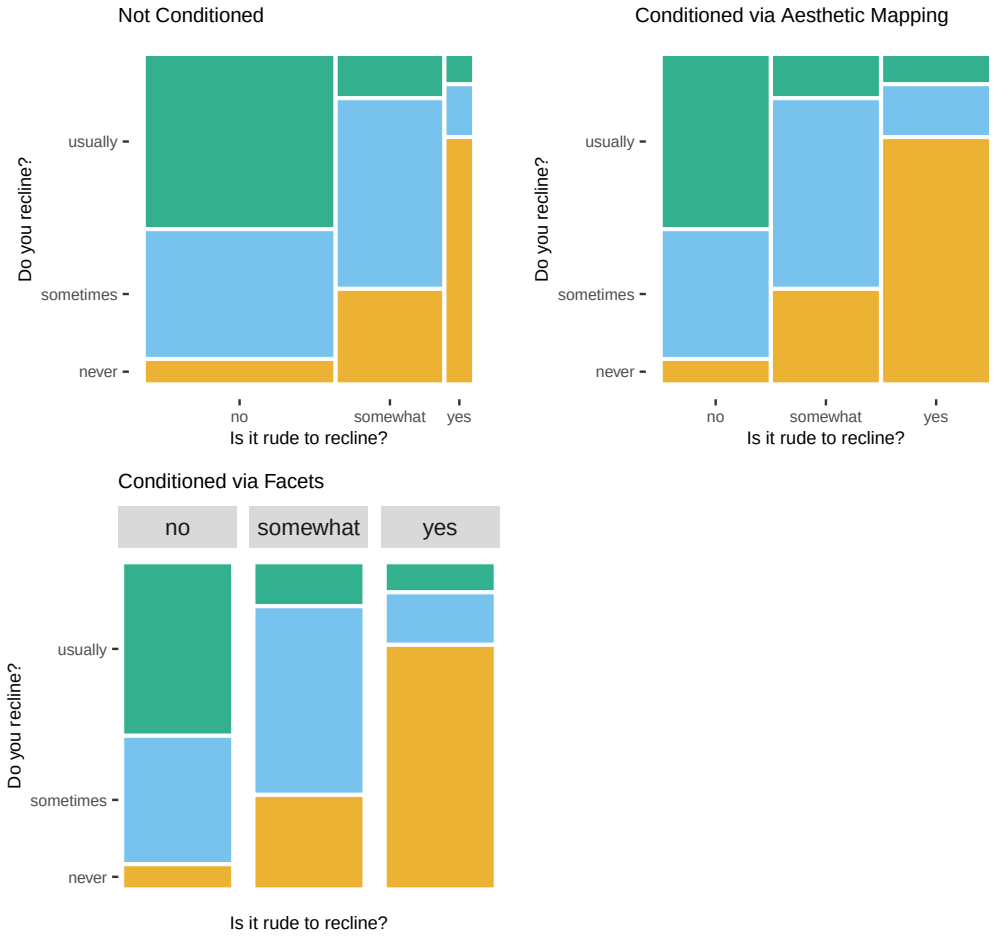


Figure 7: Mosaic plots typically depict the joint distribution (top left mosaic plot) but can instead depict the conditional distribution using facetting (bottom left mosaic plot) or the 'conds' aesthetic (top right mosaic plot). Conditioning on `rude_to_recline` may provide a clearer view of the relationship the responses to `do_you_recline` has with `rude_to_recline`. However, we lose information on the marginal distribution of `rude_to_recline`, and the area of a tile is only comparable to the other tiles within that same level of `rude_to_recline`.

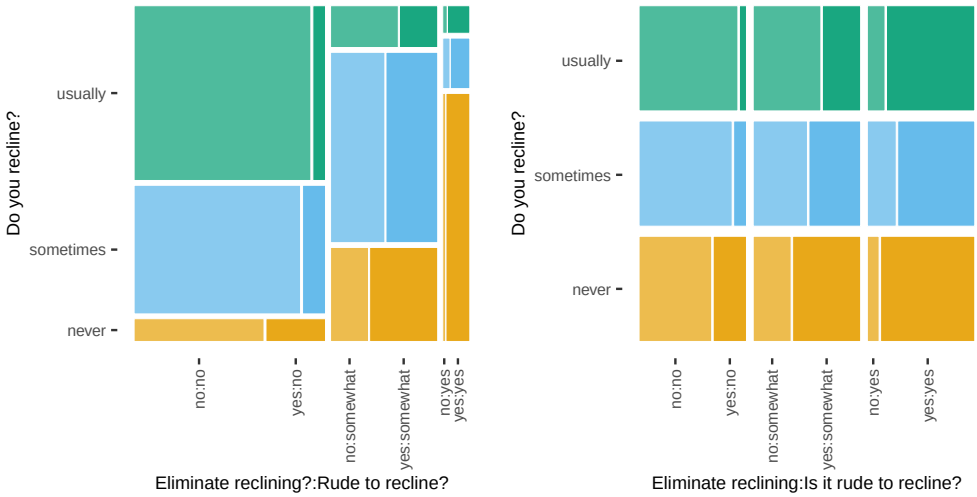


Figure 8: An example of using conditioning to focus on one relationship. The left mosaic plot represents the joint distribution of reclining tendencies, the perceived rudeness of reclining, and the desire to eliminate reclining, providing an overall picture. In the mosaic plot on the right, conditioning on `rude_to_recline` and `do_you_recline` removes the representation of the joint distribution of reclining tendencies and the perceived rudeness of reclining but provides a clearer view of the conditional distribution of `eliminate_reclining`.

```
# not conditioned
ggplot(flights) +
  geom_mosaic(aes(x = product(eliminate_reclining, do_you_recline,
                              rude_to_recline),
                  fill = do_you_recline,
                  alpha = eliminate_reclining))

# conditioned
ggplot(data = flights) +
  geom_mosaic(aes(x = product(eliminate_reclining),
                  conds = product(do_you_recline, rude_to_recline),
                  fill = do_you_recline,
                  alpha = eliminate_reclining))
```

The conditioning completed in Figure 8 provides a better view of how responses to `eliminate_reclining` relate to `do_you_recline` and `rude_to_recline` responses. We can see that desires to eliminate reclining are less likely when the respondent reclines their seat. `eliminate_reclining` may have a stronger association with responses to `rude_to_recline`, but that comparison is more difficult to make since it is comparing areas rather than positions along a common scale.

When conditioning on multiple variables, it is again important to consider how the variables' order impacts the plot. The two mosaic plots in Figure 9 display the same data, but the direction of the final split is different. The orientation of the final split determines which comparisons are positioned along a common scale.

```
# conditioned order 1
ggplot(data = flights) +
  geom_mosaic(aes(x = product(eliminate_reclining),
                  conds = product(do_you_recline, rude_to_recline),
                  fill = do_you_recline,
                  alpha = eliminate_reclining))

# conditioned order 2
ggplot(data = flights) +
  geom_mosaic(aes(x = product(eliminate_reclining),
                  conds = product(rude_to_recline, do_you_recline),
                  fill = do_you_recline,
```

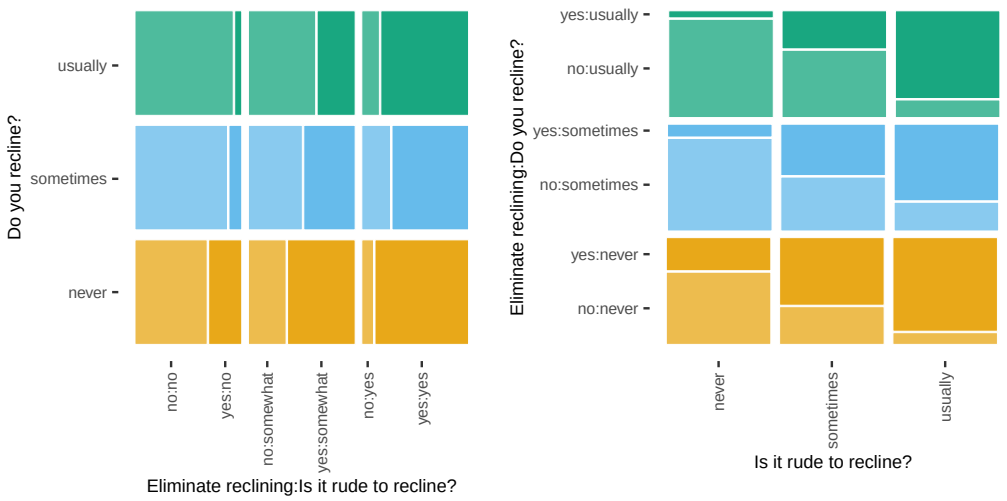


Figure 9: The two mosaic plots both portray the distribution of `eliminate_reclining` conditioned on `do_you_recline` and `rude_to_recline`, but the direction of the final split is different. The lefthand plot is better suited to compare `eliminate_reclining` with `do_you_recline`, while the righthand plot is better suited to compare `eliminate_reclining` with `rude_to_recline`. The comparison of interest should influence the order of the conditioning variables.

```
                alpha = eliminate_reclining)) +
  coord_flip()
```

Figure 9 rearranges the conditioned variables' order to gauge the strength of the relationship between `eliminate_reclining` and `rude_to_recline`. For a fixed level of `do_you_recline`, the desire to eliminate reclining correlates with an increasingly hostile stance on reclining. While this example flips the orders of the conditioned variables and then uses `coord_flip()`, **ggmosaic** provides additional methods of manually modifying the directions of the splits in the mosaic plot, which will be discussed in the next section.

The defined set of aesthetic mappings impacts more than the look of the final graphic; it impacts the analysis and the inquiries the plot can support. **ggmosaic** requires the `x` and `cond` aesthetics to be defined using the `product()` function to accommodate a variable number of variables. The defined set of aesthetic mappings will result in a model formula that will determine which of the potentially numerous ways the mosaic plot will represent the decomposition of the joint distribution.

Parameters

Easy customization is necessary for mosaic plots to be effective. Additional aspects of the mosaic plot that can be modified include the strategy used to partition the area into the tiles and the width of the spacing between the tiles. **ggmosaic** provides the parameters `divider` and `offset` to facilitate adjustments to the partitioning strategy and the spacing. This section demonstrates how to create generalized mosaic plots modified by these parameters and how they can be used to highlight different aspects of the data.

In **ggmosaic**, the desired type of partition is specified with the `divider` parameter (by setting `divider = " "`). The area of a mosaic plot can be partitioned into bars or spines, and partitions can be added horizontally or vertically, as shown in Figure 10 and the code below. When the area is partitioned into bars, the height is proportional to value, and the width equally divides the space. Bars can be arranged horizontally (`"hbar"`) or vertically (`"vbar"`). Alternatively, space can be partitioned into spines, where the section's width is proportional to the value, and the height occupies full range. Spines are space-filling and can be arranged horizontally (`"hspine"`) or vertically (`"vspine"`). The default divider for a single variable is `"hspine"`.

```
# default / hspine
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline),
                  fill = do_you_recline))
```

```
# vspine
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline),
                  fill = do_you_recline),
              divider = "vspine")

# hbar
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline),
                  fill = do_you_recline),
              divider = "hbar")

# vbar
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline),
                  fill = do_you_recline),
              divider = "vbar")
```

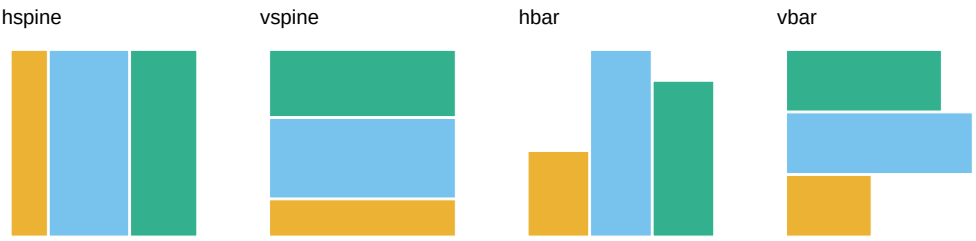


Figure 10: Examples of the four ways each dimension in a mosaic plot can be partitioned. While the values are easier to compare with the bars in this one-dimensional example, the advantages of spines become more apparent in higher-dimensional examples.

In the case of multiple variables, a type of partition must be defined for each variable. The default divider (`divider = mosaic()`) and the double-decker divider (`divider = ddecker()`) automatically select a pre-defined pattern for the partitions. For example, if three variables are plotted, the default, `divider = mosaic()`, partitions the plot with spines in alternating directions, beginning with a horizontal spine, i.e. `divider = c("hspine", "vspine", "hspine")`. Alternatively, we can declare the type of partition for each variable, e.g. `divider = c("hbar", "vspine", "vspine")`. The first partition declared in the vector will be used last in the plot. As mentioned above, an unspecified divider leads to the default `divider = mosaic()`, and the partition begins with a horizontal spine and alternate directions for each subsequent variable. To begin with a vertical spine and alternate directions from there, use `divider = mosaic(direction = "v")`. A preview of these options is available in Figure 11, and the associated code is below.

```
# default / mosaic("h")
ggplot(data = flights) +
  geom_mosaic(aes(x = product(eliminate_reclining, do_you_recline,
                              rude_to_recline),
                  fill = do_you_recline,
                  alpha = eliminate_reclining))

# mosaic("v")
ggplot(data = flights) +
  geom_mosaic(aes(x = product(eliminate_reclining, do_you_recline,
                              rude_to_recline),
                  fill = do_you_recline,
                  alpha = eliminate_reclining),
              divider = mosaic(direction = "v"))

# ddecker
```

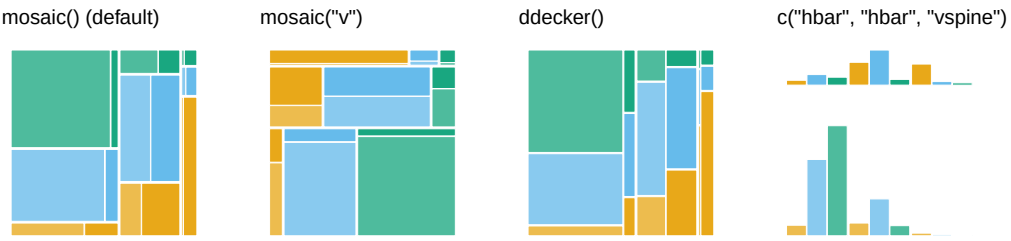


Figure 11: When multiple variables are included, a partition type must be defined for each variable, and the different partitions emphasize different relationships. The `mosaic()` and `ddecker()` functions automatically define a partition for each variable, and are shown in the first and third mosaic plots, respectively. `mosaic("v")`, shown in the second plot, is similar to `mosaic("h")`, but the coordinates are flipped and inversed. Alternatively, a type of partition can be defined manually for each dimension. `ddecker()` emphasizes the conditional distribution of the final variable. In the final plot, the divider `c("hbar", "hbar", "vspine")` creates a plot similar to a faceted bar chart that supports comparisons similar to those supported by the double-decker plot but for relative values instead of relative proportions.

```
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline, eliminate_reclining,
                              rude_to_recline),
                  fill = do_you_recline,
                  alpha = eliminate_reclining),
              divider = ddecker())

# c("hbar", "hbar", "vspine")
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline, rude_to_recline,
                              eliminate_reclining),
                  fill = do_you_recline,
                  alpha = eliminate_reclining),
              divider = c("hbar", "hbar", "vspine"))
```

The `divider` parameter provides an additional mechanism for modifying the plot's arrangement to focus on a particular relationship by enabling a user to modify any dimension of the divider to position the comparison of interest along a common scale. For instance, the double-decker plot in Figure 11 helps compare the effects changes in `eliminate_reclining` and `rude_to_recline` have on the responses to `do_you_recline` and how the relationship between `do_you_recline` and `eliminate_reclining` differs for different levels of `rude_to_recline`.

ggmosaic adopts the procedure followed by Hartigan and Kleiner (1981), Friendly (2002), Theus and Urbanek (2009), and Hofmann (2003), where an amount of space is allocated for each of the splits, with subsequent divisions receiving a smaller amount of space. The splits between the categories of the first variable are the widest and the splits decrease in width with each additional variable. Decreasing the widths of the splits with each additional variable allows the categories to group together according to the recursive strategy (Theus and Urbanek 2009) and the created spaces preserve the impact of small counts (Friendly 2002). The effect becomes apparent when an empty group is included. In this case, the spaces between the empty categories create a gap equal to the amount of space that is between non-empty categories.

For variables with many categories, it may be of interest to decrease the size of the spacing between the spines. The parameter `offset` can widen or shrink the size of the spacing between the categories of the first variable and the subsequent splits then gradually decrease in width with each additional variable. The default setting of this parameter is `offset = 0.01`, equivalent to 1% of the width of the plotting area. The code below and the corresponding mosaic plots in Figure 12 demonstrate how to use the `offset` parameter.

```
# increased spacing
ggplot(data = flights) +
  geom_mosaic(aes(x = product(rude_to_recline),
                  fill = do_you_recline),
```

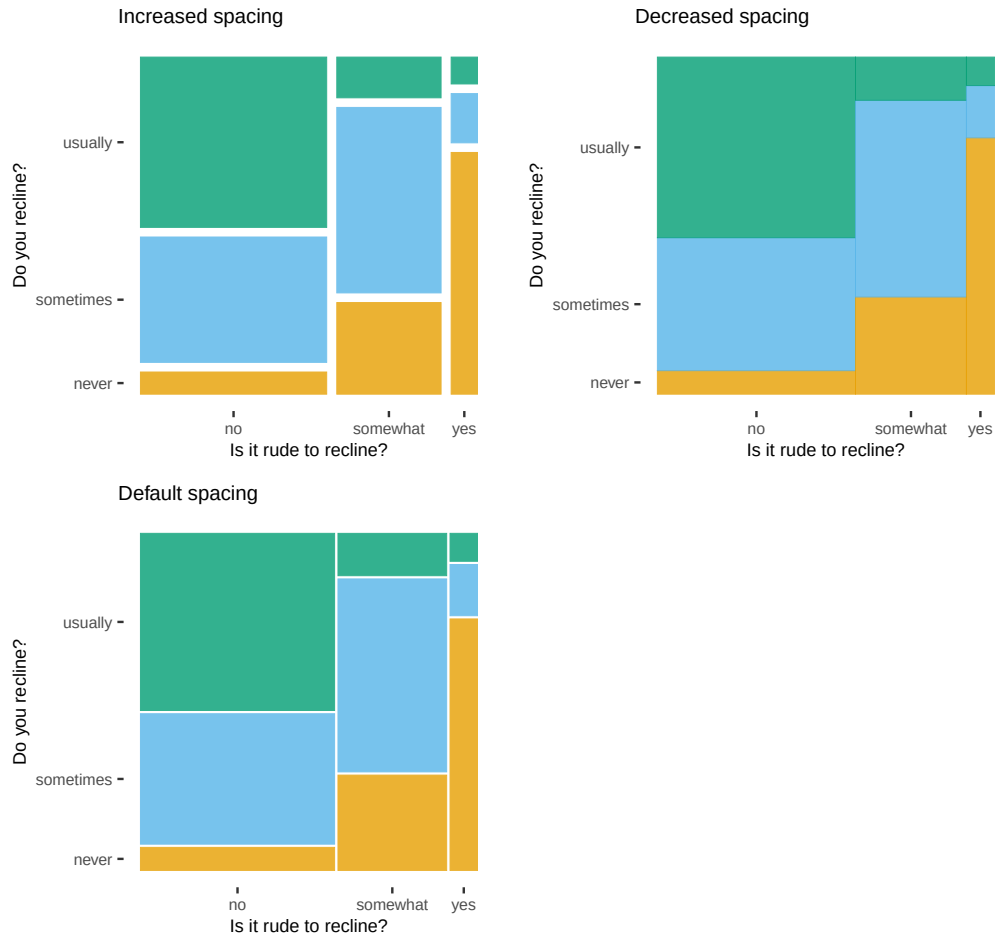


Figure 12: Three examples of how the spacing of the splits in the mosaic plot can be increased or decreased with the offset parameter.

```
              offset = 0.03)

# decreased spacing
ggplot(data = flights) +
  geom_mosaic(aes(x = product(rude_to_recline),
                  fill = do_you_recline),
              offset = 0)

# default spacing
ggplot(data = flights) +
  geom_mosaic(aes(x = product(rude_to_recline),
                  fill = do_you_recline))
```

While the examples so far include up to three variables, there is no technical limit to the number of variables in a mosaic plot. However, when many categorical variables are present, mosaic plots can quickly become cluttered and difficult to interpret, and a judgment call must be made on the practical limit to the number of variables included in the mosaic plot. This practical limit could depend on many factors, including the number of categories within each variable, the viewing size of the result mosaic plot, and the viewer's familiarity with the data.

The customization offered via the **ggmosaic** parameters allows users to easily make complex generalized mosaic plots that are more readable and simpler to interpret. The parameters provide additional dimensions of freedom with the ability to switch between bars and spines and allow users to highlight different aspects of high-dimensional categorical data to identify and communicate interesting patterns.

Figure 13: Users can create interactive mosaic plots using **ggmosaic** with **plotly**. With plotly, users gain the ability to hover over a tile to see the names of the combination of levels and the number of observations that the tile represents.

Interactivity

Having a geom designed for generalized mosaic plots allows for a `ggplotly()` hook to create interactive mosaic plots with the **plotly** package, version 4.9.3 (Sievert et al. 2016). The `ggplotly()` function translates most of the basic geoms bundled with the **ggplot2** package. To expand the functionality to custom geoms, we make use of the infrastructure provided in the **plotly** package that allows for a translation. In **ggplot2**, many geoms are special cases of other geoms. For example, `geom_line()` is equivalent to `geom_path()` once the data is sorted by the `x` variable. Because `GeomMosaic` can be reduced to the lower-level geom `GeomRect`, we were able to write a method for the `to_basic()` generic function in **plotly** (Sievert 2020). Figure 13 features an example of an interactive mosaic plot created with `ggplotly()` and the corresponding code is below.

```
p1 <- ggplot(data = flights) +
  geom_mosaic(aes(x = product(rude_to_recline),
                  fill = do_you_recline))

plotly::ggplotly(p1)
```

4 New features in ggmosaic version 0.3.3

The layered grammar implemented by **ggplot2** provides a means to add additional layers (typically resulting in additional geometric objects) to a graphic. The unique scale system in **ggmosaic**, however, makes a correct placement of items in additional layers tricky. The `x` and `y` scales in a mosaic plot are both numeric and categorical; the numeric scale determines the placement of additional objects on a range between `0` and `1`. Thus, to place an object in the center of each tile, it is necessary to know the numeric values for the corners of each of the tiles. Version 0.3.3 of **ggmosaic** introduced two additional geoms designed to build on `stat_mosaic()` and `geom_mosaic()` that bypass these calculations.

The two geoms, `geom_mosaic_text()` and `geom_mosaic_jitter()`, are provided to further enhance the generalized mosaic plots created with **ggmosaic**. These features are designed to add labels to the mosaic plot tiles and to view the tiles' density via jittered points, and are implemented as additional geom items that can layer on top of the original mosaic geom. Lastly, we also provide a new minimal theme, `theme_mosaic()`, designed for mosaic plots. The theme removes items from the plot background in order to reduce clutter and increase readability. Together, these features add

new functionality to mosaic plots to increase their usability and facilitate more profound insights into high-dimensional categorical data.

A labeling geom

The flexibility of generalized mosaic plots can lead to inadequate space around the perimeter of the plot to label each of the categories for the variables displayed, making labeling a challenge. To ease the burden on the axis labels, `geom_mosaic_text()` applies labels to the tiles. This section introduces `geom_mosaic_text()`, its parameters, and its customization options with sample code and graphics.

One aspect of mosaic plots is that while the text and tick marks on the axes may be aligned with the correct category levels on one side of the plot, the alignments may not be appropriate for category levels on the opposite side of the plot. Thus, it may be advantageous to label the tiles to ensure the group identities are apparent to a viewer. `geom_mosaic_text()` provides the means to place text, or labels, in each of the tiles. `geom_mosaic_text()` has its counterpart, `stat_mosaic_text()`, perform the necessary mapping calculations to place each tile's label in the center of the tile. Figure 14 features an example created with `geom_mosaic()` with `geom_mosaic_text()` added to place text in each of the tiles corresponding to the tiles' combination of categories. The associated code is below.

```
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline, rude_to_recline),
                  fill = do_you_recline)) +
  geom_mosaic_text(aes(x = product(do_you_recline, rude_to_recline)))
```

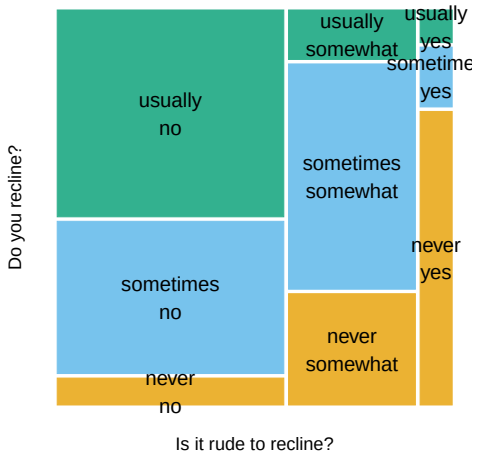


Figure 14: In this example, a geom of labels is layered on top of the mosaic geom to produce a mosaic plot with labels centered in each tile. Directly labeling each tile can alleviate issues caused by inadequate space around the perimeter for labels.

As a default, the label text contains the combinations of category levels the tile represents. Alternative labels may be desired and are achievable through the `label` aesthetic. For example, to label the tiles of a mosaic with counts, that variable can be mapped to the `label` aesthetic. The counts, however, need not be contained in the underlying data before plotting; the function `after_stat()` supports aesthetic mappings of variables calculated by the stat, in this case, the variable `.wt` calculated in `stat_mosaic_label()` (see Figure 15 and the code below).

```
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline, rude_to_recline),
                  fill = do_you_recline)) +
  geom_mosaic_text(aes(x = product(do_you_recline, rude_to_recline),
                       label = after_stat(.wt)))
```

To help label the areas of the plots where labels may be densely packed and overlapping, **ggmosaic** uses the **ggplot2** extension package, ggrepel (Slowikowski 2021). The geoms provided by ggrepel help ensure the text is readable by repelling the labels away from each other, data points, and edges of the plot panel. In addition to the standard `"Text"` and `"Label"` geoms, **ggmosaic**'s use of **ggrepel** allows

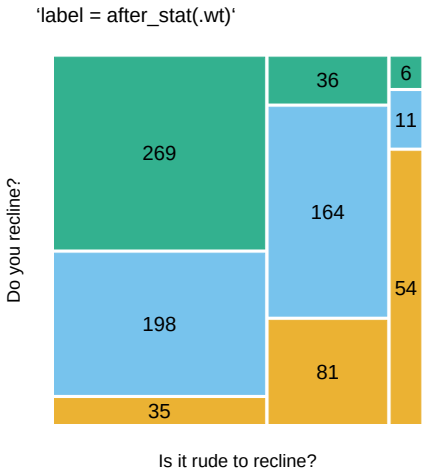


Figure 15: Variables calculated by the associated stat can be used for the label. In this example, `.wt` is used to place the frequencies represented by the tile in the center of that tile, providing a glance at the size of the data and removing potential guessing work aimed at comparing areas of unaligned tiles.

the use of the `"TextRepel"` and `"LabelRepel"` geoms within mosaic plots. Thus, `geom_mosaic_text()` provides access to the features of four geoms. To access these four geoms, the parameters `as.label` and `repel` are introduced; their use is demonstrated in Figure 16, and the code is below. Furthermore, the parameter `repel_params` is available to use with either of the **ggrepel** options, and the parameter `check_overlap` is available to use with the `"Text"` geom.

```
# default / text
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline, rude_to_recline),
                  fill = do_you_recline)) +
  geom_mosaic_text(aes(x = product(do_you_recline, rude_to_recline)))

# label
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline, rude_to_recline),
                  fill = do_you_recline)) +
  geom_mosaic_text(aes(x = product(do_you_recline, rude_to_recline)),
                   as.label = TRUE)

# repel text
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline, rude_to_recline),
                  fill = do_you_recline)) +
  geom_mosaic_text(aes(x = product(do_you_recline, rude_to_recline)),
                   repel = TRUE)

# repel label
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline, rude_to_recline),
                  fill = do_you_recline)) +
  geom_mosaic_text(aes(x = product(do_you_recline, rude_to_recline)),
                   repel = TRUE, as.label = TRUE)
```

Mosaic plots with many categories can quickly become unreadable, `geom_mosaic_text()` helps alleviate congestion-related confusion by providing quick and effective labeling in various formats. The parameters `repel` and `as.label` provide access to three additional geoms. The geom `geom_mosaic_text()` can add the text geom, the label geom, the **ggrepel** text geom, or the **ggrepel** label geom. The parameters `repel_params` and `check_overlap` provide access to the parameters native to the **ggrepel** geoms and label geoms, respectively. Labels add value to the mosaic plot as they ensure easy and correct identification of the tiles, which, in turn, eases the visual analysis.

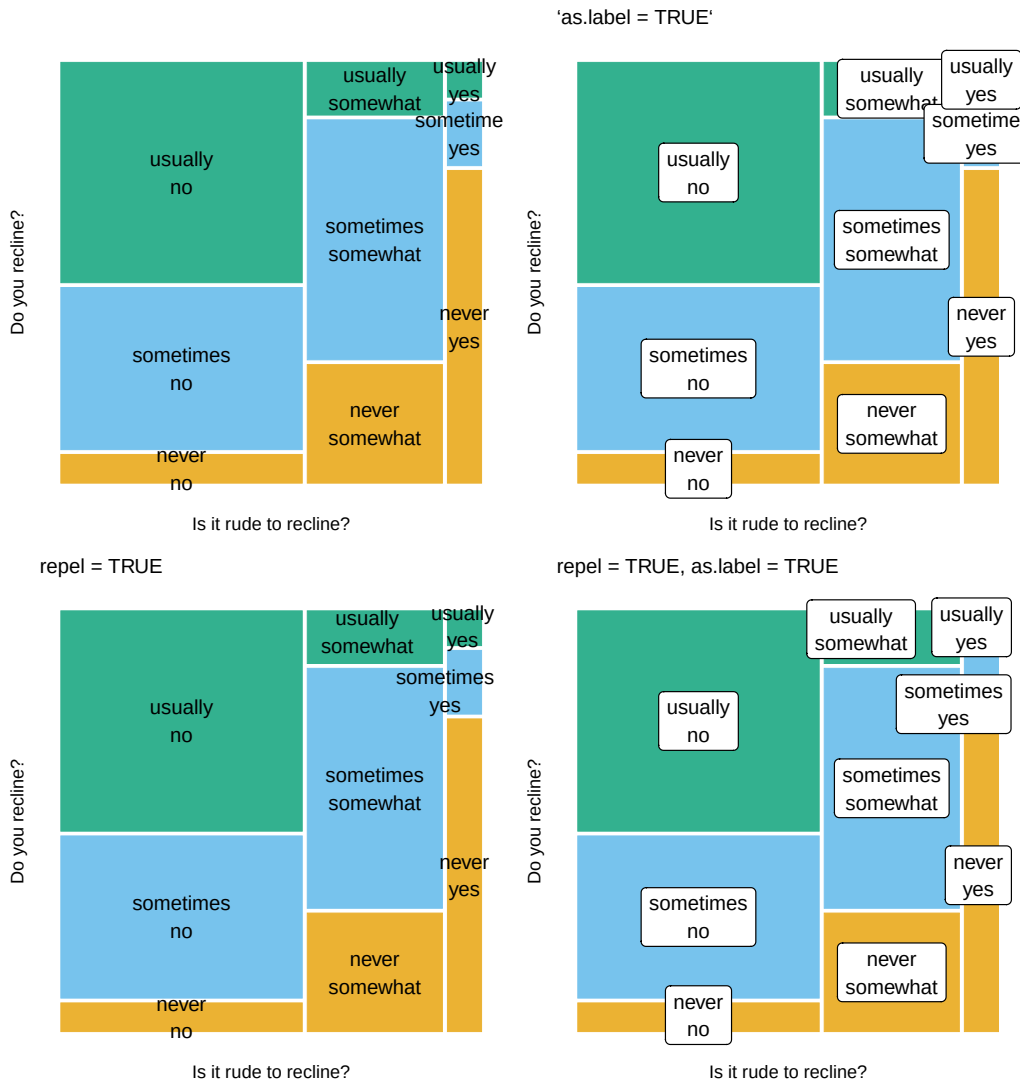


Figure 16: Examples of the four geoms accessible to `geom_mosaic_text()`. The geoms, from left to right and top to bottom, are `"Text"`, `"Label"`, `"TextRepel"`, and `"LabelRepel"`.

A jittering geom

In constructing of a mosaic plot, the tile area is proportional to the number of observations that the tile represents. In other words, the density of each tile in a mosaic plot is constant throughout the plot. While the number of observations, or density, is typically masked in a mosaic plot, a user can visualize these individual data points with `geom_mosaic_jitter()`. This added layer of visualization unlocks a range of applications, from adding additional aesthetic mappings, both included in the formula and not, to model diagnosis. Research suggests visualizations that incorporate individuals in charts allow viewers to digest and understand probabilities and risks involved more easily (Galesic, Garcia-Retamero, and Gigerenzer 2009; Ancker et al. 2006). These features further extend the ability of generalized mosaic plots to communicate interesting features of high-dimensional categorical data.

When used in conjunction with `geom_mosaic()`, `geom_mosaic_jitter()` adds a layer of jittered points superimposed on the mosaic plot. The number of points in each rectangle is equal to the number of observations that the rectangle represents. The result is an even dispersal of points throughout the one-by-one square mosaic plot.

When conditioning on a variable, `geom_mosaic_jitter()` provides a visual representation of the differences between the conditional probability and the joint probability; the spread of the points throughout the mosaic plot can help decipher a conditioning variable's effect on a mosaic plot's construction. In Figure 17, conditioning on the variable `rude_to_recline` causes a change in the density of the jittered points. The visual difference serves as an effective tool for teaching the concept. The plots are based on the following lines of code:

```
# not conditioned
ggplot(data = flights) +
  geom_mosaic(aes(x = product(rude_to_recline),
                  fill = do_you_recline),
              alpha = 0.3) +
  geom_mosaic_jitter(aes(x = product(rude_to_recline),
                         color = do_you_recline))

# conditioned
ggplot(data = flights) +
  geom_mosaic(aes(x = product(do_you_recline),
                  conds = product(rude_to_recline),
                  fill = do_you_recline),
              alpha = 0.3)  +
  geom_mosaic_jitter(aes(x = product(do_you_recline),
                         conds = product(rude_to_recline),
                         color = do_you_recline))
```

Including the jittered points generates an awareness of the size of the data, something customarily masked in a mosaic plot. For smaller data sets such as the `titanic` data set, using `geom_mosaic_jitter()` may encourage engagement as the points connect with the individuals in the data. This connection may provide a more compelling and profound visual impact.

`geom_mosaic_jitter()` is more effective when the `alpha` argument is used in both `geom_mosaic_jitter()` and `geom_mosaic()` to create semi-transparent jittered points and semi-transparent rectangles. An `alpha` value of `0.3` for `geom_mosaic()` and an `alpha` value of `0.7` for `geom_mosaic_jitter()` is aesthetically pleasing.

`geom_mosaic_jitter()` introduces an additional parameter, `drop_level`. The `drop_level` parameter controls which level defines the color of the generated points. In other words, if a `color` aesthetic is defined, should that variable be included in the formula? If the formula includes the `color` aesthetic, `drop_level = FALSE`, the colored points are at the top level. If the formula does not include the `color` aesthetic, `drop_level = TRUE`, color is added to the points one level down. Figure 18 provides an example of these two options.

```
# drop level
ggplot(data = flights) +
  geom_mosaic(aes(x = product(rude_to_recline)),
              alpha = 0.1) +
  geom_mosaic_jitter(aes(x = product(rude_to_recline),
                         color = do_you_recline),
                     drop_level = TRUE)
# do not drop level
```

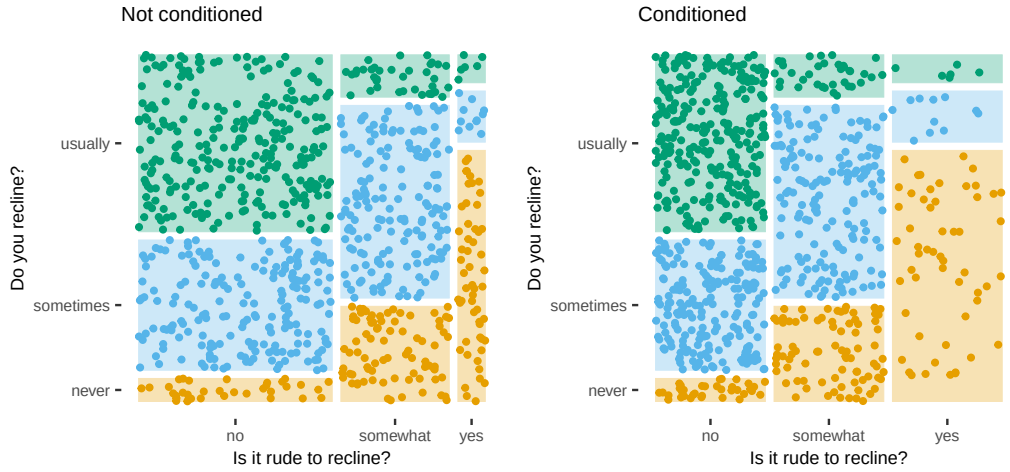


Figure 17: This example highlights the differences between the conditional (right) and joint (left) probability with the density of the jittered dots. When conditioning, the columns are equally sized, and the jittered points condense into the leftmost column, previously the widest column, rather than being evenly dispersed.

```
ggplot(data = flights) +
  geom_mosaic(aes(x = product(rude_to_recline)),
              alpha = 0.1) +
  geom_mosaic_jitter(aes(x = product(rude_to_recline),
                         color = do_you_recline),
                     drop_level = FALSE)
```

An additional aesthetic, `weight2`, is implemented in `geom_mosaic_jitter()`. The `weight2` aesthetic allows the number of points generated within each tile to be different from the number of points the cell represents.

While mosaic plots are typically drawn according to the observed values, they can be drawn according to the model's expected values.

```
flights_model %>% glimpse()
```

```
#> Rows: 9
#> Columns: 4
#> $ do_you_recline  <fct> never, never, never, sometimes, sometimes, sometimes, ~
#> $ rude_to_recline <fct> no, somewhat, yes, no, somewhat, yes, no, somewhat, yes
#> $ Observed        <int> 35, 81, 54, 198, 164, 11, 269, 36, 6
#> $ Expected        <dbl> 100, 56, 14, 219, 123, 31, 183, 102, 26
```

Figure 19 displays the observed values on the left and the expected values from the independence model on the right. The difference between the two plots represents the lack of fit. The plot is based on the following lines of code:

```
flights_model %>%
  gather("wt_type", "wt", Expected:Observed) %>%
  ggplot() +
  geom_mosaic(aes(weight = wt,
                  x = product(rude_to_recline),
                  fill = do_you_recline)) +
  facet_wrap(vars(wt_type))
```

In Figure 19, the mosaic plot drawn according to the observed values represents the data space, whereas the mosaic plot drawn according to the expected values represents the model space. Rather than requiring two plots, jittering connects in one plot both the data space with the model space highlighting the differences between the two.

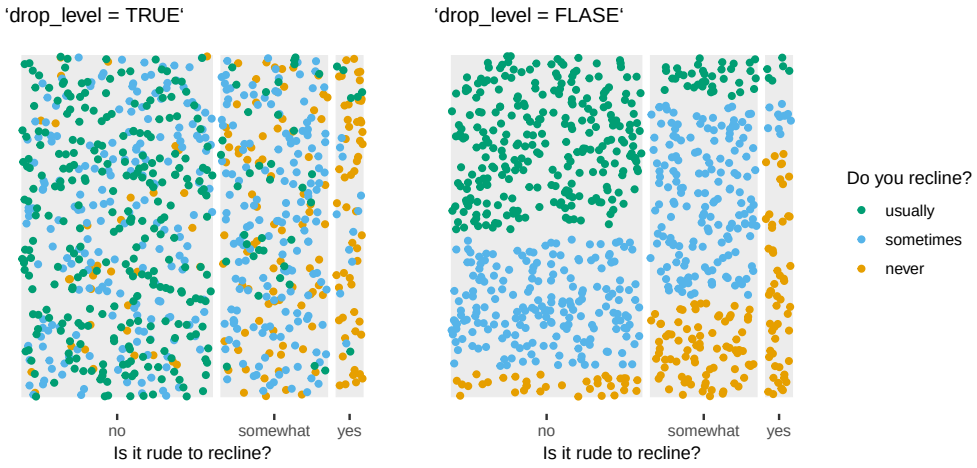


Figure 18: The parameter `drop_level` controls which level defines the colors of the generated points. On the left, color is added to the points one level down, and a mixing of the colored points occurs. In the plot on the right, the heights of the sorted points' spaces represent the conditional distribution of `do_you_recline` given `rude_to_recline`.

In Figure 20, the jittered points represent the observed values, while the tiles' size represents the expected values. We can evaluate the fit of the model according to how evenly the points spread throughout the plot. For example, the overcrowded points seen in the leftmost bottom tile communicate that the independence model underestimates the number of respondents that think it is rude to recline and never recline their seats.

```
ggplot(flights_model) +
  geom_mosaic(aes(weight = Expected,
                  x = product(do_you_recline, rude_to_recline)),
              alpha = .2) +
  geom_mosaic_jitter(aes(weight2 = Observed,
                         weight = Expected,
                         x = product(do_you_recline, rude_to_recline)))
```

The jitter geom, with its additional aesthetic mappings, provides a convenient visual aide that can help solidify what the mosaic plot represents and help communicate the differences between conditional and joint probabilities. The implementation of the jitter geom allows the user to extend the functionality of mosaic plots beyond what has previously been available.

A custom theme

Mosaic plots have two main characteristics that set mosaic plots apart from other graphics. First, at its foundation, a mosaic plot is a one-by-one square, and the area of each tile in the mosaic plot is proportional to the frequency of the combination of categories that the tile represents. Second, mosaic plots do not have a standard coordinate system. Rather, mosaic plots have a coordinate system based on recursive partitioning that can integrate several variables. These two characteristics clash with the default **ggplot2** theme. For this reason, version 0.3.3 of **ggmosaic** includes a custom theme for mosaic plots, `theme_mosaic()`, that can be added to the plot in the same manner as any other **ggplot2** theme. (Figure 21)

The plot grid lines suffer from the same issue as the axis labels; while the grid lines may be aligned with the correct category levels on one side of the plot, the alignments may not be appropriate for category levels on the opposite side of the plot. `theme_mosaic()` removes all grid lines but does not remove the axis labels and ticks.

Seeking a more faithful representation of a mosaic plot, `theme_mosaic()` enforces a fixed aspect ratio of 1. When faceting, the aspect ratio should be modified according to the number of panels and the direction of the faceting. For example, in Figure 22, the faceting represents the responses to `do_you_recline` conditioned on the responses to `rude_to_recline`. The conditioning variable,

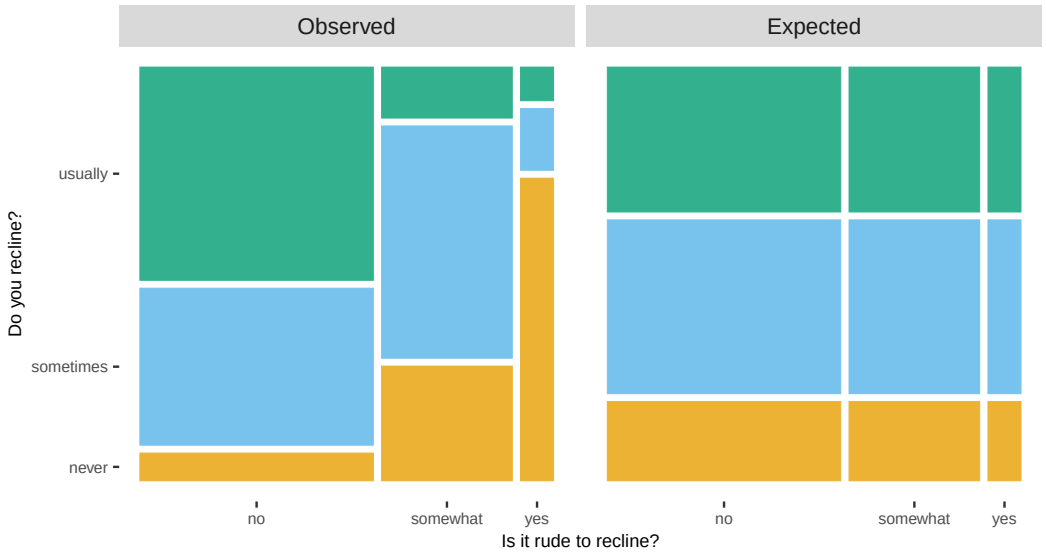


Figure 19: While mosaic plots are typically drawn according to the observed values, they can be drawn according to the model's expected values. The mosaic plot on the right displays the expected values from the independence model, recognizable by the lattice structure. The differences in the heights of the tiles in the observed data mosaic plot on the left reveal a strong association between the two variables, and the differences between the two plots highlight the lack of model fit.

`rude_to_recline`, contains three categories, and the faceting represents a horizontal spine. Hence, the aspect ratio is modified from `1` (the default) to `3`.

The custom theme, `theme_mosaic()`, seeks to help a viewer extract the correct information from the plot by providing a more suitable default aspect ratio and removing the axes lines that are not always appropriate across the entire plot. As with any theme, however, `theme_mosaic()` is merely a suggestion of a starting-off point, and it can be modified as seen fit for each individual plot.

5 Interactive exploratory mosaic plot building

Mosaic plots help identify interesting relationships in high-dimensional categorical data and are an influential tool for exploratory data analysis (EDA). Because of the complexities that arise from comparing many categories, it is often necessary to iterate through many of the potential mosaic plots and obtain many views on the data. The addition of interactivity to the generation of mosaic plots can ease this process and help mosaic plots become more valuable and insightful (Hofmann 2003).

To facilitate exploring data with mosaic plots, **ggmosaic** version 0.3.4 includes a Shiny application that can be launched with the function `ggmosaic_app()` (Figure 23). This app accommodates structural changes to the mosaic plot with the press of certain keystrokes or buttons provided in the side panel. The app enables quick iterations between visualizations, providing a mechanism for discoveries and achieving more profound insight.

The app is organized into two tabs (Figure 23), "MOSAIC PLOT" and "DATA", setting the stage for data set and variable selection. Creating a mosaic plot consists of several steps. Using the drop-down menu provided on the left-hand side, the user first selects from one of the three data sets exported with **ggmosaic**, the `titanic` data set, the `happy` data set, or the `fly` data set (the default selection). After the data set is selected, the user can create mosaic plots by selecting various variables to include and different dividers to be used, all completed with keystrokes or a set of buttons in the side panel.

The arrow keys (up, down, left, right) add, remove, or switch variables. The ordering of the variables can be quickly modified, allowing the user to find a sensible order for the variables in a streamlined manner. Additionally, the 'h', 'v', 's', and 'b' keys switch the type of divider. The 'h' and 'v' keys switch between horizontal and vertical spines or bars, switching between 'hspine' and 'vpsine' or 'hbar' and 'vbar'. Similarly, the 's' and 'b' keys can be used to select to split the categories into spines or bars, switching between 'hspine' and 'hbar' or 'vspine' and 'vbar'. If the user does not press the 'h', 'v', 's', or 'b' keys, the 'mosaic()' divider is the default, corresponding to the default divider in `geom_mosaic()`. The 'h', 'v', 's', or 'b' keystrokes and buttons will only affect the top-level variable. To modify the divider used on a lower-level variable, the user must backtrack via the down arrow key.

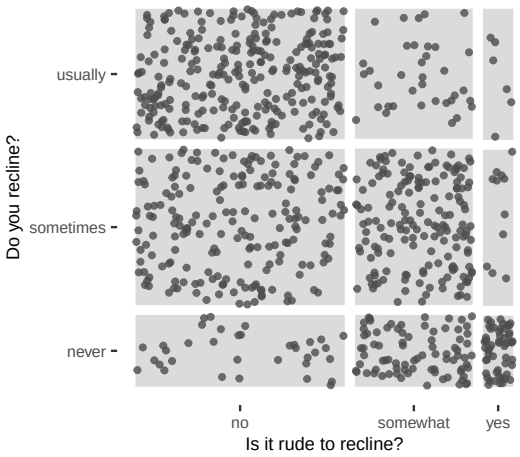


Figure 20: In this example, the tiles of the mosaic plot are drawn according to the expected values. A jitter geom representing the observed values is layered on top. The uneven dispersal of the points indicates a lack of fit.

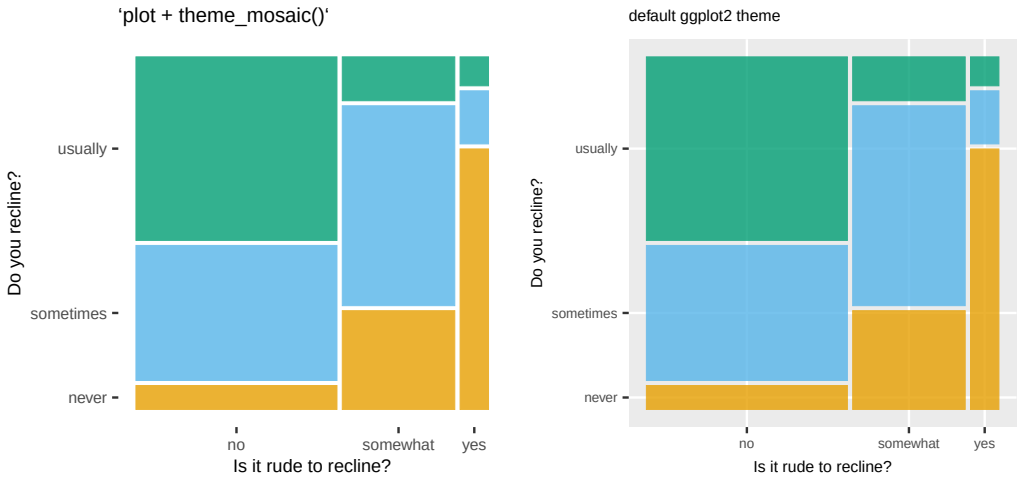


Figure 21: This example highlights the differences between the custom theme (left) and the default theme (right). The custom theme for mosaic plots seeks to minimize clutter by removing the plot background and the grid lines.

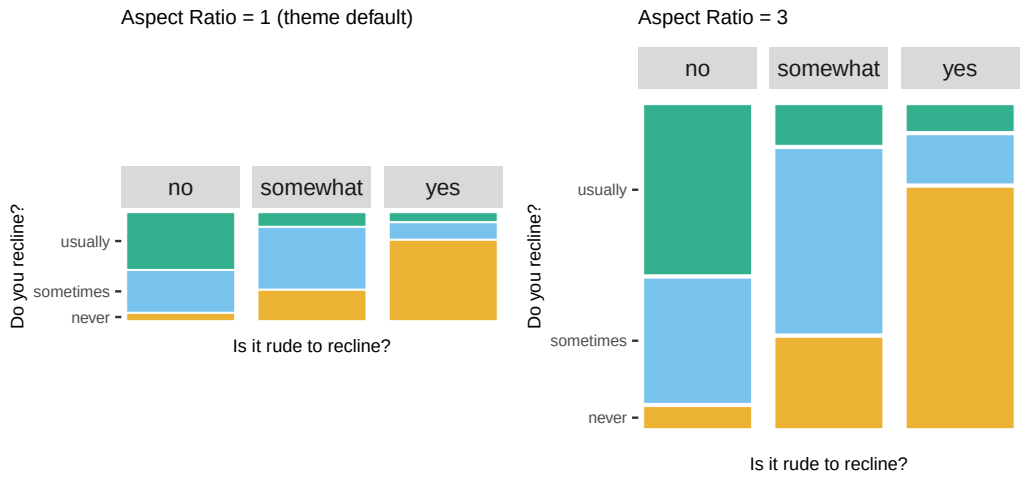


Figure 22: When using faceting, the aspect ratio might need to be adjusted according to the desired outcome. The default (left) represents each facet as a 1-by-1 mosaic plot, whereas an aspect ratio of 3 (right) represents each facet as one equal-sized column within a 1-by-1 mosaic plot.

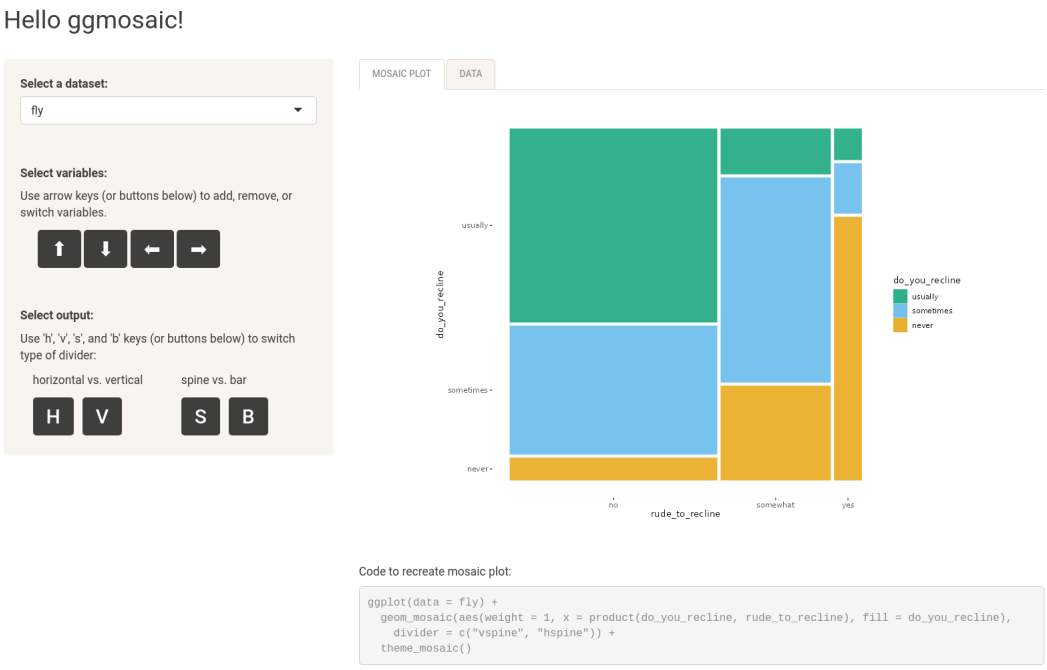


Figure 23: A snapshot of the Shiny application that can be launched with the function `ggmosaic_app()`. The Shiny app facilitates learning how to create mosaic plots. The user can explore one of three data sets provided with the package with mosaic plots created and modified via keystrokes or buttons, and no coding is required.

The Shiny app accommodates a better understanding of the myriad of possible forms a mosaic plot can take by accommodating a thorough search through the variables and structural changes to the mosaic plot with the simple press of certain keystrokes. The app provides an exploratory setting for visualizing many mosaic plots, and provide the user with the code necessary to recreate the selected mosaic plot.

6 Summary

By bringing mosaic plots into the **ggplot2** infrastructure, **ggmosaic** provides a highly customizable framework for generalized mosaic plots with a familiar syntax. The latest release of **ggmosaic** introduces novel uses of mosaic plots and exemplifies the opportunity the methods of visualizing multidimensional categorical data have for growth.

This manuscript is based on version 0.3.3 of the **ggmosaic** package. It can be installed from CRAN. The development version is available from the GitHub repository.

7 References

References

Ancker, Jessica S., Yalini Senathirajah, Rita Kukafka, and Justin B. Starren. 2006. "Design Features of Graphs in Health Risk Communication: A Systematic Review." *Journal of the American Medical Informatics Association* 13 (6): 608–18. https://doi.org/10.1197/jamia.M2115.

Chang, Winston, Joe Cheng, JJ Allaire, Carson Sievert, Barret Schloerke, Yihui Xie, Jeff Allen, Jonathan McPherson, Alan Dipert, and Barbara Borges. 2021. *Shiny: Web Application Framework for r*. https://CRAN.R-project.org/package=shiny.

Cleveland, W. S., and R. 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.

Emerson, John W. 1998. "Mosaic displays in S-PLUS: A general implementation and a case study." *Statistical Computing and Graphics Newsletter* 9 (1): 17–23.

Friendly, Michael. 2002. "A Brief History of the Mosaic Display." *Journal of Computational and Graphical Statistics* 11 (1): 89–107. https://doi.org/10.1198/106186002317375631.

———. 2016. "Working with categorical data with R and the vcd and vcdExtra packages." In. https://cran.r-project.org/web/packages/vcdExtra/vignettes/vcd-tutorial.pdf.

Galesic, Mirta, Rocio Garcia-Retamero, and Gerd Gigerenzer. 2009. "Using Icon Arrays to Communicate Medical Risks: Overcoming Low Numeracy." *Health Psychology* 28 (2): 210–16. https://doi.org/10.1037/a0014474.

Hartigan, John A., and Beat Kleiner. 1981. "Mosaics for Contingency Tables." In *Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface*, 268–73. Fairfax Station, VA: Interface Foundation of North America, Inc. https://doi.org/10.1007/978-1-4613-9464-8_37.

———. 1984. "A Mosaic of Television Ratings." *The American Statistician* 38: 32–35. https://doi.org/10.1080/00031305.1984.10482869.

Hickey, Walt. 2014. "41 Percent of Fliers Think You're Rude If You Recline Your Seat." *FiveThirtyEight*. http://fivethirtyeight.com/datalab/airplane-etiquette-recline-seat/.

Hofmann, Heike. 2003. "Constructing and Reading Mosaicplots." *Computational Statistics and Data Analysis* 43 (4): 565–80. https://doi.org/10.1016/S0167-9473(02)00293-1.

Hummel, Jürgen. 1996. "Linked Bar Charts: Analysing Categorical Data Graphically." *Computational Statistics* 11 (1): 23–33.

Kleiner, B., and J. A. Hartigan. 1981. "Representing Points in Many Dimensions by Trees and Castles." *Journal of the American Statistical Association* 76 (374): 260–69. https://doi.org/10.1080/01621459.1981.10477638.

Kuhn, Max, Davis Vaughan, and Emil Hvitfeldt. 2022. *Yardstick: Tidy Characterizations of Model Performance*. https://CRAN.R-project.org/package=yardstick.

Meyer, David, Achim Zeileis, and Kurt Hornik. 2020. *Vcd: Visualizing Categorical Data*. https://CRAN.R-project.org/package=vcd.

Oldford, Wayne, Erle Holgersen, Ben Lafreniere, and Tianlu Zhu. 2018. *Eikosograms: The Picture of Probability*. https://CRAN.R-project.org/package=eikosograms.

Playfair, William, Howard Wainer, and Ian Spence. 2005. *Playfair's Commercial and Political Atlas and Statistical Breviary*. Cambridge University Press.

Sarkar, Deepayan. 2020. *Lattice: Trellis Graphics for r*. https://CRAN.R-project.org/package=lattice.

Sarkar, Deepayan, and Felix Andrews. 2016. *latticeExtra: Extra Graphical Utilities Based on Lattice*. https://CRAN.R-project.org/package=latticeExtra.

Sievert, Carson. 2020. *Interactive Web-Based Data Visualization with R, Plotly, and Shiny*. Chapman; Hall/CRC. https://plotly-r.com.

Sievert, Carson, Chris Parmer, Toby Hocking, Scott Chamberlain, Karthik Ram, Marianne Corvellec, and Pedro Despouy. 2016. *Plotly: Create Interactive Web Graphics via 'Plotly.js'*. https://CRAN.R-project.org/package=plotly.

Slowikowski, Kamil. 2021. *Ggrepel: Automatically Position Non-Overlapping Text Labels with 'Ggplot2'*. https://CRAN.R-project.org/package=ggrepel.

Theus, Martin, and Simon Urbanek. 2009. *Interactive Graphics for Data Analysis: Principles and Examples*. Chapman & Hall/CRC Computer Science & Data Analysis. CRC Press.

United States Census office. 9th census, 1870, and Francis Amasa Walker. 1874. "Statistical atlas of the United States based on the results of the ninth census, 1870 with contributions from many eminent men of science and several departments of the government." digitized version provided through Library of Congress, https://www.loc.gov/item/05019329/.

Wickham, Hadley. 2010. "A Layered Grammar of Graphics." *Journal of Computational and Graphical Statistics* 19: 3–28. https://doi.org/10.1198/jcgs.2009.07098.

Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus

Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2020. *Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. https://CRAN.R-project.org/package=ggplot2.

Wickham, Hadley, and Heike Hofmann. 2011. "Product Plots." *IEEE Transactions on Visualization and Computer Graphics* 17 (12): 2223–30. https://doi.org/10.1109/tvcg.2011.227.

———. 2016. *Productplots: Product Plots for r*. https://CRAN.R-project.org/package=productplots.

Wilkinson, Leland. 1999. *The Grammar of Graphics*. Springer New York. https://doi.org/10.1007/978-1-4757-3100-2.

Haley Jeppson
Department of Statistics
Iowa State University
2438 Osborn Dr
Ames, IA 50011
ORCiD: 0000-0003-2524-4063
hjeppson@iastate.edu

Heike Hofmann
Department of Statistics
Center for Statistics and Applications in Forensic Evidence
Iowa State University
2438 Osborn Dr
Ames, IA 50011
ORCiD: 0000-0001-6216-5183
hofmann@iastate.edu

A Study in Reproducibility: The Congruent Matching Cells Algorithm and cmcR Package

by Joseph Zemmels, Susan VanderPlas, and Heike Hofmann

Abstract Scientific research is driven by our ability to use methods, procedures, and materials from previous studies and further research by adding to it. As the need for computationally-intensive methods to analyze large amounts of data grows, the criteria needed to achieve reproducibility, specifically computational reproducibility, have become more sophisticated. In general, prosaic descriptions of algorithms are not detailed or precise enough to ensure complete reproducibility of a method. Results may be sensitive to conditions not commonly specified in written-word descriptions such as implicit parameter settings or the programming language used. To achieve true computational reproducibility, it is necessary to provide all intermediate data and code used to produce published results. In this paper, we consider a class of algorithms developed to perform firearm evidence identification on cartridge case evidence known as the *Congruent Matching Cells* (CMC) methods. To date, these algorithms have been published as textual descriptions only. We introduce the first open-source implementation of the Congruent Matching Cells methods in the R package **cmcR**. We have structured the **cmcR** package as a set of sequential, modularized functions intended to ease the process of parameter experimentation. We use **cmcR** and a novel variance ratio statistic to explore the CMC methodology and demonstrate how to fill in the gaps when provided with computationally ambiguous descriptions of algorithms.

1 Introduction

Forensic examinations are intended to provide an objective assessment of the probative value of a piece of evidence. Typically, this assessment of probative value is performed by a forensic examiner who visually inspects the evidence to determine whether it matches evidence found on a suspect. The process by which an examiner arrives at their evidentiary conclusion is largely opaque and has been criticized (President's Council of Advisors on Sci. & Tech. 2016) because its subjectivity does not allow for an estimation of error rates. In response, National Research Council (2009) pushed to augment subjective decisions made by forensic examiners with automatic algorithms that objectively assess evidence and can be explained during court testimony. In addition to the objectivity of these algorithms, there is an additional benefit: we expect that an algorithm with the same random seed run on the same data multiple times will produce the same answer; that is, that the results are repeatable. This is extremely beneficial because it allows the prosecution and defense to come to the same conclusion given objective evidence or data.

Repeatability and reproducibility

Repeatability in forensic labs is enforced primarily using standard operating procedures (SOPs), which specify the steps taken for any given evaluation, along with the concentrations of any chemicals used, the range of acceptable machine settings, and any calibration procedures required to be completed before the evidence is evaluated. When labs use computational procedures, this SOP is augmented with specific algorithms, which are themselves SOPs intended for use by man and machine. Algorithms are generally described on two levels: we need both the conceptual description (intended for the human using the algorithm) and the procedural definition (which provides the computer hardware with a precise set of instructions). For scientific and forensic repeatability and reproducibility, it is essential to have both pieces: the algorithm description is critical for establishing human understanding and justifying the method's use in court, but no less important is the computer code which provides the higher degree of precision necessary to ensure the results obtained are similar no matter who evaluates the evidence. As with SOPs in lab settings, the code parameters function like specific chemical concentrations; without those details, the SOP would be incomplete and the results produced would be too variable to be accepted in court.

The National Academy of Sciences, Engineering, and Medicine (2019) defines *reproducibility* as "obtaining consistent computational results using the same input data, computational steps, methods, code, and conditions of analysis." This form of reproducibility requires that the input data, code, method, and computational environment are all described and made available to the community. In many situations, this level of reproducibility is not provided – not just in forensics but in many other

applied disciplines. In forensics in particular, it is easier to list the exceptions: reproducible algorithms have been proposed in sub-disciplines including DNA (Tvedebrink, Andersen, and Curran 2020; Goor, Hoffman, and Riley 2020; Tyner et al. 2019), glass (Curran, Champod, and Buckleton 2000; Park and Tyner 2019), handwriting (Crawford 2020), shoe prints (Park and Carriquiry 2020), and ballistic evidence (Hare, Hofmann, and Carriquiry 2017; Tai and Eddy 2018).

We find it useful to instead consider a more inclusive hierarchy of reproducibility. Algorithms at higher tiers of the hierarchy are more easily reproducible in the sense that fewer resources are required to (re)-implement the algorithm.

Definition 1 *Hierarchy of Reproducibility*

Conceptual description *The algorithm is described and demonstrated in a scientific publication.*

Pseudocode *The algorithm is described at a high level of detail with pseudocode implementation provided, and results are demonstrated in a scientific publication.*

Reproducible data *The algorithm is described and demonstrated in a scientific publication, and input data are available in supplementary material.*

Comparable results *The algorithm is described and demonstrated in a scientific publication, and input data and numerical results are provided in supplementary material.*

Full reproducibility *The algorithm is described and demonstrated in a scientific publication, and the input data, source code, parameter settings, and numerical results are provided in supplementary material.*

To aid in comprehension of an algorithm, it is useful to supplement conceptual descriptions with pseudocode. However, a conceptual description and pseudocode alone do not contain sufficient detail (e.g., parameter settings) to ensure computational reproducibility. Implementing algorithms based on conceptual descriptions or pseudocode requires enumerating and testing possible parameter choices which, depending on their complexity, can be a lengthy and expensive process. In contrast, implementing fully reproducible algorithms requires only as much time as it takes to emulate the original development environment. Commonly identified reasons for unreproducible results include (1) ambiguity in how procedures were implemented, (2) missing or incomplete data, and (3) missing or incomplete computer code to replicate all statistical analyses (Leek and Jager 2017). In particular, for statistical algorithms which depend on input data, we find that full reproducibility depends on the provision of both original data and any manual pre-processing applied to said data, as this manual process is not reproducible by itself. In combination with the code, the algorithm description, and the numerical results presented in the paper, it should be possible to fully reproduce the results of a paper.

In this paper, we demonstrate the importance of higher levels of reproducibility by examining the Congruent Matching Cells (CMC) algorithm for cartridge case comparisons and developing an open-source, fully reproducible version for general use in the forensics community.

The Congruent Matching Cells algorithm

A *cartridge case* is the portion of firearm ammunition that encases a projectile (e.g., bullet, shots, or slug) along with the explosive used to propel the projectile through the firearm. When a firearm is discharged, the projectile is propelled down the barrel of the firearm, while the cartridge case is forced towards the back of the barrel. It strikes the back wall, known as the *breech face*, of the barrel with considerable force, thereby imprinting any markings on the breech face onto the cartridge case and creating the so-called *breech face impressions*. These markings are used in forensic examinations to determine whether two cartridge cases have been fired by the same firearm. During a forensic examination, two pieces of ballistic evidence are placed under a *comparison microscope*. Comparison microscopes allow for a side-by-side comparison of two objects within the same viewfinder, as seen in Figure 1. A pair of breech face images is aligned along the thin black line in the middle of the images. The degree to which these breech face markings can be aligned is used to determine whether the two cartridge cases came from the same source; i.e., were fired from the same firearm. These breech face impressions are considered analogous to a firearm's "fingerprint" left on a cartridge case (Thompson 2017).

The Congruent Matching Cells (CMC) pipeline is a collection of algorithms to process and compare cartridge case evidence (Song 2013). Since its introduction, the pipeline and its extensions (Tong, Song, and Chu 2015; Chen et al. 2017; Song et al. 2018) have shown promise in being able to differentiate between matching and non-matching cartridge cases. However, so far the CMC pipelines have only been introduced in the form of conceptual descriptions. Further, the cartridge case scans used to validate the pipelines are only available in their raw, unprocessed forms on the NIST Ballistics

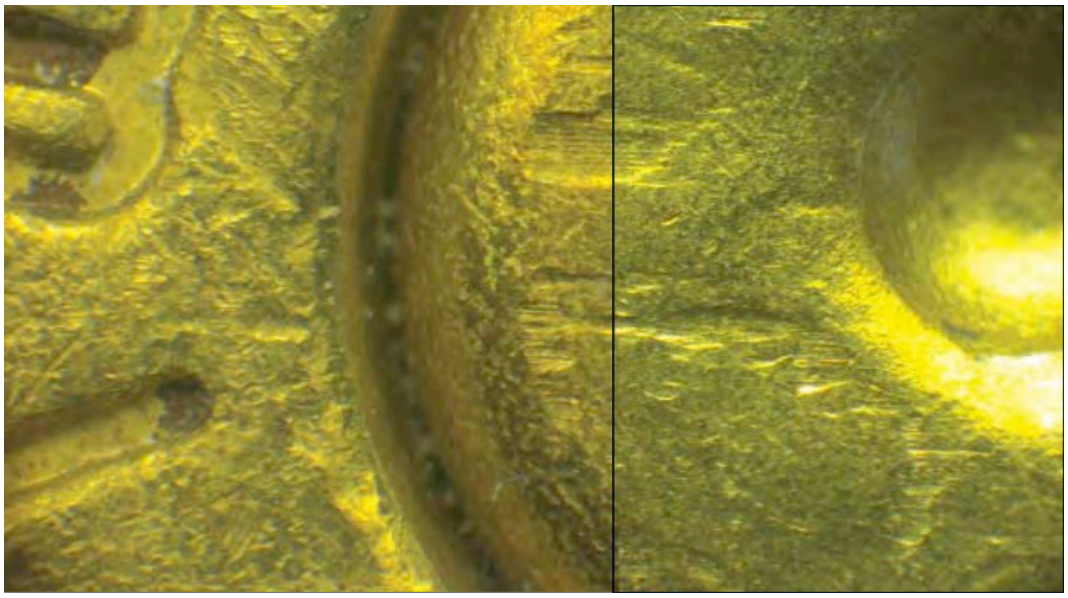


Figure 1: A cartridge case pair with visible breech face impressions under a microscope. A thin line can be seen separating the two views. The degree to which the markings coincide is used to conclude whether the pair comes from the same source.

Toolmark Research Database (Zheng, Soons, and Thompson 2016). While it is clear that the creators of the CMC pipeline have a working implementation, the wider forensic science community only has access to conceptual descriptions of the pipeline and summary statistics describing its performance. In our hierarchy of reproducibility, this puts the CMC algorithm somewhere between the conceptual description and reproducible data stage: the steps are described but no code is available, and the raw data are available but manual pre-processing steps make this raw data insufficient to replicate the pipeline even with newly written code.

The development of the CMC algorithm seems to be representative of how many forensic algorithms are developed: after an algorithm is introduced, researchers build upon the foundation laid by the original algorithm in subsequent papers. These changes are often incremental in nature and reflect a growing understanding of the algorithm's behavior. While this cycle of scientific progress certainly is not unique to forensic algorithms, given the gravity of the application it is imperative that these incremental improvements not be unnecessarily delayed. As such, we believe that the forensic community at-large would benefit greatly by establishing an open-source foundation for their algorithms upon which additional improvements can be developed. Using open-source algorithms are cheaper to use than writing one's own code, enables the process of peer review by providing an accessible benchmark, and helps other research groups or companies stay on the leading edge of technology development (The Linux Foundation 2017).

Here, we describe the process of implementing the CMC pipeline for the comparison of marks on spent cartridge cases, using the descriptions from two published papers, Song et al. (2014) and Tong, Song, and Chu (2015). Our R package, **cmcR**, provides an open-source implementation of the CMC pipeline. We use **cmcR** to illustrate how ambiguities in the textual description of an algorithm can lead to highly divergent results. In particular, our implementation highlights an extreme sensitivity to processing and parameter decisions that has not been discussed previously. Additionally, we argue that our implementation can be used as a template for future implementations of forensic pattern-matching algorithms to not only ensure transparency and auditability, but also to facilitate incremental improvements in forensic algorithms.

In the remainder of this paper, we describe a general, reproducible, and open-source CMC pipeline which encompasses those discussed in Song (2013), Song et al. (2014), and Tong, Song, and Chu (2015). Song (2013) lays out the conceptual framework for the original CMC pipeline later implemented in Song et al. (2014) and Tong et al. (2014). An improvement of the pipeline presented in Tong, Song, and Chu (2015) and used in subsequent papers is referred to as the "High CMC" method (Chen et al. 2017). However, it should be noted that what the authors refer to as the original and High CMC decision rules are variations of one step of a larger CMC pipeline.

The **cmcR** package contains implementations designed for use with 3D topographical scans of the original decision rule described in Song (2013) and Song et al. (2014) and the High CMC decision rule described in Tong, Song, and Chu (2015). The source code to the full **cmcR** package is accessible at

https://github.com/CSAFE-ISU/cmcR.

2 The CMC pipeline

In this section, we examine the process of implementing the CMC pipeline for automatic comparisons of 3D cartridge case scans. At each step, we will discuss how we filled in the gaps of the original description during the creation of **cmcR**.

All of the CMC pipelines can be broken down into three broad stages: (1) pre-processing, (2) cell-based similarity feature extraction, and (3) application of a decision rule as illustrated in Figure 2. In the following sections we break each of these stages further into a set of modular steps. One advantage of modularizing these algorithms is that we can implement an algorithm as a set of sequential procedures. This allows us to test new variations against the old implementation in a coherent, unified framework.

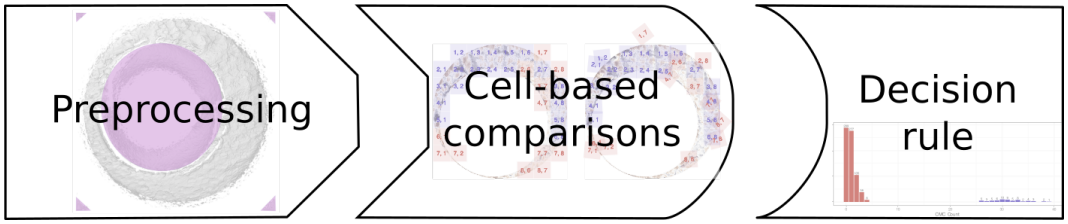


Figure 2: The stages of CMC pipelines. In the pre-processing stage, each scan is prepared for analysis, removing extraneous information and noise. Then, each scan is broken up into cells, which are numerically compared to cells in the other scan to determine an optimal alignment. Finally, each of the scores arising from the cells in the second stage are compared to a reference distribution to determine whether the scans originate from the same source or from different sources.

The primary difference between the two pipelines presented here, using the original and High CMC decision rules, lies in how the decision rules are utilized to separate matching vs. non-matching cartridge case pairs. In addition, there are also several small differences in the parameters used in the pre-processing and comparison procedures.

Initial data

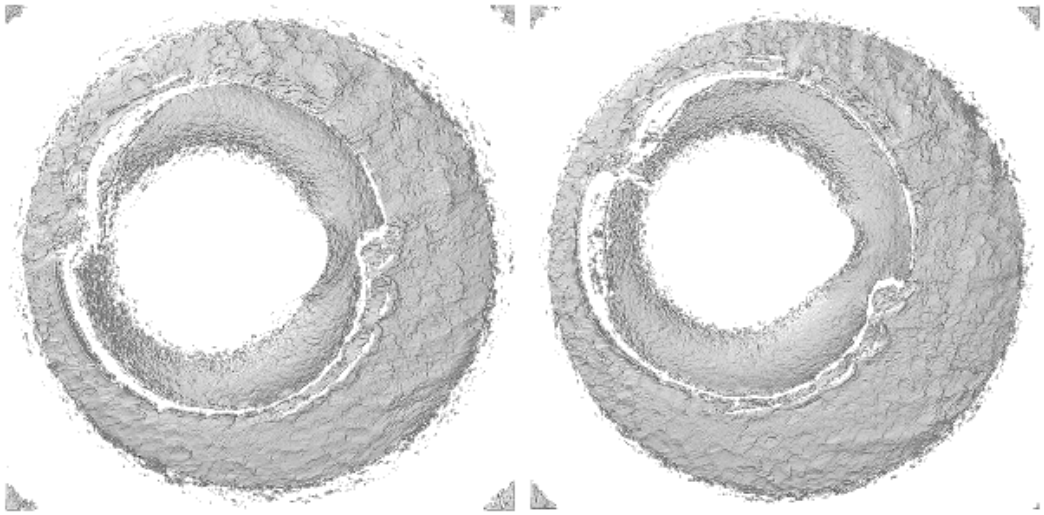
Digital microscopy is capable of precision measurements of surface topology at high resolutions. Using a 3D microscope, we can obtain scans of breech face impressions at the micron level ($1\mu m = 10^{-3}mm = 10^{-6}m$). These 3D topological scans are used as input to automated comparison algorithms, such as the CMC pipeline originally proposed in Song (2013). We will use the same data set referenced in Song et al. (2014) and Tong, Song, and Chu (2015) to illustrate usage of the **cmcR** package. These 3D scans of cartridge cases are available from the NIST Ballistics Toolmark Research Database (Zheng, Soons, and Thompson 2016). The strings defined below refer to three cartridge case scans available on the NBTRD from Fadul et al. (2011) and will be used throughout the remainder of this paper.

```
library(cmcR)

nbtrd_url <- "https://tsapps.nist.gov/NRBTD/Studies/CartridgeMeasurement"

x3p_ids <- c("DownloadMeasurement/2d9cc51f-6f66-40a0-973a-a9292dbee36d",
             "DownloadMeasurement/cb296c98-39f5-46eb-abff-320a2f5568e8",
             "DownloadMeasurement/8ae0b86d-210a-41fd-ad75-8212f9522f96")

file_names <- c("fadul1-1.x3p","fadul1-2.x3p","fadul2-1.x3p")

purrr::walk2(.x = x3p_ids,
             .y = file_names,
             .f = function(x3p_id,file_name){
               download.file(url = file.path(nbtrd_url, x3p_id),
                             destfile = paste0("data/",file_name),mode = "wb")
             })
```

Cartridge case scans are commonly stored in the ISO standard x3p file format ("Geometrical product specifications (GPS) — Surface texture: Areal — Part 72: XML file format x3p" 2017). x3p

is a container format which consists of a single surface matrix representing the height value of the breech face surface and metadata concerning the parameters under which the scan was taken (size, resolution, creator, microscope, microscopy software versions, etc.). The **x3ptools** package provides functionality to work with the format in R (Hofmann et al. 2020).

Figure 3 shows the surface matrices of a known match (KM) pair of cartridge cases from a study by Fadul et al. (2011). In this study, a total of 40 cartridge cases were scanned with a lateral resolution of 6.25 microns (micrometers) per pixel. The surface matrices are approximately 1200×1200 pixels in size corresponding to an area of about 3.8×3.8 mm^2.



(a) (b)

Figure 3: Unprocessed surface matrices of the known-match Fadul 1-1 and Fadul 1-2 Fadul et al. 2011. The observations in the corners of these surface matrices are artifacts of the staging area in which these scans were taken. The holes on the interior of the primer surfaces are caused by the firing pin striking the primer during the firing process. The region of the primer around this hole does not come into uniform contact with the breech face of the firearm.

Only certain regions of a cartridge case contain identifying breech face impression markings. Song (2013) defines "valid correlation regions" as regions where "the individual characteristics of the ballistics signature are found that can be used effectively for ballistics identification." Prior to applying the CMC comparison procedure, cartridge scans must undergo some pre-processing to isolate the valid correlation regions.

Pre-processing procedures

During the pre-processing stage, we apply sequential steps to prepare each cartridge case for analysis. The goal of this process is to remove the edges and center of the scan which did not come into contact with the breech face, as well as any artifacts of the scan and microscope staging which do not accurately represent the breech face surface. The various iterations of the CMC algorithm describe different variations of these steps. A summary of these steps is shown in Figure 4.

Translating the pre-processing steps in Figure 4 into an implementation requires the implementer to decide between potentially many implicit parameter choices. For example, Table 1 compares the pre-processing procedures as described in Song et al. (2014) to considerations that need to be made when implementing the procedures. Depending on one's interpretation of the description, there are many possible implementations that satisfy the described procedure - in contrast, there was only one implementation that led to the original results. While not explicitly mentioned in Song et al. (2014), Song et al. (2018) indicates that the "trimming" of the unwanted regions of the scan is performed manually. It is difficult to replicate manual steps as part of a reproducible pipeline; the best solution is for the authors to provide intermediate data after the manual steps have been completed.

The pre-processing procedures are implemented via modularized functions of the form `preProcess_*`. Modularizing the steps of the pre-processing procedures makes the overall process easier to understand and allows for experimentation. Figure 5 shows an overview of the pre-processing framework for the Fadul 1-1 breech face from reading the scan (left) to an analysis-ready region (right). For each scan in Figure 5, eleven height value percentiles: the Minimum (0th), 1st, 2.5th, 10th, 25th, Median

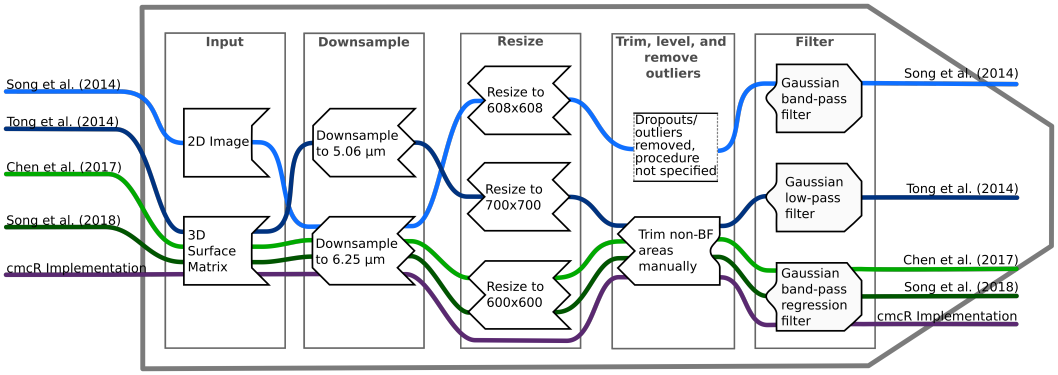


Figure 4: Overview of the set of pre-processing steps used in the CMC algorithms. Where a procedure step is not discussed or explicitly not applied in the paper, the path traverses empty space.

Table 1: Description of pre-processing procedures from Song et al. 2014 vs. considerations that need to be made when implementing these procedures. Each of these considerations requires the implementer to decide between potentially many choices.

Description from Song et al. (2014)	**Implementation Considerations**
"Trim off the inside firing pin surface and other areas outside the breech face mark, so that only breech face impression data remain for correlation."	Removal of firing pin hole, primer exterior, global trend, and primer roll-off
"Identify and remove dropouts or outliers."	Definition of outliers, what "removal" of dropouts or outliers means
"Apply a band-pass Gaussian regression filter with 40 μm short cut-off length and 400 μm long cut-off length to remove low frequency components, including surface curvature, form error, waviness and high frequency components which mainly arise from the instrument noise."	Wavelength cut-off parameters, specific implementation of the filter

(50th), 75th, 90th, 97.5th, 99th, and Maximum (100th) are mapped to a purple-to-orange color gradient. This mapping is chosen to highlight the extreme values in each scan.

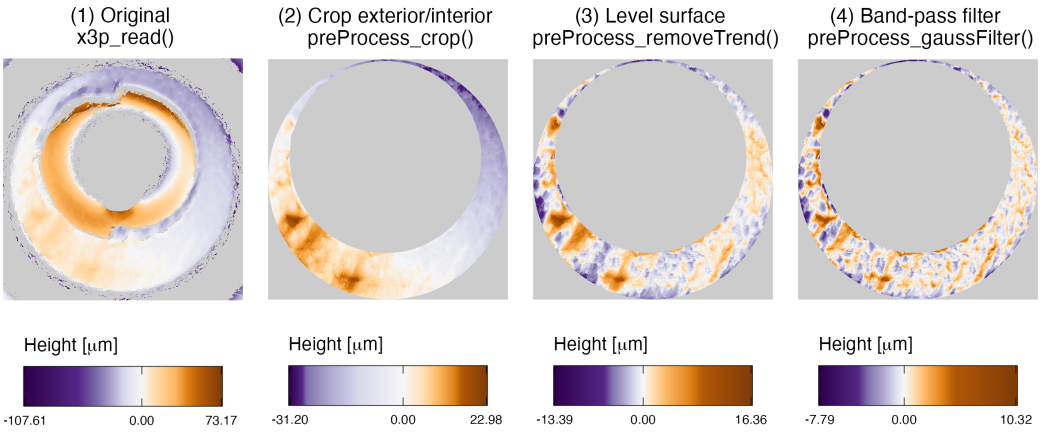


Figure 5: Illustration of the sequential application of pre-processing steps implemented in **cmcR**. We map the cartridge case surface height values to a divergent purple-white-orange color scale to emphasize deviations from the median height value (represented here as 0 micrometers). At each stage, the variability in height across the scan decreases as we emphasize the regions containing breech face impressions.

We demonstrate usage of the `preProcess_*` functions on the Fadul 1-1 scan. Each code chunk is followed up with an explanation of the functions used.

```
# Step (1)
fadul1.1 <- x3ptools::x3p_read("data/fadul1-1.x3p")
```

We begin with a 3D scan. Typically, we downsample scans to about 25% of their size by only retaining every other row and column in the surface matrix. The breech faces in Fadul et al. (2011) were initially scanned at a resolution of 3.125 μm per pixel. Downsampling reduces the resolution to 6.25 μm per pixel. Step (1) in Figure 5 shows an unprocessed breech face scan.

```
# Step (2)
fadul1.1_cropped <- fadul1.1 %>%
  cmcR::preProcess_crop(region = "exterior") %>%
  cmcR::preProcess_crop(region = "interior")
```

We then use a labeling algorithm to identify three major regions of the scan: the exterior of the cartridge case primer, the breech face impression region of interest, and the firing pin impression region in the center of the scan (Hesselink, Meijster, and Bron 2001; Barthelme 2019). We remove observations outside of the breech face impression region (i.e., replaced with `NA`). The resulting breech face scan, like the one shown in step (2) of Figure 5, is reproducible assuming the same parameters are used. The `preProcess_crop` function removes the exterior and firing pin impression region on the interior based on the `region` argument.

```
# Step (3)
fadul1.1_deTrended <- fadul1.1_cropped %>%
  preProcess_removeTrend(statistic = "quantile", tau = .5, method = "fn")
```

In step (3), we remove the southwest-to-northeast trend observable in steps (1) and (2) of Figure 5 by subtracting the estimated conditional median height value. The result of the `preProcess_removeTrend` function the median-leveled breech face scan in step (3) of Figure 5.

```
# Step (4)
fadul1.1_processed <- fadul1.1_deTrended %>%
  preProcess_gaussFilter(filtertype = "bp", wavelength = c(16,500)) %>%
  x3ptools::x3p_sample(m = 2)
```

Finally, we apply a band-pass Gaussian filter to the surface values to attenuate noise and unwanted large-scale structure. Step (4) of Figure 5 shows the effect of the `preProcess_gaussFilter` function. There is currently no determination or removal of outliers in the **cmcR** package's pre-processing procedures. Instead, we rely on the low-pass portion of the Gaussian filter to reduce the effects of any high-frequency noise.

Figure 6 displays the processed Fadul 1-1 and Fadul 1-2 scans; the second matrix is processed using the same parameters. Next, similarity features are extracted from a processed cartridge case pair in the cell-based comparison procedure.

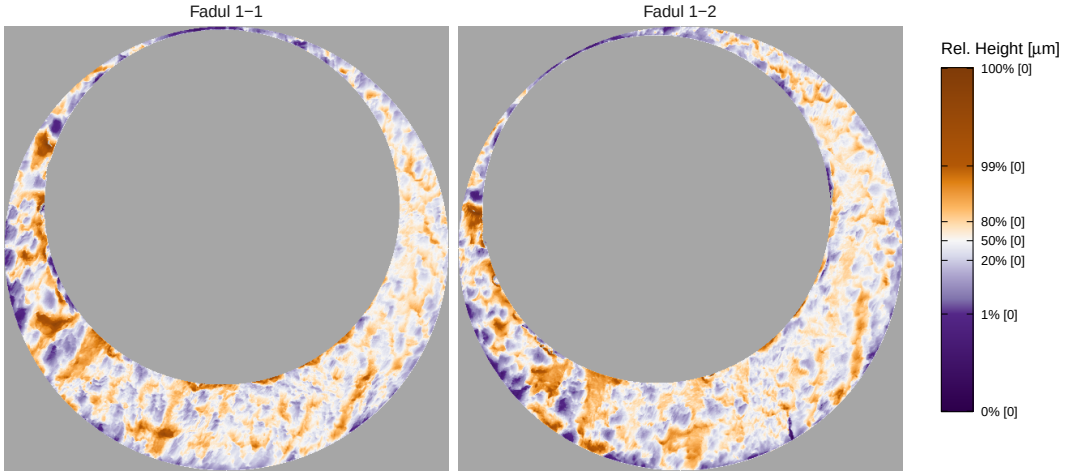


Figure 6: Fadul 1-1 and Fadul 1-2 after pre-processing. Similar striated markings are now easier to visually identify on both surfaces. It is now clearer that one of the scans needs to be rotated to align better with the other.

"Correlation cell" comparison procedure

As described in Song (2013), breech face markings are not uniformly impressed upon a cartridge case during the firing process. As such, only certain sections of the cartridge case are used in a comparison. In the CMC pipeline as proposed by Song (2013) two scans are compared by partitioning one breech face scan into a grid of so-called "correlation cells". These cells are compared individually to their best-matching counterpart on the other scan. If a large proportion of these correlation cells are highly similar to their counterparts on the other breech face scan, this is considered as evidence that the markings on the two cartridge cases were made by the same source. The number of highly similar cells is defined as the *CMC count C* (Song 2013) of the breech-face comparison. The CMC count is considered to be a more robust measure of similarity than the correlation calculated between two full scans.

Figure 7 illustrates the cell-based comparison procedure between two cartridge case scans. The scan on the left serves as the reference; it is divided into a grid of 8×8 cells.

Figure 8 shows the steps of the correlation cell comparison process in each of the papers as well as the **cmcR** implementation. Each cell is paired with an associated larger region in the other scan. The absolute location of each cell and region in their respective surface matrices remain constant. However, the scan on the right is rotated to determine the rotation at which the two scans are the most "similar," as quantified by the *cross-correlation function* (CCF).

For real-valued matrices A and B of dimension $M \times N$ and $P \times Q$, respectively, the cross-correlation function, denoted $(A \star B)$ is defined as

$$(A \star B)[m,n] = \sum_{i=1}^{M} \sum_{j=1}^{N} A[i,j]B[(i+m),(j+n)],$$

where $1 \leq m \leq M+P-1$ and $1 \leq n \leq N+Q-1$. By this definition, the $[m,n]$th element of the resulting $M+P-1 \times N+Q-1$ CCF matrix quantifies the similarity between matrices A and B for a translation of matrix B by m pixels horizontally and n pixel vertically. The index at which the CCF attains a maximum represents the optimal translation needed to align B with A. The CCF as

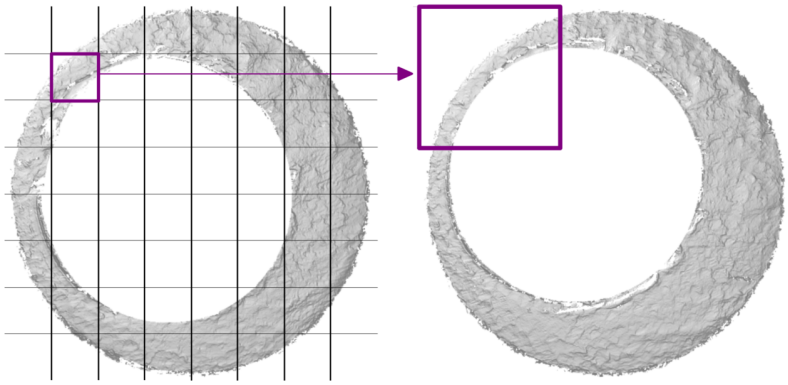


Figure 7: Illustration of comparing a cell in the reference cartridge case scan (left) to a larger region in a questioned cartridge case scan (right). Every one of the cells in the reference cartridge case is similarly paired with a region in the questioned cartridge case. To determine the rotation at which the two cartridge cases align, the cell-region pairs are compared for various rotations of the questioned cartridge case.

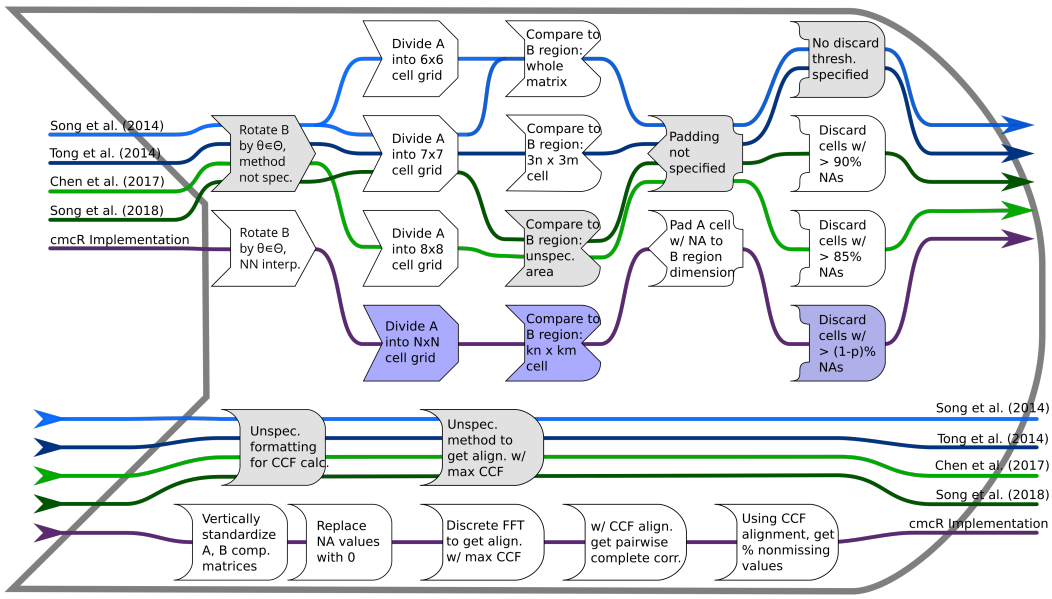


Figure 8: Each CMC implementation uses a slightly different procedure to obtain a similarity score between two cartridge cases. Steps which are implemented with additional user-specified parameters are shaded purple; steps which are described but without sufficient detail are shaded grey.

defined need not be bounded between -1 and 1. However, it is common to normalize the CCF for interpretability, and this is the convention adopted in the **cmcR** package.

Prior to calculating the CCF, the matrices A and B are standardized through subtraction of their respective means and division by their respective standard deviations. This is referred to as the *Areal Cross-Correlation Function* (ACCF) in some CMC papers (Ott, Thompson, and Song 2017). A direct calculation of the CCF for breech face scans based on the definition above is prohibitively slow. While computationally feasible alternatives exist, Song (2013) and other CMC papers do not specify the algorithm used to calculate the CCF.

Published descriptions of the CMC algorithm do not detail how the CCF is calculated. In image processing, it is common to use an implementation based on the Fast Fourier Transform (Brown 1992). This implementation leverages the Cross-Correlation Theorem, which states that for matrices A and B, the CCF can be expressed in terms of a frequency-domain pointwise product:

$$(A \star B)[m,n] = \mathcal{F}^{-1}\left(\overline{\mathcal{F}(A)} \odot \mathcal{F}(B)\right)[m,n],$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the discrete Fourier and inverse discrete Fourier transforms, respectively, and $\overline{\mathcal{F}(A)}$ denotes the complex conjugate (Brigham 1988). Because the product on the right-hand side is calculated pointwise, we trade the moving sum computations from the definition of the CCF for two forward Fourier transformations, a pointwise product, and an inverse Fourier transformation. The Fast Fourier Transform (FFT) algorithm can be used to reduce the computational load considerably. Our implementation of this FFT-based CCF calculation is adapted from the **cartridges3D** package (Tai 2021).

No computational shortcut comes without some trade-offs, though, and this FFT-based CCF calculation is no different. The FFT does not tolerate missing values, and breech faces are not continuous surfaces – all of the white regions in Figure 7 correspond to missing values. While it is unclear how the CCF is implemented in the CMC papers, the **cmcR** package adopts the following conventions:

- Only cells with a minimum proportion of non-missing pixels are assessed. This minimum threshold differs across CMC papers (15% in Chen et al. (2017) vs. 10% in Song et al. (2018), as shown in Figure 8), and is referenced but not specified in several other papers (Tong et al. 2014; Song et al. 2014; Chu, Tong, and Song 2013). The `comparison_calcPropMissing` function computes the proportion of a matrix that is missing (`NA`-valued).
- Missing values are replaced with the overall mean value when the FFT-based CCF is computed (using function `comparison_replaceMissing`).
- The optimal translation is determined using the FFT-based CCF (using `comparison_fft_ccf`).
- Based on the optimal translation determined from the FFT-based CCF, we compute the pairwise complete CCF directly, avoiding any distortion of the CCF computation based on compensation for missing values (using function `comparison_cor`).

All of the steps dealing with cell-based comparisons are implemented as functions of the form `comparison_*`. Similar to the `preProcess_*` functions, the `comparison_*` functions can be chained together through a sequence of pipes. Below, we use the `comparison_allTogether` function to perform the entire cell-based comparison procedure in one call. The comparison procedure is performed twice: once with Fadul 1-1 considered the "reference" scan divided into cells that are compared to the "target" scan Fadul 1-2 and again with the roles reversed.

```
# Fill in most of the arguments first
comp_w_pars <- purrr::partial(.f = comparison_allTogether,
                              numCells = c(8,8), maxMissingProp = .85)

# Then, map the remaining values to theta
kmComparisonFeatures <- purrr::map_dfr(
  seq(-30,30,by = 3),
  ~comp_w_pars(reference = fadul1.1, target = fadul1.2, theta = .))

kmComparisonFeatures_rev <- purrr::map_dfr(
  seq(-30,30,by = 3),
  ~comp_w_pars(reference = fadul1.2, target = fadul1.1, theta = .))
```

The `comparison_allTogether` function consists of the following steps wrapped into a single convenience function:

Table 2: Example of output from correlation cell comparison procedure between Fadul 1-1 and Fadul 1-2 rotated by -24 degrees. Due to the large proportion of missing values that are replaced to compute the FFT-based correlation, the pairwise-complete correlation is most often greater than the FFT-based correlation.

Cell Index	Pairwise-comp. corr.	FFT-based corr.	Δx	Δy	θ
2, 7	0.432	0.228	-14	-33	-24
2, 8	0.464	0.176	9	-44	-24
3, 1	0.841	0.478	-7	15	-24
3, 8	0.699	0.277	-7	5	-24
4, 1	0.850	0.375	-4	11	-24

- `comparison_cellDivision`: Divide the reference scan into cells
- `comparison_getTargetRegions`: Extract regions associated with each reference cell from the target scan
- `comparison_calcPropMissing`: Compute missing proportions and filter out cells with a proportion of missing values above the threshold.
- `comparison_standardizeHeights`: Standardize height values
- `comparison_replaceMissing`: Replace missing values
- `comparison_fft_ccf`: Compute CCF and estimated translations using FFT
- `comparison_alignedTargetCell`: Extract a matrix from the target scan corresponding to the region of the target scan to which the reference cell aligns
- `cor`: Calculate the pairwise-complete correlation between each cell pair

The `comparison_allTogether` is called repeatedly while rotating the target scan by a set of rotation angles. When implementing the High CMC decision rule (Tong, Song, and Chu 2015), both combinations of reference and target scan are examined (e.g. A-B and B-A).

Table 2 shows several rows of the data frame output of the `comparison_allTogether` function for the comparison of Fadul 1-1 vs. Fadul 1-2 considering Fadul 1-1 as the reference scan. Although we used a grid of 8×8 cells, there were only 26 cell-region pairs that contained a sufficient proportion of non-missing values (15% in this example). The features derived from the correlation cell procedure (CCF_{max}, Δx, Δy, θ) are then used to measure the similarity between scans.

Decision rule

For each cell on the reference scan, we calculate the translation $(\Delta x, \Delta y)$ and cross-correlation across rotations by a set of angles θ of the target scan. The task is to determine whether multiple cells come to a "consensus" on a particular translation and rotation. If such a consensus is reached, then there is evidence that a true aligning translation and rotation exists and the cartridge cases match. The CMC decision rules principally differ in how they identify consensus among the $\Delta x, \Delta y, \theta$ values. Here, we describe the two pipelines implemented in the **cmcR** package: using the original decision rule described in Song et al. (2014) and the High CMC decision rule proposed in Tong, Song, and Chu (2015).

The Original CMC decision rule

This section briefly describes the decision rule used in the first CMC paper (Song 2013). For a thorough explanation of the procedure, refer to the CMC Decision Rule Description vignette of the **cmcR** package.

Let x_i, y_i, θ_i denote the translation and rotation parameters which produce the highest CCF for the alignment of cell-region pair i, $i = 1, ..., n$ where n is the total number of cell-region pairs containing a sufficient proportion of non-missing values. Song (2013) propose the median as a consensus $(x_{\text{ref}}, y_{\text{ref}}, \theta_{\text{ref}})$ across the cell-region pairs. Then, the distance between each (x_i, y_i, θ_i) and

Table 3: Different thresholds for translation, rotation, and CCF_{max} are used across different papers. The range in CCF_{max} is particularly notable.

Paper	**Translation T_x, T_y (in pixels)**	**Rotation θ (in degrees)**	CCF_{max}
Song et al. (2014)	20	6	0.60
Tong et al. (2014)	30	3	0.25
Tong et al. (2015)	15	3	0.55
Chen et al. (2017)	20	3	0.40
Song et al. (2018)	20	6	0.50

$(x_{\text{ref}}, y_{\text{ref}}, \theta_{\text{ref}})$ is compared to thresholds $T_x, T_y, T_\theta, T_{\text{CCF}}$. A cell-region pair i is declared a "match" if all of the following conditions hold:

$$\begin{aligned} |x_i - x_{\text{ref}}| &\leq T_x, \\ |y_i - y_{\text{ref}}| &\leq T_y, \\ |\theta_i - \theta_{\text{ref}}| &\leq T_\theta, \\ \text{CCF}_{\text{max},i} &\geq T_{\text{CCF}}. \end{aligned} \tag{1}$$

The number of matching cell-region pairs, the "CMC count," is used as a measure of similarity between the two cartridge cases. Song et al. (2014) indicate that the thresholds $T_x, T_y, T_\theta, T_{\text{CCF}}$ need to be determined experimentally. Table 3 summarizes the thresholds used in various CMC papers.

Unlike the original CMC pipeline, the High CMC decision rule considers multiple rotations for each cell-region pair.

The High CMC decision rule

For the High CMC decision rule, two scans are compared in both directions - i.e., each scan takes on the role of the reference scan that is partitioned into a grid of cells. Tong, Song, and Chu (2015) claim that some matching cell-region pairs "may be mistakenly excluded from the CMC count" under the original decision rule because they attain the largest CCF at a rotation outside the range allowed by T_θ "by chance."

Tong, Song, and Chu (2015) introduce consensus values across all cell-region pairs for each rotation angle θ and calculate a θ-dependent CMC count as the sum of matches observed. Under the High CMC rule, a cell-region pair i is defined as a match conditional on a particular rotation θ if it satisfies the following three conditions:

$$\begin{aligned} |x_{i,\theta} - x_{ref,\theta}| &\leq T_x \\ |y_{i,\theta} - y_{ref,\theta}| &\leq T_y \\ \text{CCF}_{i,\theta} &\geq T_{\text{CCF}}. \end{aligned} \tag{2}$$

The θ-dependent CMC count, CMC_θ, is defined as the sum of matching cell-region pairs.

Tong, Song, and Chu (2015) assert that for a truly matching cartridge case pair, the relationship between θ and CMC_θ should exhibit a "prominent peak" near the true rotation value. That is, CMC_θ should be largest when the scans are close to being correctly aligned. Further, non-matching pairs should exhibit a "relatively flat and random [...] pattern" across the CMC_θ values.

To determine whether a "prominent peak" exists in the relationship between θ and CMC_θ, Tong, Song, and Chu (2015) consider an interval of rotation angles with large associated CMC_θ values. Let $CMC_{\max} = \max_\theta CMC_\theta$ be the maximum CMC_θ count across all rotation angles. For $\tau > 0$, define $S(\tau) = \{\theta : CMC_\theta > (CMC_{\max} - \tau)\}$ as the set of rotations with "large" CMC_θ values. Tong, Song,

and Chu (2015) consider the "angular range" as $R(\tau) = |\max_\theta S(\tau) - \min_\theta S(\tau)|$. If $R(\tau)$ is small, then there is evidence that many cells agree on a single rotation and that the scans match. To arrive at a CMC count similarity score, Tong, Song, and Chu (2015) suggest a value for τ of 1 and determine:

> If the angular range of the "high CMCs" is within the range T_θ, identify the CMCs for each rotation angle in this range and combine them to give the number of CMCs for this comparison in place of the original CMC number.

If the angular range is larger than T_θ, we say that the cartridge case pair "fails" the High CMC criteria and the original CMC number is used. The High CMC decision rule returns a CMC count at least as large as the original decision rule.

Implementation of decision rules

In this section, we implement the decision rules in **cmcR** for both the original and High CMC decision rules. For illustrative purposes, we consider a set of thresholds: $T_x = T_y = 20$, $T_\theta = 6$, and $T_{\text{CCF}} = 0.5$.

Decision rules in **cmcR** are implemented as functions of the form `decision_*`. In particular, the `decision_CMC` function applies both the original and High CMC decision rules depending on if the parameter τ is set. The code below demonstrates the use of `decision_CMC` on the features `kmComparisonFeatures`, extracted from the comparison of scans Fadul 1-1 vs. Fadul 1-2. Convserely, `kmComparisonFeatures_rev` contains the features from a comparison of Fadul 1-2 vs. Fadul 1-1. For comparison, we also compute the CMCs under both decision rules for the comparison between the non-match pair Fadul 1-1 and Fadul 2-1 (not shown to avoid redundancy).

```
kmComparison_cmcs <- kmComparisonFeatures %>% mutate(
  originalMethodClassif =
    decision_CMC(cellIndex = cellIndex, x = x, y = y, theta = theta,
                 corr = pairwiseCompCor, xThresh = 20, thetaThresh = 6,
                 corrThresh = .5),
  highCMCClassif =
    decision_CMC(cellIndex = cellIndex, x = x, y = y, theta = theta,
                 corr = pairwiseCompCor, xThresh = 20, thetaThresh = 6,
                 corrThresh = .5, tau = 1))
```

We use the `cmcPlot` function to visualize congruent matching cells (CMCs) and non-congruent matching cells (non-CMCs). Figure 9 shows the CMCs and non-CMCs in blue and red, respectively, based on the original decision rule. The (red) non-CMC patches are shown in the position where the maximum CCF value in the target scan is attained. The top row shows 18 CMCs in blue and 8 non-CMCs in red when Fadul 1-1 is treated as the reference and Fadul 1-2 the target. The bottom row shows the 17 CMCs and 13 non-CMCs when the roles are reversed. There is no discussion in Song (2013) about combining the results from these two comparison directions, but Tong, Song, and Chu (2015) propose using the minimum of the two CMC counts (17 in this example).

Similarly, CMCs and non-CMCs determined under the High CMC decision rule are shown in Figure 10. Treating Fadul 1-1 and Fadul 1-2 as the reference scan yields 20 and 18 CMCs, respectively. Combining the results as described above, the final High CMC count is 24.

In contrast, Figure 11 shows the CMC results for a comparison between Fadul 1-1 and a known non-match scan, Fadul 2-1, under the exact same processing conditions. Only two cells are classified as congruent matching cells under the original decision rule when Fadul 1-1 is the reference scan. No cells are classified as CMCs in the other direction. While not shown, this pair fails the High CMC criteria and thus was assigned 0 CMCs under the High CMC decision rule.

3 Discussion

Ambiguity in algorithmic descriptions

During the implementation process we encountered ambiguous descriptions of the various CMC pipelines. We include the pre-processing and cell-based comparison procedures in the description of CMC methodology to emphasize how sensitive the final results are to decisions made in these first two steps. The pre-processing and cell-based comparison procedures are discussed only briefly, if at all, in Song et al. (2014), Tong et al. (2014), Tong, Song, and Chu (2015), or Chen et al. (2017). However, the results reported often indicate a sensitivity to these procedures. Ambiguities range from minor implicit parameter choices (e.g., the convergence criteria for the robust Gaussian regression filter (Brinkman

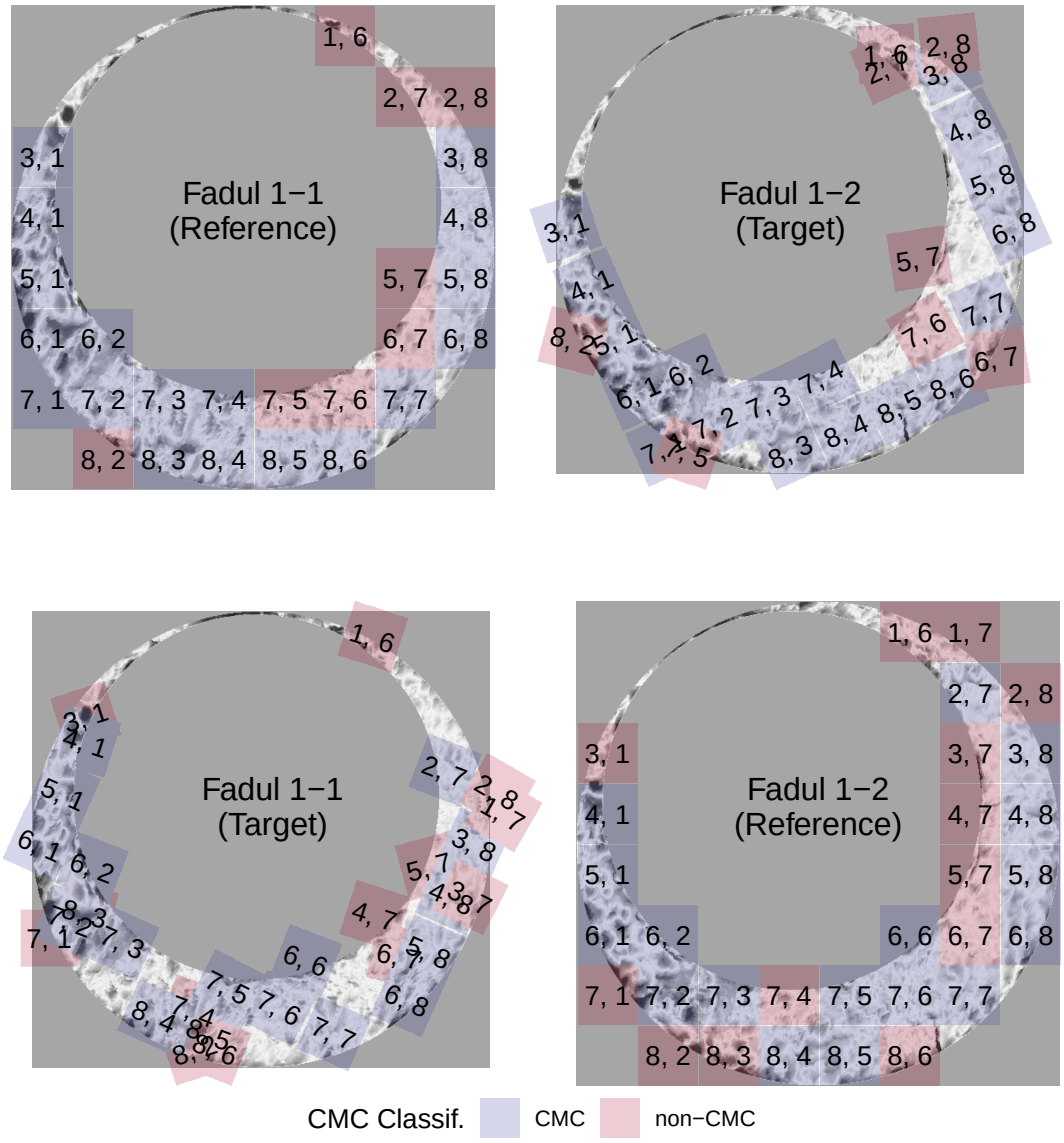


Figure 9: CMC results for the comparison between Fadul 1-1 and Fadul 1-2 using the original decision rule. The two plots in the top row show the 18 CMCs when Fadul 1-1 is treated as the "reference" cartridge case to which Fadul 1-2 (the "target") is compared. The second row shows the 17 CMCs when the roles are reversed. Red cells indicate where cells not identified as congruent achieve the maximum pairwise-complete correlation across all rotations of the target scan.

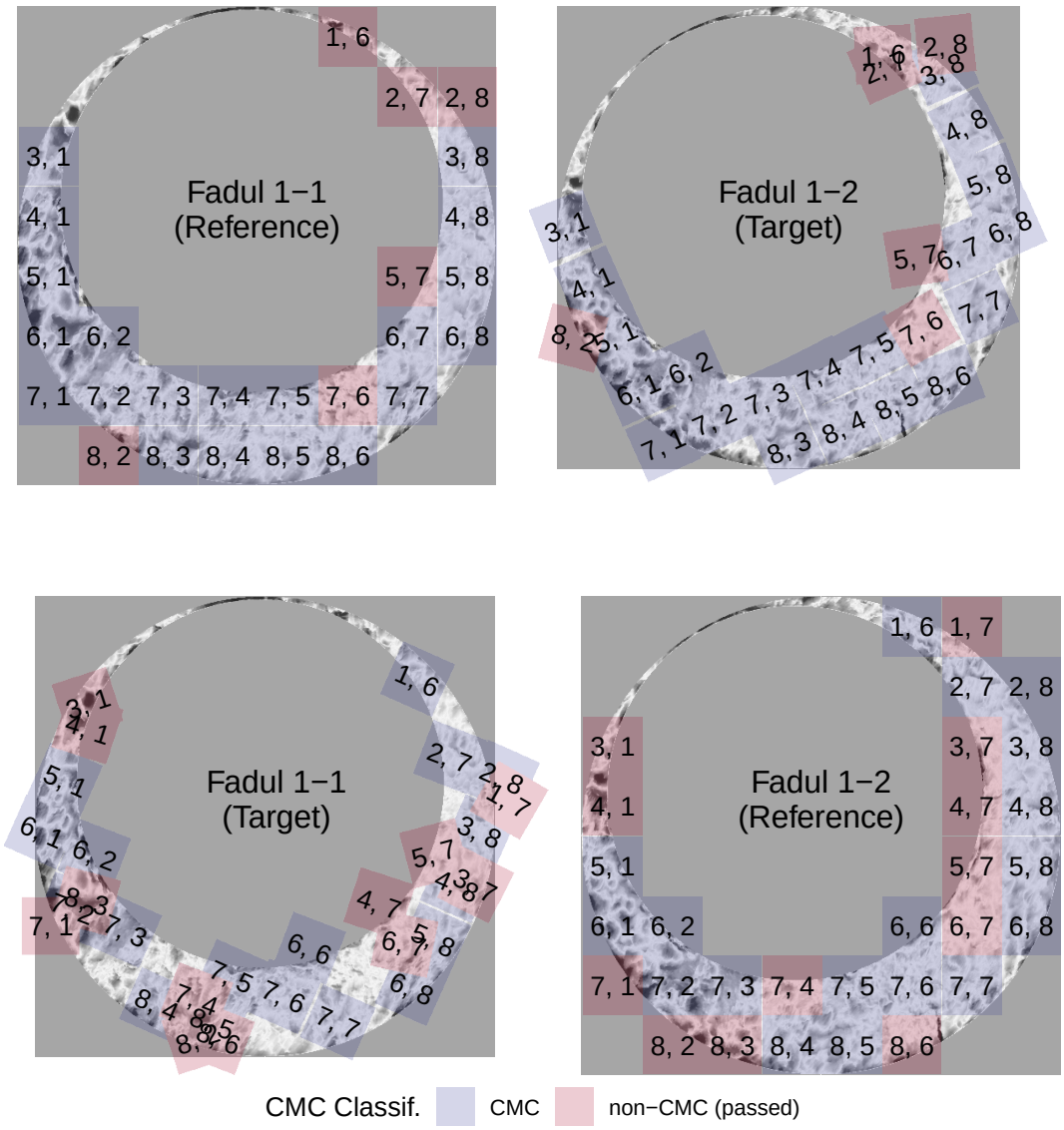


Figure 10: Applying the High CMC decision rule to the comparison of Fadul 1-1 and Fadul 1-2 results in 20 CMCs when Fadul 1-1 is treated as the reference (top) and 18 CMCs when Fadul 1-2 is treated as the reference (bottom). Although the individual comparisons do not yield considerably more CMCs than under the original CMC pipeline, Tong et al. (2015) indicate that the High CMCs from both comparisons are combined as the final High CMC count (each cell is counted at most once). Combining the results means that the High CMC decision rule tends to produce higher CMC counts than the original CMC pipeline. In this example, the combined High CMC count is 24 CMCs.

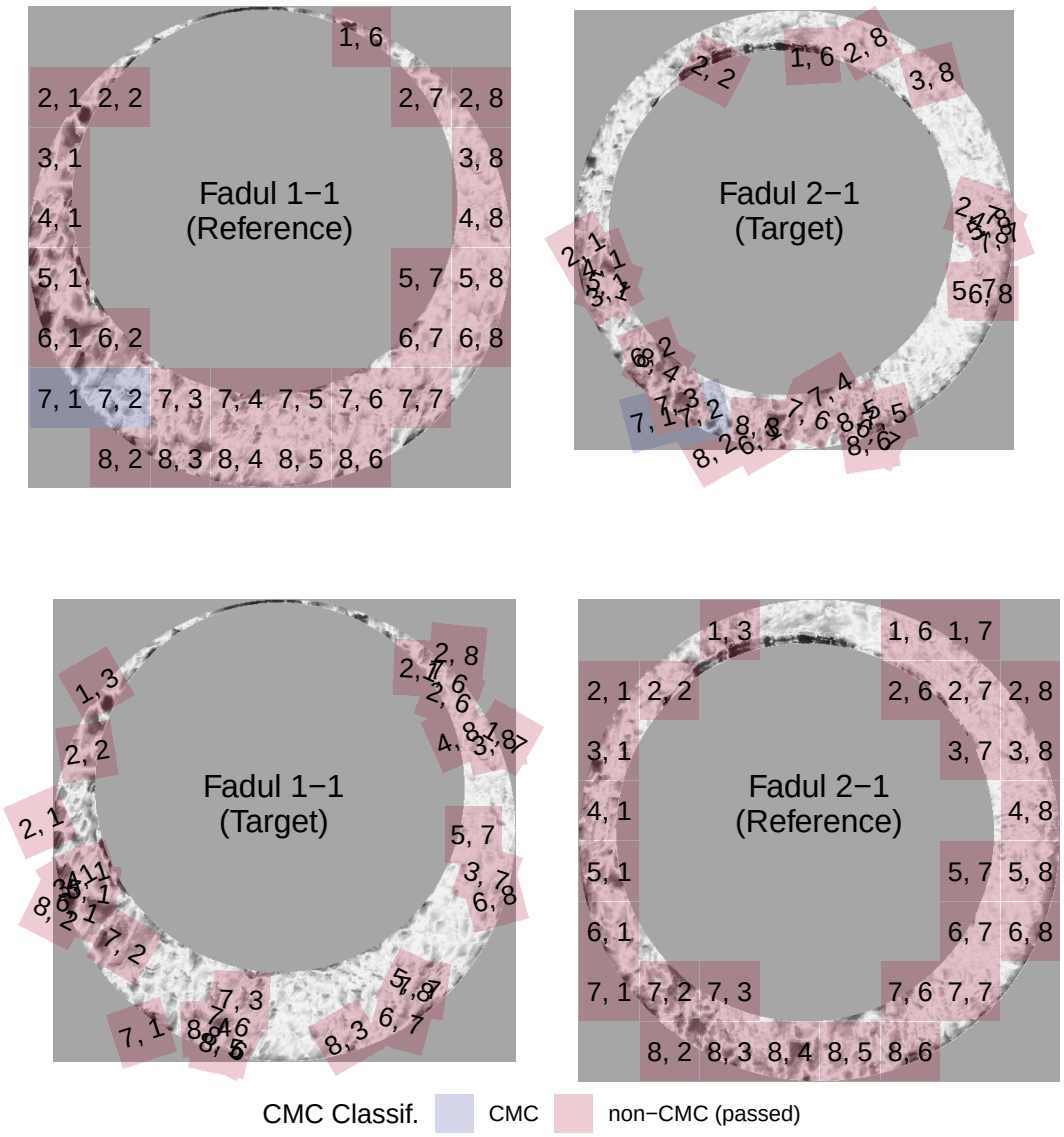


Figure 11: Applying both decision rules to the comparison between the non-match pair Fadul 1-1 and Fadul 2-1 results in 2 CMCs under the original decision rule (shown above) and 0 CMCs under the High CMC decision rule (not shown). The seemingly random behavior of the red cells exemplifies the assumption that cells in a non-match comparison do not exhibit an observable pattern. Random chance should be the prevailing factor in classifying non-match cells as CMCs.

and Bodschwinna 2003)) to procedures that are fundamental to feature calculation (e.g., how the cross-correlation is calculated). We bring up these ambiguities to demonstrate the difficulties that we faced when translating the conceptual description of the CMC pipeline into an actual pipeline. While many of these choices are unlikely to affect the results dramatically, we believe that any amount of variability that exists solely because of uncertainty in how the method was intended to be implemented is both unnecessary and dangerous in this application.

The only solution to such ambiguity is to enumerate, implement, and pare-down the possible choices that could have been made to arrive to published results. Unsurprisingly, this process takes a considerable amount of time and resources that would be better spent furthering the state of the field. During the creation of the **cmcR** package, the process of re-implementing the comparison and decision steps of the pipeline was fairly straightforward. Emulating the pre-processing procedures used, on the other hand, took months of trial and error. Even after this effort, we still have no assurances that our implementation would match the results of the original implementation if applied to other data sets.

In the next section, we describe the process of resolving these ambiguities in the CMC pipeline descriptions. In doing so, we abstract a set of principles by which pipelines and results can be rendered both computationally reproducible and more thoroughly understood.

CMC pattern matching pipeline

As described in the initial data section, the set of cartridge case scans from Fadul et al. (2011) is commonly used to compare the performance of various classification methods (Song et al. 2014; Tong, Song, and Chu 2015; Chen et al. 2017). This set consists of 40 cartridge cases and 780 total comparisons: 63 known match comparisons and 717 known non-match comparisons. Scans of each breech face impression were taken with a Nanofocus Confocal Light Microscope at 10 fold magnification for a nominal lateral resolution of 3.125 microns per pixel and published to the NBTRD (Zheng, Soons, and Thompson 2016). We also use the Weller et al. (2012) data set of 95 cartridge cases for comparison. For the Weller et al. (2012) dataset, we manually isolated the breech face impression regions using the FiX3P software (accessible here: `https://github.com/talenfisher/fix3p`). We compare results from the **cmcR** package to published results using processed scans available through the Iowa State University DataShare repository (Zemmels, Hofmann, and Vanderplas 2022). Our goal is to show that results obtained from **cmcR** are similar, at least qualitatively, to previously published results. However, justification for any differences will ultimately involve educated guesses due to the closed-source nature of the original implementations.

For each cartridge case pair, we calculate CMC counts under both the original and High CMC decision rules. In practice, we classify a cartridge case pair as "matching" if its CMC count surpasses some threshold; 6 CMCs being the generally accepted threshold in many papers (Tong, Song, and Chu 2015; Song et al. 2018; Song 2013). However, this threshold has been shown to not generalize well to all proposed methods and cartridge case data sets (Chen et al. 2017). We instead use an optimization criterion to select parameters. In doing so, we will demonstrate the sensitivity of the pipeline to parameter choice. Additionally, we introduce a set of principles designed to reduce the need for brute-force searches across parameter settings when re-implementing algorithms without accompanying code. Adherence to these principles yields not only computationally reproducible results, but also improves a reader's understanding of a proposed pipeline.

Processing condition sensitivity

Choosing threshold values $T_x, T_y, T_\theta, T_{\text{CCF}}$ for translation, rotation, and maximum cross-correlation is crucial in declaring a particular cell-region pair "congruent." However, many combinations of these thresholds yield perfect separation between the matching and non-matching CMC count distributions. Therefore, choosing parameters based on maximizing classification accuracy does not lead to an obvious, single set of parameters. We instead consider the ratio of between- and within-group variability to measure separation between match and non-match CMC counts.

Let C_{ij} denote the CMC count assigned to the jth cartridge case pair, $j = 1, ..., n_i$ from the ith group, $i = 1, 2$ representing matches and non-matches, respectively. For each set of thresholds we calculate the **Variance Ratio** r as:

$$r = r\left(T_x, T_y, T_\theta, T_{\text{CCF}}\right) = \frac{\sum_{i=1}^{2}\left(\overline{C}_{i.} - \overline{C}_{..}\right)^2}{\sum_{i=1}^{2}\frac{1}{n_i - 1}\sum_{j=1}^{n_i}\left(C_{ij} - \overline{C}_{i.}\right)^2},$$

where $\overline{C}_{i.}$ denotes the within-group CMC count average and $\overline{C}_{..}$ denotes the grand CMC count average. Greater separation between and less variability within the match and non-match CMC count

distributions yields larger r values.

For example, Figure 12 shows results for the original decision rule and the High CMC decision rule for parameters $T_x = 20 = T_y$ pixels, $T_{\text{CCF}} = 0.5$, and $T_\theta = 6$. Despite both decision rules resulting in separation between the matching and non-matching CMC count distributions, the High CMC decision rule yields greater separation as evidenced by the larger r value.

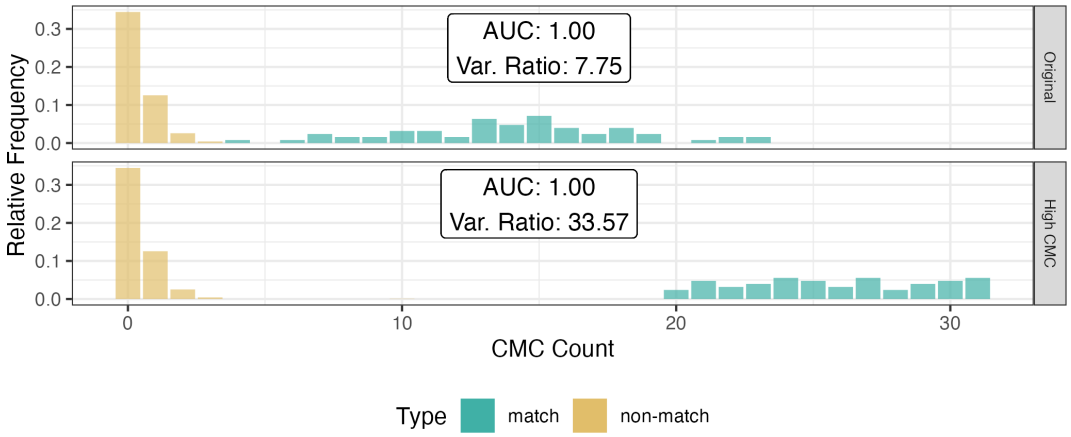


Figure 12: CMC count relative frequencies under the original decision rule and the High CMC decision rule for $T_{\Delta x} = 20 = T_{\Delta y}$ pixels, $T_{\text{CCF}} = 0.5$, and $T_\theta = 6$ degrees. An AUC $= 1$ corresponds to perfect separation of the match and non-match CMC count distributions. We can see that, for this set of processing parameters, the High CMC decision rule yields higher CMC counts for known matches that the original decision rule while known non-matches have the same distribution under both methods.

To explore the pipeline's sensitivity, we consider five dimensions that have a demonstrable impact on CMC counts:

- the decision rule (original or High CMC) used,
- whether the global trend is removed during pre-processing, and
- choice of congruency thresholds: translation T_x, T_y, rotation T_θ, and cross-correlation T_{CCF}.

Choosing a single parameter setting that results in perfect identification is not enough to generally understand the algorithm. Instead, we use the variance ratio r to identify promising ranges of parameters. Figure 13 shows the value of the variance ratio under different parameter settings. We see that the High CMC decision rule yields better separation than the original decision rule under any parameter setting. The largest variance ratio values are achieved for thresholds $T_x, T_y \in [10, 20]$, $T_\theta = 6$, and $T_{\text{CCF}} \in [0.4, 0.5]$. Interestingly, considering Table 3, only the parameters used in Song et al. (2018) fall into these ranges.

As shown in Figure 13, de-trending breech-scans in the pre-processing stage emerges as a critical step to achieve good algorithmic results. This step is not explicitly mentioned in the written-word descriptions of the algorithm in Song (2013), Tong et al. (2014), Tong, Song, and Chu (2015), Chen et al. (2017), or Song et al. (2018), though it appears from their examples that it was used in the process. Figure 13 also illustrates how breaking a pipeline up into modularized steps eases experimentation. We will expand upon this idea in the next section.

We compare the best results from **cmcR** to results presented in previous papers. In particular, we have calculated variance ratio statistics shown in Figure 14 based on CMC counts reported in Song (2013), Tong et al. (2014), Tong, Song, and Chu (2015), Chen et al. (2017), and Song et al. (2018). The last row in each facet shows the variance ratio values obtained from **cmcR**. We see that the implementation provided in **cmcR** yields comparable results to previous CMC papers.

4 Conclusion

Reproducibility is an indispensable component of scientific validity (Goodman, Fanelli, and Ioannidis 2016). In this paper, we demonstrate at least three ways reproducibility can go awry: ambiguity in procedural implementation, missing or incomplete data, and missing or incomplete code. In forensics, many matching algorithms are commonly presented in the form of conceptual descriptions with accompanying results. There is sound reasoning to this; conceptual descriptions are more easily understood by humans compared to computer code. However, using the CMC pipelines as an example

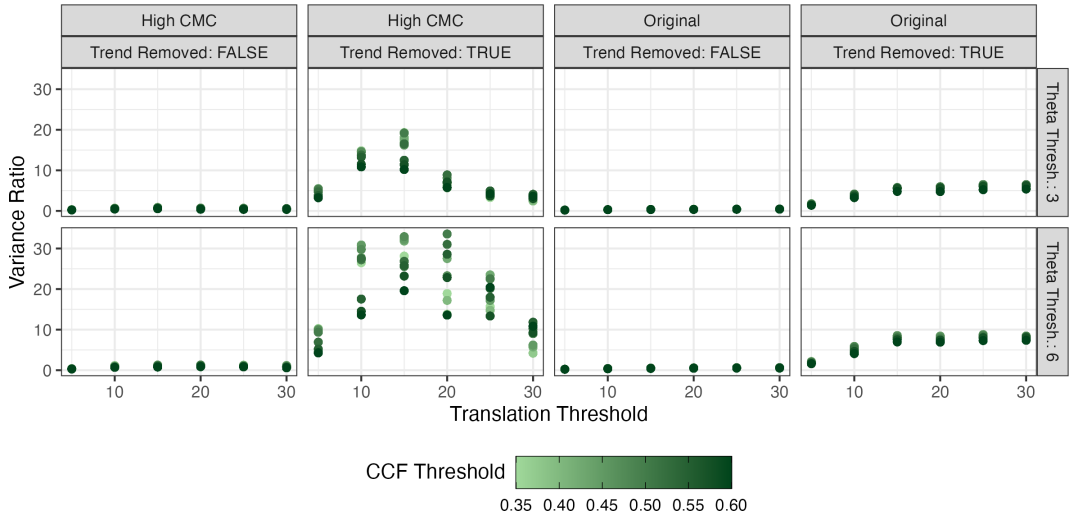


Figure 13: Variance ratio values are plotted for different parameter settings. High variance ratios are indicative of a a good separation between CMC counts for known matching pairs and known-non matching pairs. The High CMC decision rule generally performs better than the original decision rule. Removing the trend during pre-processing has a major impact on the effectiveness of the CMC pipeline. In this setting, translation thresholds $T_x, T_y \in [15, 20]$, a rotation threshold $T_\theta = 6$, and a CCF threshold $T_{\text{CCF}} \in [0.4, 0.5]$ lead to a separation of results.

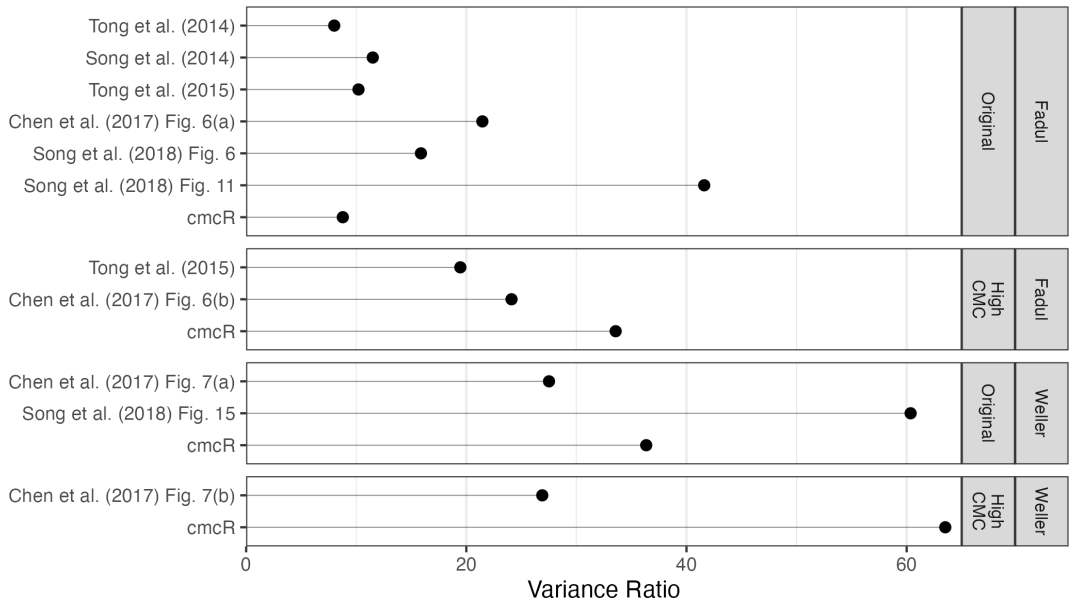


Figure 14: Variance ratios based on results reported in various CMC papers. The High CMC decision rule tends to outperform the original decision rule. However, it should be emphasized that each paper uses very different processing and parameter settings meaning the results are difficult to compare. The values labeled "cmcR" show the largest variance ratio values for the original and High CMC decision rules based on a limited grid search. These results indicate that the CMC pipeline implementation provided in **cmcR** yields comparable results to previous CMC papers.

we have observed the gaps that can exist when translating a conceptual description of an algorithm to a genuine implementation. This is largely due to the fact that conceptual descriptions rarely detail implicit parameter choices required to run an algorithm. Consequently, there are multiple choices that are compatible with the description of an algorithm in a publication. This is dangerous in a forensics context because if many parameter settings are valid but only a narrow range lead to the same conclusion, it is entirely possible that witnesses for the prosecution and defense comef to different conclusions. In order to prevent such misunderstandings, it is not enough to have guidelines for parameter settings and/or a sensitivity study – it is also necessary to standardize the specific computer code. The parameter values are only useful within the context of a single software package or pipeline.

These principles of open, accessible, interoperable code are also critical for a fair (in the legal sense) justice system: the defense has access to the code to understand the evidence against them, lawyers and examiners can assess the utility of the analysis method, and judges can determine whether a method is admissible in court. Transparent and intuitive open-source algorithms, such as **cmcR**, should be considered the gold standard in allowing the forensic science community to validate a pipeline.

Our contribution to the CMC literature is the open-source implementation, which fills the gaps in the human-friendly descriptions in the original papers. In addition, because we have structured the **cmcR** implementation as a modular pipeline, it is easier to improve upon the CMC method and document the effects of specific changes to the algorithm compared to previous versions. The modularization creates an explicit framework to assess the utility and effectiveness of each piece of the algorithm, and allows us to independently manipulate each step while monitoring the downstream impact on the results. Additionally, it allows future collaborators to improve on pieces of the pipeline, adding new options and improving the method without having to re-invent the wheel. Indeed, re-implementing steps of the pipeline is at best a useful academic exercise and at worst a waste of time and resources that could be spent actually improving the pipeline. Even after many months of trial and error, although we have succeeded in obtaining qualitatively similar results on two data sets, it is difficult to know whether our implementation will behave the same as previous implementations on external data sets. Generalizability is an important assessment for any computational algorithm (Vanderplas et al. 2020).

Our application is far from unique: some journals have adopted policies encouraging or requiring that authors provide code and data sufficient to reproduce the statistical analyses, with the goal of building a "culture of reproducibility" in their respective fields (Peng 2009, 2011; Stodden, Guo, and Ma 2013). Peer-review and scientific progress in the truest sense requires that *all* pre-processed data, code, and results be made openly available (Kwong 2017; Desai and Kroll 2017). Our experience with the CMC algorithm suggests that these standards should be adopted by the forensic science community, leveraging open-source ecosystems like R and software sharing platforms such as Github. We firmly believe that the forensic community should not go only halfway, trading a subjective, human black box for objective, proprietary algorithms that are similarly opaque and unauditable. Open, fully reproducible packages like **cmcR** allow research groups to make incremental changes, compare different approaches, and accelerate the pace of research and development.

5 Acknowledgement

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

We greatly appreciate the constructive feedback from the two anonymous reviewers. Special thanks also to all the developers and open-source contributors of R, knitr (Xie 2015, 2014), rticles (Allaire et al. 2021), and the tidyverse (Wickham et al. 2019), without whom this project would not have been possible.

6 Computational details

```
sessionInfo()

#> R version 4.2.2 (2022-10-31)
#> Platform: aarch64-apple-darwin20 (64-bit)
#> Running under: macOS Ventura 13.0
```

```
#>
#> Matrix products: default
#> BLAS:  /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRblas.0.dylib
#> LAPACK: /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib
#>
#> locale:
#> [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
#>
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets  methods   base
#>
#> other attached packages:
#>  [1] patchwork_1.1.2 rgl_1.0.1       x3ptools_0.0.3  forcats_1.0.0
#>  [5] stringr_1.5.0   dplyr_1.1.0     purrr_1.0.1     readr_2.1.3
#>  [9] tidyr_1.3.0     tibble_3.1.8    ggplot2_3.4.0   tidyverse_1.3.2
#> [13] cmcR_0.1.11
#>
#> loaded via a namespace (and not attached):
#>   [1] fs_1.6.1            lubridate_1.9.1     webshot_0.5.4
#>   [4] httr_1.4.4          hunspell_3.0.2      tools_4.2.2
#>   [7] backports_1.4.1     utf8_1.2.3          R6_2.5.1
#>  [10] DBI_1.1.3           colorspace_2.1-0    withr_2.5.0
#>  [13] readbitmap_0.1.5    gridExtra_2.3       tidyselect_1.2.0
#>  [16] compiler_4.2.2      extrafontdb_1.0     textshaping_0.3.6
#>  [19] quantreg_5.94       cli_3.6.0           rvest_1.0.3
#>  [22] SparseM_1.81        xml2_1.3.3          labeling_0.4.2
#>  [25] bookdown_0.32       scales_1.2.1        systemfonts_1.0.4
#>  [28] digest_0.6.31       tiff_0.1-11         yulab.utils_0.0.6
#>  [31] svglite_2.1.1       rmarkdown_2.20      base64enc_0.1-3
#>  [34] jpeg_0.1-10         pkgconfig_2.0.3     htmltools_0.5.4
#>  [37] extrafont_0.19      dbplyr_2.3.0        fastmap_1.1.0
#>  [40] htmlwidgets_1.6.1   rlang_1.0.6         readxl_1.4.2
#>  [43] rstudioapi_0.14     farver_2.1.1        gridGraphics_0.5-1
#>  [46] generics_0.1.3      zoo_1.8-11          jsonlite_1.8.4
#>  [49] googlesheets4_1.0.1 magrittr_2.0.3      kableExtra_1.3.4
#>  [52] ggplotify_0.1.0     Matrix_1.5-3        Rcpp_1.0.10
#>  [55] munsell_0.5.0       fansi_1.0.4         ggnewscale_0.4.8
#>  [58] lifecycle_1.0.3     stringi_1.7.12      yaml_2.3.7
#>  [61] MASS_7.3-58.2       grid_4.2.2          crayon_1.5.2
#>  [64] lattice_0.20-45     cowplot_1.1.1       splines_4.2.2
#>  [67] haven_2.5.1         hms_1.1.2           magick_2.7.3
#>  [70] knitr_1.42          pillar_1.8.1        igraph_1.3.5
#>  [73] codetools_0.2-19    imager_0.42.18      reprex_2.0.2
#>  [76] glue_1.6.2          evaluate_0.20       rjtools_1.0.9.9001
#>  [79] bmp_0.3             BiocManager_1.30.19 modelr_0.1.10
#>  [82] vctrs_0.5.2         png_0.1-8           tzdb_0.3.0
#>  [85] yesno_0.1.2         MatrixModels_0.5-1  Rttf2pt1_1.3.12
#>  [88] cellranger_1.1.0    gtable_0.3.1        assertthat_0.2.1
#>  [91] xfun_0.37           cranlogs_2.1.1      broom_1.0.3
#>  [94] pracma_2.4.2        ragg_1.2.5          viridisLite_0.4.1
#>  [97] survival_3.5-0      googledrive_2.0.0   gargle_1.3.0
#> [100] timechange_0.2.0    ellipsis_0.3.2
```

References

Allaire, JJ, Yihui Xie, R Foundation, Hadley Wickham, Journal of Statistical Software, Ramnath Vaidyanathan, Association for Computing Machinery, et al. 2021. *Rticles: Article Formats for r Markdown*. https://CRAN.R-project.org/package=rticles.

Barthelme, Simon. 2019. *Imager: Image Processing Library Based on 'CImg'*. https://CRAN.R-project.org/package=imager.

Brigham, E. Oran. 1988. *The Fast Fourier Transform and Its Applications*. USA: Prentice-Hall, Inc.

Brinkman, S., and H. Bodschwinna. 2003. "Advanced Gaussian Filters." In *Advanced Techniques for Assessment Surface Topography: Development of a Basis for 3D Surface Texture Standards "SURFSTAND"*,

edited by L. Blunt and X. Jiang. United States: Elsevier Inc. https://doi.org/10.1016/B978-1-903996-11-9.X5000-2.

Brown, Lisa Gottesfeld. 1992. "A Survey of Image Registration Techniques." *ACM Computing Surveys* 24: 325–76. https://dl.acm.org/doi/10.1145/146370.146374.

Chen, Zhe, John Song, Wei Chu, Johannes A. Soons, and Xuezeng Zhao. 2017. "A Convergence Algorithm for Correlation of Breech Face Images Based on the Congruent Matching Cells (CMC) Method." *Forensic Science International* 280 (November): 213–23. https://doi.org/10.1016/j.forsciint.2017.08.033.

Chu, Wei, Mingsi Tong, and John Song. 2013. "Validation Tests for the Congruent Matching Cells (CMC) Method Using Cartridge Cases Fired with Consecutively Manufactured Pistol Slides." *Journal of the Association of Firearms and Toolmarks Examiners* 45 (4): 6. https://www.nist.gov/publications/validation-tests-congruent-matching-cells-cmc-method-using-cartridge-cases-fired.

Crawford, Amy. 2020. "Bayesian Hierarchical Modeling for the Forensic Evaluation of Handwritten Documents." {Ph.D thesis}, Iowa State University. https://doi.org/10.31274/etd-20200624-257.

Curran, James Michael, Tacha Natalie Hicks Champod, and John S Buckleton, eds. 2000. *Forensic Interpretation of Glass Evidence*. Boca Raton, FL: CRC Press.

Desai, Deven R., and Joshua A. Kroll. 2017. "Trust but Verify: A Guide to Algorithms and the Law." *Harvard Journal of Law & Technology (Harvard JOLT)* 31 (1): 1–64.https://ssrn.com/abstract=2959472.

Fadul, T., G. Hernandez, S. Stoiloff, and Gulati Sneh. 2011. "An Empirical Study to Improve the Scientific Foundation of Forensic Firearm and Tool Mark Identification Utilizing 10 Consecutively Manufactured Slides." https://www.ojp.gov/ncjrs/virtual-library/abstracts/empirical-study-improve-scientific-foundation-forensic-firearm-and.

"Geometrical product specifications (GPS) — Surface texture: Areal — Part 72: XML file format x3p." 2017. Standard. Vol. 2014. Geneva, CH: International Organization for Standardization. https://www.iso.org/standard/62310.html.

Goodman, Steven N., Daniele Fanelli, and John P. A. Ioannidis. 2016. "What Does Research Reproducibility Mean?" *Science Translational Medicine* 8 (341): 341ps12–12. https://doi.org/10.1126/scitranslmed.aaf5027.

Goor, Robert, Douglas Hoffman, and George Riley. 2020. "Novel Method for Accurately Assessing Pull-up Artifacts in STR Analysis." *Forensic Science International: Genetics* 51 (November): 102410. https://doi.org/10.1016/j.fsigen.2020.102410.

Hare, Eric, Heike Hofmann, and Alicia Carriquiry. 2017. "Automatic Matching of Bullet Land Impressions." *The Annals of Applied Statistics* 11 (4): 2332–56. http://arxiv.org/abs/1601.05788.

Hesselink, Wim H., Arnold Meijster, and Coenraad Bron. 2001. "Concurrent Determination of Connected Components." *Science of Computer Programming* 41 (2): 173–94. https://doi.org/10.1016/S0167-6423(01)00007-7.

Hofmann, Heike, Susan Vanderplas, Ganesh Krishnan, and Eric Hare. 2020. *x3ptools: Tools for Working with 3D Surface Measurements*. https://cran.r-project.org/web/packages/x3ptools/index.html.

Kwong, Katherine. 2017. "The Algorithm Says You Did It: The Use of Black Box Algorithms to Analyze Complex DNA Evidence Notes." *Harvard Journal of Law & Technology (Harvard JOLT)* 31 (1): 275–302. https://jolt.law.harvard.edu/assets/articlePDFs/v31/31HarvJLTech275.pdf.

Leek, Jeffrey T., and Leah R. Jager. 2017. "Is Most Published Research Really False?" *Annual Review of Statistics and Its Application* 4 (1): 109–22. https://doi.org/10.1146/annurev-statistics-060116-054104.

National Academy of Sciences, Engineering, and Medicine. 2019. *Reproducibility and Replicability in Science*. National Academies Press. https://doi.org/10.17226/25303.

National Research Council. 2009. *Strengthening Forensic Science in the United States: A Path Forward*. Washington, DC: The National Academies Press. https://doi.org/10.17226/12589.

Ott, Daniel, Robert Thompson, and Junfeng Song. 2017. "Applying 3D Measurements and Computer Matching Algorithms to Two Firearm Examination Proficiency Tests." *Forensic Science International* 271 (February): 98–106. https://doi.org/10.1016/j.forsciint.2016.12.014.

Park, Soyoung, and Alicia Carriquiry. 2020. "An Algorithm to Compare Two-Dimensional Footwear Outsole Images Using Maximum Cliques and Speeded-up Robust Feature." *Statistical Analysis and Data Mining: The ASA Data Science Journal* 13 (2): 188–99. https://doi.org/10.1002/sam.11449.

Park, Soyoung, and Sam Tyner. 2019. "Evaluation and Comparison of Methods for Forensic Glass Source Conclusions." *Forensic Science International* 305 (December): 110003. https://doi.org/10.1016/j.forsciint.2019.110003.

Peng, Roger D. 2009. "Reproducible Research and Biostatistics." *Biostatistics* 10 (3): 405–8. https://doi.org/10.1093/biostatistics/kxp014.

———. 2011. "Reproducible Research in Computational Science." *Science* 334 (6060): 1226–27. https://www.jstor.org/stable/41352177.

President's Council of Advisors on Sci. & Tech. 2016. "Forensic Science in Criminal Courts: Ensuring Scientific Validity of Feature-Comparison Methods." https://obamawhitehouse.archives.gov/sites/default/files/microsites/ostp/PCAST/pcast_forensic_science_report_final.pdf.

Song, John. 2013. "Proposed 'NIST Ballistics Identification System (NBIS)' Based on 3D Topography Measurements on Correlation Cells." *American Firearm and Tool Mark Examiners Journal* 45 (2): 11. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=910868.

Song, John, Wei Chu, Mingsi Tong, and Johannes Soons. 2014. "3D Topography Measurements on Correlation Cells—a New Approach to Forensic Ballistics Identifications." *Measurement Science and Technology* 25 (6): 064005. https://doi.org/10.1088/0957-0233/25/6/064005.

Song, John, Theodore V. Vorburger, Wei Chu, James Yen, Johannes A. Soons, Daniel B. Ott, and Nien Fan Zhang. 2018. "Estimating Error Rates for Firearm Evidence Identifications in Forensic Science." *Forensic Science International* 284 (March): 15–32. https://doi.org/10.1016/j.forsciint.2017.12.013.

Stodden, Victoria, Peixuan Guo, and Zhaokun Ma. 2013. "Toward Reproducible Computational Research: An Empirical Analysis of Data and Code Policy Adoption by Journals." Edited by Dmitri Zaykin. *PLoS ONE* 8 (6): e67111. https://doi.org/10.1371/journal.pone.0067111.

Tai, Xiao Hui. 2021. *cartridges3D: Algorithm to Compare Cartridge Case Images*. https://github.com/xhtai/cartridges3D.

Tai, Xiao Hui, and William F. Eddy. 2018. "A Fully Automatic Method for Comparing Cartridge Case Images," *Journal of Forensic Sciences* 63 (2): 440–48. http://doi.wiley.com/10.1111/1556-4029.13577.

The Linux Foundation. 2017. "Using open source software to speed up development and gain business advantage." https://www.linuxfoundation.org/blog/using-open-source-software-to-speed-development-and-gain-business-advantage/.

Thompson, Robert. 2017. *Firearm Identification in the Forensic Science Laboratory*. National District Attorneys Association. https://doi.org/10.13140/RG.2.2.16250.59846.

Tong, Mingsi, John Song, and Wei Chu. 2015. "An Improved Algorithm of Congruent Matching Cells (CMC) Method for Firearm Evidence Identifications." *Journal of Research of the National Institute of Standards and Technology* 120 (April): 102. https://doi.org/10.6028/jres.120.008.

Tong, Mingsi, John Song, Wei Chu, and Robert M. Thompson. 2014. "Fired Cartridge Case Identification Using Optical Images and the Congruent Matching Cells (CMC) Method." *Journal of Research of the National Institute of Standards and Technology* 119 (November): 575. https://doi.org/10.6028/jres.119.023.

Tvedebrink, Torben, Mikkel Meyer Andersen, and James Michael Curran. 2020. "DNAtools: Tools for Analysing Forensic Genetic DNA Data." *Journal of Open Source Software* 5 (45): 1981. https://doi.org/10.21105/joss.01981.

Tyner, Sam, Soyoung Park, Ganesh Krishnan, Karen Pan, Eric Hare, Amanda Luby, Xiao Hui Tai, Heike Hofmann, and Guillermo Basulto-Elias. 2019. "Sctyner/OpenForSciR: Create DOI for Open Forensic Science in r." Zenodo. https://doi.org/10.5281/ZENODO.3418141.

Vanderplas, Susan, Melissa Nally, Tylor Klep, Cristina Cadevall, and Heike Hofmann. 2020. "Comparison of Three Similarity Scores for Bullet LEA Matching." *Forensic Science International*, January. https://doi.org/10.1016/j.forsciint.2020.110167.

Weller, Todd J., Alan Zheng, Robert Thompson, and Fred Tulleners. 2012. "Confocal Microscopy Analysis of Breech Face Marks on Fired Cartridge Cases from 10 Consecutively Manufactured Pistol Slides" 57 (4). https://doi.org/10.1111/j.1556-4029.2012.02072.x.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.

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. http://www.crcpress.com/product/isbn/9781466561595.

———. 2015. *Dynamic Documents with R and Knitr*. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. https://yihui.org/knitr/.

Zemmels, Joseph, Heike Hofmann, and Susan Vanderplas. 2022. "Zemmels et al. (2022) Cartridge Case Scans." Iowa State University. https://doi.org/10.25380/IASTATE.19686297.V1.

Zheng, Xiaoyu A., Johannes A. Soons, and Robert M. Thompson. 2016. "NIST Ballistics Toolmark Research Database." https://tsapps.nist.gov/NRBTD/.

Joseph Zemmels
Center for Statistics and Applications in Forensic Evidence
Department of Statistics
Iowa State University
2438 Osborn Drive
Ames, IA 50011

jzemmels@iastate.edu

Susan VanderPlas
University of Nebraska - Lincoln
Department of Statistics
340 Hardin Hall North Wing
3310 Holdrege St.
Lincoln, NE 68583
susan.vanderplas@unl.edu

Heike Hofmann
Center for Statistics and Applications in Forensic Evidence
Department of Statistics
Iowa State University
2438 Osborn Drive
Ames, IA 50011
hofmann@iastate.edu

Bootstrapping Clustered Data in R using lmeresampler

by Adam Loy and Jenna Korobova

Abstract Linear mixed-effects models are commonly used to analyze clustered data structures. There are numerous packages to fit these models in R and conduct likelihood-based inference. The implementation of resampling-based procedures for inference are more limited. In this paper, we introduce the **lmeresampler** package for bootstrapping nested linear mixed-effects models fit via **lme4** or **nlme**. Bootstrap estimation allows for bias correction, adjusted standard errors and confidence intervals for small samples sizes and when distributional assumptions break down. We will also illustrate how bootstrap resampling can be used to diagnose this model class. In addition, **lmeresampler** makes it easy to construct interval estimates of functions of model parameters.

Introduction

Clustered data structures occur in a wide range of studies. For example, students are organized within classrooms, schools, and districts, imposing a correlation structure that must be accounted for in the modeling process. Similarly, cholesterol measurements could be tracked across time for a number of subjects, resulting in measurements being grouped by subject. Other names for clustered data structures include grouped, nested, multilevel, hierarchical, longitudinal, repeated measurements, and blocked data. The covariance structure imposed by clustered data is commonly modeled using linear mixed-effects (LME) models, also referred to as hierarchical linear or multilevel linear models (J. C. Pinheiro and Bates 2000; Raudenbush and Bryk 2002; Goldstein 2011).

In this paper, we restrict attention to the Gaussian response LME model for clustered data structures. For cluster $i = 1, \ldots, g$, this model is expressed as

$$\underset{(n_i \times 1)}{\boldsymbol{y}_i} = \underset{(n_i \times p)}{\boldsymbol{X}_i} \underset{(p \times 1)}{\boldsymbol{\beta}} + \underset{(n_i \times q)}{\boldsymbol{Z}_i} \underset{(q \times 1)}{\boldsymbol{b}_i} + \underset{(n_i \times 1)}{\boldsymbol{\varepsilon}_i}, \tag{1}$$

where

$$\boldsymbol{\varepsilon}_i \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I}_{n_i}), \boldsymbol{b}_i \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{D}), \tag{2}$$

where $\boldsymbol{\varepsilon}_i$ and $\boldsymbol{b}_i$ are independent for all i, $\boldsymbol{\varepsilon}_i$ is independent of $\boldsymbol{\varepsilon}_j$ for $i \neq j$, and $\boldsymbol{b}_i$ is independent of $\boldsymbol{b}_j$ for $i \neq j$. Here, $\boldsymbol{0}$ denotes a vector of zeros of length n_i (the number of observations in group i), $\boldsymbol{I}_{n_i}$ denotes the n_i-dimensional identity matrix, and $\boldsymbol{D}$ is a $q \times q$ covariance matrix.

In R, the two most popular packages to fit LME models are **nlme** (J. Pinheiro et al. 2017) and **lme4** (Bates et al. 2015). Both packages fit LME models using either maximum likelihood or restricted maximum likelihood methods. These methods rely on the distributional assumptions placed on the residual quantities, $\boldsymbol{\varepsilon}_i$ and $\boldsymbol{b}_i$, as well as large enough sample sizes. While some aspects of LME models are quite robust to model misspecification, others are more sensitive, leading to biased estimates and/or incorrect standard errors. Jacqmin-Gadda et al. (2007) found that inference for the fixed effects is robust if the distribution of the error terms, $\boldsymbol{\varepsilon}_i$, is non-normal or heteroscedastic, but that variance components and random effects are biased if the covariance structure for the error terms is misspecified. The fixed intercept is not robust to misspecification of the random effects (Hui, Müller, and Welsh 2021). In addition, misspecification of the random effects distribution can lead to biased estimates of the variance components and undercoverage of the confidence intervals (Hui, Müller, and Welsh 2021). In cases where distributional assumptions are suspect, bootstrapping provides an alternative inferential approach that leads to consistent, bias-corrected parameter estimates, standard errors, and confidence intervals. Standard errors and confidence intervals for functions of model parameters are also easily calculated using a bootstrap procedure, and are available even in situations where closed-form solutions are not.

A variety of bootstrap procedures for clustered data and the LME model have been proposed and investigated, including the cases (non-parametric) bootstrap, the residual bootstrap, the parametric bootstrap, the random effect block (REB) bootstrap, and the wild bootstrap (Morris 2002; Carpenter, Goldstein, and Rasbash 2003; Field and Welsh 2007; Chambers and Chandra 2013; Modugno and Giannerini 2015). Van der Leeden, Meijer, and Busing (2008) provide a thorough overview of bootstrapping LME models. Sánchez-Espigares and Ocaña (2009) developed a full-featured bootstrapping framework for LME models in R; however, this package is not available and there appears to be no active development. Consequently, R users must pick and choose packages based on what bootstrap procedure they wish to implement. The parametric bootstrap is available in **lme4** via the `bootMer()` function, as is the semi-parametric (residual) bootstrap proposed by Morris (2002). The `simulateY()` function in **nlmeU**

(Galecki and Burzykowski 2013) makes it easy to simulate values of the response variable for `lme` model objects; however, the user is required to implement the remainder of the parametric bootstrap. Chambers and Chandra (2013) made R code available to implement their REB bootstrap as well as the parametric bootstrap and the residual bootstrap proposed by Carpenter, Goldstein, and Rasbash (2003) for LME models fix via `nlme::lme()`. Unfortunately, these functions were not published on CRAN or extended to models fit via `lme4::lmer()`. Bootstrap procedures for specific inferential tasks have also been implemented. The parametric bootstrap has been implemented to carry out likelihood ratio tests for the presence of variance components in **RLRsim** (Scheipl, Greven, and Kuechenhoff 2008). **pbkrtest** (Halekoh and Højsgaard 2014) provides a Kenward-Roger approximation for performing F-tests using a parametric bootstrap approach for LME models and generalized LME models. **rptR** uses the parametric bootstrap to estimate repeatabilities for LME models (Stoffel, Nakagawa, and Schielzeth 2017). Finally, constrained inference for LMEs using the residual bootstrap is implemented in **CLME** (Farnan, Ivanova, and Peddada 2014).

In this paper, we introduce the **lmeresampler** package which implements a suite of bootstrap methods for LME models fit using either **nlme** or **lme4** in a single package. **lmeresampler** provides a unified `bootstrap()` command to reduce cognitive load on the user and also provides access to procedures that were not previously available on CRAN. In the next section, we will clarify some notation and terminology for LME models. In Bootstrap procedures for clustered data, we provide an overview of the bootstrap methods implemented in **lmeresampler**. We then give an Overview of lmeresampler, discuss a variety of Example applications for LME models, and show how to Bootstrapping in parallel works in **lmeresampler**. We conclude with a Summary and highlight areas for future development.

Bootstrap procedures for clustered data

In **lmeresampler**, we implement five bootstrap procedures for clustered data: a cases (i.e., nonparamteric) bootstrap, a residual bootstrap (i.e., semiparametric), a parametric bootstrap, a wild bootstrap, and a block bootstrap. In this section, we provide an overview of these bootstrap approaches. Our discussion focuses on two-level models, but procedures generalize to higher-level models unless otherwise noted.

The cases bootstrap

The cases bootstrap is a fully non-parametric bootstrap that resamples the clusters in the data set to generate bootstrap resamples. Depending on the nature of the data, this resampling can be done only for the top-level cluster, only at the observation-level within a cluster, or at both levels. The choice of the exact resampling scheme should be dictated by the way the data were generated, since the cases bootstrap generates new data by mimicking this process. Van der Leeden, Meijer, and Busing (2008) provide a cogent explanation of how to select a resampling scheme. To help ground the idea of resampling, consider a two-level hierarchical data set where students are organized into schools.

One version of the cases bootstrap is implemented by only resampling the clusters. This version of the bootstrap is what Field and Welsh (2007) term the cluster bootstrap and Goldstein (2011) term the non-parametric bootstrap. We would choose this resampling scheme, for example, if schools were chosen at random and then all students within each school were observed. In this case, the bootstrap proceeds as follows:

1. Draw a random sample, with replacement, of size g from the clusters.
2. For each selected cluster, k, extract all of the cases to form the bootstrap sample $\left(\boldsymbol{y}_k^*, \boldsymbol{X}_k^*, \boldsymbol{Z}_k^*\right)$. Since entire clusters are sampled, the total sample size may differ from the that of the original data set.
3. Refit the model to the bootstrap sample and extract the parameter estimates of interest.
4. Repeat steps 1–3 B times.

An alternative version of the cases bootstrap only resamples the observations within clusters, which makes sense in our example if the schools were fixed and students were randomly sampled within schools.

1. For each cluster $i = 1, \ldots, g$, draw a random sample of the rows for cluster i, with replacement, to form the bootstrap sample $\left(\boldsymbol{y}_i^*, \boldsymbol{X}_i^*, \boldsymbol{Z}_i^*\right)$.
2. Refit the model to the bootstrap sample and extract the parameter estimates of interest.
3. Repeat steps 1–2 B times.

A third version of the cases bootstrap resamples both clusters and cases within clusters. This is what Field and Welsh (2007) term the two-state bootstrap. We would choose this resampling scheme if both schools and students were sampled during the data collection process.

All three versions of the case bootstrap are implemented in **lmeresampler**. We explain how the resampling is specified in our Overview of lmeresampler. Regardless of which version of the cases bootstrap you choose, it requires the weakest conditions: it only requires that the hierarchical structure in the data set is correctly specified.

The parametric bootstrap

The parametric bootstrap simulates random effects and error terms from the fitted distributions to form bootstrap resamples. Consequently, it requires the strongest conditions—that is, the parametric bootstrap assumes that the model, as specified by Equations (1) and (2), is correctly specified.

Let $\widehat{\boldsymbol{\beta}}$, $\widehat{\boldsymbol{D}}$, and $\widehat{\sigma}^2$ be maximum likelihood or restricted maximum likelihood estimates of the fixed effects and variance components from the fitted model. The parametric bootstrap is then implemented through the following steps:

1. Simulate g error term vectors, $\boldsymbol{e}_i^*$, of length n_i from $\mathcal{N}\left(\boldsymbol{0}, \widehat{\sigma}^2 \boldsymbol{I}_{n_i}\right)$.
2. Simulate g random effects vectors, $\boldsymbol{b}_i^*$, from $\mathcal{N}\left(\boldsymbol{0}, \widehat{\boldsymbol{D}}\right)$.
3. Generate bootstrap responses $\boldsymbol{y}_i^* = \boldsymbol{X}_i\widehat{\boldsymbol{\beta}} + \boldsymbol{Z}_i\boldsymbol{b}_i^* + \boldsymbol{e}_i^*$.
4. Refit the model to the bootstrap responses and extract the parameter estimates of interest.
5. Repeat steps 2–4 B times.

The residual bootstrap

The residual bootstrap resamples the residual quantities from the fitted LME model in order to generate bootstrap resamples. There are three general types of residuals for LME models (Haslett and Haslett 2007; Singer, Rocha, and Nobre 2017). *Marginal residuals* correspond to the errors made using the marginal predictions, $\widehat{\boldsymbol{r}}_i = \boldsymbol{y}_i - \widehat{\boldsymbol{y}}_i = \boldsymbol{y}_i - \boldsymbol{X}_i\widehat{\boldsymbol{\beta}}$. *Conditional residuals* correspond to the errors made using predictions conditioned on the clusters, $\widehat{\boldsymbol{e}}_i = \boldsymbol{y}_i - \widehat{\boldsymbol{y}}_i|\boldsymbol{b}_i = \boldsymbol{y}_i - \boldsymbol{X}_i\widehat{\boldsymbol{\beta}} - \boldsymbol{Z}_i\widehat{\boldsymbol{b}}_i$. The *predicted random effects* are the last type of residual quantity and are defined as $\widehat{\boldsymbol{b}}_i = \widehat{\boldsymbol{D}}\boldsymbol{Z}_i'\widehat{\boldsymbol{V}}_i^{-1}\left(\boldsymbol{y}_i - \boldsymbol{X}_i\widehat{\boldsymbol{\beta}}\right)$ where $\widehat{\boldsymbol{V}}_i = \boldsymbol{Z}_i\widehat{\boldsymbol{D}}\boldsymbol{Z}_i' + \widehat{\sigma}^2\boldsymbol{I}_{n_i}$.

A naive implementation of the residual bootstrap would draw random samples, with replacement, from the estimated conditional residuals and the best linear unbiased predictors (BLUPS); however, this will consistently underestimate the variability in the data because the residuals are shrunken toward zero (Morris 2002; Carpenter, Goldstein, and Rasbash 2003). Carpenter, Goldstein, and Rasbash (2003) solve this problem by "reflating" the residual quantities so that the empirical covariance matrices match the estimated covariance matrices prior to resampling:

1. Fit the model and calculate the empirical BLUPs, $\widehat{\boldsymbol{b}}_i$, and the predicted conditional residuals, $\widehat{\boldsymbol{e}}_i$.
2. Mean center each residual quantity and reflate the centered residuals. Only the process to reflate the predicted random effects is discussed below, but the process is analogous for the conditional residuals.
    i. Arrange the random effects into a $g \times q$ matrix, where each row contains the predicted random effects from a single group. Denote this matrix as $\widehat{\boldsymbol{U}}$. Define $\widehat{\boldsymbol{\Gamma}} = \boldsymbol{I}_g \otimes \widehat{\boldsymbol{D}}$, the block diagonal covariance matrix of $\widehat{\boldsymbol{U}}$.
    ii. Calculate the empirical covariance matrix as $\boldsymbol{S} = \widehat{\boldsymbol{U}}'\widehat{\boldsymbol{U}}/g$. We follow the approach of Carpenter, Goldstein, and Rasbash (2003), dividing by the number of groups, g, rather than $g - 1$.
    iii. Find a transformation of $\widehat{\boldsymbol{U}}$, $\widehat{\boldsymbol{U}}^* = \widehat{\boldsymbol{U}}\boldsymbol{A}$, such that $\widehat{\boldsymbol{U}}^{*\prime}\widehat{\boldsymbol{U}}^*/g = \widehat{\boldsymbol{\Gamma}}$. Specifically, we will find $\boldsymbol{A}$ such that $\boldsymbol{A}'\widehat{\boldsymbol{U}}'\widehat{\boldsymbol{U}}\boldsymbol{A}/g = \boldsymbol{A}'\boldsymbol{S}\boldsymbol{A} = \widehat{\boldsymbol{\Gamma}}$. The choice of $\boldsymbol{A}$ is not unique, so we use the recommendation given by Carpenter, Goldstein, and Rasbash (2003): $\boldsymbol{A} = \left(\boldsymbol{L}_D\boldsymbol{L}_S^{-1}\right)'$ where $\boldsymbol{L}_D$ and $\boldsymbol{L}_S$ are the Cholesky factors of $\widehat{\boldsymbol{\Gamma}}$ and $\boldsymbol{S}$, respectively.
3. Draw a random sample, with replacement, from the set $\{\boldsymbol{u}_i^*\}$ of size g, where $\boldsymbol{u}_i^*$ is the ith row of the centered and reflated random effects matrix, $\widehat{\boldsymbol{U}}^*$.
4. Draw g random samples, with replacement, of sizes n_i from the set of the centered and reflated conditional residuals, $\{\boldsymbol{e}_i^*\}$.

5. Generate the bootstrap responses, $\boldsymbol{y}_i^*$, using the fitted model equation: $\boldsymbol{y}_i^* = \boldsymbol{X}_i\widehat{\boldsymbol{\beta}} + \boldsymbol{Z}_i\widehat{\boldsymbol{u}}_i^* + \boldsymbol{e}_i^*$.
6. Refit the model to the bootstrap responses and extract the parameter estimates of interest.
7. Repeat steps 3–6 B times.

Notice that the residual bootstrap is a *semiparametric* bootstrap, since it depends on the model structure (both the mean function and the covariance structure) but not the distributional conditions (Morris 2002).

The random-effects block bootstrap

Another semiparametric bootstrap is the random effect block (REB) bootstrap proposed by Chambers and Chandra (2013). The REB bootstrap can be viewed as a version of the residual bootstrap where conditional residuals are resampled from within clusters (i.e., blocks) to allow for weaker assumptions on the covariance structure of the residuals. The residual bootstrap requires that the conditional residuals are independent and identically distributed, whereas the REB bootstrap relaxes this to only require that the covariance structure of the error terms is similar across clusters. In addition, the REB bootstrap utilizes the marginal residuals to calculate non-parametric predicted random effects rather than relying on the model-based empirical best linear unbiased predictors (EBLUPS). Chambers and Chandra (2013) developed three versions of the REB bootstrap, all of which have been implemented in **lmeresampler**. We refer the reader to Chambers and Chandra (2013) for a discussion of when each should be used. It's important to note that at the time of this writing, that the REB bootstrap has only been explored and implemented for use with two-level models.

REB/0 The base algorithm for the REB bootstrap (also known as REB/0) is as follows:

1. Calculate non-parametric residual quantities for the model.
    a. Calculate the marginal residuals for each group, $\widehat{\boldsymbol{r}}_i = \boldsymbol{y}_i - \boldsymbol{X}_i\widehat{\boldsymbol{\beta}}$.
    b. Calculate the non-parametric predicted random effects, $\tilde{\boldsymbol{b}}_i = \left(\boldsymbol{Z}_i'\boldsymbol{Z}_i\right)^{-1}\boldsymbol{Z}_i'\boldsymbol{r}_i$.
    c. Calculate the non-parametric conditional residuals using the residuals quantities obtained in the previous two steps, $\tilde{\boldsymbol{e}}_i = \widehat{\boldsymbol{r}}_i - \boldsymbol{Z}_i\tilde{\boldsymbol{b}}_i$.
2. Take a random sample, with replacement, of size g from the set $\{\tilde{\boldsymbol{b}}_i\}$. Denote these resampled random effects as $\tilde{\boldsymbol{b}}_i^*$.
3. Take a random sample, with replacement, of size g from the cluster ids. For each sampled cluster, draw a random sample, with replacement, of size n_i from that cluster's vector of error terms, $\tilde{\boldsymbol{e}}_i$.
4. Generate bootstrap responses, $\boldsymbol{y}_i^*$, using the fitted model equation: $\boldsymbol{y}_i^* = \boldsymbol{X}_i\widehat{\beta} + \boldsymbol{Z}_i\tilde{\boldsymbol{b}}_i^* + \tilde{\boldsymbol{e}}_i^*$.
5. Refit the model to the bootstrap sample and extract the parameter estimates of interest.
6. Repeat steps 2–5 B times.

REB/1 The first variation of the REB bootstrap (REB/1) zero centers and reflates the residual quantities prior to resampling in order to satisfy the conditions for consistency (Shao and Tu 1995). This is the same process outlined in Step 2 of the residual bootstrap outlined above.

REB/2 The second variation of the REB bootstrap (REB/2 or postscaled REB) addresses two issues: potential non-zero covariances in the joint bootstrap distribution of the variance components and potential bias in the parameter estimates. After the REB/0 algorithm is run, the following post processing is performed:

Uncorrelate the variance components. To uncorrelate the bootstrap estimates of the variance components produced by REB/0, Chambers and Chandra (2013) propose the following procedure:

1. Apply natural logarithms to the bootstrap distribution of each variance component and form the following matrices. Note that we use ν to denote the total number of variance components.
    - $\boldsymbol{S}^*$: a $B \times \nu$ matrix formed by binding the columns of these distributions together
    - $\boldsymbol{M}^*$: a $B \times \nu$ matrix where each column contains the column mean from the corresponding column in $\boldsymbol{S}^*$
    - $\boldsymbol{D}^*$: a $B \times \nu$ matrix where each column contains the column standard deviation from the corresponding column in $\boldsymbol{S}^*$

2. Calculate $\boldsymbol{C}^* = \text{cov}\,(\boldsymbol{S}^*)$.
3. Calculate $\boldsymbol{L}^* = \boldsymbol{M}^* + \{(\boldsymbol{S}^* - \boldsymbol{M}^*)\,\boldsymbol{C}^{*-1/2}\} \circ \boldsymbol{D}^*$, where $\circ$ denotes the Hadamard (elementwise) product.
4. Exponentiate the elements of $\boldsymbol{L}^*$. The columns of $\boldsymbol{L}^*$ are then uncorrelated versions of the bootstrap variance components.

Center the bootstrap estimates at the original estimate values. To correct bias in the estimation of the fixed effects, apply a mean correction. For each parameter estimate, $\widehat{\beta}_k$, adjust the bootstrapped estimates, $\widehat{\boldsymbol{\beta}}_k^*$ as follows: $\widetilde{\boldsymbol{\beta}}_k^* = \widehat{\beta}_k \mathbf{1}_B + \widehat{\boldsymbol{\beta}}_k^* - \text{avg}\left(\widehat{\boldsymbol{\beta}}_k^*\right)$. To correct bias in the estimation of the variance components, apply a ratio correction. For each estimated variance component , $\widehat{\sigma}_v^2$, adjust the uncorrelated bootstrapped estimates, $\widehat{\boldsymbol{\sigma}}_v^{2*}$ as follows: $\widetilde{\boldsymbol{\sigma}}_v^{2*} = \widehat{\boldsymbol{\sigma}}_v^{2*} \circ \{\widehat{\sigma}_v^2 / \text{avg}\left(\widehat{\boldsymbol{\sigma}}_v^{2*}\right)\}$

The wild bootstrap

The wild bootstrap also relaxes the assumptions made on the error terms of the model, allowing heteroscedasticity both within and across groups. The wild bootstrap is well developed for the ordinary regression model (Liu 1988; Flachaire 2005; Davidson and Flachaire 2008) and Modugno and Giannerini (2015) adapt it for the nested LME model.

To begin, we can re-express model (1) as

$$\boldsymbol{y}_i = \boldsymbol{X}_i\boldsymbol{\beta} + \boldsymbol{v}_i, \text{ where } \boldsymbol{v}_i = \boldsymbol{Z}_i\boldsymbol{b}_i + \boldsymbol{\varepsilon}_i. \tag{3}$$

The wild bootstrap proceeds as follows:

1. Draw a random sample, $w_1, w_2, \ldots, w_g$, from an auxiliary distribution with mean zero and unit variance.
2. Generate bootstrap responses using the re-expressed model equation (3): $\boldsymbol{y}_i^* = \boldsymbol{X}_i\widehat{\boldsymbol{\beta}} + \tilde{\boldsymbol{v}}_i w_j$, where $\tilde{\boldsymbol{v}}_i$ is a heteroscedasticity consistent covariance matrix estimator. Modugno and Giannerini (2015) suggest using what Flachaire (2005) calls HC_2 or HC_3 in the regression context:

$$\text{HC}_2 : \tilde{\boldsymbol{v}}_i = \text{diag}\,(\boldsymbol{I}_{n_i} - \boldsymbol{H}_i)^{-1/2} \circ \boldsymbol{r}_i \tag{4}$$
$$\text{HC}_3 : \tilde{\boldsymbol{v}}_i = \text{diag}\,(\boldsymbol{I}_{n_i} - \boldsymbol{H}_i) \circ \boldsymbol{r}_i, \tag{5}$$

   where $\boldsymbol{H}_i = \boldsymbol{X}_i \left(\boldsymbol{X}_i'\boldsymbol{X}_i\right)' \boldsymbol{X}_i'$, the ith diagonal block of the orthogonal projection matrix, and $\boldsymbol{r}_i$ is the vector of marginal residuals for group i.
3. Refit the model to the bootstrap sample and extract the parameter estimates of interest.
4. Repeat steps 1–3 B times.

Five options for the auxiliary distribution are implemented in **lmeresampler**:

- The two-point distribution proposed by Mammen (1993) takes value $w_i = -(\sqrt{5}-1)/2$ with probability $p = (\sqrt{5}+1)/(2\sqrt{5})$ and $w_i = (\sqrt{5}+1)/2$ with probability $1-p$.
- The two-point Rademacher distribution places equal probability on $w_i = \pm 1$.
- The six-point distribution proposed by Webb (2013) places equal probability on $w_i = \pm\sqrt{1/2}, \pm 1, \pm\sqrt{3/2}$.
- The standard normal distribution.
- The gamma distribution with shape parameter 4 and scale parameter 1/2 (Liu 1988) that is centered to have mean zero. w_i^* are randomly sampled from the gamma distribution, and then centered via $w_i = w_i^* - 2$.

Modugno and Giannerini (2015) found the Mammen distribution to be preferred based on a Monte Carlo study, but only compared it to the Rademacher distribution.

Overview of lmeresampler

The **lmeresampler** package implements the five bootstrap procedures outlined in the previous section for Gaussian response LME models for clustered data structures fit using either **nlme** (J. Pinheiro et al. 2017) or **lme4**. The workhorse function in **lmeresampler** is

Bootstrap	Function name	Required arguments
Cases	`case_bootstrap`	`model,.f,type,B,resample,orig_data,.refit`
Residual	`resid_bootstrap`	`model,.f,type,B,orig_data,.refit`
REB	`reb_bootstrap`	`model,.f,type,B,reb_type,orig_data,.refit`
Wild	`wild_bootstrap`	`model,.f,type,B,hccme,aux.dist,orig_data,.refit`
Parametric	`parametric_bootstrap`	`model,.f,type,B,orig_data,.refit`

Table 1: Summary of the specific bootstrap functions called by '`bootstrap()`' and their required arguments.

```
bootstrap(model, .f, type, B, resample = NULL, reb_type = NULL, hccme, aux.dist,
          orig_data, .refit)
```

The four required parameters to `bootstrap()` are:

- `model`, an `lme` or `merMod` fitted model object.
- `.f`, a function defining the parameter(s) of interest that should be extracted/calculated for each bootstrap iteration.
- `type`, a character string specifying the type of bootstrap to run. Possible values include: `"parametric"`, `"residual"`, `"reb"`, `"wild"`, and `"case"`.
- `B`, the number of bootstrap resamples to generate.
- `.refit`, whether the model should be refit to the bootstrap sample. This defaults to `TRUE`.

There are also five optional parameters: `resample`, `reb_type`, `hccme`, `aux.dist`, and `orig_data`.

- If the user sets `type = "case"`, then they must also set `resample` to specify what cluster resampling scheme to use. `resample` requires a logical vector of length equal to the number of levels in the model. A value of `TRUE` in the *i*th position indicates that cases/clusters at that level should be resampled. For example, to only resample the clusters (i.e., level 2 units) in a two-level model, the user would specify `resample = c(FALSE, TRUE)`.
- If the user sets `type = "reb"`, then they must also set `reb_type` to indicate which version of the REB bootstrap to run. `reb_type` accepts the integers `0`, `1`, and `2` to indicate REB/0, REB/1, and REB/2, respectively.
- If the user sets `type = "wild"`, then they must specify both `hccme` and `aux.dist`. Currently, `hccme` can be set to `"hc2"` or `"hc3"` and `aux.dist` can be set to `"mammen"`, `"rademacher"`, `"norm"`, `"webb"`, or `"gamma"`.
- If variables are transformed within the model formula and `lmer()` is used to fit the LME model, then `orig_data` should be set to the original data frame, since this cannot be recovered from the fitted model object.

The `bootstrap()` function is a generic function that calls functions for each type of bootstrap. The user can call the specific bootstrap function directly if preferred. An overview of the specific bootstrap functions is given in Table 1.

Each of the specific bootstrap functions performs four general steps:

1. *Setup.* Key information (parameter estimates, design matrices, etc.) is extracted from the fitted model to eliminate repetitive actions during the resampling process.
2. *Resampling.* The setup information is passed to an internal resampler function to generate the `B` bootstrap samples.
3. *Refitting.* The model is refit for each of the bootstrap samples and the specified parameters are extracted/calculated.
4. *Clean up.* An internal completion function formats the original and bootstrapped quantities to return a list to the user.

Each function returns an object of class `lmeresamp`, which is a list with elements outlined in Table 2. `print()`, `summary()`, `plot()`, and `confint()` methods are available for `lmeresamp` objects.

lmeresampler also provides the `extract_parameters()` helper function to extract the fixed effects and variance components from `merMod` and `lme` objects as a named vector.

Element	Description
`observed`	values for the original model parameter estimates.
`model`	the original fitted model object.
`.f`	the function call defining the parameters of interest.
`replicates`	a $B \times p$ tibble containing the bootstrapped quantities. Each column contains a single bootstrap distribution.
`stats`	a tibble containing the `observed`, `rep.mean` (bootstrap mean), `se` (bootstrap standard error), and `bias` values for each parameter.
`B`	the number of bootstrap resamples performed.
`data`	the original data set.
`seed`	a vector of randomly generated seeds that are used by the bootstrap.
`type`	a character string specifying the type of bootstrap performed.
`call`	the user's call to the bootstrap function.
`message`	a list of length `B` giving any messages generated during refitting. An entry will be `NULL` if no message was generated.
`warning`	a list of length `B` giving any warnings generated during refitting. An entry will be `NULL` if no warning was generated.
`error`	a list of length `B` giving any errors encountered during refitting. An entry will be `NULL` if no error was encountered.

Table 2: Summary of the values returned by the bootstrap functions.

Example applications

A two-level example: JSP data

As a first application of the bootstrap for nested LME models, consider the junior school project (JSP) data (Goldstein 2011; Mortimore et al. 1988) that is stored as `jsp728` in **lmeresampler**. The data set is comprised of measurements taken on 728 elementary school students across 48 schools in London.

```
library(lmeresampler)
tibble::as_tibble(jsp728)
```

```
#> # A tibble: 728 x 9
#>    mathAge11 mathAge8 gender class     school normAge11 normAge8 schoo~1 mathA~2
#>        <dbl>    <dbl> <fct>  <fct>     <fct>      <dbl>    <dbl>   <dbl>   <dbl>
#>  1        39       36 M      nonmanual 1          1.80     1.55      22.4  13.6
#>  2        11       19 F      manual    1         -2.29    -0.980     22.4  -3.42
#>  3        32       31 F      manual    1         -0.0413   0.638     22.4   8.58
#>  4        27       23 F      nonmanual 1         -0.750   -0.460     22.4   0.579
#>  5        36       39 F      nonmanual 1          0.743    2.15      22.4  16.6
#>  6        33       25 M      manual    1          0.163   -0.182     22.4   2.58
#>  7        30       27 M      manual    1         -0.372    0.0724    22.4   4.58
#>  8        17       14 M      manual    1         -1.63    -1.52      22.4  -8.42
#>  9        33       30 M      manual    1          0.163    0.454     22.4   7.58
#> 10        20       19 M      manual    1         -1.40    -0.980     22.4  -3.42
#> # ... with 718 more rows, and abbreviated variable names 1: schoolMathAge8,
#> #   2: mathAge8c
```

Suppose we wish to fit a model using the math score at age 8, gender, and the father's social class to describe math scores at age 11, including a random intercept for school (Goldstein 2011). This LME model can be fit using the `lmer()` function in **lme4**.

```
library(lme4)
jsp_mod <- lmer(mathAge11 ~ mathAge8 + gender + class + (1 | school), data = jsp728)
```

To implement the residual bootstrap to estimate the fixed effects, we can use the `bootstrap()` function and set `type = "residual"`.

```
(jsp_boot <- bootstrap(jsp_mod, .f = fixef, type = "residual", B = 2000))
```

```
#> Bootstrap type: residual
#>
#> Number of resamples: 2000
#>
#>             term   observed   rep.mean         se          bias
#> 1    (Intercept) 14.1577509 14.1583389 0.66117666  0.0005879882
#> 2       mathAge8  0.6388895  0.6386906 0.02434348 -0.0001989161
#> 3        genderM -0.3571922 -0.3472943 0.33820746  0.0098978735
#> 4 classnonmanual  0.7200815  0.7197059 0.38367036 -0.0003755777
#>
#> There were 0 messages, 0 warnings, and 0 errors.
```

We can then calculate normal, percentile, and basic bootstrap confidence intervals via `confint()`.

```
confint(jsp_boot)
```

```
#> # A tibble: 12 x 6
#>    term           estimate   lower  upper type  level
#>    <chr>             <dbl>   <dbl>  <dbl> <chr> <dbl>
#>  1 (Intercept)      14.2   12.9    15.5   norm   0.95
#>  2 mathAge8          0.639  0.591   0.687 norm   0.95
#>  3 genderM          -0.357 -1.03    0.296 norm   0.95
#>  4 classnonmanual    0.720 -0.0315  1.47  norm   0.95
#>  5 (Intercept)      14.2   12.9    15.4   basic  0.95
#>  6 mathAge8          0.639  0.592   0.688 basic  0.95
#>  7 genderM          -0.357 -1.05    0.301 basic  0.95
#>  8 classnonmanual    0.720 -0.0246  1.46  basic  0.95
#>  9 (Intercept)      14.2   12.9    15.5   perc   0.95
#> 10 mathAge8          0.639  0.590   0.685 perc   0.95
#> 11 genderM          -0.357 -1.02    0.332 perc   0.95
#> 12 classnonmanual    0.720 -0.0203  1.46  perc   0.95
```

The default setting is to calculate all three intervals, but this can be restricted by setting the `type` parameter to `"norm"`, `"basic"`, or `"perc"`.

User-specified statistics: Estimating repeatability/intraclass correlation

The beauty of the bootstrap is its flexibility. Interval estimates can be constructed for functions of model parameters that would otherwise require more complex derivations. For example, the bootstrap can be used to estimate the intraclass correlation. The intraclass correlation measures the proportion of the total variance in the response accounted for by groups, and is an important measure of repeatability in ecology and evolutionary biology (Nakagawa and Schielzeth 2010). As a simple example, we'll consider the `BeetlesBody` data set in **rptR** (Stoffel, Nakagawa, and Schielzeth 2017). This simulated data set contains information on body length (`BodyL`) and the `Population` from which the beetles were sampled. A simple Guassian-response LME model of the form

$$y_{ij} = \beta_0 + b_i + \varepsilon_{ij}, \qquad b_i \sim \mathcal{N}(0, \sigma_b^2), \qquad \varepsilon_{ij} \sim \mathcal{N}(0, \sigma^2),$$

can be used to describe the body length of beetle j from population i. The repeatability is then calculated as $R = \sigma_b^2/(\sigma_b^2 + \sigma^2)$. Below we fit this model using `lmer()`:

```
data("BeetlesBody", package = "rptR")
(beetle_mod <- lmer(BodyL ~ (1 | Population), data = BeetlesBody))
```

```
#> Linear mixed model fit by REML ['lmerMod']
#> Formula: BodyL ~ (1 | Population)
#>    Data: BeetlesBody
#> REML criterion at convergence: 3893.268
#> Random effects:
#>  Groups     Name        Std.Dev.
#>  Population (Intercept) 1.173
#>  Residual               1.798
#> Number of obs: 960, groups:  Population, 12
```

```
#> Fixed Effects:
#> (Intercept)
#>       14.08
```

To construct a bootstrap confidence interval for the repeatability, we first must write a function to calculate it from the fitted model. Below we write a one-off function for this model to demonstrate a "typical" workflow rather than trying to be overly general.

```
repeatability <- function(object) {
  vc <- as.data.frame(VarCorr(object))
  vc$vcov[1] / (sum(vc$vcov))
}
```

The original estimate of repeatability can then be quickly calculated:

```
repeatability(beetle_mod)
```

```
#> [1] 0.2985548
```

To construct a bootstrap confidence interval for the repeatability, we run the desired bootstrap procedure, specifying `.f = repeatability` and then pass the results to `confint()`.

```
(beetle_boot <- bootstrap(beetle_mod, .f = repeatability, type = "parametric", B = 2000))
```

```
#> Bootstrap type: parametric
#>
#> Number of resamples: 2000
#>
#>    observed  rep.mean       se        bias
#> 1 0.2985548 0.2862509 0.090186 -0.01230394
#>
#> There were 0 messages, 0 warnings, and 0 errors.
```

```
(beetle_ci <- confint(beetle_boot, type = "basic"))
```

```
#> # A tibble: 1 x 6
#>   term  estimate lower upper type  level
#>   <chr>    <dbl> <dbl> <dbl> <chr> <dbl>
#> 1 ""       0.299 0.135 0.473 basic  0.95
```

Notice that the `term` column of `beetle_ci` is an empty character string since we did not have `repeatability` return a named vector.

Alternatively, we can `plot()` the results, as shown in Figure 1. The plot method for `lmeresamp` objects uses `stat_halfeye()` from **ggdist** (Kay 2021) to render a density plot with associated 66% and 95% percentile intervals.

```
plot(beetle_boot, .width = c(.5, .9)) +
  ggplot2::labs(
    title = "Bootstrap repeatabilities",
    y = "density",
    x = "repeatability"
  )
```

Bootstrap tests for a single parameter

While **lmeresampler** was designed with a focus on estimation, the bootstrap functions can be used to conduct bootstrap tests on individual parameters. For example, returning to the JSP example, we might be interested in generating approximate p-values for the fixed effects:

```
summary(jsp_mod)$coefficients
```

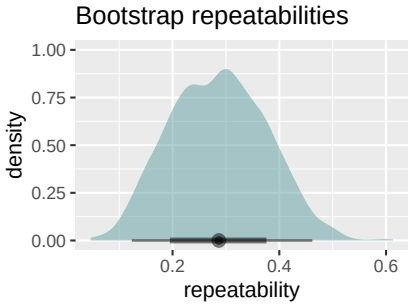


Figure 1: Density plot of the repeatabilities from the beetle model. The median bootstrap repeatability is approximatley .28 and denoted by a point under the density. 66% and 95% confidence intervals for the bootstrap repeatability are (0.217, 0.394) and (0.118, 0.475), respectively, and are displayed as line segments below the density.

```
#>                  Estimate Std. Error   t value
#> (Intercept)    14.1577509 0.73344165 19.303173
#> mathAge8        0.6388895 0.02544478 25.108865
#> genderM        -0.3571922 0.34009284 -1.050279
#> classnonmanual  0.7200815 0.38696812  1.860829
```

To generate a bootstrap p-value for a fixed effect, we must generate B bootstrap resamples under the reduced model (i.e., the null hypothesis), refit the full model, and then calculate the t-statistic for each resample, denoted t_i^*. The bootstrap p-value is then calculated as $(n_{\text{extreme}} + 1)/(B + 1)$ (Davison and Hinkley 1997; Halekoh and Højsgaard 2014).

Using the `bootstrap()` function, you can implement this procedure for a single parameter. For example, if we wish to calculate the bootstrap p-value for the `class` fixed effect we first generate $B = 1000$ bootstrap samples from the reduced model without `class`. In this example, we use the Wild bootstrap to illustrate that we are not restricted to a parametric bootstrap.

```
reduced_model <- update(jsp_mod, . ~ . - class)
reduced_boot <- bootstrap(reduced_model, type = "wild", B = 1000, hccme = "hc2",
                          aux.dist = "mammen", .refit = FALSE)
```

Next, we refit the full model, `jsp_mod`, to each simulation and extract the t-statistic for the `class` variable. Note that the function `extract_t()` extracts the specified t-statistic from the coefficient table from model summary.

```
extract_t <- function(model, term) {
  coef(summary(model))[term, "t value"]
}

tstats <- purrr::map_dbl(
  reduced_boot,
  ~refit(jsp_mod, .x) %>% extract_t(., term = "classnonmanual")
)
```

With the bootstrap t-statistics in hand, we can approximate the p-value using basic logical and arithmetic operators

```
(sum(abs(tstats) >= extract_t(jsp_mod)) + 1) / (1000 + 1)

#> [1] 0.2637363
```

While the above process is not particularly difficult to implement using the tools provided by **lmeresampler**, things get tedious if multiple parameters are of interest in this summary table. To help reduce this burden on the user, we have provided the `bootstrap_pvals()` function that will add bootstrap p-values for each term in the coefficient summary table. For `jsp_mod`, this is achieved in the below code chunk:

```
bootstrap_pvals(jsp_mod, type = "wild", B = 1000, hccme = "hc2", aux.dist = "mammen")
```

```
#> Bootstrap type: wild
#>
#> Number of resamples: 1000
#>
#> # A tibble: 4 x 5
#>   term           Estimate `Std. Error` `t value`  p.value
#>   <chr>             <dbl>        <dbl>     <dbl>    <dbl>
#> 1 (Intercept)      14.2         0.733      19.3  0.000999
#> 2 mathAge8          0.639       0.0254     25.1  0.000999
#> 3 genderM          -0.357       0.340      -1.05 0.257
#> 4 classnonmanual    0.720       0.387       1.86 0.0569
```

It's important to note that running bootstraps for each term in the model is computationally demanding. To speed up the computation, you can run the command in parallel, as we discuss below in Bootstrapping in parallel.

Model comparison

The bootstrap can be useful during model selection. For example, if you are comparing a full and reduced model where the reduced model has fewer random effects, a 50:50 mixture of χ^2 distributions is often used (Stram and Lee 1994); however, J. C. Pinheiro and Bates (2000) point out that this approximation is not always optimal. In this example, we explore the `Machine` data set discussed by J. C. Pinheiro and Bates (2000), which consists of productivity scores for six workers on three brands of machine. This data set can be loaded from **nlme**:

```
data("Machines", package = "nlme")
```

J. C. Pinheiro and Bates (2000) consider two LME models for these data. The first model has a fixed effect for the machine and a random intercept for the worker.

```
reduced_mod <- lmer(score ~ Machine + (1 | Worker), data = Machines, REML = FALSE)
```

The second model has the same fixed effects structure, but adds an additional random effect for the machine within the worker.

```
full_mod <- lmer(score ~ Machine + (1 | Worker/Machine), data = Machines, REML = FALSE)
```

J. C. Pinheiro and Bates (2000) note that the approximate null distribution given by $0.5\chi^2_0 + 0.5\chi^2_1$ is not successful when the models are fit via maximum likelihood, and that the mixture is closer to $0.65\chi^2_0 + 0.35\chi^2_1$. Instead of relying on the conventional approximation, a bootstrap test can be conducted using `bootstrap()` to simulate the responses from the reduced model.

To conduct this bootstrap test, we first extract the observed statistic obtained via `anova()` and then generate $B = 1000$ bootstrap responses from the reduced model, `fm1_machine`. Recall that specifying `.refit = FALSE` returns a data frame of the simulated responses. Here, we use a residual bootstrap for illustration.

```
observed <- anova(full_mod, reduced_mod)$Chisq[2]

reduced_boot <- bootstrap(reduced_mod, type = "residual", B = 1000, .refit = FALSE)
```

Next, we must fit both the full and reduced models to the bootstrap responses and calculate the test statistic. The user-written `compare_models()` function performs this refit and calculation for given models and bootstrap responses. The `control` argument for the full model was set to reduce the number of convergence warnings, since the null model had a variance component of 0 for machines within workers, so we expect warnings as we fit an expanded model.

```
compare_models <- function(full, reduced, newdata) {
  full_mod <- refit(full, newdata,
                    control = lmerControl(check.conv.singular = "ignore",
                                          check.conv.grad = "ignore"))
  reduced_mod <- refit(reduced, newdata)
  anova(full_mod, reduced_mod)$Chisq[2]
}

chisq_stats <- purrr::map_dbl(reduced_boot, ~compare_models(full_mod, reduced_mod, newdata = .x))
```

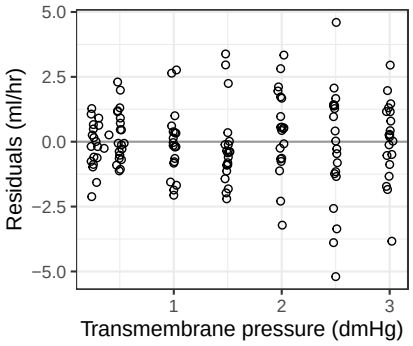


Figure 2: A plot of the conditional residuals against the transmembrane pressure for the dialyzer model. It appears that the variability of the conditional residuals increases with transmembrance pressure, but does this indicate a model condition is violated?

With the test statistics in hand, we can quickly calculate the p-value

```
(sum(chisq_stats >= observed) + 1) / (1000 + 1)
```

```
#> [1] 0.000999001
```

Simulation-based model diagnostics

Our final example illustrates how the `bootstrap()` function can be used for model diagnosis. Loy, Hofmann, and Cook (2017) propose using the lineup protocol to diagnose LME models, since artificial structures often appear in conventional residual plots for this model class that are not indicative of a model deficiency.

In this example, we consider the `Dialyzer` data set provided by **nlme**. The data arise from a study characterizing the water transportation characteristics of 20 high flux membrane dialyzers, which were introduced to reduce the time a patient spends on hemodialysis (Vonesh and Carter 1992). The dialyzers were studied in vitro using bovine blood at flow rates of either 200 or 300 ml/min. The study measured the the ultrafiltration rate (ml/hr) at even transmembrane pressures (in mmHg). J. C. Pinheiro and Bates (2000) discuss modeling these data. Here, we explore how to create a lineup of residual plots to investigate the adequacy of the initial homoscedastic LME model fit by J. C. Pinheiro and Bates (2000).

```
library(nlme)
dialyzer_mod <- lme(
  rate ~ (pressure + I(pressure^2) + I(pressure^3) + I(pressure^4)) * QB,
  data = Dialyzer,
  random = ~ pressure + I(pressure^2)
)
```

J. C. Pinheiro and Bates (2000) construct a residual plot of the conditional residuals plotted against the transmembrane pressure to explore the adequacy of the fitted model (Figure 2). There appears to be increasing spread of the conditional residuals, which would indicate that the homoscedastic model is not sufficient.

To check if this pattern is actually indicative of a problem, we construct a lineup of residual plots. To do this, we must generate data from a number of properly specified models, say 19, to serve as decoy residual plots. Then, we create a faceted set of residual plots where the observed residual plot (Figure 2) is randomly assigned to a facet. To generate the residuals from properly specified models, we use the parametric bootstrap via `bootstrap()` and extract a data frame containing the residuals from each bootstrap sample using `hlm_resid()` from **HLMdiag** (Loy and Hofmann 2014).

```
set.seed(1234)
library(HLMdiag)
sim_resids <- bootstrap(dialyzer_mod, .f = hlm_resid, type = "parametric", B = 19)
```

The simulated residuals are stored in the `replicates` element of the `sim_resids` list. `sim_resids$replicates` is a tibble containing the 19 bootstrap samples, with the the replicate number stored in the `.n` column.

```
dplyr::glimpse(sim_resids$replicates)
```

```
#> Rows: 2,660
#> Columns: 15
#> $ id              <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,~
#> $ rate            <dbl> -0.8579226, 17.0489029, 34.3658738, 44.8495547, 44.525~
#> $ pressure        <dbl> 0.240, 0.505, 0.995, 1.485, 2.020, 2.495, 2.970, 0.240~
#> $ `I(pressure^2)` <I<dbl>>   0.0576, 0.255025, 0.990025, 2.205225,   4.0804, 6~
#> $ `I(pressure^3)` <I<dbl>>     0.013824,  0.128787625,  0.985074875,  3.274759~
#> $ `I(pressure^4)` <I<dbl>>   0.00331776, 0.065037...., 0.980149...., 4.863017.~
#> $ QB              <fct> 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200,~
#> $ Subject         <ord> 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, ~
#> $ .resid          <dbl> -1.19758632, 0.65600425, -0.90104247, 1.39131045, 0.11~
#> $ .fitted         <dbl> 0.3396637, 16.3928987, 35.2669162, 43.4582443, 44.4100~
#> $ .ls.resid       <dbl> -0.51757746, 1.23968783, -1.32231163, 0.59071041, 0.30~
#> $ .ls.fitted      <dbl> -0.3403452, 15.8092151, 35.6881854, 44.2588443, 44.223~
#> $ .mar.resid      <dbl> -2.1236410, -0.3100360, -2.1298323, -0.3453118, -2.455~
#> $ .mar.fitted     <dbl> 1.265718, 17.358939, 36.495706, 45.194867, 46.981057, ~
#> $ .n              <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
```

Now, we use the `lineup()` function from **nullabor** (Buja et al. 2009) to generate the lineup data. `lineup()` will randomly insert the observed (`true`) data into the `samples` data, "encrypt" the position of the observed data, and print a message that you can later decrypt in the console.

```
library(nullabor)
lineup_data <- lineup(true = hlm_resid(dialyzer_mod), n = 19, samples = sim_resids$replicates)
```

```
#> decrypt("CLg7 X161 sO bJws6sJO qj")
```

```
dplyr::glimpse(lineup_data)
```

```
#> Rows: 2,800
#> Columns: 15
#> $ id              <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,~
#> $ rate            <dbl> -0.8579226, 17.0489029, 34.3658738, 44.8495547, 44.525~
#> $ pressure        <dbl> 0.240, 0.505, 0.995, 1.485, 2.020, 2.495, 2.970, 0.240~
#> $ `I(pressure^2)` <I<dbl>>   0.0576, 0.255025, 0.990025, 2.205225,   4.0804, 6~
#> $ `I(pressure^3)` <I<dbl>>     0.013824,  0.128787625,  0.985074875,  3.274759~
#> $ `I(pressure^4)` <I<dbl>>   0.00331776, 0.065037...., 0.980149...., 4.863017.~
#> $ QB              <fct> 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200,~
#> $ Subject         <ord> 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, ~
#> $ .resid          <dbl> -1.19758632, 0.65600425, -0.90104247, 1.39131045, 0.11~
#> $ .fitted         <dbl> 0.3396637, 16.3928987, 35.2669162, 43.4582443, 44.4100~
#> $ .ls.resid       <dbl> -0.51757746, 1.23968783, -1.32231163, 0.59071041, 0.30~
#> $ .ls.fitted      <dbl> -0.3403452, 15.8092151, 35.6881854, 44.2588443, 44.223~
#> $ .mar.resid      <dbl> -2.1236410, -0.3100360, -2.1298323, -0.3453118, -2.455~
#> $ .mar.fitted     <dbl> 1.265718, 17.358939, 36.495706, 45.194867, 46.981057, ~
#> $ .sample         <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
```

With the lineup data in hand, we can create a lineup of residual plots using `facet_wrap()`:

```
ggplot(lineup_data, aes(x = pressure, y = .resid)) +
  geom_hline(yintercept = 0, color = "gray60") +
  geom_point(shape = 1) +
  facet_wrap(~.sample) +
  theme_bw() +
  labs(x = "Transmembrane pressure (dmHg)", y = "Residuals (ml/hr)")
```

In Figure 3, the observed residual plot is in position 13. If you can discern this plot from the field of decoys, then there is evidence that the fitted homogeneous LME model is deficient. In this case, we believe 13 is discernibly different, as expected based on the discussion in J. C. Pinheiro and Bates (2000).

Since we have discovered signs of within-group heteroscedasticity, we could reformulate our model via `nlme::lme()` adding a `weights` argument using `varPower`, or we could utilize the Wild bootstrap, as illustrated below.

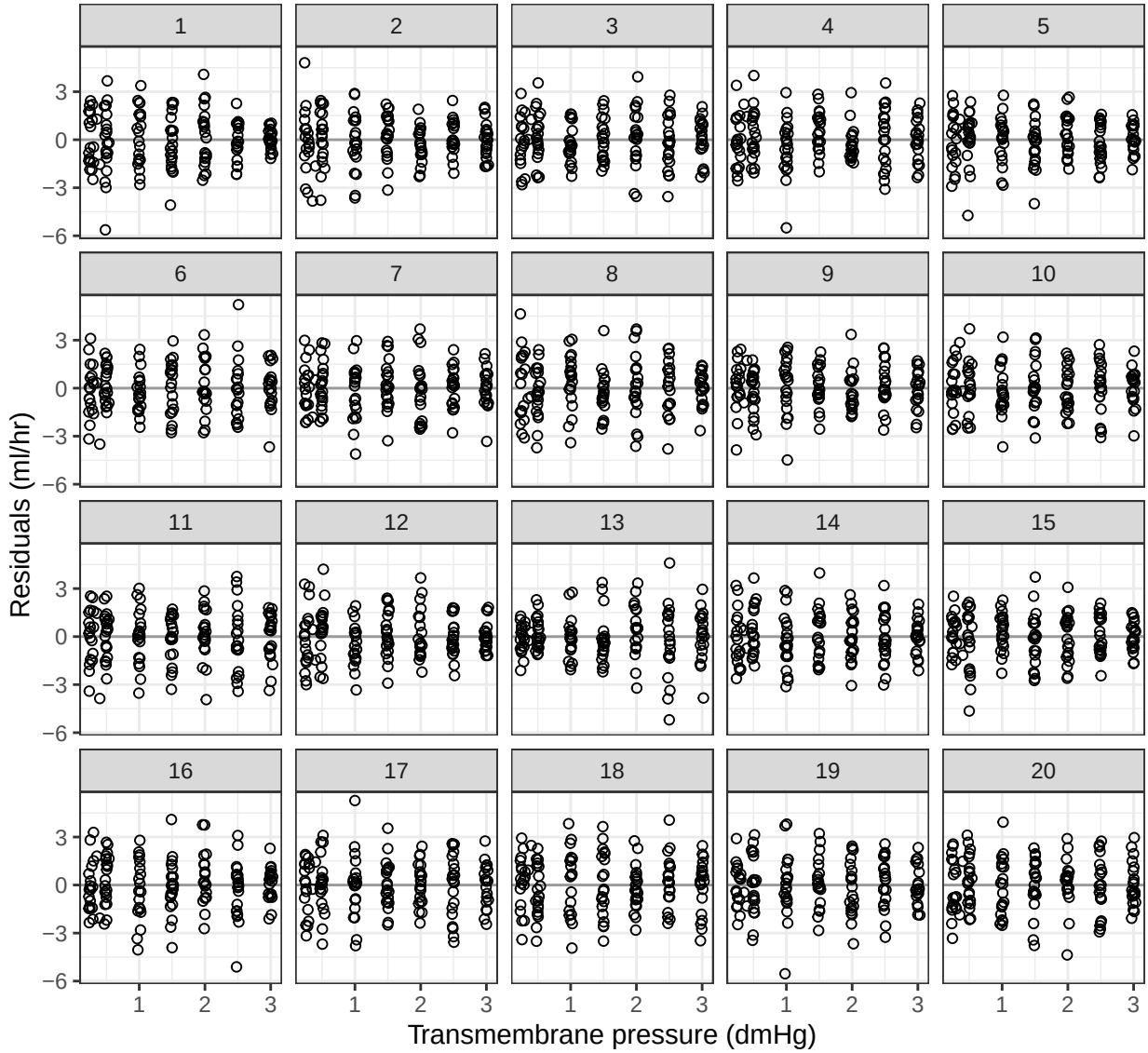


Figure 3: A lineup of the conditional residuals against the transmembrane pressure for the dialyzer model. One of the facets contains the true residual plot generated from the fitted model, the others are decoys generated using a parametric bootstrap. Facet 13 contains the observed residuals and is discernibly different from decoy plots, providing evidence of that a model condition has been violated.

```
wild_dialyzer <- bootstrap(dialyzer_mod, .f = fixef, type = "wild", B = 1000,
                           hccme = "hc2", aux.dist = "webb")

confint(wild_dialyzer, type = "perc")

#> # A tibble: 10 x 6
#>    term                estimate  lower   upper type  level
#>    <chr>                  <dbl>  <dbl>   <dbl> <chr> <dbl>
#>  1 (Intercept)          -16.0   -18.0  -13.9   perc   0.95
#>  2 pressure              88.4    77.4   99.1   perc   0.95
#>  3 I(pressure^2)        -44.3   -57.1  -30.8   perc   0.95
#>  4 I(pressure^3)          9.17    2.45  15.5   perc   0.95
#>  5 I(pressure^4)         -0.690  -1.70   0.425 perc   0.95
#>  6 QB300                 -1.26   -4.55   2.06  perc   0.95
#>  7 pressure:QB300         0.621 -16.0   16.9   perc   0.95
#>  8 I(pressure^2):QB300    3.15  -18.5   27.6   perc   0.95
#>  9 I(pressure^3):QB300    0.102 -12.0   11.1   perc   0.95
#> 10 I(pressure^4):QB300   -0.172  -1.98   1.84  perc   0.95
```

Bootstrapping in parallel

Bootstrapping is a computationally demanding task, but bootstrap iterations do not rely on each other so they are easy to implement in parallel. Rather than building parallel processing into **lmeresampler**, we created a utility function, `combine_lmeresamp()`, that allows the user to easily implement parallel processing using **doParallel** (Microsoft and Weston 2020a) and **foreach** (Microsoft and Weston 2020b). The code is thus concise and simple enough for users without much experience with parallelization, while also providing flexibility to the user.

doParallel and **foreach** default to multicore (i.e., forking) functionality when using parallel computing on UNIX operating systems and snow-like (i.e., clustering) functionality on Windows systems. In this section, we will use clustering in our example. For more information on forking, we refer the reader to the vignette in Microsoft and Weston (2020a).

The basic idea behind clustering is to execute tasks as a "cluster" of computers. Each cluster needs to be fed in information separately, and as a consequence clustering has more overhead than forking. Clusters also need to be made and stopped with each call to `foreach()` to explicitly tell the CPU when to begin and end the parallelization.

Below, we revisit the JSP example and distribute 2000 bootstrap iterations equally over two cores:

```
library(foreach)
library(doParallel)

#> Loading required package: iterators

#> Loading required package: parallel

set.seed(5678)

# Starting a cluster with 2 cores
no_cores <- 2
cl <- makeCluster(no_cores)
registerDoParallel(cores = no_cores)

# Run 1000 bootstrap iterations on each core
boot_parallel <- foreach(
  B = rep(1000, 2),
  .combine = combine_lmeresamp,
  .packages = c("lmeresampler", "lme4")
) %dopar% {
  bootstrap(jsp_mod, .f = fixef, type = "parametric", B = B)
}

# Stop the cluster
stopCluster(cl)
```

The `combine_lmeresamp()` function combines the two `lmeresamp` objects that are returned from the two `bootstrap()` calls into a single `lmeresamp` object. Consequently, working with the returned object proceeds as previously discussed.

It's important to note that running a process on two cores does not yield a runtime that is twice as fast as running the same process on one core. This is because parallelization takes some overhead to split the processes, so while runtime will substantially improve, it will not correspond exactly to the number of cores being used. For example, the runtime for the JSP example run on a single core was

```
#>    user  system elapsed
#>  23.084   1.212  24.376
```

and the runtime for the JSP run on two cores was

```
#>    user  system elapsed
#>  23.550   1.800  12.747
```

These timings were generated using `system.time()` on a MacBook Pro with a 2.9 GHz Quad-Core Intel Core i7 processor. In this set up, running the 2000 bootstrap iterations over two cores reduced the runtime by a factor of about 1.91, but this will vary based on the hardware and setting used.

Summary

In this paper, we discussed our implementation of five bootstrap procedures for nested, Gaussian-response LME models fit via the **nlme** or **lme4** packages. The `bootstrap()` function in **lmeresampler** provides a unified interface to these procedures, allowing users to easily bootstrap their fitted LME models. In our examples, we illustrated the basic usage of the bootstrap, how it can be used to estimate functions of parameters, how it can be used for testing, and how it can be used to create simulation-based visual diagnostics. The bootstrap approach to inference is computationally intensive, so we have also demonstrated how users can bootstrap in parallel.

While this paper focused solely on the nested, Gaussian-response LME model, **lmeresampler** implements bootstrap procedures for a wide class of models. Specifically, the cases, residual, and parametric bootstraps can be used to bootstrap generalized LME models fit via `lme4::glmer()`. Additionally, the parametric bootstrap works with LME models with crossed random effects, though the results may not be optimal (Mccullagh 2000). Future development of **lmeresampler** will focus on implementing additional extensions, especially for crossed data structures.

Acknowledgements

We thank Spenser Steele for his contributions to the original code base of **lmeresampler**. We also thank the reviewers and associate editor whose comments improved the quality of this paper.

References

Bates, Douglas, Martin Mächler, Ben Bolker, and Steve Walker. 2015. "Fitting Linear Mixed-Effects Models Using lme4." *Journal of Statistical Software* 67 (1): 1–48. https://doi.org/10.18637/jss.v067.i01.

Buja, Andreas, Dianne Cook, Heike Hofmann, Michael Lawrence, Eun-kyung Lee, Deborah F. Swayne, and Hadley Wickham. 2009. "Statistical Inference for Exploratory Data Analysis and Model Diagnostics." *Royal Society Philosophical Transactions A* 367 (1906): 4361–83. https://doi.org/10.1098/rsta.2009.0120.

Carpenter, James R, Harvey Goldstein, and Jon Rasbash. 2003. "A Novel Bootstrap Procedure for Assessing the Relationship Between Class Size and Achievement." *Journal of the Royal Statistical Society, Series C* 52 (4): 431–43. https://doi.org/10.1111/1467-9876.00415.

Chambers, Raymond, and Hukum Chandra. 2013. "A Random Effect Block Bootstrap for Clustered Data." *Journal of Computational and Graphical Statistics* 22 (2): 452–70. https://doi.org/10.1080/10618600.2012.681216.

Davidson, Russell, and Emmanuel Flachaire. 2008. "The Wild Bootstrap, Tamed at Last." *Journal of Econometrics* 146 (1): 162–69. https://doi.org/10.1016/j.jeconom.2008.08.003.

Davison, A. C., and D. V. Hinkley. 1997. *Boostrap Methods and Their Application*. Cambridge University Press.

Farnan, Laura, Anastasia Ivanova, and Shyamal D. Peddada. 2014. "Linear Mixed Efects Models Under Inequality Constraints with Applications." *PLOS ONE* 9 (1). https://doi.org/10.1371/journal.pone.0084778.

Field, Christopher A, and A H Welsh. 2007. "Bootstrapping Clustered Data." *Journal of the Royal Statistical Society, Series B* 69 (3): 369–90. https://doi.org/10.1111/j.1467-9868.2007.00593.x.

Flachaire, Emmanuel. 2005. "Bootstrapping Heteroskedastic Regression Models: Wild Bootstrap Vs. Pairs Bootstrap." *Computational Statistics & Data Analysis* 49 (2): 361–76. https://doi.org/10.1016/j.csda.2004.05.018.

Galecki, Andrzej, and Tomasz Burzykowski. 2013. *Linear Mixed-Effects Models Using r: A Step-by-Step Approach*. New York: Springer. https://doi.org/10.1007/978-1-4614-3900-4.

Goldstein, Harvey. 2011. *Multilevel Statistical Models*. 4th ed. West Sussex: John Wiley & Sons, Ltd. https://doi.org/10.1002/9780470973394.

Halekoh, Ulrich, and Søren Højsgaard. 2014. "A Kenward-Roger Approximation and Parametric Bootstrap Methods for Tests in Linear Mixed Models – the R Package pbkrtest." *Journal of Statistical Software* 59 (9): 1–30. https://doi.org/10.18637/jss.v059.i09.

Haslett, John, and Stephen J Haslett. 2007. "The Three Basic Types of Residuals for a Linear Model." *International Statistical Review* 75 (1): 1–24. https://doi.org/10.1111/j.1751-5823.2006.00001.x.

Hui, Francis K C, Samuel Müller, and Alan H Welsh. 2021. "Random Effects Misspecification Can Have Severe Consequences for Random Effects Inference in Linear Mixed Models." *International Statistical Review = Revue Internationale de Statistique* 89 (1): 186–206. https://doi.org/10.1111/insr.12378.

Jacqmin-Gadda, Hélène, Solenne Sibillot, Cécile Proust, Jean-Michel Molina, and Rodolphe Thiébaut. 2007. "Robustness of the Linear Mixed Model to Misspecified Error Distribution." *Computational Statistics & Data Analysis* 51 (10): 5142–54. https://doi.org/10.1016/j.csda.2006.05.021.

Kay, Matthew. 2021. *ggdist: Visualizations of Distributions and Uncertainty*. https://doi.org/10.5281/zenodo.3879620.

Liu, Regina Y. 1988. "Bootstrap Procedures Under Some Non-i.i.d. Models." *Annals of Statistics* 16 (4): 1696–1708. https://doi.org/10.1214/aos/1176351062.

Loy, Adam, and Heike Hofmann. 2014. "HLMdiag: A Suite of Diagnostics for Hierarchical Linear Models in R." *Journal of Statistical Software* 56 (5): 1–28. https://doi.org/10.18637/jss.v056.i05.

Loy, Adam, Heike Hofmann, and Dianne Cook. 2017. "Model Choice and Diagnostics for Linear Mixed-Effects Models Using Statistics on Street Corners." *Journal of Computational and Graphical Statistics* 26 (3): 478–92. https://doi.org/10.1080/10618600.2017.1330207.

Mammen, Enno. 1993. "Bootstrap and Wild Bootstrap for High Dimensional Linear Models." *Annals of Statistics* 21 (1): 255–85. https://doi.org/10.1214/aos/1176349025.

Mccullagh, Peter. 2000. "Resampling and Exchangeable Arrays." *Bernoulli* 6 (2): 285–301. https://doi.org/10.2307/3318577.

Microsoft, and Steve Weston. 2020a. *doParallel: Foreach Parallel Adaptor for the 'Parallel' Package*. https://CRAN.R-project.org/package=doParallel.

Microsoft, and Steve Weston. 2020b. *Foreach: Provides Foreach Looping Construct*. https://CRAN.R-project.org/package=foreach.

Modugno, Lucia, and Simone Giannerini. 2015. "The Wild Bootstrap for Multilevel Models." *Communications in Statistics - Theory and Methods* 44 (22): 4812–25. https://doi.org/10.1080/03610926.2013.802807.

Morris, Jeffrey S. 2002. "The BLUPs are not "best" when it comes to bootstrapping." *Statistics and Probability Letters* 56 (4): 425–30. https://doi.org/10.1016/S0167-7152(02)00041-X.

Mortimore, Peter, Pamela Sammons, Louise Stoll, and Russell Ecob. 1988. *School Matters*. University of California Press.

Nakagawa, Shinichi, and Holger Schielzeth. 2010. "Repeatability for Gaussian and Non-Gaussian Data: A Practical Guide for Biologists." *Biological Reviews of the Cambridge Philosophical Society* 85 (4): 935–56. https://doi.org/10.1111/j.1469-185X.2010.00141.x.

Pinheiro, Jose C, and Douglas M Bates. 2000. *Mixed-Effects Models in s and s-PLUS*. New York: Springer-Verlag. https://doi.org/10.1007/b98882.

Pinheiro, Jose, Douglas Bates, Saikat DebRoy, Deepayan Sarkar, and R Core Team. 2017. *nlme: Linear and Nonlinear Mixed Effects Models*. https://CRAN.R-project.org/package=nlme.

Raudenbush, Stephen W., and Anthony S. Bryk. 2002. *Hierarchical Linear Models: Applications and Data Analysis Methods*. 2nd ed. Thousand Oaks, CA: Sage Publications, Inc.

Sánchez-Espigares, José A, and Jordi Ocaña. 2009. "An R Implementation of Bootstrap Procedures for Mixed Models." Rennes, France: The R User Conference 2009. https://www.r-project.org/conferences/useR-2009/slides/SanchezEspigares+Ocana.pdf.

Scheipl, Fabian, Sonja Greven, and Helmut Kuechenhoff. 2008. "Size and Power of Tests for a Zero Random Effect Variance or Polynomial Regression in Additive and Linear Mixed Models." *Computational Statistics & Data Analysis* 52 (7): 3283–99. https://doi.org/10.1016/j.csda.2007.

10.022.
Shao, Jun, and Dongsheng Tu. 1995. *The Jackknife and Bootstrap*. Springer.
Singer, Julio M, Francisco M M Rocha, and Juvêncio S Nobre. 2017. "Graphical Tools for Detecting Departures from Linear Mixed Model Assumptions and Some Remedial Measures." *International Statistical Review = Revue Internationale de Statistique* 85 (2): 290–324. https://doi.org/10.1111/insr.12178.
Stoffel, Martin A., Shinichi Nakagawa, and Holger Schielzeth. 2017. "rptR: Repeatability Estimation and Variance Decomposition by Generalized Linear Mixed-Effects Models." *Methods in Ecology and Evolution* 8: 1639–44. https://doi.org/10.1111/2041-210X.12797.
Stram, Daniel O, and Jae Won Lee. 1994. "Variance Components Testing in the Longitudinal Mixed Effects Model." *Biometrics* 50 (4): 1171–77.
Van der Leeden, Rien, Erik Meijer, and Frank M. T. A. Busing. 2008. "Resampling Multilevel Models." In *Handbook of Multilevel Analysis*, edited by Jan de Leeuw and Erik Meijer, 401–33. Springer New York. https://doi.org/10.1007/978-0-387-73186-5_11.
Vonesh, Edward F, and Randy L Carter. 1992. "Mixed-Effects Nonlinear Regression for Unbalanced Repeated Measures." *Biometrics* 48 (1): 1–17.
Webb, Matthew D. 2013. "Reworking Wild Bootstrap Based Inference for Clustered Errors." Queen's Economics Department Working Paper. https://www.econstor.eu/handle/10419/97480.

Adam Loy
Carleton College
Northfield, MN, USA
https://aloy.rbind.io/
ORCiD: 0000-0002-5780-4611
aloy@carleton.edu

Jenna Korobova
Carleton College
Northfield, MN, USA
jenna.korobova@gmail.com

DGLMExtPois: Advances in Dealing with Over and Under-dispersion in a Double GLM Framework

by Antonio J. Sáez-Castillo, Antonio Conde-Sánchez and Francisco Martínez

Abstract In recent years the use of regression models for under-dispersed count data, such as COM-Poisson or hyper-Poisson models, has increased. In this paper the **DGLMExtPois** package is presented. **DGLMExtPois** includes a new procedure to estimate the coefficients of a hyper-Poisson regression model within a GLM framework. The estimation process uses a gradient-based algorithm to solve a nonlinear constrained optimization problem. The package also provides an implementation of the COM-Poisson model, proposed by Huang (2017), to make it easy to compare both models. The functionality of the package is illustrated by fitting a model to a real dataset. Furthermore, an experimental comparison is made with other related packages, although none of these packages allow you to fit a hyper-Poisson model.

1 Introduction

The Poisson regression model is the most common framework for modeling count data, such as the number of accidents on a stretch of road, the number of visits to the doctor, the number of recreational or shopping trips, etc (Cameron and Trivedi, 2013; Hilbe, 2011; Winkelmann, 2008). This model is constrained by its equi-dispersion assumption, that is, the mean is equal to the variance. However, this constraint is not usually met in real applications due to the existence of over-dispersion (the variance is greater than the mean) or under-dispersion (the variance is less than the mean). As over-dispersion is more common than under-dispersion, the former has received more attention from the research community.

Although the number of probabilistic models for count data affected by under-dispersion is not as high as that for over-dispersion, in recent years a significant effort has been made to provide suitable alternatives for applied researches to fit the data when the variance is below the mean. Generalized Poisson (Consul and Famoye, 1992; Wang and Famoye, 1997; Famoye et al., 2004; Zamani and Ismail, 2012; Grover et al., 2015; Sellers and Morris, 2017), Double Poisson (Efron, 1986; Zou et al., 2013; Sellers and Morris, 2017), Gamma count (Winkelmann, 1995; Oh et al., 2006; Lord et al., 2010; Daniels et al., 2010, 2011; Zeviani et al., 2014; Sellers and Morris, 2017), discrete Weibull (McShane et al., 2008; Ong et al., 2015; Klakattawi et al., 2018; Yoo, 2019) or Extended Poisson Process (Faddy and Smith, 2011; Smith and Faddy, 2016) are examples of models that may facilitate, in a regression context, a flexible dispersion structure wherein the Poisson assumption of equi-dispersion cannot be admitted because of the existence of factors that determine an excess or a shortfall of variability. In any case, Conway-Maxwell-Poisson, from now on CMP, is probably the most widely used model to fit data in the presence of under-dispersion. Sellers and Shmueli (2010) presented a formulation of CMP in a regression context and demonstrated that the model could provide good fits to data not only affected by over-dispersion but also by under-dispersion, in the presence of covariates. The number of subsequent papers applying the model in different contexts is noteworthy (Sellers et al., 2012; Sellers and Morris, 2017; Chatla and Shmueli, 2018; Abdella et al., 2019).

Nevertheless, the use of this CMP model has also been subject to criticism because of some limitations. First, there are some convergence problems in the calculation of the normalizing constant, the mean and the variance (since there are no closed expressions for them) (Francis et al., 2012). Second, and this is in our opinion its most important drawback, the original implementation of Sellers and Shmueli (2010) and, therefore, all other applications based on it, consider a log-linear relationship between the covariates and the location parameter instead of the mean, so that the regression coefficients cannot be interpreted in terms of the effect size. In fact, some authors have considered a reparameterization looking for a central tendency parameter (Guikema and Coffelt, 2008; Lord et al., 2010; Francis et al., 2012; Chanialidis et al., 2018), although it is not really the mean.

Fortunately, Huang (2017) has recently implemented a new CMP parametrization where covariates are log-linearly related to the mean and the dispersion parameter, which has been continued in subsequent works (Huang and Kim, 2019; Ribeiro et al., 2019; Forthmann et al., 2019). The **mpcmp** package is based on the work of Huang (2017). The hyper-Poisson regression model (Saez-Castillo and Conde-Sanchez, 2013; Khazraee et al., 2015; Sáez-Castillo and Conde-Sánchez, 2017), hereafter hP, provided an alternative that also presented both characteristics: on the one hand, covariates are introduced in the equation of the mean, commonly using a log-linear link, with regression

coefficients measuring the effect size; on the other hand, the dispersion parameter may also be related to the covariates to facilitate a better fit of datasets which may be affected by over- or under-dispersion. Unfortunately, although the authors provided the necessary R code for a complete fit, the computational time needed to fit real data with medium or high sample size was so huge that its usefulness in real applications was very limited.

That is why our main aim in this paper is to improve the estimation procedure of the hP model. For this purpose, we have developed an R package, called **DGLMExtPois**, in which the maximum likelihood optimization procedure has been accelerated. Of course, the package includes the common methods and functions that allow for a complete analysis, including model diagnostics. The package also provides the Huang (2017) implementation of the CMP model that admits covariates in the dispersion parameter, mainly to facilitate comparisons between hP and CMP fits. In conclusion, what we want to demonstrate is the usefulness of CMP and hP models in a genuine GLM framework to test the significance and to assess the size effect of any explanatory variable on the mean value. At the same time, other explanatory variables may take part in the explanation of the dispersion structure, providing over- or under-dispersion depending on their different values.

Comparison with other packages

There is currently no other package in R that estimates the hyper-Poisson model. There are two other packages that fit a COM-Poisson model: the **mpcmp** package that estimates the Huang's GLM implementation (Huang, 2017), named $CMP_{\mu,\nu}$, and the **COMPoissonReg** package that estimates the (Sellers and Shmueli, 2010) model, denoted as $CMP_{\theta,\nu}$. The **DGLMExtPois** package also fits the $CMP_{\mu,\nu}$ model applying an optimization process similar to the one used in fitting $hP_{\mu,\gamma}$ models. The $CMP_{\mu,\nu}$ model was included in the **DGLMExtPois** package because, at the time the package was developed, no other package estimated this model with covariates in the dispersion parameter. However, recently the **mpcmp** package has included this feature. Table 1 summarizes this comparison along with some additional features included in these packages for the analysis of the previously mentioned models.

	DGLMExtPois	**mpcmp**	**COMPoissonReg**
$hP_{\mu,\gamma}$ model	Yes	No	No
$CMP_{\mu,\nu}$ model	Yes	Yes	No
$CMP_{\theta,\nu}$ model	No	No	Yes
Covariates in dispersion parameter	Yes	Yes	Yes
Zero inflated models	No	No	Yes

Table 1: Feature comparison between packages **DGLMExtPois** (version 0.2.1), **mpcmp** (version 0.3.6) and **COMPoissonReg** (version 0.7.0).

The remainder of this paper is structured as follows. First, we theoretically describe the $hP_{\mu,\gamma}$ model and the main estimation results for fitting data. Second, we do the same for the $CMP_{\theta,\nu}$ and $CMP_{\mu,\nu}$ models. After that, we describe the **DGLMExtPois** package, illustrating the fit of a real dataset. Next, we include a comparison of fits of $hP_{\mu,\gamma}$, $CMP_{\theta,\nu}$ and $CMP_{\mu,\nu}$ models with different well-known datasets. Finally, we carry out a simulation to assess the performance of the estimates and standard errors of the $hP_{\mu,\gamma}$ model and discuss the optimization process.

2 The $hP_{\mu,\gamma}$ model

Let us consider Y as count variable of a hyper-Poisson distribution, whose probability mass function is (Saez-Castillo and Conde-Sanchez, 2013)

$$P[Y=y] = \frac{1}{{}_1F_1(1,\gamma;\lambda)}\frac{\lambda^y}{(\gamma)_y}, \qquad \lambda,\gamma>0, \tag{1}$$

where

$${}_1F_1(1,\gamma;\lambda) = \sum_{y=0}^{\infty}\frac{\lambda^y}{(\gamma)_y},$$

and $(\gamma)_y = \Gamma(\gamma+y)/\Gamma(\gamma)$. The parameter γ is known as the dispersion parameter because for $\gamma>1$ the distribution is over-dispersed and for $\gamma<1$ it is under-dispersed (for $\gamma=1$ we have the Poisson distribution).

Let $(X_1, \ldots, X_{q_1}) \in \mathbb{R}^{q_1}$ and $(Z_1, \ldots, Z_{q_2}) \in \mathbb{R}^{q_2}$ be two sets of explanatory variables or covariates. These two sets can be disjoint or overlapping (they can even be equal). In an $hP_{\mu,\gamma}$ GLM model, Y follows a hyper-Poisson distribution whose mean, μ, and dispersion parameter, γ, are functions of the covariates. In our case, log-linear relations are always considered, so $\log \mu = \mathbf{X}\boldsymbol{\beta}$ and $\log \gamma = \mathbf{Z}\boldsymbol{\delta}$, such that $\mathbf{X} = (1, X_1, \ldots, X_{q_1})$ and $\mathbf{Z} = (1, Z_1, \ldots, Z_{q_2})$ the design matrices, and $\boldsymbol{\beta}' = (\beta_0, \beta_1, \ldots, \beta_{q_1})$ and $\boldsymbol{\delta}' = (\delta_0, \delta_1, \ldots, \delta_{q_2})$ the regression coefficients. Moreover, the parameter λ is the solution of

$$\mu = \exp(\mathbf{X}\boldsymbol{\beta}) = \sum_{y=0}^{\infty} y \times P[Y = y]. \tag{2}$$

Saez-Castillo and Conde-Sanchez (2013) include a study of the relationship between $\lambda(\mu, \gamma)$, which can be interpreted as a location parameter, and the mean, μ, for a wide range of γ values.

The hP distribution belongs to the exponential family only for a given γ value. This justifies that the $hP_{\mu,\gamma}$ model can be considered as a member of the GLM family. Moreover, since the model also allows you to introduce covariates in the dispersion parameter, we refer to it as *double GLM*, although we highlight that the hP distribution does not properly belong to the exponential family for unknown γ.

The previous definition of the $hP_{\mu,\gamma}$ model (Saez-Castillo and Conde-Sanchez, 2013) was based on an alternative equation, different from (2). Concretely, the authors proposed a closed equation that involves the mean, the variance and the normalizing constant. The substitution of the previous equation by equation (2) and the change of the optimization algorithm have been the key points in the great reduction of the computational time needed to fit real data provided by the **DGLMExtPois** package.

To this end, we consider maximum likelihood for the estimation of the regression coefficients. Letting $(Y_i, X_{1i}, \ldots, X_{q_1 i}, Z_{1i}, \ldots, Z_{q_2 i})$, $i = 1, ..., n$, be a random sample, the log-likelihood is given by

$$l_{hP}(\beta, \delta) = \sum_{i=1}^{n} \{Y_i \log(\lambda(\mu_i, \gamma_i)) - \log(\Gamma(\gamma_i + Y_i)) + \log(\Gamma(\gamma_i)) - \log({}_1F_1(1, \gamma_i; \lambda(\mu_i, \gamma_i)))\} \tag{3}$$

where

$$\log \mu_i = \mathbf{x}_i'\boldsymbol{\beta}$$

and

$$\log \gamma_i = \mathbf{z}_i'\boldsymbol{\delta},$$

being $\mathbf{x}_i'$ and $\mathbf{z}_i'$ the ith row of the design matrices and each $\lambda_i = \lambda(\mu_i, \gamma_i)$ is a solution of (2) for a given μ_i and γ_i. This converts the maximum likelihood procedure into a nonlinear constrained optimization problem where the objective function is $(n + q_1 + q_2 + 2)$-dimensional, depending on n nuisance λ_i parameters, $q_1 + 1$ regression coefficients for the mean, $\boldsymbol{\beta}$, and $q_2 + 1$ regression coefficients for the dispersion parameter, $\boldsymbol{\delta}$, and it is constrained by n non-linear equations from (2). We have implemented it using a sequential quadratic programming (SQP) gradient-based algorithm (Kraft, 1994); concretely, the NLOPT_LD_SLSQP algorithm included in the **nloptr** package (Ypma et al., 2022).

In the implementation, it was necessary to compute the gradient of the objective function and the gradient of restriction (2), in which the reparameterization of $\lambda' = \log \lambda$ has been considered to ensure the positivity of λ. These expressions have been included in the appendix.

Moreover, the following results are verified:

Result 1 *For a sample $Y_1, \ldots, Y_n$ from a hyper-Poisson the score function with respect to $\boldsymbol{\beta}$ is*

$$S_{\boldsymbol{\beta}} = \frac{\partial l_{hP}}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{n} \frac{Y_i - \mu_i}{V(\mu_i, \gamma_i)} \mu_i \mathbf{x}_i' \tag{4}$$

where $V(\mu_i, \gamma_i)$ is the variance of an hP with a probability mass function given by (1), which depends on μ_i and γ_i. This way, the MLE of $\boldsymbol{\beta}$ can be characterized as the solution to the usual GLM score equations.

Result 2 *For a sample of size n from a hyper-Poisson it is verified that*

$$S_{\boldsymbol{\delta}} = \frac{\partial l_{hP}}{\partial \boldsymbol{\delta}} = \sum_{i=1}^{n} \left\{ (Y_i - \mu_i) \frac{Cov[Y_i \psi(\gamma_i + Y_i)]}{V(\mu_i, \gamma_i)} - \psi(\gamma_i + Y_i) + E[\psi(\gamma_i + Y_i)] \right\} \gamma_i \mathbf{z}_i' \tag{5}$$

where $\psi()$ is the digamma function.

The proof is included in the appendix. We have obtained estimates of the standard errors of the maximum likelihood estimators of $\boldsymbol{\beta}$ and $\boldsymbol{\delta}$ based on Cramer-Rao bound, which requires a closed expression of the Fisher information matrix.

Result 3 *The Fisher information matrix in the sample can be approximated by*

$$I_n\left(\hat{\beta},\hat{\delta}\right) = \left(\begin{array}{c|c} \sum_{i=1}^{n} E\left[S_{\hat{\beta}} S'_{\hat{\beta}}\right] & 0 \\ \hline 0 & \sum_{i=1}^{n} E\left[S_{\hat{\delta}} S'_{\hat{\delta}}\right] \end{array}\right)$$

where

$$\sum_{i=1}^{n} E\left[S_\beta S'_\beta\right] = \sum_{i=1}^{n} \frac{\mu_i^2}{V(\mu_i,\gamma_i)} \mathbf{x}_i \mathbf{x}'_i$$

$$\sum_{i=1}^{n} E\left[S_\delta S'_\delta\right] = \sum_{i=1}^{n} \left[Var\left(\psi\left(\gamma_i + Y_i\right)\right) - \frac{Cov\left(Y_i, \psi\left(\gamma_i + Y_i\right)\right)}{V(\mu_i,\gamma_i)}\right] \gamma_i^2 \mathbf{z}_i \mathbf{z}'_i.$$

are evaluated in the maximum likelihood estimates, $\widehat{\boldsymbol{\beta}}$ and $\widehat{\boldsymbol{\delta}}$. The proof is included in the appendix.

Standard error estimates would appear in the diagonal of the inverse of $I_n\left(\widehat{\boldsymbol{\beta}},\widehat{\boldsymbol{\delta}}\right)$. We have not experienced any difficulties in this computation, nor in the inversion of $\sum_{i=1}^{n} E\left[S_\beta S'_\beta\right]$. In contrast, $\sum_{i=1}^{n} E\left[S_\delta S'_\delta\right]$ is sometimes nearly singular, so the standard errors of $\widehat{\boldsymbol{\delta}}$ are not always estimated and, sometimes, are quite large. That is the reason why we recommend assessing the significance of covariates included in $\mathbf{Z}$ using the likelihood ratio test (LRT) instead of the Wald test, since it requires a convenient estimation of $\widehat{\boldsymbol{\delta}}$ standard errors.

3 The $CMP_{\theta,\nu}$ and $CMP_{\mu,\nu}$ models

The probabilities of the CMP distribution are given by

$$P[Y = y] = \frac{1}{Z(\theta,\nu)} \frac{\theta^y}{(y!)^\nu}, \qquad \theta,\nu > 0, \tag{6}$$

with

$$Z(\theta,\nu) = \sum_{y=0}^{\infty} \frac{\theta^y}{(y!)^\nu}.$$

Here, $\nu < 1$ corresponds to over-dispersion and $\nu > 1$ to under-dispersion. The distribution belongs to the bi-parametric exponential family (Huang, 2017). In this distribution, θ is a location parameter with a similar role to λ for the hP distribution. It is worth emphasizing that θ is *not* a mean, see for example Francis et al. (2012) for the exact relationship between θ and μ.

If Y is again the response count variable and $\mathbf{X}$ and $\mathbf{Z}$ are defined as above, then we denote as $CMP_{\theta,\nu}$ the model wherein

$$\log\theta = \mathbf{X}\boldsymbol{\beta}$$

and

$$\log\nu = \mathbf{Z}\boldsymbol{\delta}.$$

This is the model with constant ν defined by Sellers and Shmueli (2010), later extended to consider covariates in the ν equation (Sellers et al., 2012; Chatla and Shmueli, 2018). In contrast, the model defined by Huang (2017) is given as

$$\log\mu = \mathbf{X}\boldsymbol{\beta}$$

and

$$\log\nu = \mathbf{Z}\boldsymbol{\delta},$$

where additionally equation (2) must be verified for each observation, and the probabilities in (6). We denote this model as $CMP_{\mu,\nu}$.

Estimation and inference for the $CMP_{\theta,\nu}$ is described, for example, in Sellers and Shmueli (2010). The **COMPoissonReg** package (Sellers et al., 2018) provides adequate methods and functions to estimate such a model. Huang (2017) also provides theoretical results describing the procedure to obtain estimates of $\boldsymbol{\beta}$ and $\boldsymbol{\delta}$ using maximum-likelihood and related inference results for $CMP_{\mu,\nu}$, which are included in the **mpcmp** package.

Function	Description
`glm.hP`	Fits an $hP_{\mu,\gamma}$ model
`glm.CMP`	Fits an $CMP_{\mu,\nu}$ model
`lrtest`	Likelihood ratio chi-squared test
`residuals`	Extracts model residuals (Pearson, response and quantile) and produces a simulated envelope
`fitted`	Fitted values of the model
`plot`	Plot of residuals against fitted values and a Normal Q-Q plot
`predict`	Predictions
`confint`	Confidence intervals for $\boldsymbol{\beta}$ regression coefficients
`dhP`	Probability mass function for an hP distribution
`phP`	Distribution function for an hP distribution
`rhP`	Random generation for an hP distribution
`hP_expected`	Expected frequencies for $hP_{\mu,\gamma}$ model
`CMP_expected`	Expected frequencies for $CMP_{\mu,\nu}$ model

Table 2: Main functions in the **DGLMExtPois** package.

4 The DGLMExtPois package

At present, packages such as **COMPoissonReg** provide what a practitioner commonly requires to fit real data. Our main criticism, as we have mentioned in the introduction section, is that $\boldsymbol{\beta}$ regression coefficients cannot be interpreted in terms of the effect size. In this sense, **mpcmp** is, in our opinion, a better proposal for practitioners looking for a clearer interpretation of the $\boldsymbol{\beta}$ regression coefficients in relation to the effect of each explanatory variable on the mean of the response variable.

In this context, what the **DGLMExtPois** package provides is, first, a much faster implementation of the $hP_{\mu,\gamma}$ model than the previous existing R code and, second, our own implementation of $CMP_{\mu,\nu}$ to facilitate the comparison of the models.

DGLMExtPois has been created by trying to reproduce the syntax of GLM fits. Thus, functions `glm.hP` and `glm.CMP` provide the corresponding fits for $hP_{\mu,\gamma}$ and $CMP_{\mu,\nu}$, respectively. There are also `print` and `summary` methods to visualize the fitted models and `AIC`, `confint`, `predict`, `residuals` and `plot` methods to evaluate the goodness of fit, obtain confidence intervals of $\boldsymbol{\beta}$ regression coefficients, predictions, residuals and some associated plots, such as QQ-plots and simulated envelopes. The package additionally incorporates `expected`, a function which calculates the marginal probabilities of the Y variable, and `lrtest` to get the likelihood ratio test in nested models. Other functions are provided, for example, to work with probabilities of CMP and hP distributions. A brief description of the main functions in the package is given in Table 2.

Installation

As usual, you can install the **DGLMExtPois** package from CRAN with the code:

```
install.packages("DGLMExtPois")
```

You may encounter problems on Windows operating systems if you are not logged in with administrator rights due to the strong dependency on the **nloptr** package.

5 Numerical examples

In this section the main methods and functions in the package are illustrated by fitting a real dataset. Also, several well-known datasets in regression on count variables have been used to compare **COMPoissonReg** and **DGLMExtPois** in terms of goodness of fit, computational times and reliability. Finally, some guidelines are given on the optimization process for estimating the models.

Example of application

The dataset originates from research on customer profiles for a household supplies company. An observation corresponds to census tracts within 10-mile radius around a certain store. The response

	Poisson		$hP_{\mu,\gamma}$		$CMP_{\mu,\nu}$	
	Estimate	Std. Error	Estimate	Std. Error	Estimate	Std. Error
(Intercept)	2.9424	0.2072	2.9465	0.2155	2.9426	0.2051
nhu	0.0606	0.0142	0.0600	0.0147	0.0606	0.0141
aid	-0.0117	0.0021	-0.0115	0.0022	-0.0117	0.0021
aha	-0.0037	0.0018	-0.0038	0.0018	-0.0037	0.0018
dnc	0.1684	0.0258	0.1662	0.0269	0.1683	0.0255
ds	-0.1288	0.0162	-0.1285	0.0168	-0.1288	0.0160
(Intercept)			0.6229	0.6574	0.0219	0.1407

Table 3: Estimation of coefficients and standard errors for fitted models with a constant dispersion parameter. The last row is the estimation of δ_0 and its standard error.

variable is the number of customers of each census tract who visit the store and the explanatory variables are related to relevant demographic information for each tract. The dataset was analyzed in Neter et al. (1996) as an example of a Poisson regression model, since it is a classic instance of equidispersed count data. They consider the number of housing units (`nhu`), the average income in dollars (`aid`), the average housing unit age in years (`aha`), the distance to the nearest competitor in miles (`dnc`) and the distance to store in miles (`ds`) as covariates. Here, we consider this dataset to illustrate the performance of **DGLMExtPois** fitting models with all the explanatory variables in the mean equation and one of them, `dnc`, in the dispersion parameter equation. The dataset is also analyzed in (Bonat et al., 2018) considering the Extended Poisson-Tweedie distribution, but they also obtain an equidispersed distribution.

First, we fit a Poisson regression model, an $hP_{\mu,\gamma}$ model and a $CMP_{\mu,\nu}$ model without explanatory variables in the dispersion parameter:

```
library(DGLMExtPois)
formula.loc   <- ncust ~ nhu + aid + aha + dnc + ds # Mean formula
formula.disp0 <- ncust ~ 1                          # Dispersion parameter formula

pois <- glm(formula.loc, data = CustomerProfile, family = 'poisson')

hp0  <- glm.hP(formula.loc, formula.disp0, data = CustomerProfile)
cmp0 <- glm.CMP(formula.loc, formula.disp0, data = CustomerProfile)
c(AIC(pois), AIC(hp0), AIC(cmp0))
[1] 571.0243 571.7983 573.0006
```

The functions `glm.hP` and `glm.CMP` return S3 objects of classes `"glm_hP"` and `"glm_CMP"`, respectively, with information on the fitted models. The most suitable model according to AIC is Poisson. The fits for these models, shown in Table 3, indicate that there is equidispersion. In fact, if the LRT is carried out to determine whether the dispersion parameter is equal to 1 (Poisson model), the hypothesis is not rejected:

```
lrt_hp <- -2 * (hp0$logver + logLik(pois)[1])
pchisq(lrt_hp,1, lower.tail = FALSE)
[1] 0.2681826
lrt_cmp <- -2 * (cmp0$logver + logLik(pois)[1])
pchisq(lrt_cmp,1, lower.tail = FALSE)
[1] 0.8777006
```

However, if we introduce covariates in the dispersion parameter we find that the equidispersion hypothesis is rejected, hence the importance of being able to include variables that determine a different variability in the model. We have checked that none of the explanatory variables lead to a statistically significant improvement in the likelihood of the $CMP_{\mu,\nu}$ model when they are introduced as covariates for the dispersion parameter. That is why, from now on, we concentrate on how to use the package with the $hP_{\mu,\gamma}$ model.

Let us now consider a new $hP_{\mu,\gamma}$ model with `dnc` in the dispersion parameter equation:

```
formula.disp1 <- ncust ~ dnc        # New dispersion parameter formula
hp1           <- glm.hP(formula.loc, formula.disp1, data = CustomerProfile)
```

The summary method provides, as usual, the regression coefficient estimates, their standard errors, the corresponding t-statistics and p-values and other details about the model:

```
summary(hp1)

Call:
glm.hP(formula.mu = formula.loc, formula.gamma = formula.disp1,
    data = CustomerProfile)

Mean model coefficients (with log link):
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  2.996083   0.204212  14.671  < 2e-16 ***
nhu          0.054473   0.013960   3.902 9.54e-05 ***
aid         -0.010930   0.002068  -5.285 1.26e-07 ***
aha         -0.003631   0.001751  -2.074   0.0381 *
dnc          0.156249   0.026233   5.956 2.58e-09 ***
ds          -0.128101   0.015636  -8.193 2.55e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Dispersion model coefficients (with logit link):
            Estimate Std. Error z value Pr(>|z|)
(Intercept)    4.749      2.416   1.965   0.0494 *
dnc           -2.782      2.272  -1.224   0.2208
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

AIC: 568.1
```

The Wald test assesses whether the dnc variable is significant or not for the dispersion parameter. In any case, as previously mentioned, we also used the LRT to confirm the adequacy of hp1 versus the hp0 model, with a significance level of 0.05, resulting in the rejection of the constant dispersion hypothesis:

```
lrtest(hp0, hp1)
Statistics: 5.668851
Degrees of freedom: 1
p-value: 0.01726877
```

Since the dispersion parameter in the hp1 model depends on dnc, its values may determine over- or under-dispersion. It is interesting to note that low values of the dnc variable show over-dispersion while high values show under-dispersion. Specifically, the point at which the change occurs is 1.7069. There are 24 out of the 110 cases with $\gamma_i > 1$, and thus over-dispersed, underlying $hP(\mu_i, \gamma_i)$ distributions, see Figure 1. The plot suggests a certain correlation between under-dispersion and high response variable values.

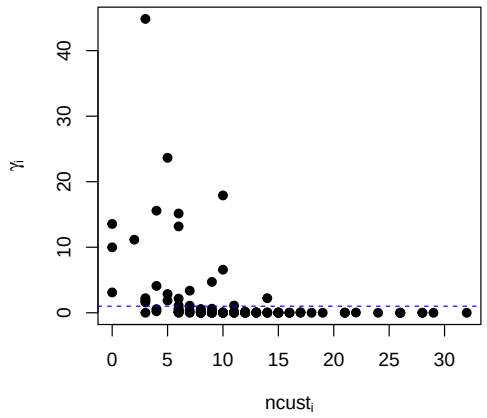
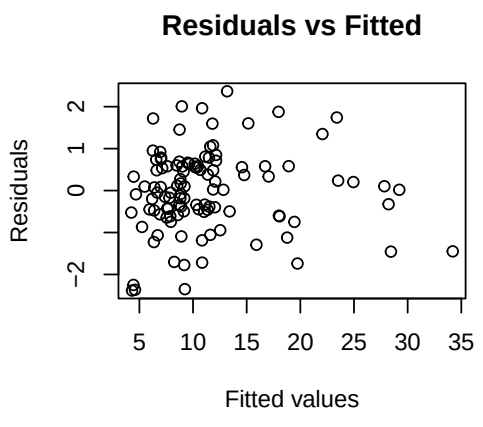
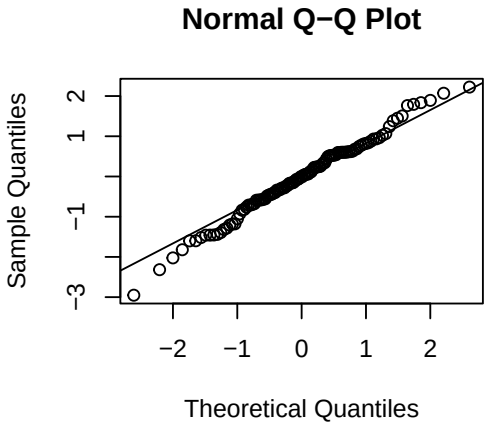


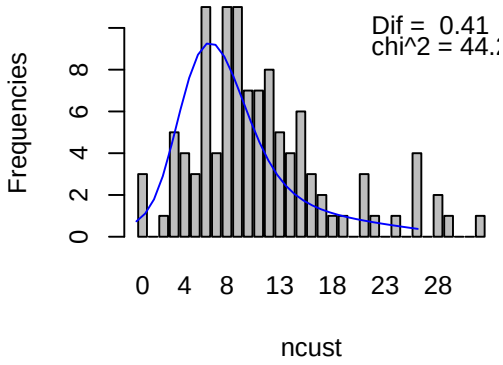
Figure 1: Value of the dispersion parameter according to the response variable. The points above the dashed line are greater than 1, so the distribution is over-dispersed. These points occur for low values of the response variable.



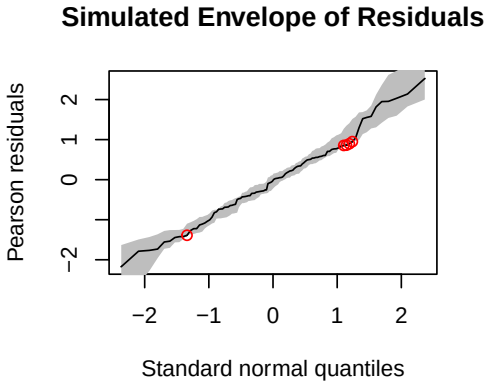
(a) Residuals versus fitted values.



(b) Normal QQ plot.



(c) Bar_plot for the observed marginal frequencies. The blue line represents the expected marginal frequencies for the response variable.



(d) Normal QQ plot with simulated envelopes of residuals. The shaded gray region contains 95% of the simulated points for each observation.

Figure 2: Regression diagnostics. The plots do not show evidence against the adequacy of the model.

```
plot(CustomerProfile$ncust, hp1$gamma,
     xlab = expression(ncust[i]),
     ylab = expression(gamma[i]),
     pch = 19
)
abline(h = 1, lty = "dashed", col = "blue")
```

The `plot` method provides two classical plots for the model diagnostics: residuals against fitted values and a normal QQ-Plot. In relation to the residuals, Pearson, deviance and response residuals are offered, although here we only calculate the Pearson type (top row of Figure 2). The package also provides the possibility to compare in a barplot the observed and expected marginal frequencies of the response variable with the `hP_expected` and the `CMP_expected` functions (Figure 2c). Finally, the diagnostic can be completed by a simulated envelope for the residuals to assess the accuracy of the model on the entire dataset (Atkinson, 1981; Garay et al., 2011). The `residuals` method provides this diagnostic tool, and in Figure 2d the cases for which the model assumptions are rejected are highlighted in red. In this case there are 5 observations that fall outside the envelope, therefore there is no evidence against the adequacy of the model.

```
par(mfrow = c(2, 2))
set.seed(21)
plot(hp1) # Residuals against fitted values and normal QQ plot

# Observed and expected marginal frequencies
expec_hp <- hP_expected(hp1)
barplot(expec_hp$observed_freq[0:33], xlab = "ncust", ylab = "Frequencies")
lines(expec_hp$frequencies[1:33], x = c(0:32), col = "blue")
text(x = c(25, 25), y = c(10, 9), c(paste("Dif = ", round(expec_hp$dif, 2)),
```

```
    paste(expression(chi ^ 2), "=", round(expec_hp$chi2, 2))), pos = 4)

# Envelope
r1 <- residuals(hp1, envelope = TRUE)
envelope_down <- apply(r1$sim_residuals, 1, min)
envelope_up <- apply(r1$sim_residuals, 1, max)
weirds <- which(r1$residuals < envelope_down | r1$residuals > envelope_up)
score_weirds <- qnorm(weirds / 111)
points(score_weirds, r1$residuals[weirds], col = "red")
```

Performance comparison

In this section eight datasets, previously used in different works, have been used to compare the performance of fitting $hP_{\mu,\gamma}$ and $CMP_{\mu,\nu}$ models with the **DGLMExtPois** package (version 0.2.1, with **nloptr** version 2.0.0), $CMP_{\theta,\nu}$ models with the **COMPoissonReg** package (version 0.7.0) and $CMP_{\mu,\nu}$ models with the **mpcmp** package (version 0.3.6). We think that the selected datasets cover a wide range of examples, with low, medium and large sample sizes, and under- and over-dispersion in their variability structure.

For each dataset we have tried to fit $hP_{\mu,\gamma}$, $CMP_{\mu,\nu}$ and $CMP_{\theta,\nu}$ models, and we have computed some goodness of fit criteria (Lord et al., 2010), such as AIC, mean prediction bias (MPB), mean absolute deviance (MAD), mean squared predictive error (MSPE) and the Pearson statistic. Additionally, we have calculated the time needed to fit the models with these R packages. The experimentation has been carried out on a computer equipped with an Intel Core i7 processor (3.20GHz) and 64 GB of RAM. Table 4 shows the AICs and the computational times obtained by the different models for the different datasets. The first column of the table lists the names of the datasets and the second column lists the number of observations of each dataset. The best AICs results are highlighted in bold. Unfortunately, the **COMPoissonReg** and **mpcmp** packages crash with some datasets, which is indicated by a dash in the table. In relation to the crashes, we clarify that all the fits have been carried out with the default optional arguments. To obtain a better approximation of the execution time needed to estimate a model, we have estimated each model 10 times and computed the average time required for the estimate. These average times are presented in Table 4 as follows. A value of 1.0 means the model has the fastest average estimate time and this fastest average time, in seconds, is shown in the last column. A value of, e.g., 41.4 means the average estimate time of the model is 41.4 times larger than the average of the fastest. Table 5 shows the results of the $hP_{\mu,\gamma}$ and $CMP_{\mu,\nu}$ models, fitted with the **DGLMExtPois** package, according to other goodness of fit criteria.

		AIC				average estimate time				
Dataset	obs.	$hP_{\mu,\gamma}$	$CMP_{\mu,\nu}$	$CMP_{\theta,\nu}$	**mpcmp**	$hP_{\mu,\gamma}$	$CMP_{\mu,\nu}$	$CMP_{\theta,\nu}$	**mpcmp**	fastest
Takeover bids	126	355.1	**354.8**	–	357.8	1.0	41.4	–	4.9	0.7
Days absent	314	1739.2	**1736.1**	–	–	1.0	5.1	–	–	23.6
Cotton bolls	125	555.1	**446.3**	466.4	447.2	1.0	53.4	10.0	11.5	0.7
Korea	162	218.2	**214.5**	–	–	1.0	6.0	–	–	2.5
Toronto	868	5088.4	**5067.4**	–	5069.4	17.6	27.6	–	1.0	9.0
Insurance	64	407.8	399.0	469.4	**394.2**	1.8	1.7	1.0	7.4	6.7
Credit card	1319	**2392.9**	–	–	–	1.0	–	–	–	1591.1
Children	1761	5380.4	**5372.4**	5393.2	5374.4	512.5	583.5	4.6	1.0	6.3

Table 4: Comparison of models based on AIC (the smallest value is highlighted in bold) and relative average estimate time. For the estimate times a value of 1.0 means the model is the fastest on average for the dataset. A value of, e.g., 41.4 means the average estimate time of the model is 41.4 times larger than the average of the fastest. The fastest time is expressed in seconds. The symbol – means that the model could not be estimated.

Next, the datasets used in the comparison are described, together with some comments about the fits for the different models:

- **Takeover bids**. The response variable is the number of bids received by 126 US firms that were successful targets of tender offers between 1978–1985. We consider 9 explanatory variables on the defensive actions taken by the management of the target firm, firm-specific characteristics and the intervention taken for both the mean and the dispersion parameter with a log-linear link. See Saez-Castillo and Conde-Sanchez (2013) for more details about these explanatory variables and other references where the dataset was analyzed. For this dataset the **COMPoissonReg** package crashes with the $CMP_{\theta,\nu}$ model. The lowest AIC is obtained by the $CMP_{\mu,\nu}$ model.

	MPB		MAD		MSPE		Pearson	
Dataset	$hP_{\mu,\gamma}$	$CMP_{\mu,\nu}$	$hP_{\mu,\gamma}$	$CMP_{\mu,\nu}$	$hP_{\mu,\gamma}$	$CMP_{\mu,\nu}$	$hP_{\mu,\gamma}$	$CMP_{\mu,\nu}$
Takeover bids	**0.0005**	-0.0834	**0.83**	0.90	**1.66**	1.99	128.4	127.4
Days absent	**0.0287**	0.0257	4.51	**4.51**	**40.20**	40.20	374.3	370.2
Cotton bolls	**0.0009**	0.0183	**0.98**	1.00	**1.60**	1.73	29.6	131.4
Korea	**-0.0052**	0.0206	0.36	**0.35**	0.24	**0.24**	119.9	–
Toronto	-0.0121	**-0.0006**	4.14	**4.14**	32.67	**32.66**	951.2	935.6
Insurance	0.0830	**-0.0254**	3.51	**3.45**	25.80	**25.33**	48.0	54.4
Credit card	**0.0156**	–	**0.66**	–	**1.66**	–	9267.8	–
Children	**-0.0000**	-0.0017	0.95	**0.95**	**1.44**	1.44	1653.1	1718.1

Table 5: Comparison of models based on additional goodness of fit criteria. MPB: mean prediction bias; MAD: mean absolute deviance; MSPE: mean squared predictive error and Pearson statistic. The best results are highlighted in bold.

The main difference between the fits of $hP_{\mu,\gamma}$ and $CMP_{\mu,\nu}$ is that the former identifies 108 out of 126 of the cases as being under-dispersed, whereas the latter identifies only 88. In general, the goodness of fit measures are quite similar, being somewhat smaller for the $hP_{\mu,\gamma}$ model. In addition, the $CMP_{\mu,\nu}$ model obtained with the **DGLMExtPois** package is slower, but with a lower AIC, than the one obtained with the **mpcmp** package.

- **Days absent**. The sample refers to 314 students sampled from two urban high schools. The response variable is the number of days absent from high school, and the explanatory variables for both the mean and the dispersion parameter are gender, math's score (standardized score out of 100) and academic program (General, Academic and Vocational). The dataset has been analyzed, among others, by Huang (2017). The **COMPoissonReg** package crashes again with the $CMP_{\theta,\nu}$ model, as does the **mpcmp** package with the $CMP_{\mu,\nu}$ model. $CMP_{\mu,\nu}$ achieves a slightly better AIC than $hP_{\mu,\gamma}$, but again requires more computational time and the goodness of fit measures are quite similar. In relation to the dispersion structure, both models identify all the cases as being over-dispersed.
- **Cotton bolls**. Here, the response variable is the number of bolls produced by 125 cotton plants. The explanatory variables, again for both the mean and the dispersion parameter, are the growth stage (vegetative, flower-bud, blossom, fig and cotton boll), and the defoliation level (with 5 levels). The model also includes the square of the defoliation level. It reproduces the analysis carried out by Zeviani et al. (2014). This is one of the cases where **COMPoissonReg** provides a $CMP_{\theta,\nu}$ fit, although the one provided by the $CMP_{\mu,\nu}$ model obtains a lower AIC (very similar to both the **DLGMExtPois** and **mpcmp** packages). Here $hP_{\mu,\gamma}$ performs worse and obtains a higher AIC. Another interesting point in relation to the $CMP_{\mu,\nu}$ fit is that, regardless of the strong under-dispersion of the complete dataset, the introduction of covariates in the dispersion parameter allows us to identify 5 cases that are actually described with over-dispersed distributions. Note also that $hP_{\mu,\gamma}$ is the fastest approach.
- **Korea crashes**. The dataset contains crash count data, which is the response variable, collected at 162 railway-highway crossings in Korea. The explanatory variables include the average daily vehicle traffic (in logarithmic scale) and 7 characteristics of the crossing. See, for example, Khazraee et al. (2015) for more details. For this dataset both the **COMPoissonReg** and **mpcmp** packages crash. The $CMP_{\mu,\nu}$ fit obtains the best AIC, although the rest of the goodness of fit measures are quite similar to those provided by the $hP_{\mu,\gamma}$ fit. Once the effect of the covariates is taken into account, both fits detect over 50 cases with under-dispersed underlying distributions.
- **Toronto crashes**. The dataset provides Toronto intersection crash data. The response variable is the number of crashes at 868 intersections, and the explanatory variables for the mean and the dispersion parameter are the average annual daily traffic in the major and minor approaches to the intersection. These data have been widely used as an example of over-dispersed crash data. See, for example, Khazraee et al. (2015). Whereas **COMPoissonReg** once again leads to a crash in the fit of a $CMP_{\theta,\nu}$ model, $CMP_{\mu,\nu}$ and $hP_{\mu,\gamma}$ provide similar fits in terms of goodness of fit measures. The $CMP_{\mu,\nu}$ model obtained with the **DGLMExtPois** package has a slightly lower AIC than the model obtained with the **mpcmp** package. In this case, the **mpcmp** package is the fastest estimating the model. There are 2 under-dispersed cases for the $hP_{\mu,\gamma}$ fit.
- **Insurance claims**. Data consist of the numbers of policyholders of an insurance company who were exposed to risk, and the number of car insurance claims made by those policyholders in the third quarter of 1973. The response variable is the number of claims, with the number of policyholders being an offset for the mean. Apart from this offset, there are 3 explanatory

variables for the mean and the dispersion parameter: district of residence of the policyholder, car group and an ordered factor with the age of the insured in 4 groups. The $CMP_{\mu,\nu}$ model of the **mpcmp** package obtains the best AIC, but it is the slowest in terms of computational time. On this occasion the **COMPoissonReg** package does not fail and is the fastest, however its AIC is clearly the highest among the estimated models. Most of the cases show under-dispersion (52 and 44 for $hP_{\mu,\gamma}$ and $CMP_{\mu,\nu}$ respectively).

- **Credit card reports**. The response variable is the number of major derogatory reports. Three explanatory variables are considered: age in years plus twelfths of a year, yearly income (in USD 10,000) and average monthly credit card expenditure. For this dataset the only model that can be estimated is $hP_{\mu,\gamma}$. This is a case of over-dispersed data, but as a result of introducing covariates in the dispersion parameter, 5 under-dispersed cases appear in the $hP_{\mu,\gamma}$ model. In any case a negative binomial model fits better due to the strong over-dispersion of data.
- **Children**. In this dataset the response variable is the number of children a woman has. The explanatory variables are: age of the woman in years, years of education, nationality of the woman (German or not), belief in God (with 6 categories) and attended university (yes or no). This is an example of equidispersed data (Tutz, 2011), although our models detected under-dispersion (1164 and 1238 cases for $hP_{\mu,\gamma}$ and $CMP_{\mu,\nu}$ respectively). In this case the **mpcmp** package provides a very fast $CMP_{\mu,\nu}$ fit, but with a slightly higher AIC.

In summary, the $hP_{\mu,\gamma}$ model is the only one that can provide fits for all the datasets. Regarding the AIC, the $CMP_{\mu,\nu}$ model seems to be better than the $hP_{\mu,\gamma}$ model, although the latter usually requires less estimation time. As for the comparison of the $CMP_{\mu,\nu}$ model in the **DGLMExtPois** and **mpcmp** packages, the $CMP_{\mu,\nu}$ model of the **mpcmp** package fails in more datasets and provides better results in terms of AIC for only one dataset.

Simulation

In order to assess the performance of the estimates and standard errors in finite samples of the $hP_{\mu,\gamma}$ model a simulation has been performed. The simulation process consists of the following steps:

1. 500 values of two covariates X and Z are generated from a uniform distribution on the interval $(0,1)$.
2. The mean and γ are obtained as $\mu_i = \exp\{\beta_0 + \beta_1 x_i\}$ and $\gamma_i = \exp\{\delta_0 + \delta_1 z_i\}$, $i = 1,\ldots,500$, for several values of the regression coefficients $\beta_0, \beta_1, \delta_0$ and δ_1.
3. The λ parameter is obtained as solution of (2) by means of the base R `optimize` function.
4. Each value of Y in the sample is randomly generated using the `rhP` function.
5. Once the dataset (Y, X, Z) is available, the coefficients of the $hP_{\mu,\gamma}$ model are estimated.

This process was repeated 1000 times and the average of the estimates and standard errors, as well as the standard deviation of the estimates, were calculated. The standard deviation of the estimates allows us to check whether the standard errors have been calculated correctly, while the average of the estimates allows us to determine whether there is a bias in the estimation of the coefficients. The results obtained for different values of the coefficients are shown in Table 6.

	β_0			β_1		
Coefficients	mean	sd	s.e.	mean	sd	s.e.
Comb. I: $\boldsymbol{\beta} = (1, 0.5)$	0.998	0.0504	0.0507	0.501	0.0827	0.0829
Comb. II: $\boldsymbol{\beta} = (1, 0.5)$	0.998	0.0552	0.0555	0.501	0.0902	0.0904
Comb. III: $\boldsymbol{\beta} = (1, 0.5)$	0.998	0.0490	0.0492	0.501	0.0805	0.0805
	δ_0			δ_1		
Coefficients	mean	sd	s.e.	mean	sd	s.e.
Comb. I: $\boldsymbol{\delta} = (0.5, -1)$	0.461	0.4237	0.4232	-1.041	0.8408	0.8164
Comb. II: $\boldsymbol{\delta} = (0.5, 0.25)$	0.456	0.3834	0.3839	0.273	0.6625	0.6567
Comb. III: $\boldsymbol{\delta} = (0, -0.5)$	-0.058	0.4943	0.4866	-0.531	0.9329	0.8980

Table 6: Simulation process for $hP_{\mu,\gamma}$ models. For each combination of coefficients the mean and standard deviation (sd) of the estimates were calculated, as well as the mean of the standard errors (s.e.).

The first combination of coefficients ($\boldsymbol{\beta} = (1, 0.5)$ and $\boldsymbol{\delta} = (0.5, -1)$) allows for under- and over-dispersion situations. The coefficients in the second combination ($\boldsymbol{\beta} = (1, 0.5)$ and $\boldsymbol{\delta} = (0.5, 0.25)$) determine a model with over-dispersion only. And finally, the third combination of coefficients ($\boldsymbol{\beta} = (1, 0.5)$ and $\boldsymbol{\delta} = (0, -0.5)$) corresponds to an under-dispersed model. It can be seen that the estimates of the $\boldsymbol{\beta}$ coefficients are practically unbiased and very precise, while those of the $\boldsymbol{\delta}$ coefficients are more biased and much less precise. In any case the mean standard errors are very close to the standard deviations of the 1000 estimates, so the calculation of these standard errors seem to be correct.

A similar simulation has been performed for the $CMP_{\mu,\nu}$ model in the **DGLMExtPois** package, and the results are shown in Table 7. It can be seen that, although the estimates of the $\boldsymbol{\delta}$ coefficients are still less precise than those of the $\boldsymbol{\beta}$ coefficients, these estimates are less biased and more precise than those of the $hP_{\mu,\gamma}$ model. In this sense, the $CMP_{\mu,\nu}$ model leads to better estimates than the hyper-Poisson model.

	β_0			β_1		
Coefficients	mean	sd	s.e.	mean	sd	s.e.
Comb. I: $\boldsymbol{\beta} = (1, 0.5)$	0.998	0.0497	0.0498	0.502	0.0818	0.0812
Comb. II: $\boldsymbol{\beta} = (1, 0.5)$	0.999	0.0383	0.0386	0.501	0.0626	0.0627
Comb. III: $\boldsymbol{\beta} = (1, 0.5)$	0.998	0.0557	0.0556	0.502	0.0909	0.0911
	δ_0			δ_1		
Coefficients	mean	sd	s.e.	mean	sd	s.e.
Comb. I: $\boldsymbol{\delta} = (0.5, -1)$	0.515	0.1415	0.1415	-1.0132	0.2640	0.2641
Comb. II: $\boldsymbol{\delta} = (0.5, 0.25)$	0.513	0.1349	0.1343	0.2426	0.2362	0.2319
Comb. III: $\boldsymbol{\delta} = (0, -0.5)$	0.012	0.1526	0.1539	-0.5085	0.2807	0.2800

Table 7: Simulation for $CMP_{\mu,\nu}$ models. For each combination of coefficients the mean and standard deviation (sd) of the estimates were calculated, as well as the mean of the standard errors (s.e.).

The optimization process

As previously mentioned, the estimation of $hP_{\mu,\gamma}$ and $CMP_{\mu,\nu}$ models in the **DGLMExtPois** package was performed using the SLSQP optimizer included in the **nloptr** package. Because the SLSQP algorithm uses dense-matrix methods, it requires $O(m^2)$ storage and $O(m^3)$ time in m dimensions (where m is the number of estimated parameters, i.e., the number of observations plus the number of model coefficients), which makes it less practical for optimizing more than a few thousand parameters. Table 8 shows the datasets from the above performance comparison sorted by the dimension of the objective function when fitting an $hP_{\mu,\gamma}$ model. Also, the average estimation time for 10 estimations and the number of iterations taken by the optimizer are listed. Clearly, the dimension of the objective function has the greatest impact on optimization time, while the number of iterations plays a less important role.

Dataset	dimension	average time	iterations
Insurance	84	12.0	987
Takeover bids	146	0.7	78
Cotton bolls	155	0.7	96
Korea	180	2.5	283
Days absent	324	23.6	558
Toronto	874	158.8	111
Credit card	1327	1591.1	135
Children	1781	3221.0	61

Table 8: Optimization process for $hP_{\mu,\gamma}$ models. For each dataset, the dimension of the objective function, the average optimization time (in seconds) and the number of iterations are shown.

The `nloptr` optimization function from the **nloptr** package was used to fit the models. This function allows several termination conditions for the optimization process, of which the **DGLMExtPois** package uses, by default, the following:

- A maximum number of iterations of 1000 and,
- a fractional tolerance (`xtol_rel`) is used in the parameters x, so that the optimizer stops when $|\Delta x_i|/|x_i| < xtol_rel$, with a default value for `xtol_rel` of 0.01.

Information on the optimization process can be obtained through the component `nloptr`, of class `"nloptr"`, included in the S3 object returned by the `glm.hP` function:

```
library(mpcmp)
formula <- daysabs ~ gender + math + prog
fit <- glm.hP(formula, formula, data = attendance)
fit$nloptr$message # termination condition
[1] "NLOPT_XTOL_REACHED: Optimization stopped because xtol_rel or xtol_abs (above) was
reached."
fit$nloptr$iterations # number of iterations
[1] 558
```

The user can modify the termination conditions. For example, in the following code the second call to `glm.hP` removes the termination condition on fractional tolerance used in the parameters and set the maximum number of iterations:

```
fit1 <- glm.hP(formula, formula, data = attendance) # default termination conditions
AIC(fit1)
[1] 1739.192
fit1$nloptr$iterations
[1] 558
fit2 <- glm.hP(formula, formula, data = attendance,
               opts = list(xtol_rel = -1, maxeval = 250))
AIC(fit2)
[1] 1739.369
fit2$nloptr$iterations
[1] 250
```

Another interesting termination condition is an approximate maximum optimization time, expressed in seconds:

```
fit3 <- glm.hP(formula, formula, data = attendance,
               opts = list(xtol_rel = -1, maxeval = -1, maxtime = 5))
AIC(fit3)
[1] 1741.672
fit3$nloptr$iterations
[1] 124
```

The user can also see how the optimization process evolves:

```
fit4 <- glm.hP(formula, formula, data = attendance,
               opts = list(xtol_rel = -1, maxeval = 5, print_level = 1))
iteration: 1
        f(x) = 1550.509229
iteration: 2
        f(x) = -0.000000
iteration: 3
        f(x) = 33443.986687
iteration: 4
        f(x) = 2980.009033
iteration: 5
        f(x) = 1444.775141
```

6 Conclusion

In this paper we have presented **DGLMExtPois**, the only package on CRAN that allows fitting count data using the hyper-Poisson model. This model is able to fit data with over- or under-dispersion, or both for different levels of covariates. Additionally, **DGLMExtPois** can also fit COM-Poisson models, considering covariates in the mean and the dispersion parameter. The **mpcmp** package also presents this feature, so the results obtained by both packages can be compared. We have found that the **mpcmp**

package fails to fit some datasets and the fits obtained are, in general, somewhat worse than those obtained with the **DGLMExtPois** package. The **COMPoissonReg** package allows covariates in the location parameter, but it models a location parameter instead of the mean. In addition, this latter package fails to fit many of the datasets used in the experimentation.

The two models implemented by the **DGLMExtPois** package achieved similar results in the experimentation, with the COM-Poisson model obtaining, in general, slightly lower AICs and needing more computational time to fit the models. We have also verified, by means of a simulation, that the COM-Poisson model is more accurate in estimating the coefficients of the covariates included in the dispersion parameter. But, overall, it cannot be said that one model is clearly better than the other. Thus, two competitive models are available to deal with count data. As shown in the example, the possibility of including covariates in the dispersion parameter is suitable for situations where there may be both over- and under-dispersion for different levels of the covariates, which makes these models especially attractive compared to those where the dispersion is constant. In fact, in the example with a constant dispersion parameter, equidispersion was obtained, whereas if covariates are considered in the dispersion parameter over- or under-dispersion can be found.

It should also be noted that the algorithm used to estimate the parameters of the hyper-Poisson model is much faster than the previous algorithms (Huang, 2017; Saez-Castillo and Conde-Sanchez, 2013).

7 Acknowledgment

We would like to thank the reviewers; their comments helped improve and clarify this manuscript and the package. In memory of our friend Antonio José, this work would not have been possible without your talent and enthusiasm.

Funding: This work was supported by MCIN/AEI/10.13039/501100011033 [project PID2019-107793GB-I00].

Bibliography

G. M. Abdella, J. Kim, K. N. Al-Khalifa, and A. M. Hamouda. Penalized Conway-Maxwell-Poisson regression for modelling dispersed discrete data: The case study of motor vehicle crash frequency. *Safety Science*, 120:157–163, 2019. URL https://doi.org/10.1016/j.ssci.2019.06.036. [p121]

A. C. Atkinson. Two graphical displays for outlying and influential observations in regression. *Biometrika*, 68(1):13–20, 1981. URL https://doi.org/10.1093/biomet/68.1.13. [p128]

W. H. Bonat, B. Jørgensen, C. C. Kokonendji, J. Hinde, and C. G. B. Demétrio. Extended Poisson–Tweedie: Properties and regression models for count data. *Statistical Modelling*, 18(1):24–49, 2018. URL https://doi.org/10.1177/1471082X17715718. [p126]

A. C. Cameron and P. K. Trivedi. *Regression Analysis of Count Data*. Econometric Society Monographs. Cambridge University Press, 2 edition, 2013. URL https://doi.org/10.1017/CBO9781139013567. [p121]

C. Chanialidis, L. Evers, T. Neocleous, and A. Nobile. Efficient Bayesian inference for COM-Poisson regression models. *Statistics and Computing*, 28(3):595–608, 2018. URL https://doi.org/10.1007/s11222-017-9750-x. [p121]

S. B. Chatla and G. Shmueli. Efficient estimation of COM–Poisson regression and a generalized additive model. *Computational Statistics & Data Analysis*, 121:71–88, 2018. URL https://doi.org/10.1016/j.csda.2017.11.011. [p121, 124]

P. Consul and F. Famoye. Generalized Poisson Regression-Model. *Communications in Statistics - Theory and Methods*, 21(1):89–109, 1992. URL https://doi.org/10.1080/03610929208830766. [p121]

S. Daniels, T. Brijs, E. Nuyts, and G. Wets. Explaining variation in safety performance of roundabouts. *Accident Analysis & Prevention*, 42(2):393–402, 2010. URL https://doi.org/10.1016/j.aap.2009.08.019. [p121]

S. Daniels, T. Brijs, E. Nuyts, and G. Wets. Extended prediction models for crashes at roundabouts. *Safety Science*, 49:198–207, Sept. 2011. URL http://dx.doi.org/10.1016/j.ssci.2010.07.016. [p121]

B. Efron. Double Exponential-Families And Their Use In Generalized Linear-Regression. *Journal of the American Statistical Association*, 81(395):709–721, 1986. URL https://doi.org/10.2307/2289002. [p121]

M. J. Faddy and D. M. Smith. Analysis of count data with covariate dependence in both mean and variance. *Journal of Applied Statistics*, 38(12):2683–2694, 2011. URL https://doi.org/10.1080/02664763.2011.567250. [p121]

F. Famoye, J. Wulu, and K. Singh. On the generalized Poisson regression model with an application to accident data. *Journal of Data Science*, 2(3):287–295, 2004. URL https://doi.org/10.6339/JDS.2004.02(3).167. [p121]

B. Forthmann, D. Gühne, and P. Doebler. Revisiting dispersion in count data item response theory models: The Conway–Maxwell–Poisson counts model. *British Journal of Mathematical and Statistical Psychology*, 2019. URL https://doi.org/10.1111/bmsp.12184. [p121]

R. Francis, S. Geedipally, S. D Guikema, S. Dhavala, D. Lord, and S. Larocca. Characterizing the Performance of the Conway-Maxwell Poisson Generalized Linear Model. *Risk analysis : an official publication of the Society for Risk Analysis*, 32(1):167–183, 2012. URL https://doi.org/10.1111/j.1539-6924.2011.01659.x. [p121, 124]

A. M. Garay, E. M. Hashimoto, E. M. Ortega, and V. H. Lachos. On estimation and influence diagnostics for zero-inflated negative binomial regression models. *Computational Statistics & Data Analysis*, 55 (3):1304–1318, 2011. URL https://doi.org/10.1016/j.csda.2010.09.019. [p128]

G. Grover, R. Vajala, and P. K. Swain. On the assessment of various factors effecting the improvement in CD4 count of aids patients undergoing antiretroviral therapy using generalized Poisson regression. *Journal of Applied Statistics*, 42(6):1291–1305, 2015. URL https://doi.org/10.1080/02664763.2014.999649. [p121]

S. D. Guikema and J. P. Coffelt. A flexible count data regression model for risk analysis. *Risk Anal*, 28 (1):213–23, 2008. URL https://doi.org/10.1111/j.1539-6924.2008.01014.x. [p121]

J. M. Hilbe. *Negative Binomial Regression*. Cambridge University Press, 2 edition, 2011. URL https://doi.org/10.1017/CBO9780511973420. [p121]

A. Huang. Mean-parametrized Conway–Maxwell–Poisson regression models for dispersed counts. *Statistical Modelling*, 17(6):359–380, 2017. URL https://doi.org/10.1177/1471082X17697749. [p121, 122, 124, 130, 134]

A. Huang and A. S. I. Kim. Bayesian Conway–Maxwell–Poisson regression models for overdispersed and underdispersed counts. *Communications in Statistics - Theory and Methods*, 2019. URL https://doi.org/10.1080/03610926.2019.1682162. [p121]

S. H. Khazraee, A. J. Saez-Castillo, S. R. Geedipally, and D. Lord. Application of the Hyper-Poisson Generalized Linear Model for Analyzing Motor Vehicle Crashes. *Risk Analysis*, 35(5):919–930, 2015. URL https://doi.org/10.1111/risa.12296. [p121, 130]

H. S. Klakattawi, V. Vinciotti, and K. Yu. A Simple and Adaptive Dispersion Regression Model for Count Data. *Entropy*, 20(2), 2018. URL https://doi.org/10.3390/e20020142. [p121]

D. Kraft. Algorithm 733: TOMP–Fortran modules for optimal control calculations. *ACM Transactions on Mathematical Software (TOMS)*, 20(3):262–281, 1994. URL https://doi.org/10.1145/192115.192124. [p123]

D. Lord, S. R. Geedipally, and S. D. Guikema. Extension of the Application of Conway-Maxwell-Poisson Models: Analyzing Traffic Crash Data Exhibiting Underdispersion. *Risk Anal*, 30(8):1268–76, 2010. URL https://doi.org/10.1111/j.1539-6924.2010.01417.x. [p121, 129]

B. McShane, M. Adrian, E. T. Bradlow, and P. S. Fader. Count Models Based on Weibull Interarrival Times. *Journal of Business & Economic Statistics*, 26(3):369–378, 2008. URL https://doi.org/10.1198/073500107000000278. [p121]

J. Neter, M. H. Kutner, C. J. Nachtsheim, and W. Wasserman. *Applied Linear Statistical Models*. Irwin, Chicago, 1996. [p126]

J. Oh, S. P. Washington, and D. Nam. Accident prediction model for railway-highway interfaces. *Accid Anal Prev*, 38(2):346–56, 2006. URL https://doi.org/10.1016/j.aap.2005.10.004. [p121]

S. H. Ong, A. Biswas, S. Peiris, and Y. C. Low. Count Distribution for Generalized Weibull Duration with Applications. *Communications in Statistics - Theory and Methods*, 44(19):4203–4216, 2015. URL https://doi.org/10.1080/03610926.2015.1062105. [p121]

J. E. E. Ribeiro, W. M. Zeviani, W. H. Bonat, C. G. Demetrio, and J. Hinde. Reparametrization of COM–Poisson regression models with applications in the analysis of experimental data. *Statistical Modelling*, 2019. URL https://doi.org/10.1177/1471082X19838651. [p121]

A. Saez-Castillo and A. Conde-Sanchez. A hyper-Poisson regression model for overdispersed and underdispersed count data. *Computational Statistics & Data Analysis*, 61:148–157, 2013. URL https://doi.org/10.1016/j.csda.2012.12.009. [p121, 122, 123, 129, 134]

A. J. Sáez-Castillo and A. Conde-Sánchez. Detecting over- and under-dispersion in zero inflated data with the hyper-Poisson regression model. *Statistical Papers*, 58(1):19–33, 2017. URL https://doi.org/10.1007/s00362-015-0683-1. [p121]

K. Sellers, T. Lotze, and A. Raim. *COMPoissonReg: Conway-Maxwell Poisson (COM-Poisson) Regression*, 2018. URL https://CRAN.R-project.org/package=COMPoissonReg. R package version 0.7.0. [p124]

K. F. Sellers and D. S. Morris. Underdispersion models: Models that are "under the radar". *Communications in Statistics - Theory and Methods*, 46(24):12075–12086, 2017. URL https://doi.org/10.1080/03610926.2017.1291976. [p121]

K. F. Sellers and G. Shmueli. A flexible regression model for count data. *The Annals of Applied Statistics*, 4(2):943–961, 2010. URL https://doi.org/10.1214/09-AOAS306. [p121, 122, 124]

K. F. Sellers, S. Borle, and G. Shmueli. The COM-Poisson model for count data: a survey of methods and applications. *Applied Stochastic Models in Business and Industry*, 28(2):104–116, 2012. URL https://doi.org/10.1002/asmb.918. [p121, 124]

D. Smith and M. Faddy. Mean and Variance Modeling of Under- and Overdispersed Count Data. *Journal of Statistical Software, Articles*, 69(6):1–23, 2016. URL https://doi.org/10.18637/jss.v069.i06. [p121]

G. Tutz. *Regression for Categorical Data*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2011. URL https://doi.org/10.1017/CBO9780511842061. [p131]

W. Wang and F. Famoye. Modeling household fertility decisions with generalized Poisson regression. *Journal of Population Economics*, 10(3):273–283, 1997. URL https://doi.org/10.1007/s001480050043. [p121]

R. Winkelmann. Duration Dependence And Dispersion In Count-Data Models. *Journal of Business & Economic Statistics*, 13(4):467–474, 1995. URL https://doi.org/10.1080/07350015.1995.10524620. [p121]

R. Winkelmann. *Econometric Analysis of Count Data*. Springer, 5 edition, 2008. [p121]

H. Yoo. Application of discrete Weibull regression model with multiple imputation. *Communications for Statistical Applications and Methods*, 26:325–336, 2019. URL https://doi.org/10.29220/CSAM.2019.26.3.325. [p121]

J. Ypma, S. G. Johnson, H. W. Borchers, D. Eddelbuettel, B. Ripley, K. Hornik, J. Chiquet, A. Adler, X. Dai, A. Stamm, and J. Ooms. *nloptr: R Interface to NLopt Regression*, 2022. URL https://CRAN.R-project.org/package=nloptr. R package version 2.0.0. [p123]

H. Zamani and N. Ismail. Functional Form for the Generalized Poisson Regression Model. *Communications in Statistics - Theory and Methods*, 41(20):3666–3675, 2012. URL https://doi.org/10.1080/03610926.2011.564742. [p121]

W. M. Zeviani, P. J. R. Jr, W. H. Bonat, S. E. Shimakura, and J. A. Muniz. The Gamma-count distribution in the analysis of experimental underdispersed data. *Journal of Applied Statistics*, 41(12):2616–2626, 2014. URL https://doi.org/10.1080/02664763.2014.922168. [p121, 130]

Y. Zou, S. R. Geedipally, and D. Lord. Evaluating the double Poisson generalized linear model. *Accident Analysis & Prevention*, 59:497–505, 2013. URL https://doi.org/10.1016/j.aap.2013.07.017. [p121]

Antonio J. Sáez-Castillo
University of Jaén
Statistics and Operations Research Department
Spain
ajsaez@ujaen.es

Antonio Conde-Sánchez
University of Jaén
Statistics and Operations Research Department
Spain
aconde@ujaen.es

Francisco Martínez
University of Jaén
Department of Computer Science, Andalusian Research Institute in Data Science and Computational Intelligence (DaSCI)
Spain
fmartin@ujaen.es

Appendix

Preliminaries

Firstly, given that

$$ {}_1F_1\left(1,\gamma;\lambda\right)=\sum_{y=0}^{\infty}\frac{\lambda^y}{\left(\gamma\right)_y}, $$

it is verified that

$$ \frac{\partial}{\partial\lambda}\,{}_1F_1\left(1,\gamma;\lambda\right)=\sum_{y=0}^{\infty}\frac{y\lambda^{y-1}}{\left(\gamma\right)_y}=\frac{1}{\lambda}\sum_{y=0}^{\infty}y\frac{\lambda^y}{\left(\gamma\right)_y}=\,{}_1F_1\left(1,\gamma;\lambda\right)\frac{\mu}{\lambda} $$

and therefore

$$ \frac{\partial}{\partial\lambda}\log\left({}_1F_1\left(1,\gamma;\lambda\right)\right)=\frac{1}{{}_1F_1\left(1,\gamma;\lambda\right)}\frac{\partial}{\partial\lambda}\,{}_1F_1\left(1,\gamma;\lambda\right)=\frac{\mu}{\lambda}. \tag{7} $$

Furthermore, since

$$ \begin{aligned}\frac{\partial\left(\gamma\right)_y}{\partial\gamma}=&\frac{\partial}{\partial\gamma}\frac{\Gamma\left(\gamma+y\right)}{\Gamma\left(\gamma\right)}=\frac{\Gamma\left(\gamma+y\right)\psi\left(\gamma+y\right)\Gamma\left(\gamma\right)-\Gamma\left(\gamma+y\right)\Gamma\left(\gamma\right)\psi\left(\gamma\right)}{\Gamma\left(\gamma\right)^2}\\=&\frac{\Gamma\left(\gamma+y\right)}{\Gamma\left(\gamma\right)}\left(\psi\left(\gamma+y\right)-\psi\left(\gamma\right)\right)=\left(\gamma\right)_y\left(\psi\left(\gamma+y\right)-\psi\left(\gamma\right)\right),\end{aligned} \tag{8} $$

where we have used

$$ \frac{\partial\Gamma\left(\gamma\right)}{\partial\gamma}=\Gamma\left(\gamma\right)\psi\left(\gamma\right). $$

Thus, you get that

$$ \begin{aligned}\frac{\partial}{\partial\gamma}\,{}_1F_1\left(1,\gamma;\lambda\right)=&\sum_{y=0}^{\infty}\frac{y\lambda^{y-1}\frac{\partial\lambda}{\partial\gamma}\left(\gamma\right)_y-\lambda^y\frac{\partial\left(\gamma\right)_y}{\partial\gamma}}{\left(\gamma\right)_y^2}=\frac{1}{\lambda}\frac{\partial\lambda}{\partial\gamma}\sum_{y=0}^{\infty}y\frac{\lambda^y}{\left(\gamma\right)_y}-\sum_{y=0}^{\infty}\frac{\lambda^y}{\left(\gamma\right)_y}\left(\psi\left(\gamma+y\right)-\psi\left(\gamma\right)\right)\\=&\,{}_1F_1\left(1,\gamma;\lambda\right)\left\{\frac{\mu}{\lambda}\frac{\partial\lambda}{\partial\gamma}-E\left[\psi\left(\gamma+Y\right)\right]+\psi\left(\gamma\right)\right\},\end{aligned} $$

so

$$ \frac{\partial}{\partial\gamma}\log\left({}_1F_1\left(1,\gamma;\lambda\right)\right)=\frac{1}{{}_1F_1\left(1,\gamma;\lambda\right)}\frac{\partial}{\partial\gamma}\,{}_1F_1\left(1,\gamma;\lambda\right)=\frac{\mu}{\lambda}\frac{\partial\lambda}{\partial\gamma}-E\left[\psi\left(\gamma+Y\right)\right]+\psi\left(\gamma\right). \tag{9} $$

Proof of result 1

Using the chain rule:

$$ \frac{\partial l_{hP}}{\partial\boldsymbol{\beta}}=\frac{\partial l_{hP}}{\partial\mu_i}\frac{\partial\mu_i}{\partial\boldsymbol{\beta}}, $$

so that the score function for μ_i is:

$$\frac{\partial l_{hP}}{\partial \mu_i} = \frac{\partial l_{hP}}{\partial \lambda_i}\frac{\partial \lambda_i}{\partial \mu_i}.$$

Let us start with the first of the previous derivatives:

$$\frac{\partial l_{hP}}{\partial \lambda_i} = \sum_{i=1}^{n}\left\{\frac{Y_i}{\lambda_i} - \frac{\partial}{\partial \lambda_i}\log\left({}_1F_1\left(1,\gamma_i;\lambda_i\right)\right)\right\} = \sum_{i=1}^{n}\frac{Y_i-\mu_i}{\lambda_i}, \tag{10}$$

where the result (7) has been used.

The second of the above derivatives is obtained using implicit differentiation, taking into account that $\lambda(\mu_i,\gamma_i)$ is solution of equation (2). Thus,

$$0 = \sum_{y=0}^{\infty}\left(y-\mu\right)\frac{\lambda^y}{(\gamma)_y},$$

wherein subscript i has been omitted, and by differentiating with respect to μ

$$\begin{aligned}0 &= -\sum_{y=0}^{\infty}\frac{\lambda^y}{(\gamma)_y} + \sum_{y=0}^{\infty}\left(y-\mu\right)y\frac{\lambda^{y-1}}{(\gamma)_y}\frac{\partial\lambda}{\partial\mu} = -{}_1F_1\left(1,\gamma;\lambda\right) + \frac{1}{\lambda}\left[\sum_{y=0}^{\infty}\left(y-\mu\right)y\frac{\lambda^y}{(\gamma)_y}\right]\frac{\partial\lambda}{\partial\mu}\\ &= -{}_1F_1\left(1,\gamma;\lambda\right) + \frac{1}{\lambda}\left[\sum_{y=0}^{\infty}\left(y-\mu\right)^2\frac{\lambda^y}{(\gamma)_y}\right]\frac{\partial\lambda}{\partial\mu} = -{}_1F_1\left(1,\gamma;\lambda\right) + \frac{1}{\lambda}\,{}_1F_1\left(1,\gamma;\lambda\right)V\left(\mu,\gamma\right)\frac{\partial\lambda}{\partial\mu},\end{aligned}$$

and clearing

$$\frac{\partial\lambda}{\partial\mu} = \frac{\lambda}{V\left(\mu,\gamma\right)}. \tag{11}$$

Therefore, plugging (10) and (11) into the score function, we obtain

$$\frac{\partial l_{hP}}{\partial \mu_i} = \sum_{i=1}^{n}\frac{Y_i-\mu_i}{\lambda_i}\frac{\lambda_i}{V\left(\mu_i,\gamma_i\right)} = \sum_{i=1}^{n}\frac{Y_i-\mu_i}{V\left(\mu_i,\gamma_i\right)}. \tag{12}$$

The only thing left to consider to achieve expression (4) is that

$$\frac{\partial\mu_i}{\partial\boldsymbol{\beta}} = \mu_i\mathbf{x}_i'.$$

Proof of result 2

Using the chain rule we get

$$\frac{\partial l_{hP}}{\partial\delta} = \frac{\partial l_{hP}}{\partial\gamma_i}\frac{\partial\gamma_i}{\partial\delta}.$$

We develop the first of these derivatives (score for γ)

$$\begin{aligned}\frac{\partial l_{hP}}{\partial\gamma_i} &= \sum_{i=1}^{n}\left\{\frac{Y_i}{\lambda_i}\frac{\partial\lambda_i}{\partial\gamma_i} - \psi\left(\gamma_i+Y_i\right) + \psi\left(\gamma_i\right) - \frac{\partial}{\partial\gamma_i}\log\left({}_1F_1\left(1,\gamma_i;\lambda_i\right)\right)\right\}\\ &= \sum_{i=1}^{n}\left\{\frac{Y_i}{\lambda_i}\frac{\partial\lambda_i}{\partial\gamma_i} - \psi\left(\gamma_i+Y_i\right) + \psi\left(\gamma_i\right) - \frac{\mu_i}{\lambda_i}\frac{\partial\lambda_i}{\partial\gamma_i} + E\left[\psi\left(\gamma_i+Y_i\right)\right] - \psi\left(\gamma_i\right)\right\}\\ &= \sum_{i=1}^{n}\left\{\frac{Y_i-\mu_i}{\lambda_i}\frac{\partial\lambda_i}{\partial\gamma_i} - \psi\left(\gamma_i+Y_i\right) + E\left[\psi\left(\gamma_i+Y_i\right)\right]\right\},\end{aligned}$$

wherein we have used (9).

The derivative of λ_i with respect to γ_i is also obtained by explicit differentiation from equation (2)

with respect to γ, getting (without considering subscripts):

$$\begin{aligned}
0 =& \sum_{y=0}^{\infty} (y-\mu) \frac{y\lambda^{y-1}\frac{\partial\lambda}{\partial\gamma}(\gamma)_y - \lambda^y \frac{\partial(\gamma)_y}{\partial\gamma}}{(\gamma)_y^2} \\
=& \frac{1}{\lambda}\frac{\partial\lambda}{\partial\gamma}\sum_{y=0}^{\infty}(y-\mu)\, y\frac{\lambda^y}{(\gamma)_y} - \sum_{y=0}^{\infty}(y-\mu)\frac{\lambda^y}{(\gamma)_y}\left(\psi(\gamma+y)-\psi(\gamma)\right) \\
=& \frac{1}{\lambda}\frac{\partial\lambda}{\partial\gamma}\sum_{y=0}^{\infty}(y-\mu)^2\frac{\lambda^y}{(\gamma)_y} - \sum_{y=0}^{\infty}(y-\mu)\,\psi(\gamma+y)\frac{\lambda^y}{(\gamma)_y} \\
=& {}_1F_1(1,\gamma;\lambda)\left\{\frac{V(\mu,\lambda)}{\lambda}\frac{\partial\lambda}{\partial\gamma} - E\left[(Y-\mu)\,\psi(\gamma+Y)\right]\right\},
\end{aligned}$$

and clearing:

$$\frac{\partial\lambda}{\partial\gamma} = \lambda\frac{E\left[(Y-\mu)\,\psi(\gamma+Y)\right]}{V(\mu,\gamma)} = \lambda\frac{Cov\left[Y,\psi(\gamma+Y)\right]}{V(\mu,\gamma)}. \tag{13}$$

Therefore, replacing (13) in the score of γ we obtain

$$\frac{\partial l_{hP}}{\gamma_i} = \sum_{i=1}^{n}\left\{(Y_i-\mu_i)\frac{Cov\left[Y_i,\psi(\gamma_i+Y_i)\right]}{V(\mu_i,\gamma_i)} - \psi(\gamma_i+Y_i) + E\left[\psi(\gamma_i+Y_i)\right]\right\}. \tag{14}$$

The only thing left to consider to get the expression (5) is that

$$\frac{\partial\gamma_i}{\partial\boldsymbol{\delta}} = \gamma_i\mathbf{z}_i'.$$

Proof of result 3

The Fisher information matrix can be calculated by blocks:

$$E\left[S_\beta S_\beta'\right] = E\left[(Y-\mu)^2\right]\frac{\mu^2}{\sigma^4}XX' = \frac{\mu^2}{\sigma^2}XX'.$$

$$\begin{aligned}
E\left[S_\delta S_\delta'\right] =& \left\{\frac{E\left[(Y-\mu)^2\right]}{\sigma^4}Cov\left(Y,\psi(\gamma+Y)\right)^2 + E\left[\psi(\gamma+Y)^2\right] + E\left[\psi(\gamma+Y)\right]^2\right. \\
& - 2\frac{E\left[(Y-\mu)\,\psi(\gamma+Y)\right]}{\sigma^2}Cov\left(Y,\psi(\gamma+Y)\right) \\
& \left. +2E\left[\psi(\gamma+Y)\right]\frac{E\left[Y-\mu\right]}{\sigma^2}Cov\left(Y,\psi(\gamma+Y)\right) - 2E\left[\psi(\gamma+Y)\right]E\left[\psi(\gamma+Y)\right]\right\}ZZ'\gamma^2 \\
=& \left\{\frac{Cov\left(Y,\psi(\gamma+Y)\right)^2}{\sigma^2} + E\left[\psi(\gamma+Y)^2\right] - E\left[\psi(\gamma+Y)\right]^2 - 2\frac{Cov\left(Y,\psi(\gamma+Y)\right)^2}{\sigma^2}\right\}ZZ'\gamma^2 \\
=& \left[Var\left(\psi(\gamma+Y)\right) - \frac{Cov\left(Y,\psi(\gamma+Y)\right)^2}{\sigma^2}\right]ZZ'\gamma^2.
\end{aligned}$$

$$\begin{aligned}
E\left[S_\beta S_\delta'\right] =& \left\{\frac{E\left[(Y-\mu)^2\right]}{\sigma^4}Cov\left(Y,\psi(\gamma+Y)\right) - \frac{E\left[(Y-\mu)\left(\psi(\gamma+Y)\right)\right]}{\sigma^2} + \frac{E\left[Y-\mu\right]}{\sigma^2}E\left[\psi(\gamma+Y)\right]\right\}XZ'\mu\gamma \\
=& 0.
\end{aligned}$$

Gradient of the objective function

Considering the log-likelihood function (3) as a function of $\boldsymbol{\beta}$, $\lambda_i' = \log \lambda_i$ and $\boldsymbol{\delta}$, the gradient of the objective function is:

$$\begin{aligned}
\frac{\partial l_{hP}}{\partial \boldsymbol{\beta}} &= \mathbf{0}_{1\times(q_1+1)}.\\
\frac{\partial l_{hP}}{\partial \lambda_i'} &= \frac{\partial l_{hP}}{\partial \lambda_i}\frac{\partial \lambda_i}{\partial \lambda_i'} = \frac{1}{\lambda_i}\left(\sum_{i=1}^{n} Y_i - n\mu_i\right)\lambda_i = \sum_{i=1}^{n} Y_i - n\mu_i.\\
\frac{\partial l_{hP}}{\partial \boldsymbol{\delta}} &= \frac{\partial l_{hP}}{\partial \boldsymbol{\gamma}}\frac{\partial \boldsymbol{\gamma}}{\partial \boldsymbol{\delta}} = \left(-\sum_{i=1}^{n}\psi\left(\gamma_i + Y_i\right) + E\left[\psi\left(\gamma_i + Y_i\right)\right]\right)\gamma_i \mathbf{z}_i'.
\end{aligned}$$

The proof is straightforward from expressions (7) and (8).

Gradient of the restriction (2)

Restriction (2) can be expressed as

$$ceq_i = e^{\mathbf{x}_i'\boldsymbol{\beta}} - \sum_{y_i=0}^{\infty} y_i \frac{1}{{}_1F_1\left(1, \gamma_i; \lambda_i\right)}\frac{\lambda_i^{y_i}}{(\gamma_i)_{y_i}}, \qquad i = 1, \ldots, n,$$

whose gradient is given by:

$$\nabla ceq_i = \left(\frac{\partial ceq_i}{\partial \boldsymbol{\beta}}, \frac{\partial ceq_i}{\partial \lambda_i}, \frac{\partial ceq_i}{\partial \boldsymbol{\delta}}\right),$$

where

$$\begin{aligned}
\frac{\partial ceq_i}{\partial \boldsymbol{\beta}} &= \mu_i \mathbf{x}_i'.\\
\frac{\partial ceq_i}{\partial \lambda_i} &= -\frac{V\left(\mu_i, \gamma_i\right)}{\lambda_i} \Longrightarrow \frac{\partial ceq_i}{\partial \lambda_i'} = -V\left(\mu_i, \gamma_i\right).\\
\frac{\partial ceq_i}{\partial \boldsymbol{\delta}} &= Cov\left(Y_i, \psi\left(\gamma_i + Y_i\right)\right)\gamma_i \mathbf{z}_i'.
\end{aligned}$$

For this, you have to take into account that:

$$\begin{aligned}
\frac{\partial ceq_i}{\partial \lambda_i} &= -\sum_{y_i=0}^{\infty} y_i^2 \frac{1}{{}_1F_1\left(1, \gamma_i; \lambda_i\right)}\frac{\lambda_i^{y_i-1}}{(\gamma_i)_{y_i}} + \sum_{y_i=0}^{\infty} y_i \frac{\mu}{\lambda}\frac{1}{{}_1F_1\left(1, \gamma_i; \lambda_i\right)}\frac{\lambda_i^{y_i}}{(\gamma_i)_{y_i}}\\
&= -\frac{1}{\lambda_i}\sum_{y_i=0}^{\infty} y_i^2 \frac{1}{{}_1F_1\left(1, \gamma_i; \lambda_i\right)}\frac{\lambda_i^{y_i}}{(\gamma_i)_{y_i}} + \frac{\mu_i}{\lambda_i}\sum_{y_i=0}^{\infty} y_i \frac{1}{{}_1F_1\left(1, \gamma_i; \lambda_i\right)}\frac{\lambda_i^{y_i}}{(\gamma_i)_{y_i}}\\
&= -\frac{1}{\lambda_i}E\left[Y_i^2\right] + \frac{\mu_i^2}{\lambda_i} = -\frac{V\left(\mu_i, \gamma_i\right)}{\lambda_i},
\end{aligned}$$

where expression (7) has been used. Considering expressions (8) and (9), it is verified that:

$$\begin{aligned}
\frac{\partial ceq_i}{\partial \gamma_i} &= \sum_{y_i=0}^{\infty} y_i\left(\psi\left(\gamma_i + y_i\right) - \psi\left(\gamma_i\right)\right)\frac{1}{{}_1F_1\left(1, \gamma_i; \lambda_i\right)}\frac{\lambda_i^{y_i}}{(\gamma_i)_{y_i}} + \sum_{y_i=0}^{\infty} y_i\left(\psi\left(\gamma_i\right) - E\left[\psi\left(\gamma_i + Y_i\right)\right]\right)\frac{1}{{}_1F_1\left(1, \gamma_i; \lambda_i\right)}\frac{\lambda_i^{y_i}}{(\gamma_i)_{y_i}}\\
&= E\left[Y_i\psi\left(\gamma_i + Y_i\right)\right] - E\left[Y_i\right]E\left[\psi\left(\gamma_i + Y_i\right)\right] = Cov\left(Y_i, \psi\left(\gamma_i + Y_i\right)\right),
\end{aligned}$$

from which it is straightforward to obtain the gradient for $\boldsymbol{\delta}$.

Making Provenance Work for You

by Barbara Lerner, Emery Boose, Orenna Brand, Aaron M. Ellison, Elizabeth Fong, Matthew K. Lau, Khanh Ngo, Thomas Pasquier, Luis Perez, Margo Seltzer, Rose Sheehan, Joseph Wonsil

Abstract To be useful, scientific results must be reproducible and trustworthy. Data provenance—the history of data and how it was computed—underlies reproducibility of, and trust in, data analyses. Our work focuses on collecting data provenance from R scripts and providing tools that use the provenance to increase the reproducibility of and trust in analyses done in R. Specifically, our "End-to-end provenance tools" ("E2ETools") use data provenance to: document the computing environment and inputs and outputs of a script's execution; support script debugging and exploration; and explain differences in behavior across repeated executions of the same script. Use of these tools can help both the original author and later users of a script reproduce and trust its results.

1 Introduction

In today's data-driven world, an increasing number of people are finding themselves needing to analyze data in the course of their work. Often these people have little or no background or formal coursework in programming and may think of it solely as a tedious means to an interesting end. Writing scripts to work with data in this way is often exploratory. The researcher may be writing a script to produce a plot that enables visual understanding of the data. This understanding might then lead to a realization that the data need to be cleaned to remove bad values, and statistical tests need to be performed to determine the strength or trends of relationships. Examining these results may raise more questions and lead to more code. This type of exploratory programming can easily lead to scripts that grow over time to include both useful and irrelevant code that is difficult to understand, debug, and modify.

Creating a script and successfully running it once to analyze a dataset is one thing. Reproducing it later is another thing entirely. We might expect that re-running a script and reproducing a data analysis should be a simple matter of rerunning a program or script on the same data, but it is rarely that simple. Anyone who has tried to retrieve the version of the data and scripts used to produce the results presented in a paper will likely appreciate how difficult this can be. Data and scripts can be modified or lost. But even if care is taken to save the scripts and data, new versions of programming languages, libraries and operating systems may make scripts behave differently or be unable to run at all. In an ideal world, everything would be backwards-compatible, but in reality, what ran last week often doesn't run next week. It can be difficult to determine what went wrong, especially if programming is an occasional activity. The National Academy of Sciences report on Reproducibility and Replicability in Science (National Academies of Sciences, Engineering, and Medicine, 2019) describes at length the challenges associated with computational reproducibility of scientific results.

Motivated by an interest in supporting reproducibility of R scripts, we developed a package called **rdtLite** to collect data provenance containing a record of a script's execution and the environment in which it was executed (Lerner et al., 2018). Having done that, we then realized that the wealth of information contained in the data provenance could serve other purposes as well. This led to the development of End-to-End Provenance Tools ("E2ETools"): an evolving set of R packages that use data provenance to help users save workable copies of their data and scripts, debug them, understand how data and results of analyses were derived, discover what has changed when a script stops working, and reproduce prior results.

2 What is data provenance?

Provenance is the history of creation, ownership, chain-of-custody, and location of an object. In its original and still most-frequently used sense, provenance is used to authenticate and trace the legitimate ownership of a work of art; it confers, creates, or adds value to the work itself. But provenance can be constructed, identified, or traced for any object, including data (Becker and Chambers, 1988). Data provenance is analogous to provenance of a work of art in that it includes the history of a datum or entire dataset from the point at which it was collected (by a person or sensor), created (by a computational process), or derived (from other data). Data provenance also confers or adds value—as trustworthiness—to data, but data provenance can do more: it can be used to reproduce computational analyses and validate scientific conclusions.

More precisely, data provenance is the history of a data item ("datum") or a dataset ("data"); it describes **how** the datum or data came to be in its present state. Our E2ETools focus on *language-level*

```
# Load the mtcars data set that comes with R
data(mtcars)

# All the cars
allCars.df <- mtcars

# Create separate data frames for each number of cylinders
cars4Cyl.df <- allCars.df[allCars.df$cyl == 4, ]
cars6Cyl.df <- allCars.df[allCars.df$cyl == 6, ]
cars8Cyl.df <- allCars.df[allCars.df$cyl == 8, ]

# Create a table with the average mpg for each # cylinders
cylinders = c(4, 6, 8)
mpg = c(mean(cars4Cyl.df$mpg), mean(cars6Cyl.df$mpg), mean(cars8Cyl.df$mpg))
cyl.vs.mpg.df <- data.frame (cylinders, mpg)

# Plot it
plot(cylinders, mpg)
```

Figure 1: Source code for mtcars_example.R. This code is used to demonstrate the lineage traces provided by the debug.lineage function as described in the text.

provenance: how data are created and manipulated by a programming language such as R during the execution of a script or program. Provenance is also referred to in other computing contexts. For example, data provenance can be used to understand results of queries to a database or to the processes that were used to create or modify a file. In the remainder of this paper, however, when we say "provenance" or "data provenance", we specifically mean language-level provenance.

We associate three types of information with provenance: environment information, coarse-grained information, and fine-grained information. *Environment information* includes information about the computing environment in which the script was executed. This includes information such as the operating system version, the R version, and the versions of the R libraries used, as each of these may play a role in understanding the details of how a script behaves. *Coarse-grained information* includes the source code of the script(s), the data input to the script, the data output by the script, and plots produced by the script. *Fine-grained information* includes an execution trace. Specifically, for each line of the script that is executed, fine-grained information includes the data used on that line and any data computed by, or object created by, that line. Our E2ETools can use this fine-grained information to help a user understand exactly how any data value or object in the script was computed or derived.

3 A first example

Consider this simple example, 'mtcars_example.R', that loads in the 'cars' dataset and plots miles per gallon (mpg) as a function of the number of cylinders (cylinders) (Figure 1).

The following commands run the script, collect its provenance, and produce a textual summary of the provenance.

```
library(rdtLite)
prov.run("mtcars_example.R")
prov.summarize()
```

The provenance summary is shown in Figure 2. The environment information (lines 3–18) reports details of the computing environment in which the script was executed, such as the processor and operating system on which it ran and the version of R and R libraries used. The coarse-grained information (lines 20–36) identifies the location in the file system of the script, the input dataset, and the plot produced. The fine-grained information, which is not displayed by `prov.summarize()` but is accessible via other tools, indicates the input and output data for each line of code executed, linking them together so that one can see how the values computed in one statement are used in later statements. For example, the provenance debugger can use fine-grained information to display everything that is derived from a variable.

```
PROVENANCE SUMMARY for mtcars_example.R

ENVIRONMENT:
Executed at 2022-07-28T13.52.25EDT
Total execution time was 1.516 seconds
Script last modified at 2022-07-22T10.41.25EDT
Executed with R version 4.2.1 (2022-06-23)
Platform was x86_64, darwin17.0
Operating system was macOS Catalina 10.15.7
User interface was 2022.02.3+492 Prairie Trillium (desktop)
Document converter was 2.2.1 @ /usr/local/bin/pandoc
Provenance was collected with rdtLite1.4
Provenance is stored in /Users/blerner/tmp/prov/prov_mtcars_example
Hash algorithm is md5

LIBRARIES (loaded by script):
None (see notes below)

SCRIPTS:
1[:] /Users/blerner/Documents/Process/DataProvenance/Papers/RJournal/scripts/
    examples/mtcars_example.R

PRE-EXISTING:
None

INPUTS:
1[:] /Library/Frameworks/R.framework/Versions/4.2/Resources/library/datasets/
    data/Rdata.rds

OUTPUTS:
1[-] /Users/blerner/Documents/Process/DataProvenance/Papers/RJournal/scripts/
    dev.off.11.pdf

CONSOLE:
None

ERRORS & WARNINGS:
None

NOTES: Files are listed in the order of execution (script 1 = main script).
The status of each file in its original location is marked as follows:
File unchanged [:], File changed [+], File missing [-], Not checked [ ].
Copies of original files are available on the provenance directory.

Libraries loaded by the user's script at the time of execution are displayed.
Note that some libraries may have been loaded before execution. Use details =
TRUE to see all loaded libraries along with script, file, and message details.
```

Figure 2: Provenance summary for mtcars_example.R, showing the environment in which the script was executed, identifying the script, input and output files, and any errors or warnings encountered when the script was executed.

```
library(provDebugR)
prov.debug()
debug.lineage("cars4Cyl.df", forward = TRUE)
```

The resulting output displays the line numbers and code for everything computed, either directly or indirectly, from `cars4Cyl.df` .

```
Var cars4Cyl.df
8:    cars4Cyl.df <- allCars.df[allCars.df$cyl == 4, ]
14:   mpg = c(mean(cars4Cyl.df$mpg), mean(cars6Cyl.df ...
15:   cyl.vs.mpg.df <- data.frame (cylinders, mpg)
18:   plot(cylinders, mpg)
NA:   mtcars_example.R
```

Alternatively, a modified version of the same command

```
debug.lineage("cars4Cyl.df")
```

shows the lines of code that lead to the value for `cars4Cyl.df` being computed.

```
Var cars4Cyl.df
2:   data(mtcars)
5:   allCars.df <- mtcars
8:   cars4Cyl.df <- allCars.df[allCars.df$cyl == 4, ]
```

Having seen an introductory example of some things the E2ETools can do, we now turn to a more detailed discussion of each tool.

4 The end-to-end provenance tools

The E2ETools consist of three types of packages:

- A package to collect provenance: **rdtLite**;
- Packages that process data provenance to provide information to the user about a particular script and its execution: **provSummarizeR**, **provDebugR**, **provViz**, and **provExplainR**;
- Packages to enable tool developers to more easily use data provenance: **provParseR** and **provGraphR**.

We describe each of these packages, beginning with provenance collection. All the tools described are available on CRAN.

Collecting provenance with rdtLite

The rdtLite package collects provenance from R scripts as they execute.[1] rdtLite captures provenance data from both scripts and interactive console sessions. To capture provenance for a script, the user runs the script using the `prov.run` function.

```
library(rdtLite)
prov.run("script.R")
```

To collect provenance for an interactive session, the user begins the session with the `prov.init` function and concludes it with `prov.quit`.

```
library(rdtLite)
prov.init()
data <- read.csv("mydata.csv")
plot(data$x, data$y)
prov.quit()
```

rdtLite collects information about each file or URL read by the script, each file written by the script, and each plot created by the script. In addition, it records an execution trace of the top-level R statements. This trace identifies the statement executed. It records any variables set or used by the statement. When a variable is set, it records the type of the value, including its container (such as

[1] rdtLite is a simplified version of RDataTracker (Lerner and Boose, 2014b; Lerner et al., 2018).

vector, data frame, etc.), dimensions, and class (e.g., character, numeric). If the container is a vector of length 1, rdtLite records its data value, embedded in the provenance (which is stored in a JSON file). rdtLite can save the values of larger containers in separate snapshot files. The user controls how much data to save using the `snapshot.size` parameter in `prov.init` and `prov.run`. The default is to not save snapshots. rdtLite also records any warning or error messages generated when the statement is executed. To capture similar information about scripts that are included using the `source` function, calls to `source` must be replaced with calls to `prov.source`.

The provenance is stored in a JSON file using a format that extends the PROV-JSON standard (W3C, 2014).[2] The extended format provides structured information about fine-grained provenance, such as a list of libraries used, a mapping from functions called to the libraries from which they came, script line numbers, and data values and their types. More information about the extended JSON format is provided in the Appendix.

The JSON file is stored in a provenance directory that also contains copies of all input and output files and the R scripts executed. By default, the provenance data is stored in the R session temporary directory, but the user can change this location either at the time that `prov.run` or `prov.init` is called or by setting the `prov.dir` option, for example, in the .Rprofile file.

Upon completion of a script called with `prov.run`, or after a call to `prov.quit`, rtdLite creates and populates a directory named either '`prov_script`', where '`script`' is the name of the script file, or '`prov_console`' for an interactive session. The directory will contain:

- 'prov.json' - the JSON file containing the fine-grained provenance
- 'data' - a directory containing copies of input and output files, URLs, plots created, and snapshot files.
- 'scripts' - a directory containing a copy of the scripts for which provenance was collected.

The rdtLite default is to overwrite this information if the same script is executed again or if `prov.init` is used again in a console session. However if the `overwrite` parameter is set to FALSE, the provenance is stored in a unique, time-stamped directory, allowing provenance from multiple executions to be analyzed and compared.

Using provenance

Having the provenance is extremely valuable, but it is not particularly usable without tools that read the provenance and provide *information* or enable *reproducibility*. We next describe four tools that use provenance to help R programmers understand executions of their script. The **provSummarizeR** package provides a concise textual summary of an execution. The **provViz** package provides a graphical visualization of the provenance. The **provDebugR** package uses collected provenance to help programmers debug their code. The **provExplainR** package compares provenance from two executions to help the programmer understand changes between them. These applications exist in packages separate from **rdtLite** and would work equally well with provenance collected by other tools that produce the same JSON format.

provSummarizeR

The purpose of **provSummarizeR** is to produce a concise record of the environment in which a script was executed. This information could be particularly valuable when including a script and its results in a paper, or when sharing a script with a colleague. For an example, please see Figure 2 above. The summary includes the following information:

- The ENVIRONMENT section shows information about when the script was modified and executed, what version of R was used, what hardware and operating system were used, what R environment (such as RStudio) was used, what tool collected the provenance, where the provenance is stored, and what hash algorithm was used to store hash values for files used in the input and output of the script.
- The LIBRARIES section shows the libraries loaded by the script and their version numbers.
- The SCRIPTS section lists the main script and any scripts that are included in the execution of this script using the `source` or `prov.source` functions.
- The PRE-EXISTING section shows any variables where the script uses a value that was bound to the variable before the script started. This is a common R programming error that can lead to unexpected results if the script is run again in a different environment, where such a variable might have a different value or not be set at all.

[2] https://github.com/End-to-end-provenance/ExtendedProvJson/blob/master/JSON-format.md.

- The INPUTS and OUTPUTS sections list all input and output files, the date they were last modified, and their hash values, using the hash algorithm shown in the environment section.
- The CONSOLE section shows any output sent to the console when the script executed.
- The ERRORS & WARNINGS section lists any errors or warnings that occurred when the script executed, including the number of the line that caused them.

In our own day-to-day work, we use **provSummarizeR** to document the processing of real-time meteorological and hydrological data at Harvard Forest. Data and plots of data captured in the past 30 days, including air temperature, precipitation, stream discharge, and water temperature, are updated and posted every 15 minutes.[3] Also posted at the same site are provenance summaries for the script execution that creates the plots.

There are three functions provided to generate summaries:

```
prov.summarize(details = FALSE)
prov.summarize.file(prov.file, details = FALSE)
prov.summarize.run(r.script, details = FALSE)
```

- `prov.summarize` produces a summary for the last provenance collected in the current R session.
- `prov.summarize.file` takes the name of a JSON file containing provenance and produces a summary from it.
- `prov.summarize.run` takes the name of a file containing an R script. It runs the script, collects its provenance, and produces a summary.[4]

By passing `TRUE` for the `details` parameter, the user can see more detail about some aspects of the provenance. In particular,

- The libraries section is divided into three parts. The first part shows the libraries loaded by the script. The second part shows the libraries that were loaded before the script starts. The third part shows the libraries loaded by the rdtLite code itself.
- The information about script, inputs, and output files includes modification date and hash value.
- The information about errors and warnings includes the line number on which each occurred.

The provViz and provDebugR tools described below provide a similar set of three functions: one to use the last provenance collected, one to use a specific JSON file, and one to run a script and use its provenance.

provViz

The **provViz** package allows visual exploration of script execution as shown in Figure 3. There are two types of nodes: data nodes and procedure nodes. Data nodes represent things such as variables, files, plots, and URLs. Procedure nodes represent executed R statements. An edge from a data node to a procedure node indicates that the statement represented by the procedure node uses the data represented by the data node. For example, the edge from data item, '`7-mpg`', to procedure node, '`9-plot(cylinders,mpg)`', indicates that `mpg` was used in the call to the `plot` function. Conversely, an edge from a procedure node to a data node indicates that the procedure produced the data, for example, by assigning to a variable or writing to a file. An edge between two procedure nodes represents control flow, indicating the order in which the statements were executed.

provViz also allows the user to view the graph and explore it to examine intermediate data values or input and output files and to perform lineage queries. The node colors indicate node type. Data nodes representing variables are purple. Files are tan. Orange nodes represent standard output, while red data nodes represent warnings and errors. Yellow nodes represent R statements. Green nodes come in pairs and represent the start and end of a group of R statements. Clicking on a green node reduces the set of statements between the matching '`Start`' and '`Finish`' nodes into a single node, which is useful for making large graphs more manageable.

To see everything that depends on the value of a variable at a particular point in the execution of the script, the user can right-click on the data node and select '`Show what is computed using this value`'. This will display a subgraph containing just the data and procedure nodes that are in the lineage of the data node, as shown in Figure 4, which shows the lineage of '`3-cars4Cyl.df`'. Notice that statements that do not use the value of `cars4Cyl.df`, either directly or indirectly, are not shown.

[3] https://harvardforest.fas.harvard.edu/met-hydro-stations

[4] All three functions have additional optional parameters. For details, see the online help page.

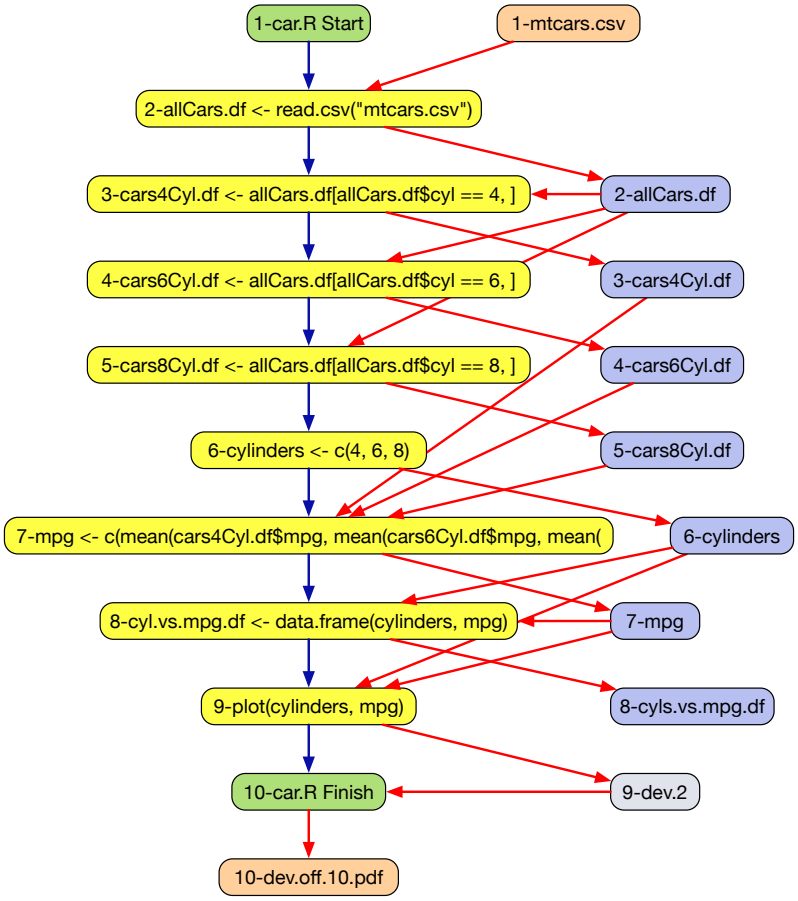


Figure 3: A provenance graph as displayed using **provViz**. Yellow nodes represent statements in the code, blue nodes represent variables, orange nodes represent files and green nodes mark the start and end of the script.

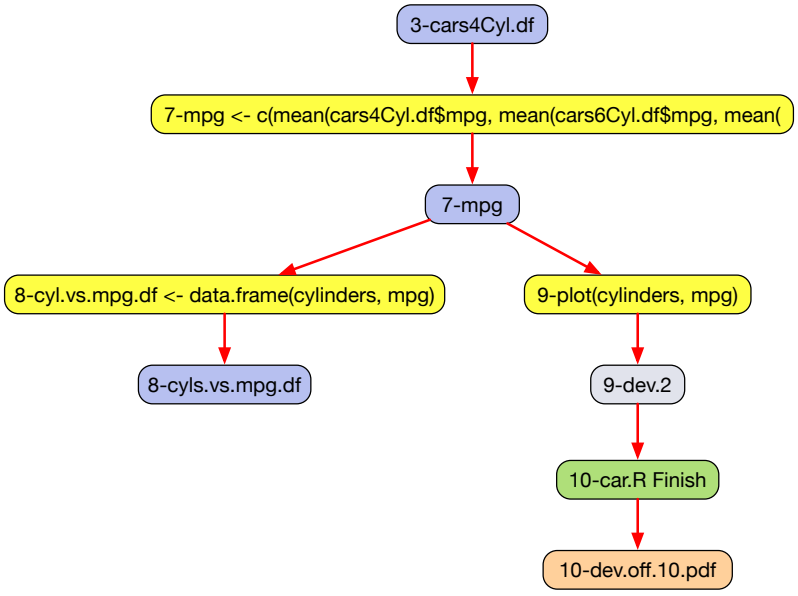


Figure 4: Displaying the Lineage of 3-cars4Cyl.df

In addition to examining data values and tracing lineages as in this example, provViz supports the following ways of exploring the provenance:

- Viewing input and output data files
- Viewing plots created
- Viewing the source code for a node or the entire script
- Comparing R scripts
- Comparing provenance graphs
- Searching for nodes by name and type
- Sorting procedure nodes based on execution time

provViZ itself is a small R program that connects to a Java program called DDG Explorer (Lerner and Boose, 2014a), which does the actual work of creating and managing the display.

provDebugR

The **provDebugR** package provides debugging support by using the provenance to help users understand the state of their script at any point during execution. It provides command-line debugging capabilities, but one could imagine building a GUI on top of these functions to produce a friendly interactive debugging environment. By using provenance, provDebugR provides insight into the entire execution and creates a rich debugging environment that provides execution context not typically available in debuggers.

For example, consider a simple, but buggy script.

```
w <- 4:6
x <- 1:3
y <- 1:10
z <- w + y
y <- c('a', 'b', 'c')
xyz <- data.frame (x, y, z)
```

Running this script produces a warning and an error.

```
Error in data.frame(x, y, z) :
  arguments imply differing number of rows: 3, 10
In addition: Warning message:
In w + y : longer object length is not a multiple of shorter object length
```

Of course, with a short script like this, a user could simply step through the script one line at a time and examine the results, but for the purposes of demonstrating the debugger, imagine that this code is buried within a large script. The lines of code might not be consecutive as shown here, and it may even be difficult to determine what lines caused the reported errors.

The debugger provides some functions that are particularly helpful for understanding warning and error messages. For example, if the user needs help understanding where a warning came from, calling `debug.warning` with no arguments lists all the warnings; when called with a warning number, it displays the lines of code leading up to the warning.

```
> debug.warning()
Possible results:

1 In  w + y :  longer object length is not a multiple of shorter object length

Pass the corresponding numeric value to the function for info on that warning
> debug.warning(1)
Warning: In  w + y :  longer object length is not a multiple of shorter object length
        1:          w <- 4:6
        3:          y <- 1:10
        4:          z <- w + y
```

By omitting lines that do not contribute to the computations that lead to the warning, the R programmer should be able to find the problem more easily.

Similarly, the user can get information about what led up to an error using `debug.error`.

```
> debug.error(stack.overflow=TRUE)
Your Error: Error in data.frame(x, y, z): arguments imply differing number
of rows: 3, 10

Code that led to error message:
1:          w <- 4:6
2:          x <- 1:3
3:          y <- 1:10
4:          z <- w + y
5:          y <- c('a', 'b', 'c')
6:          xyz <- data.frame (x, y, z)

Results from StackOverflow:
[1] "What does the error \"arguments imply differing number of rows: x, y\"
    mean?"
[2] "ggplot gives \"arguments imply differing number of rows\" error in
    geom_point while it isn't true - how to debug?"
[3] "Checkpoint function error in R- arguments imply differing number of rows:
    1, 38, 37"
[4] "qdap check_spelling Error in checkForRemoteErrors(val) : one node
    produced an error: arguments imply differing number of rows"
[5] "Creating and appending to data frame in R (Error: arguments imply
    differing number of rows: 0, 1)"
[6] "Caret and GBM: task 1 failed - \"arguments imply differing number of rows\""

Choose a numeric value that matches your error the best or q to quit:
```

Figure 5: The output of a call to debug.error, showing the titles of posts on Stack Overflow related to the error encountered in the script. The user can select an option to be taken to the corresponding Stack Overflow page.

```
> debug.error()
Your Error: Error in data.frame(x, y, z): arguments imply differing number of rows: 3, 10

Code that led to error message:
        1:          w <- 4:6
        2:          x <- 1:3
        3:          y <- 1:10
        4:          z <- w + y
        5:          y <- c('a', 'b', 'c')
        6:          xyz <- data.frame (x, y, z)
```

The `debug.error` function has an optional logical parameter, `stack.overflow`. When set to `TRUE`, `debug.error` uses the stackexchange API to search Stack Overflow for posts about similar error messages. It lists the questions asked in the top six posts. The user can select one and a tab will open in the user's browser displaying the selected post.

Figure 5 shows a sample dialog using `debug.error`. Selecting 1 results in the user's browser going to the page displayed in Figure 6.[5] By scrolling down through answers to this question (not shown here), users will ideally obtain helpful information allowing them to solve their problem quickly.

A common cause of programming errors in R is caused by automatic type conversions as occurs here:

```
x <- 1
y <- 1:10
z <- 2
x <- x + y
if (x == 2) {
  print ("x is 2")
```

[5] https://stackoverflow.com/questions/26147558/what-does-the-error-arguments-imply-differing-number-of-rows-x-y-mean

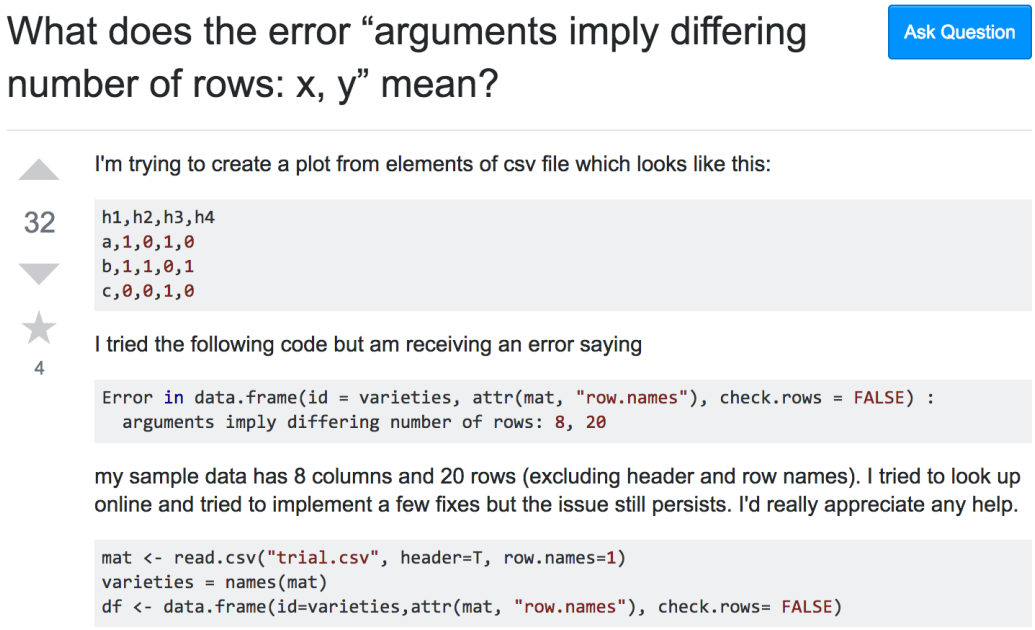


Figure 6: Stack Overflow Page to Resolve an Error

```
} else {
  print ("x is not 2")
}
```

Running this simple script produces this output.

```
Error in if (x == 2) { : the condition has length > 1
```

The programmer may be surprised or confused to get this warning message, as the assignment back to x may have been a mistake. Since R is a dynamically-typed language, there is no error at the time of the assignment, but only later when the value is used. The programmer can use `debug.variable` to quickly identify the type of x at each assignment

```
> debug.variable(x, showType=TRUE)
Var: x
        1:          1          x <- 1
  container dimension type
1 vector    1         numeric
        4:          2  3  4  5  6  7  8  9 10 11          x <- x + y
  container dimension type
2 vector    10        numeric
```

This shows that on line 4, x changed from a single element vector whose value was 1 to a 10-element vector containing the numbers 2 through 11.

Next, the programmer may want to find out why x became a vector. The `debug.lineage` function provides this information.

```
> debug.lineage(x)
Var x
        1:          x <- 1
        2:          y <- 1:10
        4:          x <- x + y
```

By showing the lines that led to x's value and type at line 4, we see the vector assignment to y in line 2, followed by the computation of x in line 4. Notice that line 3, the assignment to z, is not included in the lineage, since it played no role, either directly or indirectly in the value assigned to x. Ideally, by examining the provenance, the programmer realizes that the assignment should have been to y rather than to x.

An experienced R programmer may realize that unexpected type changes such as these can commonly lead to errors. Even if no error had been reported, they might want to check preemptively for type changes. This can be done by calling `debug.type.changes`, which reports all variables where

the container, dimension, or type of value in the container have changed, showing just the values immediately before and after the type change.

```
> debug.type.changes()
The type of variable x has changed. x was declared on line 1 in debugScript4.R.
        1: debugScript4.R, line 4
                dimension changed to: 10
                from:                     1
                code excerpt: x <- x + y
```

The `debug.line` and `debug.state` functions allow the user to inspect variable values at specific lines in the code. The `debug.line` function shows the values of all variables used or modified on a specific line.

```
> debug.line(4)
Results for line(s): 4

4: x <- x + y
         Inputs:
                1. x   1
                2. y    1  2  3  4  5  6  7  8  9 10
         Outputs:
                1. x    2  3  4  5  6  7  8  9 10 11
```

The `debug.state` function shows the values that all variables have after execution of a specific line, showing the line number where the variable was set.

```
> debug.state(4)
Results for line(s): 4

Line 4
        4:         x         2  3  4  5  6  7  8  9 10 11
        2:         y         1  2  3  4  5  6  7  8  9 10
        3:         z        2
```

Earlier we showed the `debug.lineage` function that shows the user how a particular value was computed. That was an example of **backward lineage** or **ancestry**, because it starts with a variable and goes back in time to show all the computations on which a variable depends. The `debug.lineage` function can also display **forward lineage** to show how a value is used, i.e., all the subsequent computations that depend on it. This is particularly helpful in identifying all the information that might be affected by a programmatic change or modification to an input file.

```
> debug.lineage(x, forward = TRUE)
Var x
        1:          x <- 1
        4:          x <- x + y
        5:          if (x == 2) {
```

Note that by using provenance, provDebugR is able to display information about the execution state of the script at different points in its execution without the need to set breakpoints or insert print statements and re-run the script. This is particularly helpful for stochastic processes where the output might vary on each execution, causing some bugs to be challenging to track down.

provExplainR

Whereas **provSummarizeR** provides a summary of a single script execution, **provExplainR** goes a step further and provides a textual description of the difference between two script executions. If two executions of a script produce different outputs, **provExplainR** can be used to expose differences. This can be helpful when returning to work on an old script, when porting a script to a new environment, or when inheriting a script from someone else.

The `prov.explain` function reads two provenance directories and identifies differences in the computing environment, the input data, the versions of R or its libraries, and/or the main and sourced scripts.

```
prov.explain(
  dir1 = "prov_factorial_2021-03-31T12.01.36EDT",
  dir2 = "prov_factorial_2021-04-26T16.34.16EDT")
```

Results are displayed in the console (Figure 7).

The `prov.diff.script` function can be used to identify differences between two scripts.

```
prov.diff.script(
  dir1 = "prov_MyScript_2019-08-06T15.59.18EDT",
  dir2 = "prov_MyScript_2019-08-21T16.25.58EDT")
```

This function uses the **diffobj** package to identify and display differences (Figure 8).

We are planning to extend the functionality of provExplainR so that it also helps the programmer understand the impact of any reported changes by identifying where the behavior of the two executions start to differ. We expect this will help the programmer understand more specifically why the script is behaving differently. For example, if the line of code where changes first appear involves calling a function from an updated library, the programmer will likely want to understand better what changed with the new version of the library.

Developing new provenance-based tools

In addition to end-user tools as described above, we have also made available packages intended for programmers interested in developing their own tools incorporating provenance information.

provParseR

The **provParseR** package parses the JSON provenance and provides a convenient API to access portions of the provenance. To get started the tool developer calls the `prov.parse` function.

```
prov.parse(prov.input, isFile = TRUE)
```

The `prov.input` parameter is a string that can either be the path to a JSON file containing provenance or it can be a string containing the provenance. The second parameter (`isFile`) is used to disambiguate these cases. The default assumption is that `prov.input` is the path to a file. This function returns an object whose class is `ProvInfo`. The remaining functions provided by **provParseR** are getters that are passed a `ProvInfo` object and return information, typically a data frame containing that portion of the provenance.

For example, `get.input.files` returns a data frame containing a subset of the data nodes that correspond to files read by the script. The data frame that is returned includes the following information:

- id - a unique id
- name - the file name
- value - the path to a saved copy of the file
- hash - the hash value of the file
- location - the path to the original file

The `get.environment` function returns a data frame including information about the execution environment, such as the architecture and operating system on which the script was executed, the version of R, and the modification and execution times of the script.

Two functions provide information about the R libraries used. The `get.libs` function returns the name and version of each library, and whether it was loaded by the script, loaded before the script ran, or loaded by rdtLite code. The `get.func.lib` function returns the name of each function called from a library and the library from which it came.

Other functions provide information about the R statements executed and the edges between nodes. See the package's help page for a complete list of the functions and what they do.

The provSummarizeR, provDebugR and provExplainR tools all use provParseR to extract the information they need from the JSON file.

provGraphR

The **provGraphR** package provides an API that allows a tool developer to make lineage queries over provenance, as provDebugR does. To get started, the tool developer calls the `create.graph` function.

```
You entered:
dir1 = prov_factorial_2021-03-31T12.01.36EDT
dir2 = prov_factorial_2021-04-26T16.34.16EDT
SCRIPT CHANGES: The content of the main script factorial.R has changed
Run prov.diff.script to see the changes.
### dir1 main script factorial.R was last modified at: 2021-03-31T11.58.03EDT
### dir2 main script factorial.R was last modified at: 2021-03-31T11.58.21EDT

LIBRARY CHANGES:
Library version differences:
      name dir1.version dir2.version
      base        4.0.0        4.0.5
  datasets        4.0.0        4.0.5
   ggplot2        3.3.2        3.3.3
  graphics        4.0.0        4.0.5
 grDevices        4.0.0        4.0.5
   methods        4.0.0        4.0.5
     stats        4.0.0        4.0.5
     utils        4.0.0        4.0.5

Libraries in dir2 but not in dir1: No such libraries were found
Libraries in dir1 but not in dir2:
         name version
        dplyr   1.0.0
   provDebugR     1.0
 provExplainR     1.0

INPUT FILE CHANGES:
No input files were found in dir 1
No input files were found in dir 2

ENVIRONMENT CHANGES: Value differences:
Attribute: language version
### dir1 value: R version 4.0.0 (2020-04-24)
### dir2 value: R version 4.0.5 (2021-03-31)
Attribute: scriptHash
### dir1 value: c6b976a5ba662833323d56543817671b
### dir2 value: 426ecf01ebab431cdcbb000a20c3e273
Attribute: total elapsed time
### dir1 value: 1.483
### dir2 value: 1.752
Attribute: working directory
### dir1 value: /Users/blerner/Documents/workspace/factorial-1
### dir2 value: /Users/blerner/Documents/workspace/factorial-2
Attribute: provenance directory
### dir1 value: /Users/blerner/tmp/prov/prov_factorial_2021-03-31T12.01.36EDT
### dir2 value: /Users/blerner/tmp/prov/prov_factorial_2021-04-26T16.34.16EDT
Attribute: provenance collection time
### dir1 value: 2021-03-31T12.01.36EDT
### dir2 value: 2021-04-26T16.34.16EDT

PROVENANCE TOOL CHANGES: Tool differences: No differences have been detected
```

Figure 7: Output from prov.explain describing the differences found in the provenance of two executions of factorial. Items referenced as dir1 refer to the first execution, while items referenced as dir2 refer to the second execution. In this case, the significant differences are differences in the factorial script, the library versions, and the version of R. Other less significant differences that are identified include when the script was executed, the time it took the script to execute, the directory in which the script was executed, and the directory in which the provenance is stored.

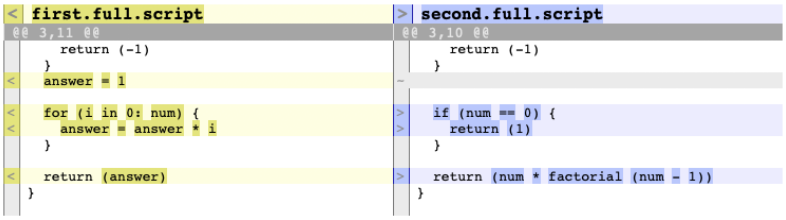


Figure 8: Comparing scripts using provExplainR.

```
create.graph(prov.input = NULL, isFile = TRUE)
```

The `create.graph` function uses the igraph package to calculate an adjacency matrix representation of the graph. The value returned by `create.graph` can be used as an argument to the `get.lineage` function to perform lineage queries. As with `prov.parse`, the default behavior is for `prov.input` to be the path to a JSON provenance file and for `isFile` to be TRUE. Alternatively, `prov.input` can be a string containing JSON provenance if `isFile` is `FALSE`.

The `get.lineage` function computes either backward or forward provenance.

```
get.lineage(adj.graph, node.id, forward = FALSE)
```

Its `node.id` parameter is the unique id assigned to each node in the graph. Using parser functions, such as `get.input.files`, `get.output.files`, `get.variables.set`, and `get.variables.used`, a tool developer can find the id of a file or variable and then obtain its lineage.

These functions provide information about how input data is used or how the values stored in an output file or a plot were computed. The return value is a vector of node ids identifying the nodes in the lineage. The functions return complete lineage, so backward provenance traces back to input files or constants, while forward lineage traces to output. This function underlies the various trace and lineage functionality provided in provDebugR.

5 Limitations

There are two techniques used to capture provenance, each with its own limitations.

First, provenance information concerning files that are read or written is done by using R's `trace` function. Specifically, we trace the low-level I/O functions provide by R, such as `writeLines`, `write.table`, `readLines`, and `read.table`, as well as I/O functions from the vroom package. We also trace plotting functions provided by the **grDevices** package, like `pdf`, and functions from the ggplot2 package, like `ggsave`. Any I/O function built on top of any traced functions will effectively be traced. However, I/O functions that instead use an external library to do the actual I/O will not be traced. It is not difficult to add new functions to trace, but it requires a modification to rdtLite for that to happen.

Second, statement-level provenance is captured by parsing each statement to find the variables used and set and then executing the statement to capture the values of variables that are modified. Each top-level statement is executed atomically. As a result, an if-statement, loop, or a function call is executed as a unit. While I/O information is captured internally to these, provenance at the level of variables is not captured on a line-by-line basis internally to these programming constructs. Provenance collection slows down the execution of scripts, and collecting more detailed provenance seems prohibitive, although it does limit the usefulness of provDebugR, in particular.

For a similar reason, a statement that uses the pipe operator is also executed as a unit. The variables used within pipes, and the final value computed by a statement that uses pipes is captured. However, the intermediate values passed through the pipe are not captured.

rdtlite may misidentify some expressions as variables when non-standard evaluation is used. For example, in the statement

```
cars6Cyl.df <- subset(allCars.df, cyl == 6)
```

`cyl` is not a variable, but rather the name of a column in the `allCars.df` data frame. In order to know that `cyl` is not a variable, rdtLite would need to know how the `subset` function evaluates its parameters. There is no general purpose way of determining this. Handling this situation would require creating a list of known functions and which parameters use non-standard evaluation. rdtLite does not do this currently.

Finally, rdtLite captures values associated with R's base types. However, it has not been extensively tested with the various class systems supported by R.

6 Related work

There are many systems that collect provenance and several excellent survey papers on provenance systems (Freire et al., 2008; Herschel et al., 2018; Pimentel et al., 2019). Provenance collection is common in workflow systems where it is built directly into the execution environment, such as in Kepler (Altintas et al., 2006), VisTrails (Koop et al., 2013), and Taverna (Missier et al., 2008). Of particular interest is the work of de Oliveira et al. (2014) who use provenance to debug long-running workflows, and Why-Diff (Thavasimani et al., 2019) which compares provenance of multiple workflow executions to find differences. Provenance collection in programming languages is much less common, with the exception of the noWorkflow (Murta et al., 2014) implementation for Python.

There has been previous work on collecting provenance for R. Much of this work collects provenance at the level of files. The rctrack package (Liu and Pounds, 2014) uses R's `trace` function to record information about files read and written and the computing environment. It saves copies of data files and scripts with the goal of being able to reproduce a computation. Similarly, **recordr** (Slaughter et al., 2018) records information about files read and written and the computing environment. It can also save copies of those files.

The **CodeDepends** (Lang et al., 2019), trackr (Becker et al., 2017), and **histry** (Becker et al., 2017) packages coordinate to provide insights and records of code execution similar to how **rdtLite** and its associated tools work. The techniques used to collect provenance and the functionality built on top of the collected provenance are different, however. The **CodeDepends** package collects dependency information from R code based on static analysis of the code, rather than through execution. The **histry** package tracks expression evaluation and weaving as with RMarkdown. The **trackr** package (Becker et al., 2017) captures the provenance of plots created by a script. Metadata about how a plot is created comes from the dependencies and provenance gathered by **CodeDepends** and **histry**. The plots can later be discovered by performing searches on the metadata.

The **adapr** package (Gelfond et al., 2018) stores hash values of data files with the R code in a GitHub repository. They assume the data themselves are stored elsewhere. Their goal is to be able to confirm that data match the data used by the code. If the data are modified, the modification will be observable, but the original data cannot be restored by adapr.

While these R provenance systems collect valuable information useful for archiving data provenance, they do not produce the fine-grained provenance needed for debugging. In contrast, **CXXR** (Silles and Runnalls, 2010; Runnalls and Silles, 2012) computes fine-grained provenance using a modified R interpreter where the read-eval-print loop is modified to collect provenance. The collected provenance is available interactively but is not stored persistently. This type of provenance can be helpful for debugging but does not support archiving the provenance.

In contrast to these, **rdtLite** saves information persistently about file inputs and outputs that is useful for archival purposes and saves fine-grained provenance useful for debugging. The E2ETools also build on top of this provenance to provide useful functionality to the user and provide building blocks to enable more tools to be built. Since the JSON provenance format is language-agnostic, the same provenance tools should be usable for different programming languages, and we are currently working on supporting Python by translating provenance collected by noWorkflow (Murta et al., 2014) into the E2ETool JSON format.

7 Conclusions and future work

Data provenance contains a wealth of information. Although provenance initially was thought of as documentation to bolster trust in the data, it has many uses beyond that. In particular, fine-grained provenance offers rich opportunities to develop tools that can be helpful for debugging, learning how a script works, maintaining scripts, and porting scripts to new environments.

Reproducibility as a Service (RaaS) (Wonsil, 2021), a web-based reproducibility tool, strongly benefits from collecting and using provenance data. This tool automatically constructs a computational environment in a Docker container for a given set of R scripts and the data they analyze. It then executes all the scripts, collecting provenance with **rdtLite** and saving all the results to a Docker image. The resulting provenance currently allows RaaS to build a report for its users and situates it perfectly to use the E2ETools in the future. For example, it could use **provSummarizeR** to generate its reports. If researchers want to compare the RaaS execution to their initial execution on their machine, RaaS could integrate **provExplainR** for easy comparisons. Finally, RaaS could also incorporate **provDebugR** to allow users to step through the execution of the scripts entirely within their browser without needing an R session or even downloading the data.

Our collaborators have used a variant of **provDebugR** to explore asynchronous collaboration between data scientists. This variant, called the Multilingual Provenance Debugger (MPD) (Yoo et al.,

2021), is not tied to the R language. Instead, it works on provenance for any language that exports to the same PROV-JSON format as **rdtLite**. An experimental feature in MPD allows users to record and annotate a debugging session as a trace to send to another collaborator, who can replay the trace step-by-step or view the whole session as a pretty-printed markdown file. We could implement similar features in **provDebugR** and extend it to include a visualization component.

Finally, another avenue for future work is the semi-automatic generation of model cards, an artifact that Mitchell et al. (2019) proposed to increase transparency for machine-learning models. One of our current collaborations includes contributions to the open-source Tribuo machine-learning library (Pocock, 2021), which contains a built-in provenance collection system focused on machine-learning provenance. Using the provenance that Tribuo generates, our collaborators built a feature to automatically generate the technical details for model cards and provide support for annotations to supplement the data on the card. We can bring a variant of this feature back into the R ecosystem as an extension of **provSummarizeR**, either directly for machine learning in R or, more generally, to build an 'analysis card' or 'script card.' As these ongoing projects demonstrate, collecting provenance is just the beginning. Developing software that builds on collected provenance to support reproducibility, understanding, and enhancement of software is the long-term goal of this work.

Acknowledgements

This work was supported by NSF grants DEB-1237491, DBI-1459519, and SSI-1450277, the Charles Bullard Fellowship program at Harvard University, and a faculty fellowship from Mount Holyoke College. This paper is a contribution of the Harvard Forest Long-Term Ecological Research (LTER) program.

The authors acknowledge intellectual contributions from the following students: Shaylyn Adams, Vasco Carinhas, Marios Dardas, Andrew Galdunski, Connor Gregorich-Trevor, Nicole Hoffler, Jennifer Johnson, Siqing (Alex) Liu, Erick Oduniyi, Antonia Oprescu, Luis Perez, Moe Pwint Phyu, Katerina Poulos, Garrett Rosenblatt, Cory Teshera-Sterne, Sofiya Toskova, Morgan Vigil, and Yujia Zhou.

1 Appendix: Extended Prov JSON format

The provenance collected by **rdtLite** uses a JSON format that extends the Prov JSON format defined by W3C W3C (2014). The W3C Prov JSON format was designed to capture workflow involving multiple activities with information flowing between them. An activity might be performed by a piece of software, or by a person. The detailed provenance captured by **rdtLite** has activities that are at the level of R statements, with the data being files and variables. The extensions use the same schema as defined by W3C, encoding the provenance data as described below.

Prov JSON has three types of elements: entities, agents, and activities. In the extended JSON used by rdtLite, information about data, libraries, and functions, as well as the runtime environment are encoded as entities. The tool used to collect the provenance is encoded as an agent. Information about statements is encoded as activities.

Prov JSON provides many types of relationships. In the extended JSON, just four of these are used. The `wasInformedBy` relationship is used to represent edges connecting statement elements. Specifically, these edges capture control flow information. The `wasGeneratedBy` relationship connects a statement element to the data elements that it generates, such as a variable that is modified, or a file that is output. The `used` relationship is used to connect a data element to the statement elements that uses the data, such as a variable used within a statement or a file input by a statement. The `used` edge also is used to record what functions are used by each statement. The `hadMember` relationship records which library each function comes from.

See `https://github.com/End-to-end-provenance/ExtendedProvJson/blob/master/JSON-format.md` for more details about this format.

Bibliography

I. Altintas, O. Barney, and E. Jaeger-Frank. Provenance collection support in the Kepler scientific workflow system. In *Proceedings of the International Provenance and Annotation Workshop*, pages 118–132, Chicago, May 2006. Springer-Verlag. [p155]

G. Becker, S. E. Moore, and M. Lawrence. trackr: A framework for enhancing discoverability and

reproducibility of data visualizations and other artifacts in r, 2017. URL https://arxiv.org/abs/1706.04440. [p155]

R. A. Becker and J. M. Chambers. Auditing of data analyses. *SIAM Journal of Scientific and Statistical Computing*, 9:747–760, 1988. [p141]

D. de Oliveira, F. Costa, V. Silva, K. Ocaña, and M. Mattoso. Debugging scientific workflows with provenance: Achievements and lessons learned. In *in Proceedings of the 29th SBBD*, pages 67–76, Brazil, October 2014. [p155]

J. Freire, D. Koop, E. Santos, and C. T. Silva. Provenance for computational tasks: A survey. *Computing in Science and Engineering*, 10(3):11–21, May/June 2008. [p155]

J. Gelfond, M. Goros, B. Hernandez, and A. Bokov. A system for an accountable data analysis process in R. *The R Journal*, 10(1):6–21, July 2018. [p155]

M. Herschel, R. Diestelkämper, and H. B. Lahmar. A survey on provenance: What for? What form? What from? *VLDB Journal*, 2018. [p155]

D. Koop, J. Freire, and C. T. Silva. Enabling reproducible science with vistrails. In *First Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE1)*, Denver, CO, November 2013. [p155]

D. T. Lang, R. Peng, D. Nolan, and G. Becker. CodeDepends. https://github.com/duncantl/CodeDepends, 2019. [Online; accessed 19-July-2022]. [p155]

B. Lerner, E. Boose, and L. Perez. Using introspection to collect provenance in R. *Informatics*, 5(12), 2018. URL http://www.mdpi.com/2227-9709/5/1/12/htm. [p141, 144]

B. S. Lerner and E. R. Boose. Poster: RDataTracker and DDG Explorer — capture, visualization and querying of provenance from R scripts. In *Proceedings of the International Provenance and Annotation Workshop*, Cologne, Germany, June 2014a. [p148]

B. S. Lerner and E. R. Boose. RDataTracker: Collecting provenance in an interactive scripting environment. In *Proceedings of 6th USENIX Workshop on the Theory and Practice of Provenance (TaPP '14)*, Cologne, Germany, June 2014b. [p144]

Z. Liu and S. Pounds. An R package that automatically collects and archives details for reproducible computing. *BMC Bioinformatics*, 15(138), 2014. [p155]

P. Missier, S. Embury, and R. Stapenhurst. Exploiting provenance to make sense of automated decisions in scientific workflows. In *Provenance and Annotation of Data and Processes: Second International Provenance and Annotation Workshop, IPAW 2008*, number 5272 in Lecture Notes in Computer Science, pages 174–185, Salt Lake City, Utah, June 2008. Springer-Verlag. [p155]

M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 220–229, 2019. [p156]

L. Murta, V. Braganholo, F. Chirigati, D. Koop, and J. Freire. noworkflow: Capturing and analyzing provenance of scripts. In *Proceedings of IPAW 2014*, Cologne, Germany, June 2014. [p155]

National Academies of Sciences, Engineering, and Medicine. *Reproducibility and Replicability in Science*. National Academies Press, Washington, DC, 2019. [p141]

J. F. Pimentel, J. Freire, L. Murta, and V. Braganholo. A survey on collecting, managing, and analyzing provenance from scripts. *ACM Comput. Surv.*, 52(3), June 2019. [p155]

A. Pocock. Tribuo: Machine learning with provenance in java, 2021. URL https://arxiv.org/abs/2110.03022. [p156]

A. Runnalls and C. Silles. Provenance tracking in R. In *Proceedings of the 4th International Conference on Provenance and Annotation of Data and Processes*, IPAW'12, pages 237–239, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-34221-9. doi: 10.1007/978-3-642-34222-6_25. [p155]

C. A. Silles and A. R. Runnalls. Provenance-awareness in R. In *Proceedings of the 3rd International Conference on Provenance and Annotation of Data and Processes*, pages 64–72, 2010. [p155]

P. Slaughter, M. B. Jones, C. Jones, and L. Palmer. recordr, 2018. URL https://github.com/NCEAS/recordr. [p155]

P. Thavasimani, J. Cała, and P. Missier. Why-diff: Exploiting provenance to understand outcome differences from non-identical reproduced workflows. *IEEE Access*, 2019. [p155]

W3C. The PROV-JSON Serialization. https://openprovenance.org/prov-json/, 2014. [Online; accessed 18-July-2022]. [p145, 156]

J. Wonsil. *Reproducibility as a service*. PhD thesis, University of British Columbia, 2021. URL https://open.library.ubc.ca/collections/ubctheses/24/items/1.0398221. [p155]

J. Yoo, A. Li, and J. Wonsil. Multilingual provenance debugger, 2021. URL https://github.com/jyoo980/MultilingualProvenanceDebugger. [p155]

Barbara Lerner
Mount Holyoke College
Computer Science Department
South Hadley, MA 01075
United States of America
blerner@mtholyoke.edu

Emery Boose
Harvard University
Harvard Forest
Petersham, MA 01366
United States of America
boose@fas.harvard.edu

Orenna Brand
Columbia University
New York, NY 10027
United States of America
o.brand@columbia.edu

Aaron M. Ellison
Sound Solutions for Sustainable Science
Boston, MA 02135
United States of America
aaron@ssforss.com

Elizabeth Fong
Mount Holyoke College
Computer Science Department
South Hadley, MA 01075
United States of America
fong22e@mtholyoke.edu

Matthew Lau
University of Hawaii West Oahu
Sustainable Community Food Systems Program
Division of Social Sciences
91-1001 Farrington Hwy, Kapolei, HI 96707
United States of America
mklau3@hawaii.edu

Khanh Ngo
Mount Holyoke College
South Hadley, MA 01075
United States of America
ngo22k@mtholyoke.edu

Thomas Pasquier
University of British Columbia
Department of Computer Science

2366 Main Mall #201, Vancouver, BC V6T 1Z4
Canada
tfjmp@cs.ubc.ca

Luis A. Perez
Harvard College[6]
Massachusetts Hall
Cambridge, MA 02138
United States of America
nautilik@deepmind.com

Margo Seltzer
University of British Columbia
Department of Computer Science
2366 Main Mall #201, Vancouver, BC V6T 1Z4
Canada
mseltzer@cs.ubc.ca

Rose Sheehan
Mount Holyoke College
South Hadley, MA 01075
United States of America
sheeh22r@mtholyoke.edu

Joseph Wonsil
University of British Columbia
Department of Computer Science
2366 Main Mall #201, Vancouver, BC V6T 1Z4
Canada
jwonsil@cs.ubc.ca

[6]Primary contributions while at Harvard College. Now at DeepMind.

remap: Regionalized Models with Spatially Smooth Predictions

by Jadon Wagstaff and Brennan Bean

Abstract Traditional spatial modeling approaches assume that data are second-order stationary, which is rarely true over large geographical areas. A simple way to model nonstationary data is to partition the space and build models for each region in the partition. This has the side effect of creating discontinuities in the prediction surface at region borders. The regional border smoothing approach ensures continuous predictions by using a weighted average of predictions from regional models. The R package **remap** is an implementation of regional border smoothing that builds a collection of spatial models. Special consideration is given to distance calculations that make **remap** package scalable to large problems. Using the **remap** package, as opposed to global spatial models, results in improved prediction accuracy on test data. These accuracy improvements, coupled with their computational feasibility, illustrate the efficacy of the **remap** approach to modeling nonstationary data.

1 Introduction

When observations exhibit spatial autocorrelation, geographic location can be leveraged to improve predictions of the response variable by considering responses in nearby observations. Typical spatial statistical models assume that the covariance between two observations can be modeled as a function of location difference, i.e., the relationship needs to be second-order stationary. For sufficiently large distances, the stationarity assumption often fails. Even when external drift or secondary variables are used, continental scale models may still not be second-order stationary.

The simplest way to use traditional spatial modeling with nonstationary data is to partition the global region into smaller sub-regions that are locally stationary and create separate models for each region. The naïve implementation of this approach leads to noncontinuous predictions at the borders of each region. Previous attempts to smooth out discontinuities at region boundaries often involved taking weighted averages of local regional model output where the weights of each local model prediction or covariance structure are a function of the distance between a new observation and the centers of each region (Fuentes, 2001; Fuentes and Smith, 2001; Gosoniu et al., 2006, 2009; Konomi et al., 2014). While the center based approach may be appropriate for symmetrical regions, it is likely not appropriate for oddly shaped or disjoint regions not well represented by their centers. In some cases, a region may not even contain its center. The center-based approach also fails to respect major geographic features that can cause sharp changes in response variables over very short distances.

The regional border smoothing approach described in this article is a novel method that uses a weighted average of predictions from local models, but gives weight to regional models based on the nearest distance to the *border* of each region rather than distance to the center of each region. With this method, regions may be chosen that more naturally reflect local climate and topography with smoothing only occurring near the borders of each region. Figure 1 is an example of the regional border smoothing approach applied to three regions with different spatial models.

The R package **remap** is a General Public License implementation of this regional border smoothing method that scales well to large problems (Wagstaff, 2022). Using **remap**, regional border smoothing is applied to two different modeling problems. The first problem is a national set of 50-year ground snow loads and the second problem is modeling April 1st snow water content for Utah. Mapped values from both projects show improvement in model accuracy using the regional border smoothing approach over global models for a variety of spatial modeling approaches.

The body of this article will proceed with a description of the regional border smoothing approach. This will be followed by an illustration of the available functions and tools in **remap** as well as two demonstrations of the software on a state and national-level data set. These examples show the utility of **remap** in producing smooth estimates when applied to spatial modeling problems over large geographical areas with irregularly shaped partitions.

Background

There are many proposed methods to model nonstationary data using locally stationary models. Haas (1990b,a) describes a moving window approach where only data within a pre-specified bounding box are used to fit a local dependence structure and then make predictions. This process is computationally intensive and may not result in continuous predictions.

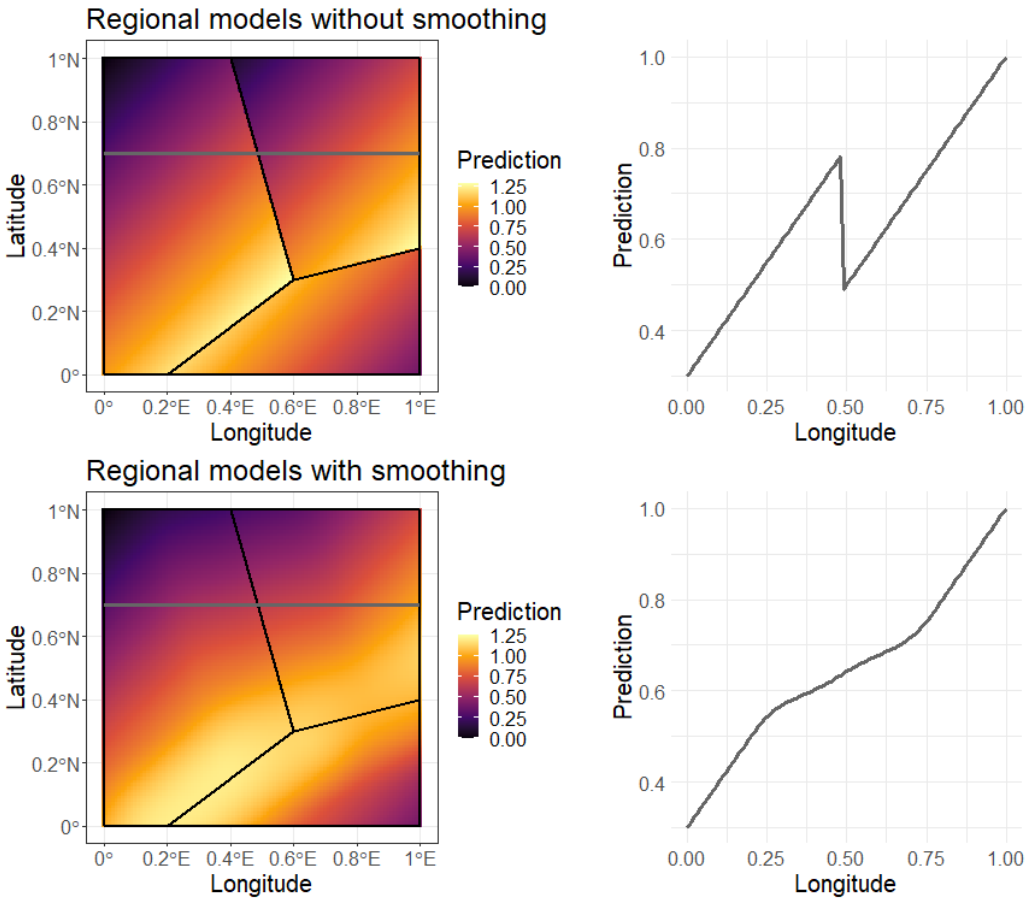


Figure 1: With (bottom) and without (top) application of regional border smoothing where predictions are a linear combination of longitude and latitude. The gray line shows predicted values at 0.7°N. Left region predicts $lon - lat + 1$, bottom region predicts $lat - lon + 1.4$, and right region predicts $lon - lat + 0.7$. Smoothing zone is 30 km ($\approx 0.27°$).

Fuentes (2001) and Fuentes and Smith (2001) propose a method for nonstationary problems where a global covariance structure that changes continuously as a function of location is used in a Gaussian process model. The data are first partitioned into locally stationary regions and a global covariance structure is calculated by taking a weighted average of regional covariance structures. Weights are based on the distance from the prediction location to a point in each region, usually the center. Regions are either given *a priori* or by using subgrids chosen using the Bayesian information criterion.

Applications of local partition modeling approaches include Kim et al. (2005), who describe a method to deal with sudden changes in spatial covariance structure that occur between layers of rock strata. The spatial domain is partitioned into independent regions using Voronoi Tessellations (Green and Sibson, 1978), with each region fit using an independent Gaussian model. The resulting global model has sharp changes at the borders of each region, which was desirable given the context of the problem. Konomi et al. (2014) illustrate a decision tree based method for partitioning the spatial domain when modeling global Ozone levels. Heaton et al. (2017) use a hierarchical clustering method to partition the spatial domain for temperature data in Houston, TX. The hierarchical clustering method has the benefit of creating a partition that more naturally follows changes in the covariance structure rather than partitioning the space into symmetrical blocks or spheres.

Gosoniu et al. (2006, 2009) provide an additional application of local partitioning models mapping malaria risk using *a priori* partitions of West Africa. The 2006 study uses three large rectangular regions and the 2009 study uses agro-ecological regions to partition the spatial domain. In these studies, spatial random effects are modeled as a weighted sum of regional stationary effects based on the distance to region centroids. The authors note problems with sudden changes at region borders as a result of using region centroids for the weighted sum of effects.

Most of the methods discussed so far create a global covariance structure from local covariance structures of Gaussian process models (Fuentes, 2001; Fuentes and Smith, 2001; Kim et al., 2005; Konomi et al., 2014; Heaton et al., 2017). Each method has a different way of smoothing regional transitions that are specifically tied to their methodology. Many of the methods use a Bayesian framework that requires computationally expensive Markov chain Monte Carlo simulations (Kim et al., 2005; Konomi et al., 2014; Heaton et al., 2017; Gosoniu et al., 2006, 2009). Many of the techniques are only applied to purely spatial data rather than multivariate data (Kim et al., 2005; Konomi et al., 2014; Heaton et al., 2017). There is a lack of methodology that allow for smooth transitions between partitions and works for multiple modeling techniques.

The regional border smoothing method described in this article can be thought of as stitching together images to form a larger image with no sharp changes. The approach is similar to those used to combine black and white images from microscopes into a larger image (Thévenaz and Unser, 2007). Individual microscope images are aligned and the overlapping regions are smoothed by taking a weighted average of the overlapping pixel brightness. The weights are based on the distance from the pixel to the outer edge of each image with more weight being applied to pixels closer to the center of an image.

The process described in this article provides a simple way of combining regional model predictions to form a continuous global prediction surface. The method can be applied to problems that are not strictly spatial. For example, Osborne and Suárez-Seoane (2002) show that building models for partitioned space can improve the accuracy of large scale species distribution models. The regional border smoothing method works for any model that produces continuous predictions.

Computer code availability and competing interests

The **remap** package is available on the Comprehensive R Archive Network (see `https://cran.r-project.org/web/packages/remap/index.html`) with the most current version available at `https://github.com/jadonwagstaff/remap`. The code and data used to generate the results in this article are available as supplementary materials.

The figures and tables in this article are made using the R programming language. Figures are created with the **tidyverse** (Wickham et al., 2019), **gridExtra** (Auguie, 2017), **cowplot** (Wilke, 2020), and **maps** (Becker et al., 2021) packages. The **sf** (Pebesma, 2018), **nngeo** (Dorman, 2022), and **raster** (Hijmans, 2022) packages are used to manipulate spatial data. Kriging models are built with **automap** (Hiemstra et al., 2009) and **gstat** (Pebesma, 2004; Gräler et al., 2016) and generalized additive models are built with **mgcv** (Wood, 2011). A docker container with these packages installed and all code is available at `https://hub.docker.com/r/jadonwagstaff/remap_manuscript_code`.

This research was funded by the American Society of Civil Engineers and the Structural Engineering Institute (award number 202827). The authors have no affiliation with any organization with a direct or indirect financial interest in the subject matter discussed in the manuscript. This article is based on the first author's master's thesis (Wagstaff, 2021).

2 The regional border smoothing approach

Regional border smoothing is the process of using regional models to make predictions that are globally continuous. Regional border smoothing may be used on any spatially referenced data ($\boldsymbol{X}$) in conjunction with a modeling approach and a finite set of regions $\mathcal{R}_p$ where each $\mathcal{R}_{p_i} \in \mathcal{R}_p$, $i = 1 \dots m$, is a closed set of points contained in the region of interest $\mathcal{R}$. The intention is that $\mathcal{R}_p$ is a set of non-overlapping polygons with shared borders and $\bigcup_{i=1}^{m} \mathcal{R}_{p_i} = \mathcal{R}$, but these are not necessary conditions. While $\mathcal{R}_p$ does not meet the strict definition of a partition, $\mathcal{R}_p$ will be referred to as a partition throughout this article.

The border smoothing approach described in this article was originally designed for regression-based models, but can be used with any modeling approach that produces a continuous response. This technically includes classification techniques that make continuous probability predictions prior to classifying based on probability threshold, such as logistic regression. Presumably, the predictions from a chosen modeling approach will result in continuous predictions as a function of location.

Modeling regions are defined by the borders of $\mathcal{R}_p$. The data contained by a modeling region $\mathcal{R}_{p_i}$ are used to inform a distinct regression model ($f_{p_i}(\boldsymbol{X})$) for that region. In some cases it may be desirable to include observations near each $\mathcal{R}_{p_i}$ when building regional regression models. In these cases, data within $\mathcal{R}_{p_i}$ and within a buffer zone around each modeling region are used to build each $f_{p_i}(\boldsymbol{X})$ (Figure 2). Using points within a buffer zone that extends beyond the region boundaries avoids edge extrapolation when using a regional model for interpolation in the smoothing zones of neighboring regions.

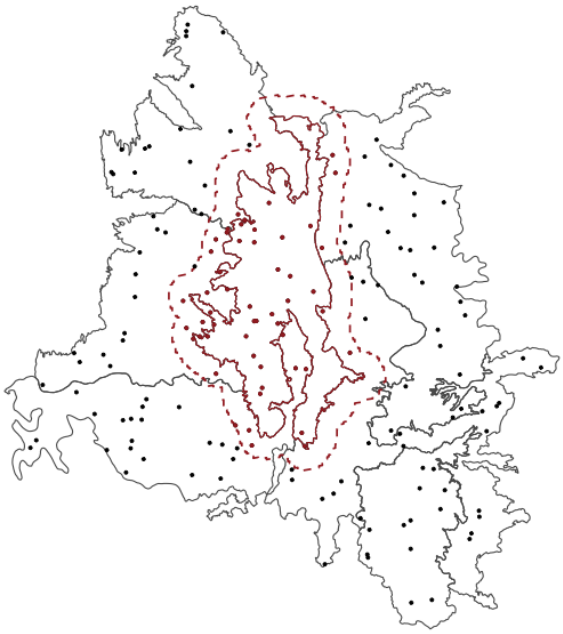


Figure 2: Example showing how observations are selected within a region's buffer zone. Seven arbitrary regions represented as polygons contain 169 observations represented as points. During the regional modeling process, a model is built for each of the seven regions. For this example, all points within 50km of a region are used to build each model. For the highlighted region, all observations within a 50km buffer zone (dashed line) are used to build that region's model. Highlighted points are observations used to build the highlighted region's model.

Simply making predictions within each $\mathcal{R}_{p_i}$ using each corresponding $f_{p_i}(\boldsymbol{X})$ results in noncontinuous predictions at region boundaries. Regional border smoothing results in continuous predictions for the entire space by taking a weighted average of the predictions provided by each $f_{p_i}(\boldsymbol{X})$. The smoothed global prediction surface is continuous, but not necessarily differentiable.

Smoothing at borders

Let $\hat{y}_1, \hat{y}_2, ..., \hat{y}_m$ represent predictions from regional models $f_{p_1}(\boldsymbol{x}^*), f_{p_2}(\boldsymbol{x}^*), ..., f_{p_m}(\boldsymbol{x}^*)$ for location of interest $\boldsymbol{x}^*$ where $\boldsymbol{x}^*$ represents both spatial and non-spatial information. A final prediction $\hat{y}'$ is calculated by the weighted average of each $\hat{y}$, i.e.,

$$\hat{y}' = \frac{\sum_{i=1}^{m} w(d_i|S)\hat{y}_i}{\sum_{i=1}^{m} w(d_i|S)}. \tag{1}$$

The weights are calculated based on the smallest great-circle distances $(d_1, d_2, ..., d_m)$ between the location of $\boldsymbol{x}^*$ and the boundaries of $\mathcal{R}_{p_1}, \mathcal{R}_{p_2}, ..., \mathcal{R}_{p_m}$. If $\boldsymbol{x}^*$ is located within a region, the distance between $\boldsymbol{x}^*$ and that region is 0. The weight $w(d_i|S)$ given to $\hat{y}_i$ is non-zero when d_i is within some threshold S, i.e.,

$$w(d_i|S) = \begin{cases} \left(\frac{S-d_i}{S}\right)^2 & d_i \leq S \\ 0 & d_i > S. \end{cases} \tag{2}$$

Consider now the special case when a prediction is made in region j and $d_{i\neq j} > S$, then $w(d_{i\neq j}|S) = 0$ and Equation 1 reduces to $\hat{y}' = \hat{y}_j$. As prediction location in region j approach $\mathcal{R}_{p_k}$, then the weight of $\hat{y}_k$ increases gradually and $\hat{y}' = \frac{\hat{y}_j + ((S-d_k)/S)^2 \hat{y}_k}{1+((S-d_k)/s)^2}$. At the border of regions j and k, $\hat{y} = \left(\hat{y}_j + \hat{y}_k\right)/2$. Finally, as prediction locations progress into $\mathcal{R}_{p_k}$, weights for y_{i_j} decrease gradually to zero and $\hat{y}' = \frac{\left((S-d_j)/S\right)^2 \hat{y}_j + \hat{y}_k}{\left((S-d_j)/S\right)^2 + 1}$. All locations within S of a region are referred to as the smoothing zone for that region.

Standard error approximations

The smoothing approach described in this paper represents a spatially weighted ensemble of model predictions along region boundaries, with each model (possibly) estimating prediction standard error (SE) or variance. There is no consensus in the literature on how SE should be calculated for ensemble model predictions. The methods for SE calculation that do exist tend to be specific to model type (e.g. Wager et al. (2014)) or averaging approach (e.g. Hoeting et al. (1999)). For this reason, the method for estimating SE in **remap** relies on the general properties of variance for the summation of random variables.

Suppose that each $\hat{y}_i$ is an unbiased estimate from of $f_{p_i}(\boldsymbol{x}^*)$ with the SE of $\hat{y}_i$ represented as $\hat{\sigma}_i$. Then the combined model SE for $\hat{y}'$ can be represented as

$$\hat{\sigma}' = \sqrt{\frac{\sum_{i=1}^{m} w(d_i|S)^2 \hat{\sigma}_i^2 + \sum_{i=1}^{m} \sum_{j<i} 2w(d_i|S)w(d_j|S)\rho_{ij}\hat{\sigma}_i\hat{\sigma}_j}{\left(\sum_i w(d_i|S)\right)^2}}, \tag{3}$$

where ρ_{ij} represents the correlation between predictions for model i and j. One primary difficulty with estimating $\hat{\sigma}'$ in the smoothing zones is a lack of information regarding ρ_{ij}. It is highly likely that SE estimates are correlated for models in adjacent regions. In the absence of a computationally efficient and theoretically robust estimate for ρ_{ij} the Cauchy–Schwarz inequality can be used to provide an upper bound on $\hat{\sigma}'$ given as

$$\hat{\sigma}' \leq \sqrt{\frac{\sum_{i=1}^{m} w(d_i|S)^2 \hat{\sigma}_i^2 + \sum_{i=1}^{m} \sum_{j<i} 2w(d_i|S)w(d_j|S)\hat{\sigma}_i\hat{\sigma}_j}{\left(\sum_i w(d_i|S)\right)^2}}, \tag{4}$$

where ρ_{ij} has been replaced with a constant value of one.

The **remap** package provides a method for estimating the upper bound of the prediction SE using the inequality specified in (4). It is important to note that it may not always be appropriate to combine model SE estimates in this way. This is particularly true for models that ignore spatial autocorrelation. It is also important to remember that the primary focus of **remap** is to provide a practical approach for smoothing model predictions, and not to preserve the theoretical integrity of the SE estimates. It is up to the end user to determine that the SE calculations are valid and appropriate for their modeling purposes.

A note on smoothing

The "smoothing" described throughout this article refers to smoothing in the colloquial sense. A continuous prediction surface is created with a steady transition between regions. The prediction surface is not always differentiable. If a region is not convex, the rate of change in the distance to a region can shift suddenly at locations that are equidistant to different parts of the region. Figure 3 shows an example of a prediction surface near a non-convex region where the prediction surface is not differentiably smooth.

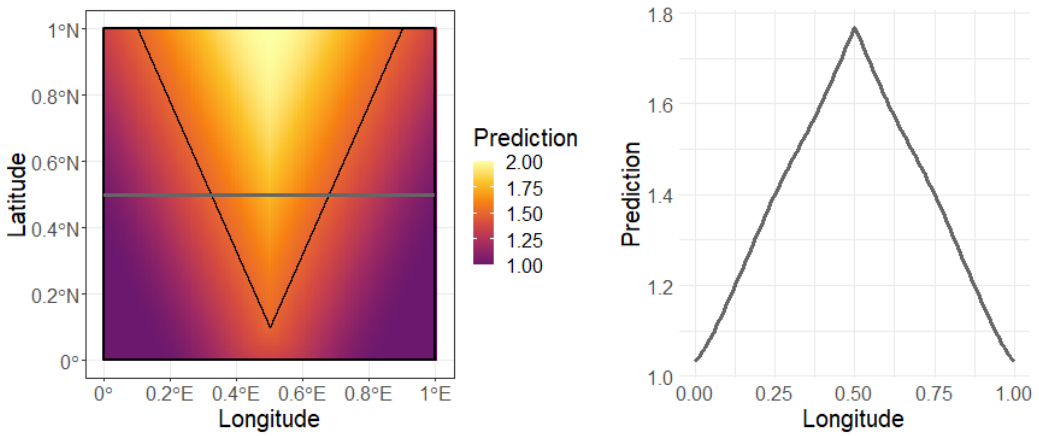


Figure 3: Example of where predictions are not differentiable near non-convex regions. The bottom region predicts a constant value of one and the top region predicts a constant value of two with a smoothing zone of 40 km (≈ 0.36°). The gray line shows predicted values along the 0.5°N transect. This example illustrates that the smoothing approach always produces continuous predictions, but differentiability is dependent on the shape of the polygon partition.

Since each region is a closed set of points, d_i is continuous. It therefore follows that the weight function $w(d_i|S)$ described in Equation 2 is continuous as $\lim_{x\to 0^+} w(x|S) = w(0|S) = 1$ and $\lim_{x\to S^-} w(x|S) = w(S|S) = 0$ for $x \in [0,\infty)$ and $S > 0$. Since the right side of Equation 1 is the multiplication and addition of continuous functions, it is guaranteed to have $\max(w(d_i|S)) > 0$ as long as $\boldsymbol{x}_s^*$ is within S units of any $\mathcal{R}_p$ and it follows that Equation $\hat{y}'$ is also continuous. Given continuous predictions as a function of location for each region, the regional border smoothing method is guaranteed to be continuous for any location within the smoothing zone of at least one region. Once all $d_i > S$, then the denominator of y' is equal to zero, which means y' is no longer well-defined. As long as all of the data $\boldsymbol{X}$ are located within $\mathcal{R}_p$, any prediction location outside of $\mathcal{R}_p$ is spatial extrapolation that is generally discouraged.

3 remap

The R package **remap** is an implementation of the regional border smoothing approach to spatial modeling. The function `remap` creates a set of regionalized models given:

- A set of observations as spatially projected or geographic points.
- A set of regions as spatially projected or geographic polygons.
- A desired buffer zone distance.
- A modeling function to apply to observations in each region.

Predictions can be made on new observations given a regionalized model and the smoothing parameter `smooth` used for weighted averages in smoothing zones (variable S in Equation 2). Detailed descriptions of function parameters can be found in the package documentation. Some working examples using the **remap** package are provided via the vignette which accompanies the package. The development version of the code for **remap** can be found at https://github.com/jadonwagstaff/remap.

Calculating distances

The weights for regional predictions require the distances from all observations to the boundary of each region. The modeling process of the `remap` function also requires the distances from all observations to

each region to assign the correct observations to each regional model. The process of fine-tuning a model can result in recalculating these distances many times. To avoid recalculating distances, the function `redist` is included in the **remap** package to pre-compute the distances from a given set of observations to a given set of regions. These pre-computed distances can be supplied as a parameter to the `remap` function to reduce computational time while fine-tuning models.

Calculating distances in the **remap** package takes advantage of tools already available for spatial analysis in the R package **sf** (Pebesma, 2018). The function `sf::st_distance` is used to find either Euclidean or great-circle distances depending on the spatial projection of the locations and regions. The `sf::st_distance` function uses the **S2** geometry library (Google, 2022) when calculating great-circle distances. The **S2** geometry library is designed specifically to efficiently compute distances between nearby objects given large sets of geographic data. Regardless, calculating distances can still take a lot of computational time if the regions are complex and/or there are a lot of observations.

The naïve approach to regional border smoothing is to find all distances between each location and each region. This approach may be necessary during the modeling process if any region does not contain a required minimum number of observations (detailed in the implementation considerations section). If predictions are being made using new observations, then distances do not need to be calculated between every observation and every region.

Because the weight for predictions in different regions is zero when the distance to those regions is greater than the smoothing parameter (Equation 2), distances do not need to be calculated between *every* region and *every* prediction location. To determine which observations require distance calculations, an approximate polygon is constructed that encompasses the original region plus the smoothing zone around the region. The function `sf::st_within` is used to determine which observations are within the new polygon and equivalently, which observations are within the smoothing zone of the original region. Observations in the approximate polygon become candidates for precise distance calculations.

An approximate polygon that contains a region and the region's smoothing zone is created using the `sf::st_buffer` function. For geographic coordinates, `sf::st_buffer` requires a buffer value in degrees. Since the great-circle distance between a unit degree of longitude changes with latitude, Equation 5 is used to find c: the shortest distance (measured in kilometers) required to move one degree longitude at the observation in the data set that is nearest to a geographical pole. The set $\boldsymbol{\lambda}$ is the set of Latitude values of the observations and 6380 km is the radius of the earth at a pole (rounded down). A sufficient buffer value in degrees for `sf::st_buffer` is calculated by dividing the smoothing zone length in km by c. The resulting buffered polygon contains all of the observations within the smoothing zone of the original polygon,

$$c = \frac{\pi * 6350}{180} * cos\left(\frac{\pi * \max\left(|\boldsymbol{\lambda}|\right)}{180}\right). \tag{5}$$

Assuming each region contains a sufficient number of observations to fit each regional model, the same process used for reducing the number of distance calculations for model predictions can be used to reduce the number of distance calculations required for model training. The function `redist` is able to restrict distance calculations to only observations within a certain buffer zone or smoothing zone of each region using the `max_dist` parameter. This eliminates the need to calculate distances to points with a known weight of zero when smoothing.

Practical implementation considerations

Reliable regression results depend upon sufficient sample sizes, which differ based on the variability of the response and dimensionality of the inputs. A strict (and obvious) minimum sample size for the **remap** function is one observation per region, though this threshold will rarely, if ever, ensure reasonable results. An additional parameter `min_n` is included in the `remap` function to specify a minimum number of observations be used to build each model. If a region and the buffer around that region do not contain the minimum number of observations specified, the `min_n` observations closest to the boundaries of the region are used to train that region's model.

The **remap** package has the ability to make predictions outside of all modeling regions. If the prediction location is within the smoothing zone of a region, then Equation 1 applies and the nearest region will have the most weight. If the prediction location is outside the smoothing zone of all regions, then the model from the closest region is used to make a prediction. This may result in non-continuous transitions in predictions when the closest region changes across geographic space, but this is only possible at locations outside of the smoothing zone of all regions. As a general rule, extrapolation of predictions to a location beyond those represented in the input data is not recommended. See Figure 4 for a visual depiction of how predictions behave outside of all modeling regions.

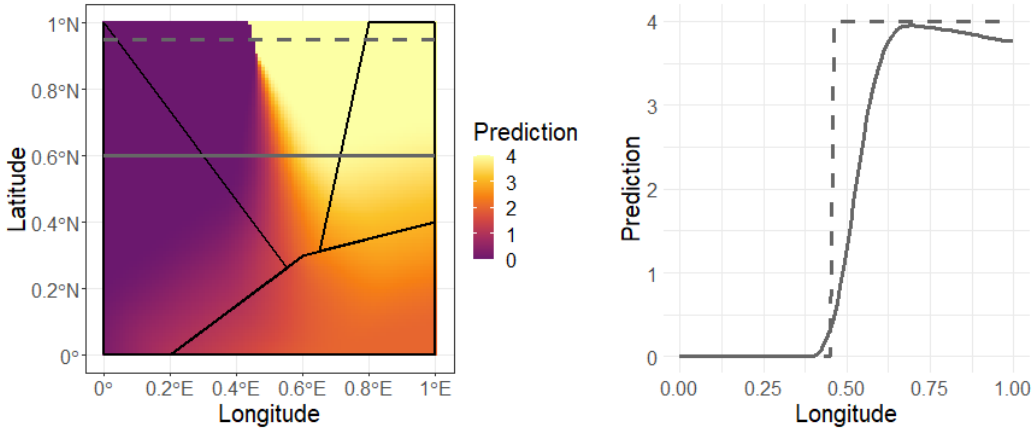


Figure 4: Example of what happens when predictions are made outside of any modeling region. The gray line shows predicted values along the 0.6°N transect. The dashed gray line shows predicted values along the 0.95°N transect. The left region predicts a constant 0, the bottom region predicts a constant 2, and the right region predicts a constant 4. Smoothing zone is 35 km ($\approx 0.32°$).

The ability to predict outside of all regions means that the **remap** package can make smooth predictions across small gaps at polygon borders, provided the gaps are no larger than two times the smoothing zone distance. This encourages the use of `rgeos::gSimplify` (Bivand and Rundel, 2021) or `sf::st_simplify` (Pebesma, 2018) to simplify complex polygons and ease the computational burden associated with distance calculations, even if those simplifications slightly compromise the topology of the original geometry.

4 Applications

Design snow loads

Snow loads are obtained from weather stations throughout the United States in the National Oceanic and Atmospheric Administration's (NOAA) Global Historical Climatology Network (Menne et al., 2012). Snow loads are either measured directly, or estimated from measured snow depth using a depth to load conversion model. Engineers have historically used estimates of 50-year ground snow loads when designing structures. The 50-year snow load is traditionally obtained by fitting a probability distribution to yearly maximum snow load measurements and extracting the 98^{th} percentile. A recent effort by the American Society of Civil Engineers has resulted in a new set of 50-year ground snow loads at 7964 measurement locations (Bean et al., 2021).

Building a geospatial snow load model with remap

Building design requirements call for continuous maps that estimate design loads between measurement locations. This has historically been accomplished using various mapping techniques (Tobiasson et al., 2002; Liel et al., 2017; Bean et al., 2019). The problem is that the relationship between predictor variables and snow load can change drastically on a continental scale. For example, the typical loads at 1500 meter elevation in the Rocky Mountains of Montana are much lower than loads at 1500 meters in the Cascade Mountains of Washington State (Figure 5). Commonly used geospatial models for mapping are not well suited for the nonstationary nature of this problem.

Snow loads are typically assumed to share a log-linear relationship with elevation. This relationship can be modeled directly using ordinary least squares (OLS), which ignores any potential spatial dependencies among the observations. A generalized additive model (GAM) built with the **mgcv** R package (Wood, 2011) characterizes the log of 50-year loads as a function of elevation and a spatial smoother called splines on the sphere (Wood, 2003). Universal kriging interpolates values using a Gaussian process model after accounting for the log-linear trend in elevation. Variograms are individually fit within each region using the **automap** R package (Hiemstra et al., 2009).

Geographic regions defined by the US Environmental Protection Agency (EPA) define regions with similar ecology and climate called eco-regions (Commission for Environmental Cooperation, 1997). The eco-regions provide a natural partition of the conterminous United States and give no regard to political boundaries. Snow load can be modeled using observations within each eco-region where

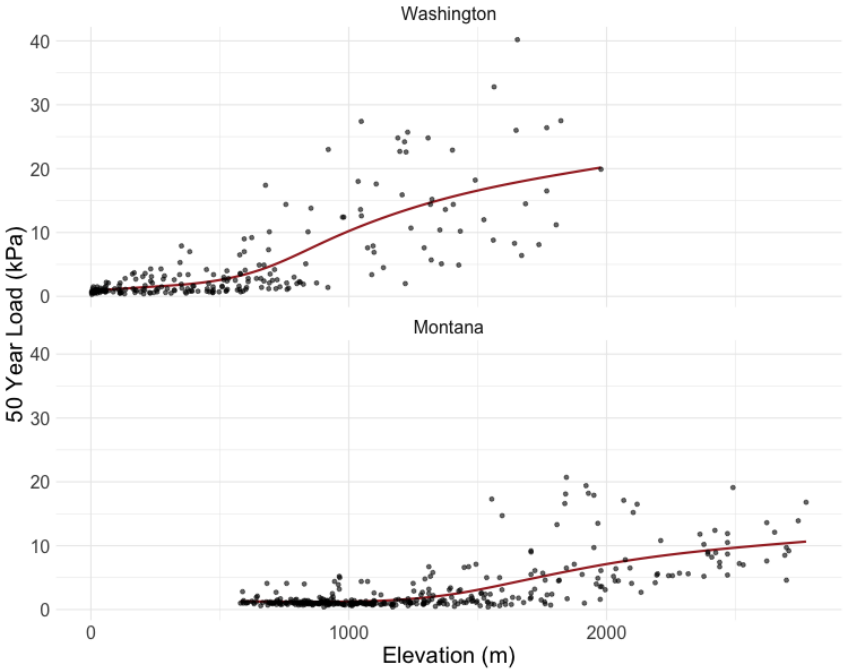


Figure 5: 50-year events for measurement locations within Washington and Montana. Trend lines are fit with cubic regression splines. In both regions, loads eventually increase as elevation increases, but the inflection points are about 1000 m apart and the rate of change is different. This demonstrates the need for an approach which adapts to the changing relationship between predictor variables and response in different regions.

the relationship between predictor variables and the response are more consistent on a local level. The **remap** package facilitates modeling in separate eco-regions and creates a smooth model on the national scale. There are 86 eco-regions that fall within the conterminous United States, so 86 separate models are built using the `remap` function with a buffer zone of 50 km and a `min_n` of 150 observations. The regional models are smoothed to a single continuous model using a `smooth` parameter of 25 km.

The following code demonstrates how `remap` is used to build these models. The script and data for the following examples are provided as supplementary materials with this article. The example data include three `"sf"` objects: a spatial points data frame with 50-year snow loads at locations within the US and Canada (`loads`), a spatial polygons data frame of eco-regions (`eco3`), a spatial polygon of the conterminous United States (`cont_us`), and a `"raster"` object with elevations comprising a 0.8 km grid over the conterminous United States (`grd`).

```
library(tidyverse)
library(sf)
library(raster)
library(mgcv)
library(automap)
library(gstat)
library(remap)
load("wagstaff-bean.RData")
```

The eco-regions have more complex borders than is necessary for this problem, so they can be simplified to the same extent as shown in Figure 6.

```
eco3_simp <- eco3 %>%
  sf::st_simplify(dTolerance = 10000) %>%
  dplyr::filter(!sf::st_is_empty(.)) %>%
  sf::st_cast("MULTIPOLYGON")
```

Since multiple models are built with the same set of measurement locations, pre-calculating the distances from measurement location to region boundaries can save some computational time.

```
eco3_dist <- redist(loads, regions = eco3_simp, region_id = ECO3)
```

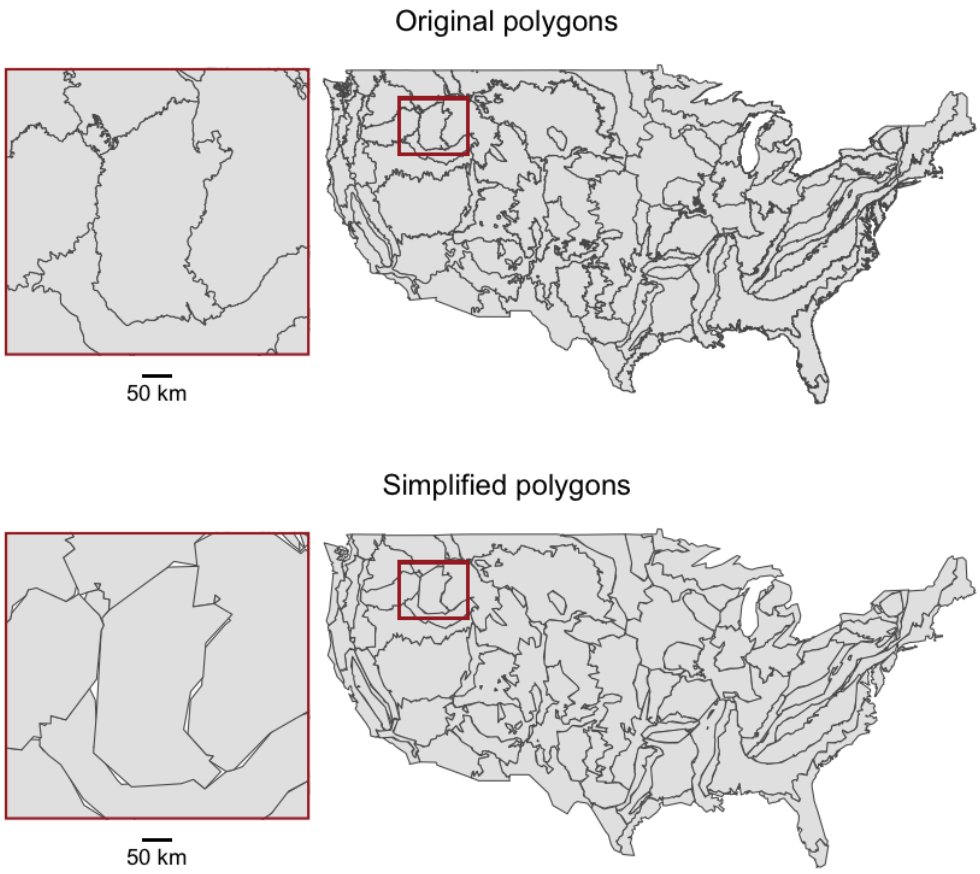


Figure 6: Results of simplification of eco-regions. The highlighted region is an area with some of the most severe gaps between polygons. Notice that the width of the gaps are still much smaller than two times the smoothing parameter (2×25 km).

Linear models and GAMs are easiest to build with remap. The `remap` function accepts additional arguments for a selected model function but passes the spatial data to the function as an unnamed parameter. Since the `data` parameter is the second parameter of the `stats::lm` function, we need to formally specify the first argument of the function (i.e. the `formula` parameter) to pass along to `stats::lm`, rather than relying on the `stats::lm` function defaults. Otherwise the data for each sub-region will be sent as the wrong parameter.

```
lmod <- remap(loads, regions = eco3_simp, region_id = ECO3,
              buffer = 50, min_n = 150,
              distances = eco3_dist,
              model_function = stats::lm,
              formula = log(EVENT50) ~ ELEVATION)

lmod
#> remap model with 86 regional models
```

The `lmod` object has class `"remap"` and contains a `models` object and a `regions` object. The `models` object is a list of `gam` models where the model names correspond to each modeling region ID. The `regions` object is an `"sf"` multipolygon object with one row per region and the first column is the region IDs.

```
head(names(lmod$models), 3)
#> [1] "10.1.2" "10.1.3" "10.1.4"

class(lmod$models[[1]])
#> [1] "lm"

head(lmod$regions, 3)
```

```
#> Simple feature collection with 3 features and 1 field
#> Geometry type: MULTIPOLYGON
#> Dimension:     XY
#> Bounding box:  xmin: -121.3721 ymin: 40.15434 xmax: -105.4847 ymax: 49.39082
#> Geodetic CRS:  WGS 84
#>     ECO3                           geom
#> 1 10.1.2 MULTIPOLYGON (((-115.8435 4...
#> 2 10.1.3 MULTIPOLYGON (((-120.5262 4...
#> 3 10.1.4 MULTIPOLYGON (((-108.8632 4...
```

Similar to the `stats::lm` function, the `mgcv::gam` function needs a `formula` parameter and a `family` parameter passed to the `remap` function since `data` is the third parameter of `mgcv::gam`. In this example a wrapper function for `mgcv::gam` is written to allow a custom predict function which either return predicted values (`se.fit = FALSE`) or standard errors (`se.fit = TRUE`).

```
gam_wrap <- function(data, ...) {
  model <- mgcv::gam(data, ...)
  class(model) <- "gam_wrap"
  return(model)
}

predict.gam_wrap <- function(object, data, se.fit = FALSE) {
  class(object) <- "gam"
  if (se.fit) { # return standard errors
    return(predict(object, data, se.fit = TRUE)$se.fit)
  } else { # return predictions
    return(predict(object, data))
  }
}

gm <- remap(loads,
            regions = eco3_simp, region_id = ECO3,
            buffer = 50, min_n = 150,
            distances = eco3_dist,
            model_function = gam_wrap,
            formula = log(EVENT50) ~ s(ELEVATION, k = 15) +
              s(LATITUDE, LONGITUDE, bs = 'sos', k = 75),
            family = stats::gaussian)

predict(gm, loads[1:3, ], smooth = 25)
#> [1] -0.003297382  0.132882034 -0.170530457

predict(gm, loads[1:3, ], smooth = 25, se = TRUE, se.fit = TRUE)
#> Upper bound for standard error calculated at each location.
#> Reminder: make sure that the predict function outputs a vector of standard
#> error values for each regional model in your remap object.
#> [1] 0.3177003 0.3383906 0.2142151
```

The kriging example requires a bit more work to implement in **remap** since the `automap::autoKrige` and `gstat::krige` functions do not accept data as an `"sf"` object. A custom modeling function can be written to pass to **remap** and a custom predict function can be written for **remap** to use for predictions.

```
projection <- sf::st_crs("+proj=laea +x_0=0 +y_0=0 +lon_0=-100 +lat_0=45")

krig <- function(data, formula) {
  data <- data %>%
    sf::st_transform(projection) %>%
    sf::as_Spatial()

  out <- list(data = data, formula = formula)
  class(out) <- "krig"

  return(out)
}
```

```
predict.krig <- function(object, data) {
  if (nrow(data) == 0) return(NULL)
    data <- data %>%
      sf::st_transform(projection) %>%
      sf::as_Spatial()

    variogram_object <- automap::autofitVariogram(
      formula = object$formula,
      input_data = object$data,
      model = "Sph")

    k <- gstat::krige(formula = object$formula,
                      locations = object$data,
                      newdata = data,
                      model = variogram_object$var_model,
                      debug.level = 0)

    return(k$var1.pred)
}

kg <- remap(loads,
            regions = eco3_simp, region_id = ECO3,
            buffer = 50, min_n = 150,
            distances = eco3_dist,
            model_function = krig,
            formula = log(EVENT50) ~ ELEVATION)
```

Since the buffer distance is greater than the smoothing distance, the resulting "remap" object can be used to interpolate the same way that a global kriging model interpolates between points.

```
loads_us <- sf::st_filter(loads, cont_us)

kg_preds <- exp(predict(kg, loads_us, smooth = 25))

all(dplyr::near(kg_preds, loads_us$EVENT50))
#> [1] TRUE
```

Snow load modeling methods are compared in Table 1 using national scale models and models built with the **remap** package on EPA Level III Eco-Regions (Commission for Environmental Cooperation, 1997). Table 1 shows that the **remap** package framework improves the cross validated accuracy of every spatial modeling method. This demonstrates that the **remap** package has the ability to generally improve modeling results, as improvements are not isolated to a single spatial modeling case. Even though the regional models are made up of 86 different regional models, the estimated 50-year loads are smooth across the prediction surface.

Model	MSE $\times 10^2$		Improvement
	National	Regional	
GAM	8.1	5.5	32%
Kriging	7.2	5.9	18%
OLS	89.3	17.8	80%

Table 1: Ten fold cross-validation results from modeling the log of 50-year snow loads. Mean squared error (MSE) is multiplied by 10^2 for readability. Improvement is (national - regional) / national.

Computational challenges

National snow load maps require a sufficiently fine resolution to be feasibly used to design buildings in topographically complex areas. To accomplish this, snow load maps are created that match the resolution of PRISM climate output (PRISM Climate Group, 2020), which maps at a 0.8 km resolution for the conterminous United States. This means that predicted snow loads are calculated at 12,113,556 locations using a model created by the **remap** package with 86 different regions. Elevation values

for the grid are obtained from the United States Geological Survey (United States Geological Survey, 2020a).

Since the **remap** package uses the distances between prediction locations and regions to make a smooth model, calculating these distances is a potential bottleneck in the prediction process. Also, since the prediction area is so large, geographic coordinates rather than projected coordinates must be used to find distances. Prior to **sf** version 1.0.0 (Pebesma, 2018), the **S2** geometry library (Google, 2022) had not been implemented. Without the **S2** geometry library, simply calculating all geographic distances would have taken nearly half a year to run on a typical personal computer, as illustrated in Table 2. The test computer used for Table 2 has 16 GB of RAM and an Intel Core I7-7820HQ CPU which runs at 2.9 GHz. Maps for the national design snow loads were developed before **S2** had been integrated into **sf**, so the steps described later in this section had previously been necessary to use **remap** to build these maps.

Remedial steps	Run time in hours		
	Geographic	Geographic (**S2**)	Projected
None	4200*	8.1*	100.0*
Simplify polygons	14*	2.3*	1.0
+ set `max_dist` to 25 km	1.1	0.3	0.8
+ run in parallel on 4 cores	0.8	0.3	0.8

Table 2: Run times for `redist` to calculate geographic and projected distances from 0.8 km grid points to 86 different eco-regions (polygons) in the conterminous US. Remedial steps are additive, i.e., each row in the table includes all previous remedial steps. * Approximate run time based on random sample of 10,000 grid points.

The first step to reduce the run time for distance calculations is to simplify the polygons as discussed in the implementation considerations. Using a tolerance of 10000 with the function `sf::st_simplify`, the size of the polygons are reduced from 21 MB to 0.2 MB. The results of polygon simplification are visualized in Figure 6. When making predictions, the `smooth` parameter is set to 25 km; therefore, the `max_dist` parameter of `redist` can be set to 25 km to further reduce run time. The distances may also be calculated in parallel; however, on the test computer the improvements are marginal. The relevant distances may be calculated for all points using great circle distance in as fast as 20 minutes (Table 2). This tremendous computational speed-up is made possible by the ability of the **remap** package to smoothly fill in the gaps that occur in aggressively simplified polygons.

Here is an example of the code used to find distances from grid points to eco-regions. Due to space and time constraints, the 0.8 km prediction grid is aggregated to a 7.2 km grid (`grd`) over the conterminous United States. The grid is also converted to an `"sf"` spatial points data frame for use with **remap**.

```
ag_grd <- grd %>%
  raster::aggregate(9) %>%
  raster::rasterToPoints() %>%
  as.data.frame() %>%
  mutate(LONGITUDE = x, LATITUDE = y) %>%
  sf::st_as_sf(coords = c("x", "y"), crs = 4326) %>%
  filter(sf::st_intersects(., cont_us, sparse = FALSE)[,1])

grd_dist <- redist(ag_grd, regions = eco3_simp, region_id = ECO3, max_dist = 25)
```

Smoothed regional predictions are made automatically by using the generic `predict` function on a `"remap"` object and setting the `smooth` parameter to value greater than zero. Below is an example of predicting values on the grid points using the 50-year load regionalized GAM model. Since we already calculated distances from `grd` to `eco3_simp`, the `grd_dist` object can be passed to the `distances` parameter of the `predict` function. The distance matrix is optional and `remap` will calculate distances internally if not provided with a distance matrix. Predictions are smoothed to 25 km on either side of each border. Note that predictions from `gm` are on the log scale.

```
gm_preds <- predict(gm, ag_grd, smooth = 25, distances = grd_dist)
gm_preds <- exp(gm_preds)
```

Here, the predictions from the regionalized GAM reveal some extrapolation issues when making predictions at elevations outside of the range of available data within each sub-region. To get a better visualization of the predictions, the predicted values can be truncated to the range observed in the input data.

```
range(gm_preds)
#> [1]   0.04273189 444.34484622
range(loads$EVENT50)
#> [1]  0.1 40.2

gm_preds[gm_preds > 40.2] <- 40.2
gm_preds[gm_preds < 0.1] <- 0.1
```

The smooth predictions from the gm model are visualized in Figure 7 with the following code.

```
ggplot(cont_us) +
  geom_tile(data = ag_grd %>% dplyr::mutate(EVENT50 = gm_preds),
            aes(x = LONGITUDE, y = LATITUDE, fill = EVENT50)) +
  geom_sf(fill = NA, color = NA) +
  scale_fill_viridis_c(option = "inferno",
                       trans = "log10",
                       breaks = c(0.1, 1, 7, 40),
                       labels = c("0.1 kPa", "1 kPa", "7 kPa", "40 kPa"),
                       name = "50-year event") +
  theme_void()
```

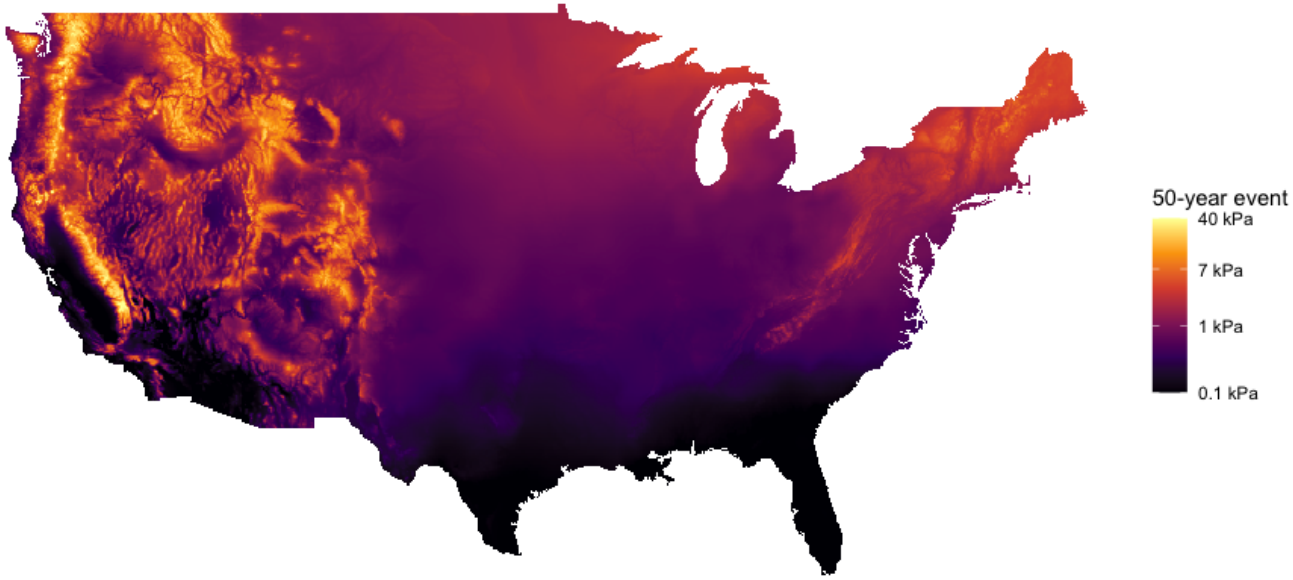


Figure 7: Prediction map for 50-year loads from the regional GAM model built with the **remap** package using 7,936 observations wthin 86 eco-regions. GAM models are built using a buffer zone of 50 m and predictions are smoothed using a smoothing zone of 25 m. Values are restricted to be within the range of values observed in the load data. This figure illustrates how **remap** functions can be used to create continuous predictions of snow loads for all of the Conterminous United States.

Utah snowpack

The state of Utah assesses snow water content or snowpack every April 1st to plan for yearly water resource availability. These measurements are much more variable than the 50-year snow loads since the values are based on a single measurement at each location instead of the 98th percentile of a distribution fit to annual maximum values at each location. This example uses direct measurements of the water content made via Snowpack Telemetry (SNOTEL) stations included in NOAA's Global Historical Climatology Network (Menne et al., 2012). In addition to SNOTEL station data, Snow Course data collected by the National Water and Climate Center are also included in the data set (Natural Resources Conservation Service, 2017). SNOTEL and Snow Course data measure the water equivalent of snow depth (WESD) and are located in the mountainous areas of Utah above 1777 m elevation. This negates the need to estimate water content from snow depth, which is typically the case when measuring snow at non-mountainous locations in the state.

SNOTEL stations and Snow Course data within 100 km of the border of Utah are included in the data set. There are 3511 April 1st WESD observations available for modeling which range from 0 to 1746 mm of water. The observations have 178 unique locations and span 30 years (1986-2015). Each year is modeled separately with no regard for any temporal correlations and each year has at least 97 observations.

Models for GAM, kriging, and OLS are created using the same model structure as the national snow load example, but a different response variable. Since there are 142 zero values for WESD within the data, the log of (WESD + 1) is used as the response variable for the April 1st snowpack. Predictions are also constrained to be between [1, max(WESD)] to avoid excessive extrapolation. The regions used for regional modeling are watershed boundaries defined by the USGS (United States Geological Survey, 2020b). Watersheds are defined by a hierarchy of hydrologic unit codes (HUC) with a two-digit designation for continental scale watersheds (HUC2) and a four-digit designation that partitions each HUC2 region (HUC4). There are four HUC2 regions and 12 HUC4 regions within the boundaries of the state of Utah (see Figure 8). A buffer zone of 20 km, a smoothing parameter of 10 km, and a `min_n` of 30 observations are used when modeling over both HUC2 and HUC4 regions with the `remap` function.

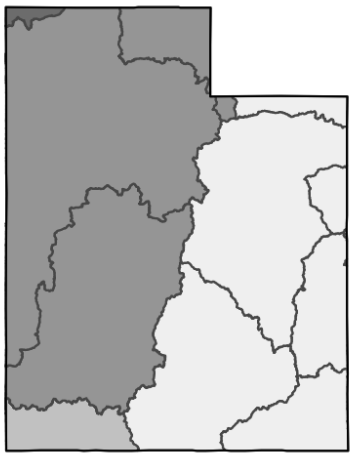


Figure 8: HUC4 regions (lines) and HUC2 regions (shades) in Utah.

The following code shows how **remap** models are built for the 2011 snowpack values using HUC2 watershed regions. Included in the supplied example data are an `sf` points data frame with snowpack Utah snowpack values (`utapr1`, and an `sf` polygons data frame of HUC2 and HUC4 watersheds within Utah (`utws`). Note that the maximum number of knots (`k`) in the GAM formula are reduced to accommodate a smaller `min_n` in the `remap` function.

```
utsp2011 <- utapr1 %>%
  dplyr::filter(YEAR == 2011) %>%
  dplyr::mutate(WESD = WESD + 1)

utlmod <- remap(utsp2011,
                regions = utws, region_id = HUC2,
                buffer = 20, min_n = 30,
                model_function = stats::lm,
                formula = log(WESD) ~ ELEVATION)

utgm <- remap(utsp2011,
              regions = utws, region_id = HUC2,
              buffer = 20, min_n = 30,
              model_function = mgcv::gam,
              formula = log(WESD) ~ s(ELEVATION, k = 5) +
                s(LATITUDE, LONGITUDE, bs = 'sos', k = 20),
              family = gaussian)

utkg <- remap(utsp2011,
              regions = utws, region_id = HUC2,
              buffer = 20, min_n = 30,
```

```
model_function = krig,
formula = log(WESD) ~ ELEVATION)
```

Table 3 shows that the gains in accuracy are more modest in this application than in the national example shown previously. This is expected as the partitions are on the state level instead of the national level. Nevertheless, the consistent improvement in accuracy at various scales using different polygons and different input data highlight the general ability of the **remap** package to improve predictive accuracy without sacrificing continuity.

Model	MSE $\times 10^2$		
	State	HUC2	HUC4
GAM	88	75(15%)	78(11%)
Kriging	90	87 (3%)	83 (8%)
OLS	140	111(21%)	89(36%)

Table 3: Ten fold cross-validation mean squared error from modeling log(WESD + 1) for Utah snowpack. Mean squared error (MSE) is multiplied by 10^2 for readability. Improvement in parentheses is calculated with (state - HUC) / state.

5 Conclusions

Partitioning a space for geospatial modeling is a practical approach for handling nonstationary data; however, smooth transitions in mapped values are a desirable constraint of many projects, such as the design snow load requirements set forth by the American Society of Engineers. The **remap** R package provides a ready-to-use framework for regional models with smooth transitions at region borders that overcomes the computational difficulties of naïve implementations.

The **remap** package creates continuous prediction surfaces by weighting regional predictions based on the proximity to region borders. Smoothing at region borders makes the package particularly equipped to handle irregularly shaped regions not well represented by their geographic centers. Because the **remap** package has the ability to smooth over small gaps formed when simplifying polygons, computation times can be drastically reduced without sacrificing the continuity of the predictions. Methods that automatically subset the number of required distance calculations further reduce computational times. This article has demonstrated accuracy improvements using the **remap** package on two separate data sets with two sets of polygon inputs. These examples highlight the feasibility of applying the **remap** framework to large spatial regression modeling problems.

6 Acknowledgements

The development of **remap** was funded in part by the American Society of Civil Engineers (ASCE) and the Structural Engineering Institute in partnership with several organizations including (in alphabetical order): Factory Mutual, Metal Building Manufacturer's Association, National Council of Structural Engineering Associations, Nucor, Simpson Gumpertz and Heger, the State of Montana, the Steel Deck Institute, the Steel Joist Institute, Structural Engineers Association of Montana, Wiss Janney and Elstner Associates. Further, the authors would like to thank Dr. Kevin Moon of Utah State University for his insight on the ensemble SE estimation problem.

Bibliography

B. Auguie. *gridExtra: Miscellaneous Functions for "Grid" Graphics*, 2017. URL https://CRAN.R-project.org/package=gridExtra. R package version 2.3. [p162]

B. Bean, M. Maguire, and Y. Sun. Comparing design ground snow load prediction in Utah and Idaho. *Journal of Cold Regions Engineering*, 33(3):04019010, 2019. URL https://doi.org/10.1061/(ASCE)CR.1943-5495.0000190. [p167]

B. Bean, M. Maguire, Y. Sun, J. Wagstaff, S. A. Al-Rubaye, J. Wheeler, S. Jarman, and M. Rogers. The 2020 national snow load study. Technical Report 276, Utah State University Department of

Mathematics and Statistics, Logan, UT, Feb 2021. URL https://doi.org/10.26077/200k-pr86. [p167]

R. A. Becker, A. R. Wilks, R. Brownrigg, T. P. Minka, and A. Deckmyn. *maps: Draw Geographical Maps*, 2021. URL https://CRAN.R-project.org/package=maps. R package version 3.4.1. [p162]

R. Bivand and C. Rundel. *rgeos: Interface to Geometry Engine - Open Source ('GEOS')*, 2021. URL https://CRAN.R-project.org/package=rgeos. R package version 0.5-9. [p167]

Commission for Environmental Cooperation. Ecological regions of North America: Toward a common perspective. Technical report, Commission for Environmental Cooperation, 1997. URL http://www3.cec.org/islandora/en/item/1701-ecological-regions-north-america-toward-common-perspective. Accessed: 2020-04-09. [p167, 171]

M. Dorman. *nngeo: k-Nearest Neighbor Join for Spatial Data*, 2022. URL https://CRAN.R-project.org/package=nngeo. R package version 0.4.6. [p162]

M. Fuentes. A high frequency kriging approach for non-stationary environmental processes. *Environmetrics*, 12(5):469–83, 2001. URL https://doi.org/10.1002/env.473. [p160, 162]

M. Fuentes and R. L. Smith. A new class of nonstationary spatial models. Technical report, North Carolina State University, 2001. URL https://repository.lib.ncsu.edu/bitstream/handle/1840.4/213/nonstat.pdf?sequence=1. [p160, 162]

Google. S2, 2022. URL https://s2geometry.io/. Source code. [p166, 172]

L. Gosoniu, P. Vounatsou, N. Sogoba, and T. Smith. Bayesian modelling of geostatistical malaria risk data. *Geospatial Health*, 1:127–39, 2006. URL https://doi.org/10.4081/gh.2006.287. [p160, 162]

L. Gosoniu, P. Vounatsou, N. Sogoba, N. Maire, and T. Smith. Mapping malaria risk in West Africa using a Bayesian nonparametric non-stationary model. *Computational Statistics & Data Analysis*, 53 (9):3358–71, 2009. URL https://doi.org/10.1016/j.csda.2009.02.022. [p160, 162]

P. J. Green and R. Sibson. Computing Dirichlet tessellations in the plane. *The Computer Journal*, 21(2): 168–73, 1978. URL https://doi.org/10.1093/comjnl/21.2.168. [p162]

B. Gräler, E. Pebesma, and G. Heuvelink. Spatio-temporal interpolation using gstat. *The R Journal*, 8: 204–218, 2016. URL https://doi.org/10.32614/RJ-2016-014. [p162]

T. C. Haas. Kriging and automated variogram modeling within a moving window. *Atmospheric Environment. Part A. General Topics*, 24(7):1759–69, 1990a. URL https://doi.org/10.1016/0960-1686(90)90508-K. [p160]

T. C. Haas. Lognormal and moving window methods of estimating acid deposition. *Journal of the American Statistical Association*, 85(412):950–63, 1990b. URL https://doi.org/10.1080/01621459.1990.10474966. [p160]

M. J. Heaton, W. F. Christensen, and M. A. Terres. Nonstationary Gaussian process models using spatial hierarchical clustering from finite differences. *Technometrics*, 59(1):93–101, 2017. URL https://doi.org/10.1080/00401706.2015.1102763. [p162]

P. Hiemstra, E. Pebesma, C. Twenhöfel, and G. Heuvelink. Real-time automatic interpolation of ambient gamma dose rates from the Dutch radioactivity monitoring network. *Computers & Geosciences*, 35(8): 1711–21, 2009. URL https://doi.org/10.1016/j.cageo.2008.10.011. [p162, 167]

R. J. Hijmans. *raster: Geographic Data Analysis and Modeling*, 2022. URL https://CRAN.R-project.org/package=raster. R package version 3.6-3. [p162]

J. A. Hoeting, D. Madigan, A. E. Raftery, and C. T. Volinsky. Bayesian model averaging: a tutorial (with comments by M. Clyde, David Draper and EI George, and a rejoinder by the authors). *Statistical science*, 14(4):382–417, 1999. URL https://doi.org/10.1214/ss/1009212519. [p164]

H.-M. Kim, B. K. Mallick, and C. C. Holmes. Analyzing nonstationary spatial data using piecewise Gaussian processes. *Journal of the American Statistical Association*, 100(470):653–68, June 2005. URL https://doi.org/10.1198/016214504000002014. [p162]

B. A. Konomi, H. Sang, and B. K. Mallick. Adaptive Bayesian nonstationary modeling for large spatial datasets using covariance approximations. *Journal of Computational and Graphical Statistics*, 23(3): 802–29, 2014. URL https://doi.org/10.1080/10618600.2013.812872. [p160, 162]

A. B. Liel, D. J. DeBock, J. R. Harris, B. R. Ellingwood, and J. M. Torrents. Reliability-based design snow loads. II: Reliability assessment and mapping procedures. *Journal of Structural Engineering*, 143(7):04017047, 2017. URL https://doi.org/10.1061/(ASCE)ST.1943-541X.0001732. [p167]

M. Menne, I. Durre, B. Korzeniewski, R. Vose, B. Gleason, and T. Houston. Global historical climatology network - daily, version 3.26, 2012. URL https://doi.org/10.7289/V5D21VHZ. Accessed: 2020-06-04. [p167, 173]

Natural Resources Conservation Service. Snow course stations, 2017. URL https://wcc.sc.egov.usda.gov/reportGenerator. Accessed: 2020-04-03. [p173]

P. E. Osborne and S. Suárez-Seoane. Should data be partitioned spatially before building large-scale distribution models? *Ecological Modelling*, 157(2):249–59, 2002. URL https://doi.org/10.1016/S0304-3800(02)00198-9. [p162]

E. Pebesma. Simple features for R: Standardized support for spatial vector data. *The R Journal*, 10(1): 439–46, 2018. URL https://doi.org/10.32614/RJ-2018-009. [p162, 166, 167, 172]

E. J. Pebesma. Multivariable geostatistics in S: the gstat package. *Computers & Geosciences*, 30:683–691, 2004. URL https://doi.org/10.1016/j.cageo.2004.03.012. [p162]

PRISM Climate Group. 30-year normals, 2020. URL https://prism.oregonstate.edu/normals/. [p171]

P. Thévenaz and M. Unser. User-friendly semiautomated assembly of accurate image mosaics in microscopy. *Microscopy Research and Technique*, 70(2):135–46, 2007. URL https://doi.org/10.1002/jemt.20393. [p162]

W. Tobiasson, J. Buska, A. Greatorex, J. Tirey, and J. Fisher. Ground snow loads for New Hampshire. Technical report, Cold Regions Research and Engineering Laboratory, 2002. URL https://www.senh.org/wp-content/uploads/2010/12/tr02-6.pdf. [p167]

United States Geological Survey. The national map, 2020a. URL https://www.usgs.gov/core-science-systems/national-geospatial-program/national-map. Accessed: 2020-04-01. [p172]

United States Geological Survey. National hydrography dataset, 2020b. URL https://www.usgs.gov/core-science-systems/ngp/national-hydrography/access-national-hydrography-products. Accessed: 2020-10-16. [p174]

S. Wager, T. Hastie, and B. Efron. Confidence intervals for random forests: The jackknife and the infinitesimal jackknife. *The Journal of Machine Learning Research*, 15(1):1625–1651, 2014. URL http://jmlr.org/papers/v15/wager14a.html. [p164]

J. Wagstaff. Regionalized models with spatially continuous predictions at the borders. Master's thesis, Utah State University, 2021. URL https://doi.org/10.26076/f25f-3104. Document No. 8065. [p162]

J. Wagstaff. *remap: Regional Spatial Modeling with Continuous Borders*, 2022. URL https://CRAN.R-project.org/package=remap. R package version 0.3.0. [p160]

H. Wickham, M. Averick, J. Bryan, W. Chang, L. D. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. L. Pedersen, E. Miller, S. M. Bache, K. Müller, J. Ooms, D. Robinson, D. P. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani. Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686, 2019. URL https://doi.org/10.21105/joss.01686. [p162]

C. O. Wilke. *cowplot: Streamlined Plot Theme and Plot Annotations for ggplot2*, 2020. URL https://CRAN.R-project.org/package=cowplot. R package version 1.1.1. [p162]

S. N. Wood. Thin plate regression splines. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 65(1):95–114, 2003. URL https://doi.org/10.1111/1467-9868.00374. [p167]

S. N. Wood. Fast stable restricted maximum likelihood and marginal likelihood estimation of semi-parametric generalized linear models. *Journal of the Royal Statistical Society (B)*, 73(1):3–36, 2011. URL https://doi.org/10.1111/j.1467-9868.2010.00749.x. [p162, 167]

Jadon Wagstaff
Huntsman Cancer Institute, University of Utah
Department of Oncological Sciences
2000 Circle of Hope
Salt Lake City, UT 84112, USA
jadonw@gmail.com

Brennan Bean
Utah State University
Department of Mathematics and Statistics
3900 Old Main Hill
Logan, UT 84322, USA
Brennan.Bean@usu.edu

HostSwitch: An R Package to Simulate the Extent of Host-Switching by a Consumer

by Valeria Trivellone, Sabrina B. L. Araujo and Bernd Panassiti

Abstract In biology a general definition for host switch is when an organism (consumer) uses a new host (which represents a resource). The host switch process by a consumer may happen through its pre-existing capability to use a sub-optimal resource. The **HostSwitch** R package provides functions to simulate the dynamics of host switching (extent and frequency) in the population of a consumer that interacts with current and potential hosts over the generations. The **HostSwitch** package is based on a Individual-Based mock-up model published in FORTRAN by Araujo et al. (2015). The package largely improve the previous mock-up model, by implementing numerous new functionalities such as comparison and evaluation of simulations with several customizable parameters to accommodate several types of biological consumer-host associations, an interactive visualization of the model, an in-depth description of the parameters in a biological context. Finally, we provided three real world scenarios taken from the literature selected from ecology, agriculture and parasitology. This package is intended to reach researchers in the broad field of biology interested in simulating the process of host switch of different types of symbiotic biological associations.

1 Introduction

In several branches of biology (such as for example ecology, evolution, parasitology) a general definition for host-switching (or host shift) is when a consumer uses a newly colonized host, which represents its resource. Different spatial and temporal outcomes may result from the new host-consumer association depending on whether or not the colonization is successful. Studies on the evolution of biotic associations relies on an increasing body of literature covering all prototypical examples of symbiotic relationship categories (Thompson, 2010). Symbiosis *sensu lato* is defined here as any interaction between two organisms of different species. The possible influence between interacting organisms has been placed in a continuum of association types defined by their role, direction, and extension; and it varies from mutual consumption to unidirectional exploitation (Dimijian, 2000a,b). In the interspecific associations, the interacting organisms may play the role of strict consumer (e.g., predator) or resource (e.g., prey). Evolution of the associations may include several concatenated events of speciation affecting one or both species and it is driven by four main processes: cospeciation, host switch, failure to speciate and "missing the boat" (Page, 2002). While the prevalent paradigm considers cospeciation to be the main process driving evolution of most biological associations, recent evidence showed that, given the opportunity, a consumer may use a sub-optimal resource (or host) by host-switching without the need for any genetic innovation. This may explain the rapid origin of novel associations (i.e. colonization of novel hosts at the ecological time scale) eventually followed by speciation (at the evolutionary time) as well the observed incongruences of the paired phylogenies (see Brooks et al. (2019) for a review). Computer simulation modeling provide a valid tool to understand ecological and evolutionary dynamic of interacting species. To theoretically support the importance of host switch events, an Individual-Based Model (IBM) has been proposed by Araujo et al. (2015) (mock-up model hereafter). The model simulates the extent of host-switching in a host-parasite association formalized as the probability of an individual to disperse and successfully colonize a novel host. Recently, Feronato et al. (2021) provided a further add-on to the model by exploring the significance and the interaction of three parameters thought to be of paramount importance for the acquisition of a new host by a parasite. By using simulated data, these two initial papers provided important insights on the dynamic of host-switches for a parasite species. The main results were that host switch on a new host does not require prior evolutionary novelty, pathogens may survive on sub-optimal hosts which results in increased chance of host-switching to hosts more distant, and some parameters facilitate host switching (e.g., mutation and reproductive rate).

Both models were coded and run in FORTRAN language. Although available in an executable version (Windows, Linux, and MacOS), the previous mock-up model still lacks a user-friendly interface, and the manipulation of parameters is broadly limited. The current access to the model is restricted to certain groups of users who know how to compile and code in FORTRAN.

We present here a user-friendly R package, called **HostSwitch**, which improves the earlier version published in FORTRAN (Araujo et al., 2015; Feronato et al., 2021) in three main ways:

(1) by increasing the accessibility of the model to researchers that are not familiar with FORTRAN;

(2) by including customizable parameters, some previously unreleased (e.g., *jump_back*), that allow us to extend the mock-up model from strict pathology (i.e., simulating host switches by pathogen) to

other branches of biology such as ecology, microbiology, agriculture (covering a very broader spectrum of symbiotic *sensu lato* associations, see Implementation and Usage scenarios sections for further details) reaching a broader audience;

(3) by providing in-depth descriptions of the parameters in a biological context, and examples of possible uses with real world data (see Usage scenarios section).

To our knowledge, there are no R packages that simulate the events of host switch using the theoretical approach briefly presented above and widely discussed in previous papers (Araujo et al., 2015; Agosta et al., 2010; Brooks et al., 2019). However, is worth to mention that there are one putatively similar packages that may be used to simulate host switches. Notably, the package **EpiILM** (Vineetha Warriyar et al., 2020) uses discrete-time individual-level models to simulate the dynamic of infection disease transmission. The **EpiILM** package differs from the one present here for a fundamental point: **HostSwitch** simulate the host switch using the point of view of the consumer, rather than the host, and formalize the probability as random encounters of new hosts different to one other. For further details of the model formalize in **EpiILM** are presented in Deardon et al. (2010).

In Section "The model", we describe the mathematical framework for simulating the dispersion and the survival of individuals of a population on a novel resource which describes the event of host-switching. In Section "Implementation", we describe the implementation of the mock-up model in the **HostSwitch** package, including a description of the arguments of the main functions. In Section "Usage scenarios" we used empirical data gathered from the literature to simulate and compare different scenarios of host switch. We also present possible hypotheses and research questions that can be explored using the simulation approach.

2 The model

The simulation model of **HostSwitch** aims to measure the dynamics of host switching (extent and frequency) in the population of an organism (hereafter Consumer) that interacts with current and potential hosts (hereafter Resource) over generations. A successful host switch implies that a Consumer may colonize a new Resource, which in turn imposes selection pressure that impacts the Consumers' survival. The host-switching relies on a mechanism of ecological readjustment or ecological fitting, i.e. the capability of the Consumer to use a similar Resource even if sub-optimal (Janzen, 1985; Agosta and Klemens, 2008). The fundamental aspect of the **HostSwitch** simulation model is to track, summarize and compare the dispersion and successful host switch events in a new Resource by the populations of the Consumer. Although the model and the basic parameters have been previously described in Araujo et al. (2015) and Feronato et al. (2021), we provide here a revised description of the modeling dynamic which accommodates all symbiotic (*sensu lato*) associations.

The Resource is characterized by a real number, p_{Opt}, randomly selected from a uniform distribution ranging from p_{Res_min} to p_{Res_max}. This number represents the optimum phenotype imposed on Consumers by the Resource. Besides, the Resource imposes a carrying capacity K on the Consumer population. Individuals of the Consumer have a phenotype which can evolve over generations due to the emergence of novelties (hereafter mutations). Each individual consumer i is characterized by one phenotype p_i. The simulation starts (generation n = 0) with all M consumer individuals having the same phenotype value ($p_i = p_{Ind}$) which is equal to the average value in the resource range ((p_{Res_min} + p_{Res_max})/2). At each generation, a novel Resource is offered and all Consumers have a probability mig to migrate from the current to the novel Resource. The number of dispersing individuals is calculated by assigning to each individual a value that follows the random uniform distribution. All the individuals with a value lower than mig disperse to the new Resource. Then, the assigned value may be interpreted ecologically as intrinsic or extrinsic characteristics involved in the dispersal event (e.g., morphological features, environmental constraints). The parameter mig defines a criterion for inclusion, with higher values allowing more individuals to disperse at each generation. The dispersion event has two possible outcomes: no migration (m1) and migration (m2).

- $(m1)$ No individual disperses to the novel Resource and the model continues in the current Resource through the selection event. The Resource imposes on each individual a selection and the survival probability is calculated using the following equation:

$$P(p_i, p_{Opt}) = \exp\left[-\frac{(p_i - p_{Opt})^2}{2\sigma_{sel}^2}\right], \tag{1}$$

  where p_i is the phenotype of i-th Consumer attempting to survive on the Resource; p_{Opt} is the optimum phenotype the Consumer should have to maximize survival success on the Resource; σ_{sel} is the standard deviation of a Gaussian distribution which is inversely related to the strength of selection. It may be interpreted as a proxy for the amplitude of the fitness space

of the Consumer, the higher the σ_{sel}, the lower the selection and the higher the probability of surviving. Ecologically this value may be related to the niche breadth for the Consumer.

- $(m2)$ Some individuals disperse and they go through a selection event on the novel Resource following eqn 1 (but with new p_{Opt} values, randomly defined). Here, the survival event has two possible outcomes: no survival (s1) and survival (s2).
  - $(s1)$ None of the individuals survive and the model continues with the portion of individuals that did not migrate and are in the current (original) Resource, if dispersing individuals are not allowed to come back to the current Resource, or with individuals that did not migrate and those jumping back to the current Resource. The model imposes to these individuals a survival probability according to eqn 1.
  - $(s2)$ Some individuals survive and the model continues only with those individuals.

All other individuals are ignored in subsequent steps. The novel Resource then becomes current Resource and the individuals will reproduce with a net reproduction rate b, limited to the carrying capacity K. The offspring's phenotype (inherit from the parents, plus variation δ) is assigned to each individual and calculated using the normal probability function:

$$P(\delta) = \exp\left[\frac{-\delta^2}{2\sigma^2_{mut}}\right], \tag{2}$$

where δ is random phenotypic variation assigned to each individual; σ_{mut} is the standard deviation for mutation. The parameter δ is randomly defined from a Gaussian distribution centered in zero and with a standard deviation σ_{mut}. Offspring phenotypes are equal to the arithmetic average of their parent's phenotypes plus δ (i.e the extent of genetic novelty introduced with the reproduction). The descendants replace their parents and will populate generation n+1 that will start over with another dispersion event to a novel Resource.

The overview of the main steps of the model are summarized in Figure 1.

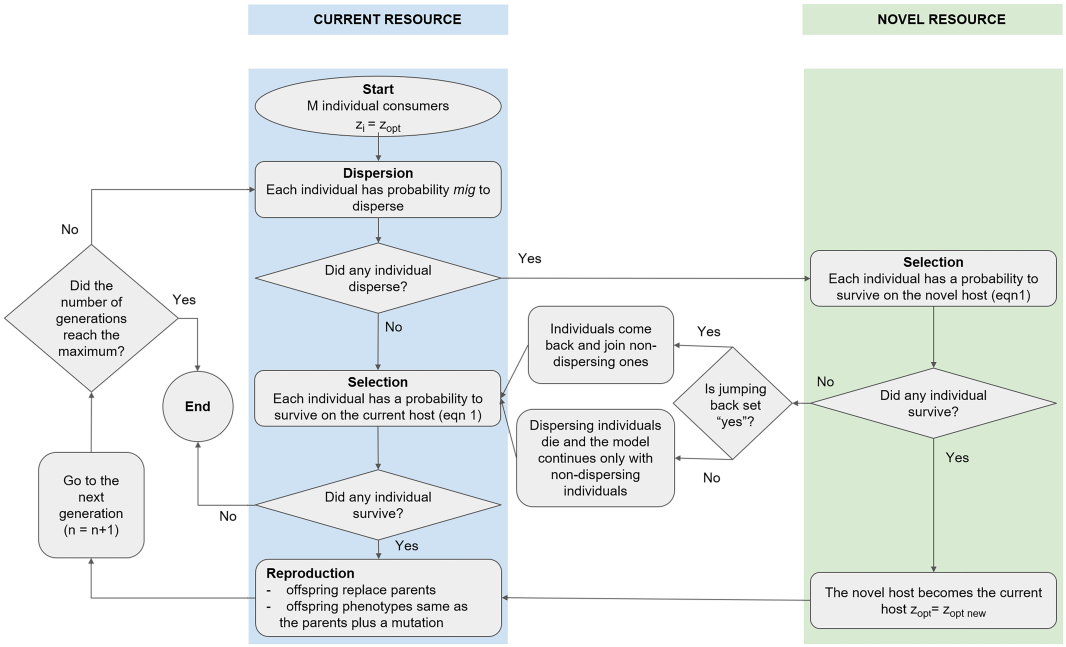


Figure 1: Overview of the host-switching simulation model. At each generation every individual consumer has the chance to disperse to a novel Resource (on the right in green) and colonize it, or to jump back to the original Resource (on the left in blue). Selection (see eqn 1) is imposed by the current and the novel Resource. Individuals that survive the selection start a new generation after the reproduction event (see eqn 2). The framework is largely modified from Figure 1 after Araujo et al. (2015), more details in the text. z_i = phenotype of i-th Consumer; z_{opt} = phenotype that the Consumer should have to fit the novel Resource; $z_{opt\ new}$ = phenotype that the Consumer retains on the current Resource after successful colonization of novel Resource; *mig* = a parameter defining the probability of dispersion of each Consumer.

3 Implementation

The **HostSwitch** R package is available for download from CRAN at https://cran.r-project.org/web/packages/HostSwitch/index.html. Latest versions of future developments of the R-package will be available on Github at https://github.com/berndpanassiti/HostSwitch.

This package provides an easy-to-use toolbox for users who want to simulate the host-switching of consumers and currently includes six functions (Table 1).

The package is installed and loaded by typing:

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

Table 1: Description of the six functions in **HostSwitch** package

Function	Description
`plotHostSwitch`	Plots dispersal and colonization (host-switching events) of Consumers on a novel Resource offered at each generation given the values of parameters related to carrying capacity, fitness space, migration, reproduction, selection, and biological model.
`shinyHostSwitch`	Plots simulations of Consumer's host-switching on interactive web-based front-end using Shiny App.
`simHostSwitch`	Simulates the number of dispersion and successful host switch events by individuals of the Consumer until all individuals die.
`summaryHostSwitch`	Generates summary statistics for the object generated by `simHostSwitch` function.
`survivalProbability`	Includes the formula to calculate the survival probability used in the `simHostSwitch` function.
`testHostSwitch`	Tests the significance of the difference between two objects generated by `simHostSwitch` function.

The **HostSwitch** package is implemented around two main functions: `survivalProbability` and `simHostSwitch`. The former includes the formula of the survival probability (eqn 1), and the latter simulates host switches according to Figure 1 and encompasses dispersion, selection and reproduction events. `simHostSwitch` uses the `survivalProbability` function to calculate the probability of survival of the individuals on the current and new Resources.

The arguments of the `simHostSwitch` function are:

- *K*: carrying capacity, positive integer (min = 1, max = 1000), default value: 100.
- *b*: net reproduction rate; average number of offspring that a population of the Consumer produces at each generation, integer value (min = 0), default value: 10.
- *mig*: define the number of migrating individuals at each generation, numeric value (min = 0, max = 1), default value: 0.01. Larger values indicate higher probability to migrate.
- *sd*: standard deviation for mutation, numeric value (min = 0, max = 10), default value: 0.2. It corresponds to σ_{mut} in (eqn 2).
- *sigma*: standard deviation of the survival function, numeric value (min = 0, max = 10), default value: 1. It corresponds to σ_{sel} in (eqn 1).
- *pRes_min*: smallest optimum phenotype value imposed by the Resource, numeric value (min = 1, max = *pRes_max*), default value: 1.
- *pRes_max*: highest optimum phenotype value imposed by the Resource, numeric value (min = *pRes_min*, max = 100), default value: 10.
- *n_generations*: number of generations, positive integer (min = 1, max = 50000), default value: 200.

- *jump_back*: option for Consumers that do not survive on the novel Resource. If "yes" the consumer(s) is able to jump back to the current Resource and will be considered in the selective pressure and reproduction stage for the n+1 generation, if "no" (default) it cannot jump back and dies on the new host.
- *seed*: a single value useful for creating simulations or random objects that can be reproduced, positive integer (>0), default value: NULL.
- *n_sim*: number of simulations, positive integer (min = 1, max = 50000), default value: 1.
- *nInitConsumer*: propagule size (or number of initial individuals) at the generation n = 0, default value: 20.
- *data*, *column*: instead of the arguments outlined above, the user may pass to the function a vector (*column* =) saved in a matrix or dataframe (*data* =) which specifies all or some of the arguments above, default value: NULL. The default value will be set if the argument is not provided. The arguments provided in dataframe will be overwritten if provided as an argument in the `simHostSwitch` function.

Using the `simHostSwitch` function, the R command below generates an object of class 'HostSwitch' which is a list of 18 elements, six simulated quantities and 12 used parameters (arguments of the function).

```
# generate a simulated 'HostSwitch' object using default values for the parameters
> sim1 <- simHostSwitch(n_sim= 3, seed = 2)
```

The simulated quantities are:

- *pRes_sim*: a vector of the optimum phenotypes (one for each generation) that Consumers should have to be favored by the current Resource;
- *pRes_new_sim*: a vector of the optimum phenotypes (one for each generation) that Consumers should have to be favored by the novel Resource;
- *pInd_sim*: list of vectors that includes the individual phenotype values of the Consumers in the population of each generation;
- *pInd_jump_sim*: vector of number of migrating individuals at each generation. The vector length is always equal to the 'n_generation' parameter, if the simulation ends before the 'n_generation' value then the vector will include a 'NA' by default;
- *pInd_whichjump_sim*: list of vectors that extracts the individual phenotype values of the Consumers who disperse in a novel Resource in each population and generation;
- *pInd_whichsurv_sim*: list of vectors that extracts the individual phenotype values of the Consumers who successful colonize a novel Resource in each population and generation;

The function `summaryHostSwitch` creates a summary of basic statistics for phenotypes, dispersion and host switch events. The argument *warmup* defines the number of generations to be excluded from summary statistics. If *warmup* = 1 the generation at time 0 is excluded from summary, if *warmup* = 2 the generations at times 0 and 1 are excluded and so on. If *warmup* = NULL all generations are considered for summary statistics. Possible values are NULL or positive integer (min=1, max=50), default value = 1.
The example below provides the averages of the quantities of interest calculated for three simulations (*n_sim* = 3)

```
# generate a summary for the simulated 'HostSwitch' object
> summaryHostSwitch(sim1)
An object of class summaryHostSwitch
Summary of HostSwitch simulations

General settings of individual based model:
K:100, b:10, mig:0.01, sd:0.2, sigma:1, pRes_min:1, pRes_max:10
n_generations:200, jump_back:no, seed:2, n_sim:3, warmup:1, nInitConsumer: 20

Summary of phenotypes:
         Min. 1st Qu. Median Mean 3rd Qu. Max.
pRes     3.48    3.94   4.41 4.33    4.75 5.10
pRes_new 5.21    5.33   5.45 5.49    5.64 5.83
pInd     3.68    4.17   4.65 4.64    5.11 5.57

Summary of host switches by consumers:
```

```
                                      Mean Max
Total events of dispersion:         29.67000  43
Number of successful host switches: 12.33333  16
```

The testHostSwitch function measures if there is a significant difference between the means of three simulated quantities of two HostSwitch objects (*simulated_quantities*1 and *simulated_quantities*2). The argument *parameter* selects the quantity to compare, i.e. "j" (total number of dispersing events), "s" (total number of successful host switch events), and "d" (distance between the *pRes_sim* and *pRes_new_sim* for the generations where a successful host switch occurs, hereafter phenotype distance). The argument *test* defines the inferential statistic to be used, "t" parametric (t-test) or "w" nonparametric (wilcoxon rank test). The *warmup* is defined as above, and the argument *plot* provides the box plot of the results based on the *ggplot* function. When comparing quantities, the user needs to take into account all the arguments provided to the function simHostSwitch, including the number of simulations (*n_sim*). A warning message will be produced if *n_sim*=1 for both HostSwitch objects. The following code generates another HostSwitch object (sim2), with *mig* = 0.9 and default values for the others arguments. We compare the quantities from the objects sim1 and sim2 using testHostSwitch:

```
> sim2 <- simHostSwitch(mig = 0.9, n_sim= 3, seed = 3)

# compare the average values of the simulated quantities ("j", "s", "d") saved in the
# objects sim1 and sim2
> testHostSwitch(simulated_quantities1 = sim1, simulated_quantities2 = sim2,
  parameter = "j", test = "t", warmup = NULL, plot = FALSE)
  An object of class testHostSwitch
  Test result comparing 2 HostSwitch simulations using Welch Two Sample t-test
  t: 2.74044, df:2.278635, p.value:0.09663877

> testHostSwitch(simulated_quantities1 = sim1, simulated_quantities2 = sim2,
  parameter = "s", test = "t", warmup = NULL, plot = FALSE)
  An object of class testHostSwitch
  Test result comparing 2 HostSwitch simulations using Welch Two Sample t-test
  t: 2.505218, df:3.545114, p.value:0.0743971

> testHostSwitch(simulated_quantities1 = sim1, simulated_quantities2 = sim2,
  parameter = "d", test = "t", warmup = NULL, plot = FALSE)
  An object of class testHostSwitch
  Test result comparing 2 HostSwitch simulations using Welch Two Sample t-test
  t: 1.271303, df:12.32786, p.value:0.2270841
```

The function returns the results of the test and the plot (if plot = TRUE). In the example above there is no significant difference between the two simulations, for all the simulated quantities ("j", "s", "d").

The plotHostSwitch and shinyHostSwitch functions both allow visualizing the results of a simulation, the first in the static plot window of the R console and the second one in a dynamic interface. The S3 method plotHostSwitch function graphically summarizes the simulated output. The following code generates the plot for the third simulation of the object sim1:

```
> plotHostSwitch(HostSwitch_simulated_quantities = sim1, n_sim = 3)
```

The function shinyHostSwitch relies on the package **shiny**, and by running the code:

```
> shinyHostSwitch()
```

a popup window displays the plot for a simulation run with default values. The panel on the left includes slider bars for each argument of simHostSwitch function. The plot reacts dynamically to user input after pressing the Refresh simulation button.

4 Usage scenarios

In this section, we provide examples of how the functions described above can be applied to real data. We conducted a literature review with the aim to select studies providing data about the life cycle parameters of consumers interacting with their resources. The data were all gathered from

experimental settings and are related to three biotic interaction models representing distinct fields of research: wildlife ecology, agricultural pests, zoonotic pathogens. These data saved in the *parli* list object are included in the **HostSwitch** package. The list contains three matrices: $Cephaloleia, $Cacopsylla, and $SarsMers.

The complete code to reproduce results for the three biological scenarios presented is provided as an R Markdown file in the supplementary material for this article. Each scenario has two chunks of code: one to run the test and one to create plots. Using the Knit button in RStudio the results will be embedded beneath the code chunk in a .pdf document (see supplementary material for an example).

Cephaloleia - Zingeriberales (wildlife ecology)

Beetle species in the genus *Cephaloleia* (Coleoptera, Chrysomelidae, Cassidinae) are herbivorous insects feeding on monocots within the order Zingiberales and are known to be strictly co-evolved with their host plants during the last 35-60 Million years in the Neotropical region. This herbivore-plant association has been used earlier as a good model to test macroevolutionary hypotheses of host use and to investigate drivers of the evolution of the biological interactions (García-Robledo and Horvitz, 2011; Schmitt and Frank, 2013). Previous studies revealed host switch events in at least 7 species of *Cephaloleia* beetles in Costa Rica, and detailed information of life-cycle parameters have been reported (García-Robledo and Horvitz, 2011; Garcia-Robledo et al., 2010).

To inform our model, we used data on net reproduction rate for two species of *Cephaloleia* (the generalist *C. belti* and the specialist *C. placida*) provided by García-Robledo and Horvitz (2011). On their native host plants, *C. belti* showed the highest net reproduction rate (b = 4.6) and *C. placida* the lowest (b = 1.4) (see the lower value of the parameter for each species from Table 7 in García-Robledo and Horvitz (2011)). Suppose we are interested in simulating the host switch events of the two closely related taxa with different diet breadth, *C. belti* and *C. placida*. We may want to test the hypothesis that a higher net reproduction rate significantly increases the number of dispersion events and the successful colonization of new hosts (number of host switches). In this example, we simulated the dispersion and host switch events of the generalist *C. belti* (known to be associated with different families in Zingeriberales in Costa Rica) and the specialist *C. placida* (hosted by *Renealmia alpinia*). For each species, the simulation was run for 200 generations, K=1000 (carrying capacity not being a limiting factor), and 1000 runs. Because the other parameters, namely *pRes_min*, *pRes_max*, *sd* (standard deviation of mutation), and sigma (standard deviation of survival), are not available in the literature for the two species the default values of the `simHostSwitch` function were used. The effect of the net reproduction rate on the simulated quantities (dispersion events j, successful host switches s, and distance between current and novel Resource d) was tested, along with the assumption that b does not vary switching from the current to the novel resource across generations. We also tested the influence of the two parameters *mig* (migration probability) and *jump_back* (possibility to jump back to the original host) with the assumption that they may rapidly vary under different ecological conditions. We arbitrarily set the variation for *mig* as low = 0.01 or high = 0.1, and for *jump_back* as "no" or "yes". All possible combinations between the species were compared (Table 2).

The difference between the average values of j, s, and d was evaluated with the paired t-test using the function `testHostSwitch`. The null hypothesis is that the pairwise difference between the two average values is equal. The code chunk "Scenario 1: Cephaloleia-Zingeriberales (wildlife ecology)" (supplementary material) uses the data in *parli*$*Cephaloleia* object to generate a large list (simResult) with eight `simHostSwitch` objects, 4 for *C. belti* (hereafter Cb) and 4 for *C. placida* (Cp). The objects differ for the combinations of the parameters described above. The dataframe *testResult.Cephaloleia* saves the p-values for all the 16 combinations and all simulation objects defined by the user using the vector "simulations". All the objects are saved in the Global environment and a reshaped table with p-values of the t-test is visualized on the console and in Table 2. The code chunk "Plot for Cephaloleia" generates 3 plots for each of the tested quantities (j, s, and d); Figure 2 shows example box plots for the first comparison.

The results in Table 2 and Figure 2 show that the differences in the average number of dispersion and host switch events between *C. belti* and *C. placida* is significant, regardless of the values of probability of migration and the possibility to jump back to the old host. *Cephaloleia belti* shows higher values of j and s, this means that a four times higher net reproduction rate drives a higher probability of migration and successful host switches over time. As far as the distance between current and novel Resources is concerned, only seven out of 16 combinations showed a non-significant difference, all of them including *C. belti* with high probability of migration.

Table 2: Difference between the average values of three estimated quantities (total number of dispersing events *j*, total number of successful host switches *s*, phenotype distance *d*) calculated from 200 simulations for each of the 16 independent combinations of two *Cephaloleia* species, (*C. belti* and *C. placida*). Fixed parameter from real data: net reproduction rate, b = 4.6 *C. placida* and b = 1.4 for *C. placida*. Arbitrary variation for two parameters: *mig*, probability of migration Low = 0.01 (default value) or High = 0.1; *jump_back*, possibility to jump back to the original Resource No (default value) or Yes. Not significant p-values are shown in gray. For significant differences, the combination of the species with a higher value for the estimated quantities is shown in red.

C. belti (mig x jump_back)	**C. placida** (mig x jump_back)	**j**	**s**	**d**
Low x No	Low x No	<0.001	<0.001	<0.001
Low x No	High x No	<0.001	<0.001	<0.001
Low x No	Low x Yes	<0.001	<0.001	<0.001
Low x No	High x Yes	<0.001	<0.001	<0.001
High x No	Low x No	<0.001	<0.001	>0.05
High x No	High x No	<0.001	<0.001	<0.05
High x No	Low x Yes	<0.001	<0.001	>0.05
High x No	High x Yes	<0.001	<0.001	>0.05
Low x Yes	Low x No	<0.001	<0.001	<0.001
Low x Yes	High x No	<0.001	<0.001	<0.001
Low x Yes	Low x Yes	<0.001	<0.001	<0.001
Low x Yes	High x Yes	<0.001	<0.001	<0.001
High x Yes	Low x No	<0.001	<0.001	>0.05
High x Yes	High x No	<0.001	<0.001	>0.05
High x Yes	Low x Yes	<0.001	<0.001	>0.05
High x Yes	High x Yes	<0.001	<0.001	>0.05

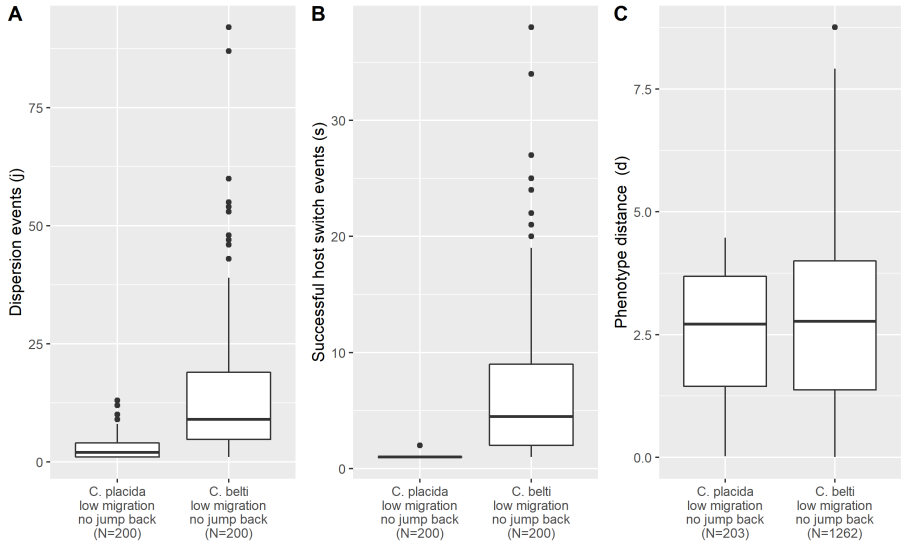


Figure 2: Comparisons between the three simulated quantities (A- total number of dispersing events *j*, B- total number of successful host switches *s*, and C- phenotype distance *d*) calculated for populations of *Cephaloleia placida* and *C. belti*. The populations are characterized by a low migration value and no possibility to jump back to the original Resource. The simulated quantities are calculated using the `simHostSwitch` function. The quantity *d* is defined as the difference between the phenotype of current and new Resource for generations where a successful host switch occurs. Each box plot depicts the distribution of the simulated quantities. For plot A and B, N represents the number of simulations (200). For plot C, N represents the total number of successful host switches calculated for 200 simulations.

Cacopsylla melanoneura - Rosaceae (agricultural pests)

The hemipteran insect *Cacopsylla melanoneura* (Hemiptera, Psyllidae) is one of the known vectors of phytoplasmas, plant pathogenic bacteria in Class Mollicutes, and their association with host plants in the family of Rosaceae is known to be restricted to Central Europe (Janik et al., 2020). This group of pathogens is associated with severe diseases of orchards, including apple proliferation disease associated with '*Candidatus* (*Ca.*) Phytoplasma (P.) mali', pear decline ('*Ca.* P. pyri') and European stone fruit yellows ('*Ca.* P. prunorum'). The epidemiology of this vector-borne pathogen closely depends on the range of host plants that may serve as inoculum sources (reservoirs), and on the specific relationship between insect vectors and plants. Although *C. melanoneura* has been reported as a competent vector of '*Ca.* P. mali' in Italy (Tedeschi et al., 2002), the degree of preference for natural host plants may vary across countries affecting its role in phytoplasma spread, which is still largely unclear (Malagnini et al., 2010). Earlier investigations aimed to characterize aspects of its autoecology, such as life cycle, cryptic genetic variation, and habitat preference that may influence the associations with host plants (Malagnini et al., 2013). *Cacopsylla melanoneura* is known to be an oligophagous insect on hawthorn (*Crataegus* spp.), apple (*Malus* spp.), medlar (*Mespilus germanica*) and pear (*Pyrus communis*), and coniferous species are used as shelter plants during the winter (Jackson et al., 1990; Lal, 1934; Ossiannilsson, 1992). Moreover, the existence of *C. melanoneura* host races which show a morphological and genetic differentiation along with specific association with different host plants (apple or hawthorn) is likely in populations from different geographical regions (Lazarev, 1972, 1974; Lauterer, 1999; Malagnini et al., 2013). Available evidence suggests that sympatric speciation via host shift may be hypothesized. Even within the family Rosaceae only, the suite of host plants of *C. melanoneura* may vary locally through host switches with important consequences for the ecology of the insect vector and on the epidemiology of the phytoplasma disease in cultivated orchard trees.

We used the large amount of data available to inform our model and to compare two distinct populations of *C. melanoneura*: one adapted to apple (hereafter CmA) and the other to hawthorn (CmH). We used the experimental data provided by Malagnini et al. (2013) to calculate the net reproduction rate of the populations of *C. melanoneura* using eqn 2 of García-Robledo and Horvitz (2011). The calculated value *b* for CmA was about twice (2.196) that of CmH (1.022). To calculate the probability of migration *mig* we used the results of a choice test carried out using a dynamic olfactometer and provided in Figure 6 by Mayer et al. (2011). The values for *mig* for CmA were retrieved from the experiment with a population collected on apple tree that had a probability of 0.38 to migrate on apple. For CmH the probability was 0.49. The standard deviation of survival *sigma* for CmA was retrieved by Malagnini et al. (2010) (see Results section) reporting a value of 3.68, and because no data was available for CmH we hypothesized a higher value 7 because hawthorn was reported as the ancestral host of *C. melanoneura* (Jackson et al., 1990).

Suppose we are interested in simulating the host switch events of two populations of *C. melanoneura* to novel closely related hosts in family Rosaceae. We may want to test the hypothesis that twice a value of the standard deviation of survival *sigma* for CmH is more important than a *b* two times higher for CmA, driving a significant increase in the number of dispersion events and more successful colonization of new hosts (number of host switches) for the population CmH than CmA. The parameter *mig* (migration) was also provided from real data but only slightly different between the two populations. Each simulation was run with 200 generations, K=1000 (carrying capacity not being a limiting factor), and 1000 runs. The parameters *pRes_min* and *pRes_max* are not available in the literature and the default values were used. The effect of combined *sigma* and *b* values on the simulated quantities (dispersion events *j*, successful host switches *s*, and distance between current and novel Resource *d*) was tested, along with the assumption that both do not vary from the current to the novel Resource across generations. We arbitrarily set the variation for standard deviation of mutation *sd* as low = 0.2 or high = 9, and for the possibility to jump back to the original host *jump_back* as "no" or "yes". All possible combinations between the populations were tested (Table 3).

We used the code chunk "Scenario 2: Cacopsylla melanoneura-Rosaceae (agricultural pests)" (supplementary file) to test the difference between the average values of simulated quantities (v*j*, *s*, and *d*) for each combination of the populations of *C. melanoneura*. The null hypothesis is that the pairwise difference between the two average values is equal. The p-values of the t-test are reported in (Table 3. The code chunk "Plot for Cacopsylla" generates three plots for each of the tested quantities and in Figure 3 an example for the first combination is reported.

The results in Table 3 show that the difference in the average number of dispersion events is significant for all combinations except the one where we allowed the population adapted to hawthorn to come back to the original host and mutation is high for both populations. The comparisons of host switch events were significant except for three comparisons where mutation was set as high. When "j" and "s" are significantly different, the population of CmA shows higher values except for 5 combinations. These results suggest that a two times higher net reproduction rate drives a higher probability of successful host switch for CmA over time, masking the contribution of a two times

Table 3: Difference between the average values of three estimated quantities (total number of dispersing events *j*, total number of successful host switches *s*, phenotype distance *d*) calculated from 200 simulations for each of the 16 independent combinations of two *Cacopsylla melanoneura* populations, *C. melanoneura* on Apple and *C. melanoneura* on Hawthorn. Fixed parameters from real data were: net reproduction rate, $b = 2.196$, probability of migration, $mig = 0.37$, and survival probability $sigma = 3.68$ for *C. melanoneura* from apple and $b = 1.022$; $mig = 0.49$ and $sigma = 7$ for *C. melanoneura* on hawthorn. Arbitrary variation for two parameters: *sd*, standard deviation of mutation Low = 0.2 (default value) or High = 9; *jump_back*, possibility to jump back to the old Resource No (default value) or Yes. Not significant p-values are shown in gray. or significant differences, the combination of the species with a higher value for the estimated quantities is shown in red.

C. melanoneura Apple (sd x jump_back)	**C. melanoneura Hawthorn** (sd x jump_back)	**j**	**s**	**d**
Low x No	Low x No	<0.001	<0.001	>0.05
Low x No	High x No	<0.001	<0.001	>0.05
Low x No	Low x Yes	<0.001	<0.001	>0.05
Low x No	High x Yes	<0.001	<0.001	<0.01
High x No	Low x No	<0.01	<0.001	<0.01
High x No	High x No	<0.01	>0.05	>0.05
High x No	Low x Yes	<0.001	<0.001	<0.001
High x No	High x Yes	>0.05	<0.001	>0.05
Low x Yes	Low x No	<0.001	<0.001	>0.05
Low x Yes	High x No	<0.001	<0.001	>0.05
Low x Yes	Low x Yes	<0.001	<0.001	>0.05
Low x Yes	High x Yes	<0.001	<0.001	<0.01
High x Yes	Low x No	<0.001	<0.001	>0.05
High x Yes	High x No	<0.001	>0.05	>0.05
High x Yes	Low x Yes	<0.001	<0.001	<0.01
High x Yes	High x Yes	<0.001	>0.05	>0.05

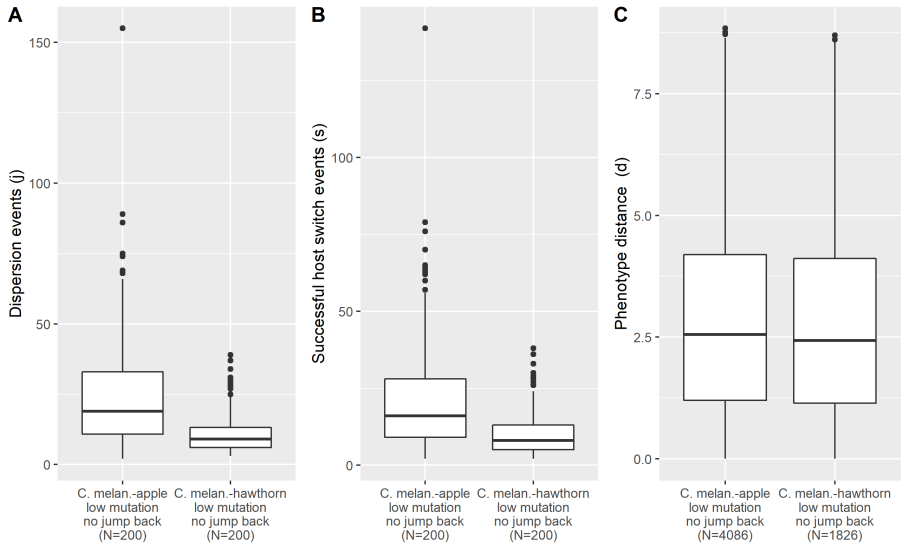


Figure 3: Comparisons between the three simulated quantities (A- total number of dispersing events *j*, B- total number of successful host switches *s*, and C- phenotype distance *d*) calculated for populations of *Cacopsylla melanoneura* (C. melan.) colonizing apple trees and for populations of *C. melanoneura* colonizing hawthorn. The populations are characterized by a low mutation value and no possibility to jump back to the original Resource. The simulated quantities are calculated using the `simHostSwitch` function. The quantity *d* is defined as the difference between the phenotype of current and new Resource for generations where a successful host switch occurs. Each box plot depicts the distribution of the simulated quantities. For plot A and B, N represents the number of simulations (200). For plot C, N represents the total number of successful host switches calculated for 200 simulations.

higher value of the standard deviation of survival. As far as the distance between current and novel Resources is concerned, 11 out of 16 combinations showed a non-significant difference, but no specific pattern has been revealed.

Sarbecovirus spp. and *Merbacovirus* sp. (SARS-MERS) - Mammals (zoonotic pathogens)

The Severe Acute Respiratory Syndrome - Corona Virus (SARS-CoV) is a virus in the family *Coronaviridae* causing respiratory disease and was first identified in humans at the end of February 2003 in China. The Middle East Respiratory Syndrome - Corona Virus (MERS- CoV) (*Coronaviridae*, *Merbacovirus* sp.) was first isolated from humans in 2012 in Saudi Arabia. Lately, a SARS-like coronavirus (SARS-CoV-2, *Sarbecovirus* spp.) causing severe acute respiratory infections was first identified in December 2019 in Wuhan (China). The emergence of these viruses in humans appears to have been driven by stepping-stone switches (Parrish et al., 2008; Rodriguez-Morales et al., 2020), from bats to camels to humans, in the case of MERS-CoV and from bats to pangolins (among the others) to humans, in the case of SARS-CoV-2. As for many other human pathogens, to anticipate the emergence of future outbreaks is critical for appropriate measures to mitigate the consequences and costs associated with epidemic events.

The usage scenario presented here offers an opportunity to apply our model to the biological association between viruses and their mammalian hosts. For this specific example we used the data provided by Kim et al. (2021) on viral dynamics of SARS-CoV-2, SARS-CoV, and MERS-CoV in humans. In particular we selected the two coronaviruses SARS-CoV-2 (hereafter Sars) and MERS-CoV (Mers), and two estimated population parameters: Maximum rate constant for viral replication and critical inhibition level (see Table 1 in Kim et al. (2021)), in our model corresponding to net reproduction rate (*b*, 4 for Sars and 1.46 for Mers) and standard deviation for survival (*sigma*, 0.77 for Sars and 0.38 for Mers), respectively. The parameter *sigma* was multiplied by 10 to be adjusted to the argument requirement of the model. Suppose we are interested in simulating host switch events of Sars and Mers among closely related mammalian hosts. We may test the hypothesis that higher net reproduction rate in combination with higher standard deviation of survival (i.e. lower selection on the Resource) significantly increases the number of dispersion events and successful colonization of new hosts (number of host switches). For each virus, the simulation was run with 200 generations, K=1000 (carrying capacity not being a limiting factor), and 1000 runs. The parameter *sd* (standard deviation for mutation) was set at 0.002 for both viruses and was derived by the values of the substitution rate (per site per year) provided in Table 1 of van Dorp et al. (2020). The parameters *pRes_min*, *pRes_max* and *mig* (migration) are not available in the literature and the default values were used. The combined effect of the net reproduction rate and survival probability on the simulated quantities was tested, along with the assumption that both parameters do not vary from the current to the novel Resource across generations. In this example, we also tested the influence of low (0.01), medium (0.5) and high (0.09) values of migration probability. The possibility to jump back to the original host (*jump_back*) was set to "no" assuming that the virus is not able to come back from a new non-human mammal to human within the same generation (Table 4).

The code chunk "Scenario 3: Sarbecovirus sp. and Merbacovirus sp. (SARS-MERS)-Mammals (zoonotic pathogens)" (supplementary file) was used to test the difference between the average values of simulated quantities (v*j*, *s*, and *d*) for each combination of the two viruses. The null hypothesis is that the pairwise difference between the two average values is equal. The p-values for all the 9 combinations are reported in (Table 4). The code chunk "Plot for SarsMers" generates three plots for each of the tested quantities and in Figure 4 an example for the first combination is reported.

The results in Table 4 and Figure 4 show that the differences in the average number of dispersion and host switch events between Sars and Mers are significant for all the simulations. The population of Sars shows higher values of the estimated quantities, except when Mers has high migration probability and Sars has low migration probability. This means that a combination of higher net reproduction rate and lower selection on the resource (both current and novel) drives to a higher probability of successful host switches over time. As far as the distance between current and novel resources is concerned, only 3 out of 9 combinations showed a non-significant difference, and for all these Mers has a high migration. In comparing the data from these coronaviruses, our purpose was merely academic and we did not mean to provide any evidence for the emergence of re-emergence of the present pandemic SARS-CoV-2 virus. Nevertheless, modeling support may certainly be of value when informed with detailed real data collected to address specific research questions.

Table 4: Difference between the average values of three estimated quantities (total number of dispersing events *j*, total number of successful host switches *s*, phenotype distance *d*) calculated from 200 simulations for each of the 16 independent combinations of the two coronaviruses, Sars (*Sarbecovirus* sp.) and Mers (*Merbacovirus* sp.). Fixed parameters from real data were: net reproduction rate $b = 4$, standard deviation of survival $sigma = 7.7$, and standard deviation of mutation $sd = 0.002$ for Sars and $b = 1.46$; $sigma = 0.38$ and $sd = 0.002$ for Mers. Arbitrary variation for *mig*, probability of migration was set, Low = 0.01 (default value), Medium = 0.5 or High = 0.9. Not significant p-values are shown in gray. For significant differences, the combination of the species with a higher value for the estimated quantities is shown in red.

Sars (mig)	**Mers** (mig)	**j**	**s**	**d**
Low	Low	<0.001	<0.001	<0.001
Medium	Low	<0.001	<0.001	<0.001
High	Low	<0.001	<0.001	<0.001
Low	Medium	<0.001	<0.001	<0.001
Medium	Medium	<0.001	<0.001	<0.001
High	Medium	<0.001	<0.001	<0.001
Low	High	<0.001	<0.001	>0.005
Medium	High	<0.001	<0.001	>0.005
High	High	<0.001	<0.001	>0.005

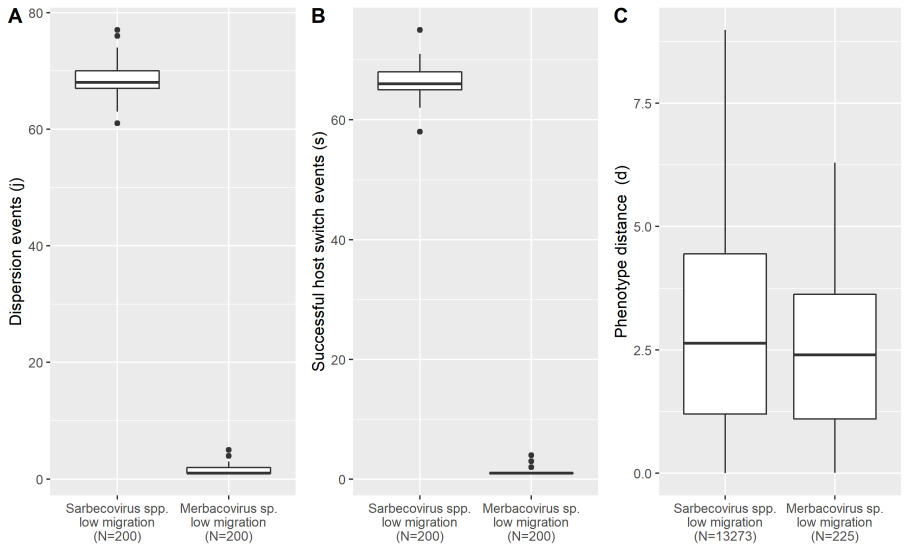


Figure 4: Comparisons between the three simulated quantities (A- total number of dispersing events *j*, B- total number of successful host switches *s*, and C- phenotype distance *d*) calculated for populations of *Sarbecovirus* spp. and *Merbacovirus* sp. The populations are characterized by a low probability of migration value. The simulated quantities are calculated using the `simHostSwitch` function. The quantity *d* is defined as the difference between the phenotype of current and new Resource for generations where a successful host switch occurs. Each box plot depicts the distribution of the simulated quantities. For plot A and B, N represents the number of simulations (200). For plot C, N represents the total number of successful host switches calculated for 200 simulations.

5 Conclusion and future avenues

This paper introduces the R package **HostSwitch** that provides functions to simulate, plot and test the number of dispersion and host switch events of a Consumer. It also measures phenotype distance between a current and novel Resource in the case of a successful host switch. **HostSwitch** is based on an earlier model published in FORTRAN (Araujo et al., 2015). The presented R-package offers several additional functions which largely improve the previous mock-up model, and accommodates users who want to simulate the ecological event of host-switching by using real parameters collected from different types of symbiotic biological associations. For example, the **HostSwitch** package can be used to test the hypothesis that host-switching happens more often than expected by chance.

The **HostSwitch** package was tested on real ecological Consumer-Resource interactions using data from the literature. We are aware that the examples provided above have several limitations, e.g., most of the arguments to inform the model were not available in the literature and the default values have been used. The reliability of the simulation on empirical data depends on the quality of the parameters used to inform the model and on their ecological significance. Users are therefore encouraged to set up specific experiments to collect parameters from a "host-switch perspective". The **HostSwitch** package will also continue to provide a theoretical foundation for understanding the specific processes that drive host switch events. Lastly, it represents a valuable user-friendly educational tool to facilitate deeper understanding of ecological and evolutionary dynamics. As a future avenue, we plan to extend the package to allow for the modeling of microbe-mediated tritrophic interactions such as occur in associations between pathogens, vectors and alternate hosts, e.g. in the phytoplasma pathosystem consisting of an insect vector, crop plants, and a pathogen that is able to manipulate its vector (Consumer) and host (Resource).

Albeit, the present package is intended for researchers in the broad field of biology who study consumer-host associations, we encourage users from other research areas to evaluate if the flowchart presented in Figure 1 may fit and be co-opted by other biological phenomena.

6 Acknowledgment

SBLA thanks Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) that provided partial funding. The authors thank Dr. Christopher H. Dietrich for constructive feedback on the final version of the paper.

Bibliography

S. J. Agosta and J. A. Klemens. Ecological fitting by phenotypically flexible genotypes: implications for species associations, community assembly and evolution. *Ecology Letters*, 11(11):1123–1134, 2008. URL https://doi.org/10.1111/j.1461-0248.2008.01237.x. [p180]

S. J. Agosta, N. Janz, and D. R. Brooks. How specialists can be generalists: resolving the "parasite paradox" and implications for emerging infectious disease. *Zoologia (Curitiba)*, 27(2):151–162, 2010. ISSN 1984-4689. URL https://doi.org/10.1590/S1984-46702010000200001. [p180]

S. B. L. Araujo, M. P. Braga, D. R. Brooks, S. J. Agosta, E. P. Hoberg, F. W. v. Hartenthal, and W. A. Boeger. Understanding host-switching by ecological fitting. *PLOS ONE*, 10(10):e0139225, 2015. URL https://doi.org/10.1371/journal.pone.0139225. [p179, 180, 181, 191]

D. R. Brooks, E. P. Hoberg, and W. A. Boeger. *The Stockholm paradigm: climate change and emerging disease*. The University of Chicago Press, 2019. ISBN 978-0-226-63230-8 978-0-226-63244-5. URL https://press.uchicago.edu/ucp/books/book/chicago/S/bo38871306.html. [p179, 180]

R. Deardon, S. P. Brooks, B. T. Grenfell, M. J. Keeling, M. J. Tildesley, N. J. Savill, D. J. Shaw, and M. E. J. Woolhouse. Inference for individual-level models of infectious diseases in large populations. *Statistica Sinica*, 20(1):239–261, 2010. ISSN 1017-0405. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4578172/. [p180]

G. G. Dimijian. Evolving together: the biology of symbiosis, part 2. *Proceedings (Baylor University. Medical Center)*, 13(4):381–390, 2000a. ISSN 0899-8280. URL https://doi.org/10.1080/08998280.2000.11927712. [p179]

G. G. Dimijian. Evolving together: the biology of symbiosis, part 1. *Proceedings (Baylor University. Medical Center)*, 13(3):217–226, 2000b. ISSN 0899-8280. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1317043/. [p179]

S. G. Feronato, S. Araujo, and W. A. Boeger. 'Accidents waiting to happen'—insights from a simple model on the emergence of infectious agents in new hosts. *Transboundary and Emerging Diseases*, tbed.14146, 2021. ISSN 1865-1682. URL http://onlinelibrary.wiley.com/doi/abs/10.1111/tbed.14146. [p179, 180]

C. García-Robledo and C. C. Horvitz. Experimental demography and the vital rates of generalist and specialist insect herbivores on native and novel host plants. *Journal of Animal Ecology*, 80(5):976–989, 2011. URL https://doi.org/10.1111/j.1365-2656.2011.01843.x. [p185, 187]

C. Garcia-Robledo, C. C. Horvitz, C. L. Staines, et al. Larval morphology, development, and notes on the natural history of *Cephaloleia* "rolled-leaf" beetles (Coleoptera: Chrysomelidae: Cassidinae). *Zootaxa*, 2610:50–68, 2010. URL https://doi.org/10.11646/zootaxa.2610.1.3. [p185]

C. Jackson, I. Hodkinson, P. Stanley, et al. Cold-hardiness in the hawthorn psyllid *Cacopsylla melanoneura* (Förster) (Homoptera: Psylloidea). *Entomologist*, 109(4):224–230, 1990. [p187]

K. Janik, D. Barthel, T. Oppedisano, and G. Anfora. *Apple proliferation - A joint review*. Fondazione Edmund Mach/ Laimburg Research Center, 2020. ISBN 9788878430549. URL http://www.laimburg.com/downloads/Apple_Proliferation.pdf. [p187]

D. H. Janzen. On ecological fitting. *Oikos*, 45:308–310, 1985. URL https://doi.org/10.2307/3565565. [p180]

K. S. Kim, K. Ejima, S. Iwanami, Y. Fujita, H. Ohashi, Y. Koizumi, Y. Asai, S. Nakaoka, K. Watashi, K. Aihara, et al. A quantitative model used to compare within-host SARS-CoV-2, MERS-CoV, and SARS-CoV dynamics provides insights into the pathogenesis and treatment of SARS-CoV-2. *PLoS biology*, 19(3):e3001128, 2021. URL https://doi.org/10.1371/journal.pbio.3001128. [p189]

K. Lal. The biology of Scottish Psyllidae. *Transactions of the Royal Entomological Society of London*, 82(2): 363–385, 1934. [p187]

P. Lauterer. Results of the investigations on Hemiptera in Moravia, made by the Moravian museum (Psylloidea 2). *Acta Musei Moraviae, Scientiae Biologicae (Brno)*, 84:71–151, 1999. [p187]

M. A. Lazarev. *Psylla melanoneura* frst. taurica forma nov. (Homoptera: Psyllidae). An apple tree pest in the Crimea. *Proceedings of the All-Union VI Lenin Academy of Agricultural Sciences, The State Nikita Botanical Gardens, Yalta, Ukraine*, 61:101–122, 1972. [p187]

M. A. Lazarev. The 'Crimean Apple Sucker' *Psylla melanoneura* frst forma taurica, nov. (Homoptera, Psylloidea). *Nikita State Botanical Gardens, Ukraine*, pages 23–26, 1974. [p187]

V. Malagnini, F. Pedrazzoli, V. Gualandri, F. Forno, R. Zasso, A. Pozzebon, and C. Ioriatti. A study of the effects of 'Candidatus Phytoplasma mali' on the psyllid *Cacopsylla melanoneura* (Hemiptera: Psyllidae). *Journal of invertebrate pathology*, 103(1):65–67, 2010. URL https://doi.org/10.1016/j.jip.2009.11.005. [p187]

V. Malagnini, F. Pedrazzoli, C. Papetti, C. Cainelli, R. Zasso, V. Gualandri, A. Pozzebon, and C. Ioriatti. Ecological and genetic differences between *Cacopsylla melanoneura* (Hemiptera, Psyllidae) populations reveal species host plant preference. *PloS one*, 8(7):e69663, 2013. URL https://doi.org/10.1371/journal.pone.0069663. [p187]

C. J. Mayer, A. Vilcinskas, and J. Gross. Chemically mediated multitrophic interactions in a plant–insect vector-phytoplasma system compared with a partially nonvector species. *Agricultural and Forest Entomology*, 13(1):25–35, 2011. URL https://doi.org/10.1111/j.1461-9563.2010.00495.x. [p187]

F. Ossiannilsson. *The Psylloidea (Homoptera) of Fennoscandia and Denmark*, volume 26. Brill, 1992. [p187]

R. D. M. Page. Introduction. In *Tangled Trees. Phylogeny, Cospeciation and Coevolution*, pages 1–22. The University of Chicago Press, 2002. URL https://press.uchicago.edu/ucp/books/book/chicago/T/bo3634552.html. [p179]

C. R. Parrish, E. C. Holmes, D. M. Morens, E.-C. Park, D. S. Burke, C. H. Calisher, C. A. Laughlin, L. J. Saif, and P. Daszak. Cross-species virus transmission and the emergence of new epidemic diseases. *Microbiology and Molecular Biology Reviews*, 72(3):457–470, 2008. URL https://doi.org/10.1128/MMBR.00004-08. [p189]

A. J. Rodriguez-Morales, D. K. Bonilla-Aldana, G. J. Balbin-Ramon, A. A. Rabaan, R. Sah, A. Paniz-Mondolfi, P. Pagliano, and S. Esposito. History is repeating itself: Probable zoonotic spillover as the cause of the 2019 novel coronavirus epidemic. *Infez Med*, 28(1):3–5, 2020. [p189]

M. Schmitt and M. Frank. Notes on the ecology of rolled-leaf hispines (Chrysomelidae, Cassidinae) at La Gamba (Costa Rica). *ZooKeys*, 332:55, 2013. URL http://dx.doi.org/10.3897/zookeys.332.5215. [p185]

R. Tedeschi, D. Bosco, and A. Alma. Population dynamics of *Cacopsylla melanoneura* (Homoptera: Psyllidae), a vector of apple proliferation phytoplasma in northwestern Italy. *Journal of economic entomology*, 95(3):544–551, 2002. URL https://doi.org/10.1603/0022-0493-95.3.544. [p187]

J. N. Thompson. Four central points about coevolution. *Evolution: Education and Outreach*, 3(1): 7–13, 2010. ISSN 1936-6434. doi: 10.1007/s12052-009-0200-x. URL https://evolution-outreach.biomedcentral.com/articles/10.1007/s12052-009-0200-x. Number: 1 Publisher: BioMed Central. [p179]

L. van Dorp, M. Acman, D. Richard, L. P. Shaw, C. E. Ford, L. Ormond, C. J. Owen, J. Pang, C. C. Tan, F. A. Boshier, et al. Emergence of genomic diversity and recurrent mutations in SARS-CoV-2. *Infection, Genetics and Evolution*, 83:104351, 2020. URL https://doi.org/10.1016/j.meegid.2020.104351. [p189]

K. V. Vineetha Warriyar, A. Waleed, and D. Rob. Individual-Level Modelling of Infectious Disease Data: EpiILM. *The R Journal*, 12(1):87–104, 2020. doi: 10.32614/RJ-2020-020. URL https://doi.org/10.32614/RJ-2020-020. [p180]

Valeria Trivellone
University of Illinois at Urbana-Champaign
Prairie Research Institute
1816 South Oak Street, Champaign, Illinois 61820
United States of America
ORCID: http://orcid.org/0000-0003-1415-4097
valeria.trivellone@gmail.com

Sabrina B. L. Araujo
Universidade Federal do Paraná
Departamento de Física
81531-980 Curitiba, Paraná
Brazil
Laboratório de Ecologia e Evolução de Interações
Biological Interactions Universidade Federal do Paraná
P.O. Box 19073 PR 81531-980, Curitiba
Brazil
ORCiD: https://orcid.org/0000-0002-8759-8310
araujosbl@gmail.com

Bernd Panassiti
Independent researcher
Munich
Germany
ORCID: http://orcid.org/0000-0002-5899-4584
bernd.panassiti@gmail.com

OTrecod: An R Package for Data Fusion using Optimal Transportation Theory

by Gregory Guernec, Valerie Gares, Jeremy Omer, Philippe Saint-Pierre, and Nicolas Savy

Abstract The advances of information technologies often confront users with a large amount of data which is essential to integrate easily. In this context, creating a single database from multiple separate data sources can appear as an attractive but complex issue when same information of interest is stored in at least two distinct encodings. In this situation, merging the data sources consists in finding a common recoding scale to fill the incomplete information in a synthetic database. The **OTrecod** package provides R-users two functions dedicated to solve this recoding problem using optimal transportation theory. Specific arguments of these functions enrich the algorithms by relaxing distributional constraints or adding a regularization term to make the data fusion more flexible. The **OTrecod** package also provides a set of support functions dedicated to the harmonization of separate data sources, the handling of incomplete information and the selection of matching variables. This paper gives all the keys to quickly understand and master the original algorithms implemented in the **OTrecod** package, assisting step by step the user in its data fusion project.

1 Introduction

The large amount of data produced by information technology requires flexible tools to facilitate its handling. Among them, the field of data fusion (Hall and Llinas 1997; Klein 2004; Castanedo 2013) also known as statistical matching (Adamek 1994; D'Orazio, Di Zio, and Scanu 2006; Vantaggi 2008) aims to integrate the overall information from multiple data sources for a better understanding of the phenomena that interact in the population.

Assuming that two heterogeneous databases A and B share a set of common variables X while an information of interest is encoded in two distinct scales respectively: Y in A and Z in B. If Y and Z are never jointly observed, a basic data fusion objective consists in the recoding of Y in the same scale of Z (or conversely), to allow the fusion between the databases as illustrated in Table 1.

Initial

DB	ID	Y	Z	X
A	1	observed	**???**	observed
A	2			
...	...			
A	n_A			

DB	ID	Y	Z	X
B	n_A+1	**???**	observed	observed
B	n_A+2			
...	...			
B	n_A+n_B			

⇒

Final

DB	ID	Y	Z	X
A	1	observed	**predicted**	observed
A	2			
...	...			
A	n_A			
B	n_A+1	**predicted**	observed	observed
B	n_A+2			
...	...			
B	n_A+n_B			

Table 1: This package provides algorithms for merging two databases A and B where two variables of interest, Y and Z, are never jointly observed: the final result is a unique and synthetic database where Y and Z are fully completed.

Providing a solution to this recoding problem is often very attractive because it aims at giving access to more accurate and consistent information with no additional costs in a unique and bigger database. Despite this, if we exclude all R data integration packages applied in the context of genetic area like the **MultiDataSet** package (Hernandez-Ferrer et al. 2017), **OMICsPCA** package (Das and Tripathy 2022), or the **mixOmics** package (F et al. 2017) which are often only effective for quantitative data integration. To our knowledge, the **StatMatch** package (D'Orazio 2022) is actually the only one that provide a concrete solution to the problem using hot deck imputation procedures. The main reason for this relative deficiency is that this problem is, in fact, often assimilated and solved like a missing data imputation problem. According to this idea, a very large amount of works and reference books now exist about the handling of missing data (Zhu, Wang, and Samworth 2019; Little and Rubin 2019). Moreover, we can enumerate several R packages that we can sort by types of imputation methods

(Mayer et al. 2019): **mice** (van Buuren and Groothuis-Oudshoorn 2011) and **missForest** (Stekhoven and Bühlmann 2012; Stekhoven 2022) which use conditional models, **softImpute** (Hastie and Mazumder 2021) and **missMDA** (Josse and Husson 2016) which apply low-rank based models. For all these packages, imputation performances can sometimes fluctuate a lot according to the structure and the proportion of non-response encountered. Regressions and non parametric imputation approaches (like hot-deck methods from donor based family) seem to use partially, or not at all, the available information of the two databases to provide the individual predictions. Contrary to our approach, all these methods only use the set of shared variables X for the prediction of Z in A (or conversely Y in B) without really taking into account Y and its interrelations with X in their process of predictions.

The purpose of this paper is to present a new package to the `R` community called **OTrecod** which provides a simple and intuitive access to two original algorithms (Garès et al. 2020; Garès and Omer 2022) dedicated to solve these recoding problems in the data fusion context by considering them as applications of Optimal Transportation theory (OT). In fact, this theory was already applied in many areas: for example, the **transport** package (Schuhmacher et al. 2022) solves optimal transport problems in the field of image processing. A specific package called *POT: Python Optimal Transport* (Flamary et al. 2021) also exists in the *Python* software (Van Rossum and Drake Jr 1995) to solve optimization problems using optimal transportation theory in the fields of signal theory, image processing and domain adaptation. Nevertheless, all these available tools were still not really adapted to our recoding problem and the performances established by OT-based approaches to predict missing information compared to more standard processes (Garès et al. 2020; Garès and Omer 2022; Muzellec et al. 2020) have finished to confirm our decision.

In **OTrecod** the first provided algorithm, called `OUTCOME` and integrated in the `OT_outcome` function consists in finding a map that pushes the distribution of Y forward to the distribution of Z (Garès et al. 2020) while the second one, called `JOINT` and integrated in the `OT_joint` function, pushes the distribution of *(Y,X)* forward to the distribution of *(Z,X)*. Consequently, by building, these two algorithms take advantage of all the potential relationships between Y, Z, and X for the prediction of the incomplete information of Y and/or Z in A and B. Enrichments related to these algorithms and described in (Garès and Omer 2022; Cuturi 2013) are also available via the optional arguments of these two functions. In its current version, these algorithms are accompanied by original preparation (`merge_dbs`, `select_pred`) and validation (`verif_OT`) functions which are key steps of any standard statistical matching project and can be used independently of the `OUTCOME` and `JOINT` algorithms.

2 Solving recoding problems using optimal transportation theory

The optimal transportation problem

The optimal transportation (**OT**) problem was originally stated by Monge (1781) and consists in finding the cheapest way to transport a pile of sand to fill a hole. Formally the problem writes as follows.

Consider two (Radon) spaces $\mathbb{X}$ and $\mathbb{Y}$, μ^X a probability measure on $\mathbb{X}$, and μ^Y a probability measure on $\mathbb{Y}$ and c a Borel-measurable function from $\mathbb{X} \times \mathbb{Y}$ to $[0, \infty]$. The Kantorovich's formulation of the optimal transportation problem (Kantorovich 1942) consists in finding a measure $\gamma \in \Gamma(\mu^X, \mu^Y)$ that realizes the infimum:

$$\inf \left\{ \int_{\mathbb{X}\times\mathbb{Y}} c(x,y)\, \mathrm{d}\gamma(x,y) \middle| \gamma \in \Gamma(\mu^X, \mu^Y) \right\}, \tag{1}$$

where $\Gamma(\mu^X, \mu^Y)$ is the set of measures on $\mathbb{X} \times \mathbb{Y}$ with marginals μ^X on $\mathbb{X}$ and μ^Y on $\mathbb{Y}$.

This theory is applied here to solve a recoding problem of missing distributions in a data fusion area. To do so, we make use of Kantovorich's formulation adapted to the discrete case, known as Hitchcock's problem (Hitchcock 1941). Therefore, by construction, the proposed algorithms are usable for specific target variables only: categorical variables, ordinal or nominal, and discrete variables with finite number of values.

Optimal transportation of outcomes applied to data recoding

Let A and B be two databases corresponding to two independent sets of subjects. We assume without loss of generality that the two databases have equal sizes, so that they can be written as $A = \{i_1, \ldots, i_n\}$ and $B = \{j_1, \ldots, j_n\}$. Let $\left((X_i, Y_i, Z_i)\right)_{i\in A}$ and $\left((X_j, Y_j, Z_j)\right)_{j\in B}$ be two sequences of i.i.d. discrete random variables with values in $\mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$, where $\mathcal{X}$ is a finite subset of $\mathbb{R}^P$, and $\mathcal{Y}$ and $\mathcal{Z}$ are finite subsets of $\mathbb{R}$. Variables $(X_i, Y_i, Z_i), i \in A$, are i.i.d copies of (X^A, Y^A, Z^A) and

$(X_j, Y_j, Z_j), j \in B$, are i.i.d copies of (X^B, Y^B, Z^B). Moreover assume that $\{(X_i, Y_i, Z_i), i \in A\}$ are independent of $\{(X_j, Y_j, Z_j), j \in B\}$. The first version using the optimal transportation algorithm approach, described in (Garès et al. 2020), assumes that:

Assumption 1. *Y^A and Z^A respectively follow the same distribution as Y^B and Z^B.*

Assumption 2. *For all $x \in \mathcal{X}$ the probability distributions of Y^A and Z^A given that $X^A = x$ are respectively equal to those of Y^B and Z^B given that $X^B = x$.*

In this setting, the aim is to solve the recoding problem given by equation (1) that pushes μ^{Y^A} forward to μ^{Z^A}. The variable γ of (1) is a discrete measure with marginals μ^{Y^A} and μ^{Z^A}, represented by a $|\mathcal{Y}| \times |\mathcal{Z}|$ matrix. The cost function denoted as c is a $|\mathcal{Y}| \times |\mathcal{Z}|$ matrix, $(c_{y,z})_{y \in \mathcal{Y}, z \in \mathcal{Z}}$. The goal is in the identification of:

$$\gamma^* \in argmin_{\gamma \in \mathbb{R}_+^{|\mathcal{Y}| \times |\mathcal{Z}|}} \left\{ \langle \gamma \ , \ c \rangle : \gamma \mathbf{1}_{|\mathcal{Z}|} = \mu^{Y^A}, \gamma^T \mathbf{1}_{|\mathcal{Y}|} = \mu^{Z^A} \right\}, \tag{2}$$

where $\langle \cdot \ , \ \cdot \rangle$ is the dot product, $\mathbf{1}$ is a vector of ones with appropriate dimension and M^T is the transpose of matrix M. The cost function considered by Garès et al. (2020), $c_{y,z}$, measures the average distance between the profiles of shared variables of A satisfying $Y = y$ and subjects of B satisfying $Z = z$, that is:

$$c_{y,z} = \mathbb{E}\left[d(X^A, X^B) \mid Y^A = y, Z^B = z \right], \tag{3}$$

where d is a given distance function to choose on $\mathcal{X} \times \mathcal{X}$.

In fact, the above situation cannot be solved in reality, since the distributions of X^A, X^B, Y^A and Z^A are never jointly observed. As a consequence, the following unbiased empirical estimators are used: $\hat{\mu}_n^{X^A}$ of μ^{X^A} and $\hat{\mu}_n^{X^B}$ of μ^{X^B}. Because Y and Z are only available in A and B respectively, two distinct empirical estimators have to be defined:

$$\begin{aligned} \hat{\mu}_{n,y}^{Y^A} &= \frac{1}{n} \sum_{i \in A} \mathbf{1}_{\{Y_i = y\}}, \ \forall y \in \mathcal{Y}, \\ \hat{\mu}_{n,z}^{Z^A} &= \frac{1}{n} \sum_{j \in B} \mathbf{1}_{\{Z_j = z\}}, \ \forall z \in \mathcal{Z}, \end{aligned} \tag{4}$$

where $\mathbf{1}_{\{Y=y\}} = 1$ if $Y = y$ and 0 otherwise. The **assumption 1** gives: $\mu^{Z^A} = \mu^{Z^B}$ from which we can conclude that $\hat{\mu}_{n,z}^{Z^B} = \hat{\mu}_{n,z}^{Z^A}$. Finally, denoting:

$$\kappa_{n,y,z} \equiv \sum_{i \in A} \sum_{j \in B} \mathbf{1}_{\{Y_i = y, Z_j = z\}},$$

the number of pairs $(i, j) \in A \times B$ such that $Y_i = y$ and $Z_j = z$, the cost matrix c is estimated by:

$$\hat{c}_{n,y,z} = \begin{cases} \frac{1}{\kappa_{n,y,z}} \sum_{i \in A} \sum_{j \in B} \mathbf{1}_{\{Y_i = y, Z_j = z\}} \times d(X_i, X_j), & \forall y \in \mathcal{Y}, z \in \mathcal{Z} : \kappa_{n,y,z} \neq 0, \\ 0, & \forall y \in \mathcal{Y}, z \in \mathcal{Z} : \kappa_{n,y,z} = 0. \end{cases} \tag{5}$$

Plugging the values observed for these estimators in (2) yields to a linear programming model denoted:

$$\hat{\mathcal{P}}_n^0 : \begin{cases} \min \ < \hat{c}_n, \gamma > \\ \text{s.t.} \ \sum_{z \in Z} \gamma_{y,z} = \mu_{n,y}^{Y^A}, \ \forall y \in \mathcal{Y}, \\ \quad \sum_{y \in Y} \gamma_{y,z} = \mu_{n,z}^{Z^A}, \ \forall z \in \mathcal{Z}, \\ \quad \gamma_{y,z} \geq 0, \ \forall y \in \mathcal{Y}, \forall z \in \mathcal{Z}. \end{cases} \tag{6}$$

The solution $\hat{\gamma}_n$ can then be interpreted as an estimator $\hat{\mu}_n^{(Y^A, Z^A)}$ of the joint distribution of Y^A and Z^A, $\mu^{(Y^A, Z^A)}$. If this estimate is necessary, it is nevertheless insufficient here to provide the individual predictions on Z in A. These predictions are done in a second step using a nearest neighbor algorithm from which we deduce an estimation of $\mu^{Z^A | X^A = x, Y^A = y}$ (see Garès and Omer (2022) for details). In the remainder, the overall algorithm described in this section is referred to as `OUTCOME`. To improve the few drawbacks of this algorithm described in Garès et al. (2020), derived algorithms from `OUTCOME` have been developed (Garès and Omer 2022) and described in the following part.

Optimal transportation of outcomes and covariates

Using the same notations, Garès et al. (2020) propose to search for an optimal transportation map between the two joint distributions of (X^A, Y^A) and (X^A, Z^A) with marginals $\mu^{(X^A,Y^A)}$ and $\mu^{(X^A,Z^A)}$ respectively. Under Kantorovich's formulation in a discrete setting, they search for:

$$\gamma^* \in \text{argmin}_{\gamma \in \mathcal{D}} < c, \gamma >,$$

where c is a given cost matrix and $\mathcal{D}$ is the set of joint distributions with marginals $\mu^{(X^A,Y^A)}$ and $\mu^{(X^A,Z^A)}$. It is natural to see any element $\gamma \in \mathcal{D}$ as the vector of joint probabilities $\mathbb{P}((X^A = x, Y^A = y), (X^A = x', Z^A = z))$ for any $x, x' \in \mathcal{X}^2$, $y \in \mathcal{Y}$ and $z \in \mathcal{Z}$. Since this probability nullifies for all $x \neq x'$, $\gamma \in \mathcal{D}$ is defined as a vector of $\mathbb{R}^{|X|\times|\mathcal{Y}|\times|\mathcal{Z}|}$, where $\gamma_{x,y,z}$ stands for an estimation of the joint probability $\mathbb{P}(X^A = x, Y^A = y, Z^A = z)$. These notations lead to the more detailed model:

$$\mathcal{P} : \begin{cases} \min\ < c, \gamma > \\ \text{s.t.} \sum_{z \in Z} \gamma_{x,y,z} = \mu_{x,y}^{(X^A,Y^A)},\ \forall x \in \mathcal{X}, \forall y \in \mathcal{Y}, \\ \sum_{y \in Y} \gamma_{x,y,z} = \mu_{x,z}^{(X^A,Z^A)},\ \forall x \in \mathcal{X}, \forall z \in \mathcal{Z}, \\ \gamma_{x,y,z} \geq 0,\ \forall x \in \mathcal{X}, \forall y \in \mathcal{Y}, \forall z \in \mathcal{Z}. \end{cases} \tag{7}$$

The above algorithm can be solved only if the marginals $\mu^{(X^A,Y^A)}$ and $\mu^{(X^A,Z^A)}$ are known, but, based on assumption 2, unbiased estimators $\hat{\mu}_n^{X^A,Y^A}$ and $\hat{\mu}_n^{X^A,Z^A}$ can be built according to the previous subsection. For the first one it gives:

$$\hat{\mu}_n^{X^A,Y^A} = \frac{1}{n} \sum_{i \in A} \mathbf{1}_{\{Y_i = y, X_i = x\}},\ \forall x \in \mathcal{X},\ \forall y \in \mathcal{Y}. \tag{8}$$

The cost matrix introduced in the `OUTCOME` algorithm is used (3) and estimated by (5). Formally we can write:

$$c_{x,y,z} = c_{y,z},\ \forall x \in \mathcal{X}, \forall y \in \mathcal{Y}, \forall z \in \mathcal{Z}, \tag{9}$$

which does not depend on the value of x.

Plugging the values observed for these estimators in (7) yield a linear programming model denoted as $\widehat{\mathcal{P}}_n$. In contrast to OUTCOME, the algorithm that consists in solving $\widehat{\mathcal{P}}_n$ to solve the recoding problem is referred to as `JOINT` in what follows.

An estimation of the distribution of Z^A given the values of X^A and Y^A is then given by:

$$\tilde{\mu}_{n,z}^{Z^A|X^A=x,Y^A=y} = \begin{cases} \dfrac{\hat{\gamma}_{n,x,y,z}}{\hat{\mu}_{n,x,y}^{(X^A,Y^A)}}, & \forall x \in \mathcal{X}, y \in \mathcal{Y}, z \in \mathcal{Z} : \hat{\mu}_{n,x,y}^{(X^A,Y^A)} \neq 0, \\ 0, & \forall x \in \mathcal{X}, y \in \mathcal{Y}, z \in \mathcal{Z} : \hat{\mu}_{n,x,y}^{(X^A,Y^A)} = 0. \end{cases} \tag{10}$$

and an individual prediction of Z^A is then deduced using the maximum a posterior rule:

$$\widehat{z}_i^A = \text{argmax}_{z \in \mathcal{Z}}\ \tilde{\mu}_{n,z}^{Z^A|X^A=x_i,Y^A=y_i}.$$

Due to potential errors in the estimations of $\mathcal{P}$, the constraints of $\widehat{\mathcal{P}}_n$ may derive from the true values of the marginals of $\mu^{(X^A,Y^A,Z^A)}$. To deal with this situation, small violations of the constraints of $\widehat{\mathcal{P}}_n$ are allowed by enriching the initial algorithm as described in Garès and Omer (2022).

The equality constraints of $\widehat{\mathcal{P}}_n$ are then relaxed as follows:

$$\sum_{z\in Z}\gamma_{x,y,z}=\hat{\mu}_{n,x,y}^{(X^A,Y^A)}+e_{x,y}^{X,Y},\ \forall x\in\mathcal{X},\forall y\in\mathcal{Y} \tag{11}$$

$$\sum_{y\in Y}\gamma_{x,y,z}=\tilde{\mu}_{n,x,z}^{(X^A,Z^A)}+e_{x,z}^{X,Z},\ \forall x\in\mathcal{X},\forall z\in\mathcal{Z} \tag{12}$$

$$\sum_{x\in\mathcal{X},y\in\mathcal{Y}}e_{x,y}^{X,Y}=0,\ \sum_{x\in\mathcal{X},z\in\mathcal{Z}}e_{x,z}^{X,Z}=0 \tag{13}$$

$$-e_{x,y}^{X,Y,+}\leq e_{x,y}^{X,Y}\leq e_{x,y}^{X,Y,+},\ \forall x\in\mathcal{X},\forall y\in\mathcal{Y} \tag{14}$$

$$-e_{x,z}^{X,Z,+}\leq e_{x,z}^{X,Z}\leq e_{x,z}^{X,Z,+},\ \forall x\in\mathcal{X},\forall z\in\mathcal{Z} \tag{15}$$

$$\sum_{x\in\mathcal{X},y\in\mathcal{Y}}e_{x,y}^{X,Y,+}\leq\alpha_n,\ \sum_{x\in\mathcal{X},z\in\mathcal{Z}}e_{x,z}^{X,Z,+}\leq\alpha_n. \tag{16}$$

This relaxation is possible by introducing extra-variables $e^{X,Y,+}$ and $e^{X,Z,+}$ as additional constraints (14)–(15). Garès and Omer (2022) suggests to consider $\alpha_n:=\frac{\alpha}{\sqrt{n}}$ from (16), with a parameter α to calibrate numerically but proposes also a default value fixed to 0.4.

A regularization term λ given by $(\frac{\pi_{x,y,z}}{\hat{\mu}_{n,x}^{X^A}})_{x\in\mathcal{X},y\in\mathcal{Y},z\in\mathcal{Z}}$ can also be added to improve regularity in the variations of the conditional distribution $\mu^{Y^A,Z^A|X^A=x}$ with respect to x. The corresponding regularized algorithm is:

$$\widehat{\mathcal{P}}_n^R:\begin{cases}\min\ <\widehat{c}_n,\gamma>+\lambda\displaystyle\sum_{(x_i,x_j)\in E_{\mathcal{X}}}w_{i,j}\sum_{y\in\mathcal{Y},z\in\mathcal{Z}}r_{i,j,y,z}^{+}\\ \quad\text{s.t. constraints (11)–(16)}\\ \quad\dfrac{\gamma_{x_i,y,z}}{\hat{\mu}_{n,x_i}^{X^A}}-\dfrac{\gamma_{x_j,y,z}}{\hat{\mu}_{n,x_j}^{X^A}}\leq r_{i,j,y,z}^{+},\ \forall\{x_i,x_j\}\in E_{\mathcal{X}},y\in\mathcal{Y},z\in\mathcal{Z}\\ \quad\dfrac{\gamma_{x_i,y,z}}{\hat{\mu}_{n,x_i}^{X^A}}-\dfrac{\gamma_{x_j,y,z}}{\hat{\mu}_{n,x_j}^{X^A}}\geq -r_{i,j,y,z}^{+},\ \forall\{x_i,x_j\}\in E_{\mathcal{X}},y\in\mathcal{Y},z\in\mathcal{Z}\\ \quad\gamma_{x,y,z}\geq 0,\ \forall x\in\mathcal{X},\forall y\in\mathcal{Y},\forall z\in\mathcal{Z}.\end{cases} \tag{17}$$

The constant $\lambda\in\mathbb{R}^+$ is a regularization parameter to be calibrated numerically (0.1 can be considered as default value) and $E_{\mathcal{X}}\subset\mathcal{X}^2$ includes the pairs of elements of X defined as neighbors: $\left\{x_i,x_j\right\}\in E_{\mathcal{X}}$ if x_j is among the k nearest neighbors of x_i for some parameter $k\geq 1$. The method that computes a solution to the recoding problem with regularization and relaxation is called `R-JOINT`.

In a same way, a relaxation of the **assumption 1** is also proposed and added to the `OUTCOME` algorithm: this resulting method is denoted `R-OUTCOME` and is related to the following program:

$$\widehat{\mathcal{P}}_n^{0-R}:\begin{cases}\min\ <\widehat{c}_n,\gamma>\\ \quad\displaystyle\sum_{z\in\mathcal{Z}}\gamma_{y,z}=\hat{\mu}_{n,y}^{Y^A}+e_y^Y,\ \forall y\in\mathcal{Y}\\ \quad\displaystyle\sum_{y\in\mathcal{Y}}\gamma_{y,z}=\tilde{\mu}_{n,z}^{Z^A}+e_z^Z,\ \forall z\in\mathcal{Z}\\ \quad\displaystyle\sum_{y\in\mathcal{Y}}e_y^Y=0,\ \sum_{z\in\mathcal{Z}}e_z^Z=0\\ \quad-e_y^{Y,+}\leq e_y^Y\leq e_y^{Y,+},\ \forall y\in\mathcal{Y}\\ \quad-e_z^{Z,+}\leq e_z^Z\leq e_z^{Z,+},\ \forall z\in\mathcal{Z}\\ \quad\displaystyle\sum_{y\in\mathcal{Y}}e_y^{Y,+}\leq\alpha_n,\ \sum_{z\in\mathcal{Z}}e_z^{Z,+}\leq\alpha_n\\ \quad\gamma_{y,z}\geq 0,\ \forall y\in\mathcal{Y},\forall z\in\mathcal{Z}.\end{cases} \tag{18}$$

Note that algorithms `JOINT` and `R-JOINT` do not require **assumption 1**. The relaxation in `R-OUTCOME` alleviates its dependence to the satisfaction of this assumption. However, algorithms `JOINT` and `R-JOINT` require that X is a set of discrete variables (factors ordered or not are obviously allowed) while the absence of X in the linear algorithms `OUTCOME` and `R-OUTCOME` allow X to be a set of discrete and/or continuous variables. In this case, the nature of the variables X need to be considered when choosing the distance d.

3 Package installation and description

Installation

The **OTrecod** package can be installed from the Comprehensive R Archive Network (CRAN) by using the following template:

```
install.packages("OTrecod")
```

The development version of **OTrecod** is also available and can be directly installed from **GitHub** by loading the **devtools** (Wickham et al. 2022) package (`R> install_github("otrecoding/OTrecod")`).

Main functions

The two types of optimal transportation algorithms previously introduced (`OUTCOME` and `JOINT`) and their respective enrichments (`R-OUTCOME` and `R-JOINT`) are available in the **OTrecod** package via two core functions denoted `OT_outcome` and `OT_joint`. Details about their implementations in `R` are described in the following section. In this package, these algorithms of recoding are seen as fundamental steps of data fusion projects that also require often adapted preparation and validation functions. In this way, the Table 2 introduces the main functions proposed by **OTrecod** to handle a data fusion project.

`R` **Function**	**Description**
Pre-process functions	
`merge_dbs`	Harmonization of the data sources
`select_pred`	Selection of matching variables
Functions of data fusion	
`OT_outcome`	Data fusion with OT theory using the `OUTCOME` or `R-OUTCOME` algorithms.
`OT_joint`	Data fusion with OT theory using the `JOINT` or `R-JOINT` algorithms.
Post-process function	
`verif_OT`	Quality assessment of the data fusion

Table 2: A brief description of the main functions of OTrecod

All the intermediate functions integrated in the `OT_outcome` and `OT_joint` functions (`proxim_dist`, `avg_dist_closest`, `indiv_grp_closest`, `indiv_grp_optimal`), and their related documentations, are all included and usable separately in the package. They have been kept available for users to ensure a great flexibility as other interesting functions like `power_set` that returns the power set of a set. This function did not exist on `R` until now and could be of interest for specialists of algebra. These functions are not described here but detailed in the related pdf manual of the package.

4 Functionalities overview

Expected structure of the input databases as arguments

The functions of recoding `OT_outcome` and `OT_joint` require a specific structure of data.frame as input arguments. Described in Table 3, it must be the result of two overlayed databases made of at least four variables:

- A first variable, discrete or categorical, corresponding to the database identifier, stored in factor or not, but with only two classes or levels (for example: *A* and *B*, *1* and *2* or otherwise).
- The target variable of the first database (or top database) denoted *Y* for example, whose values related to the second database are missing. This variable can be discrete or categorical stored in factor, ordered factor or not.
- In the same way, the target variable of the second database (or below database) denoted *Z* for example, whose values related to the first database are missing.
- At least one shared variable (defined as a variable with the same label and the same encoding in the two distinct data sources). The type of shared variables can be continuous, categorical stored

in factor or not, complete or not. Nevertheless, few constraints must be noticed related to this question. First, in a critical situation where only one shared variable exists, this latter cannot be incomplete. Second, continuous shared variables are actually not allowed in the current version of the function `OT_joint`, therefore, these variables must be transformed beforehand.

DB	Y	Z	X_1	X_2	X_3
1	(600-800]	<NA>	M	Yes	50
1	(600-800]	<NA>	M	No	32
1	[200-600]	<NA>	W	No	31
2	<NA>	G1	M	No	47
2	<NA>	G3	W	Yes	43
2	<NA>	G2	W	No	23
2	<NA>	G4	M	Yes	22
2	<NA>	G2	W	Yes	47

Table 3: Example of expected structure for two databases 1 and 2 with three shared variables X_1, X_2, X_3

As additional examples, users can also refer to the databases `simu_data` and `tab_test` provided in the package with expected structures. Note that class objects are not expected here as input arguments of these functions to allow users to freely work with or without the use of the pre-process functions provided in the package.

Choice of solver

The package **OTrecod** uses the `ROI` optimization infrastructure (Theußl, Schwendinger, and Hornik 2017) to solve the optimization problems related to the `OUTCOME` and `JOINT` algorithms. The solver GLPK (The GNU Linear Programming Kit (Makhorin 2011)) is the default solver actually integrated in the `OT_outcome` and `OT_joint` functions for handling linear problems with linear constraints. The ROI infrastructure makes easy for users to switch solvers for comparisons. In many situations, some of them can noticeably reduce the running time of the functions.

For example, the solver Clp (Forrest, Nuez, and Lougee-Heimer 2004) for COINT-OR Linear Programming, known to be particularly convenient in linear and quadratic situations, can be easily installed by users via the related plug-in available in `ROI` (searchable with the instruction `ROI_available_solvers()`) and following the few instructions detailed in (Theußl, Schwendinger, and Hornik 2020) or via the dedicated website.

An illustrative example

In California (United States), from 1999 to 2018, the Public Schools Accountability Act (PSAA) imposed on its California Department of Education (CDE) to provide annually the results of an Academic Performance Index (API) which established a ranking of the best public schools of the state.

This numeric score, indicator of school's performance levels, could vary from 200 to 1000 and the performance objective to reach for each school was 800. Information related to the 418 schools (identified by `cds`) of Nevada (County 29) and to the 362 schools of San Benito (County 35), was respectively collected in two databases, `api29` and `api35`, available in the package. The distributions of all the variables in the two databases are provided by following the `R` commands:

```
library(OTrecod)
data(api29); data(api35)
summary(api29) #--------------------------------------------------
```

```
#>      cds                apicl_2000  stype    awards       acs.core
#>  Length:418         [200-600] : 93   E:300   No  : 98   Min.   :16.00
#>  Class :character   (600-800] :180   M: 68   Yes :303   1st Qu.:26.00
#>  Mode  :character   (800-1000]:145   H: 50   NA's: 17   Median :30.00
#>                                                         Mean   :31.97
#>                                                         3rd Qu.:39.00
#>                                                         Max.   :50.00
```

```
#>     api.stu         acs.k3.20   grad.sch         ell          mobility
#>  Min.   : 108.0   <=20   :190   0   : 86   [0-10]  :153   [0-20]  :362
#>  1st Qu.: 336.2   >20    :108   1-10:180   (10-30] : 83   (20-100]: 56
#>  Median : 447.5   unknown:120   >10 :152   (30-50] : 60
#>  Mean   : 577.8                            (50-100]: 93
#>  3rd Qu.: 641.5                            NA's    : 29
#>  Max.   :2352.0
#>       meals     full
#>  [0-25]  :200   1: 85
#>  (25-50] : 56   2:333
#>  (50-75] :100
#>  (75-100]: 62
#>
#>

summary(api35) #---------------------------------------------------

#>      cds            apicl_1999 stype    awards       acs.core
#>  Length:362         G1:91      E:257   No  :111   Min.   :16.00
#>  Class :character   G2:90      M: 67   Yes :237   1st Qu.:25.00
#>  Mode  :character   G3:90      H: 38   NA's: 14   Median :30.00
#>                     G4:91                         Mean   :31.81
#>                                                   3rd Qu.:39.00
#>                                                   Max.   :50.00
#>     api.stu         acs.k3.20   grad.sch         ell       mobility
#>  Min.   : 102.0   <=20   :227   0   : 50   [0-10]  :164   1:213
#>  1st Qu.: 363.2   >20    : 30   1-10:241   (10-30] : 99   2:149
#>  Median : 460.0   unknown:105   >10 : 71   (30-50] : 64
#>  Mean   : 577.2                            (50-100]: 10
#>  3rd Qu.: 624.0                            NA's    : 25
#>  Max.   :2460.0
#>       meals     full
#>  [0-25]  : 77   1:244
#>  (25-50] : 92   2:118
#>  (50-75] :122
#>  (75-100]: 71
#>
#>
```

The two databases seem to share a majority of variables (same labels, same encodings) of different types, therefore inferential statistics could be ideally considered by combining all the information of the two databases to study the effects of social factors on the results of the API score in 2000.

Nevertheless, while this target variable called `apicl_2000` is correctly stored in the database `api29` and encoded as a three levels ordered factors clearly defined: `[200-600]`, `(600-800]` and `(800-1000]`, the only information related to the API score in the database `api35` is the variable `apicl_1999` for the API score collected in 1999, encoded in four unknown balanced classes (`G1`, `G2`, `G3` and `G4`). As no school is common to the two counties, we easily deduce that these two variables have never been jointly observed.

By choosing these two variables as outcomes (called also target variables), the objective of the following examples consists in creating a synthetic database where the missing information related to the API score in 2000 is fully completed in `api35` by illustrating the use of the main functions of the package.

Harmonization of two datasources using `merge_dbs`

The function `merge_dbs` is an optional pre-process function dedicated to data fusion projects that merges two raw databases by detecting possible discrepancies among the respective sets of variables from one database to another. The current discrepancy situations detected by the function follow specific rules described below:

- any variable (other than a target one) whose label (or name) is not common to the two data sources is automatically excluded. By default, the remaining variables are denoted shared variables.

- among the subset of shared variables, any variable stored in a different format (or type) from one datasource to another will be automatically discarded from the subset previously generated and its label saved in an output object called REMOVE1.
- among the remaining subset of shared variables, a factor variable (ordered or not) stored with different levels or number of levels from one data source to another will be automatically discarded from the subset and its label saved in an output object called REMOVE2.

These situations sometimes require reconciliation actions which are not supported by the actual version of the merge_dbs function. Therefore, when reconciliation actions are required by the user, they will have to be treated a posteriori and outside the function.

Applied to our example and according to the introduced rules, the first step of our data fusion project consists in studying the coherence between the variables of the databases api29 and api35 via the following R code:

```
step1 = merge_dbs(DB1 = api29, DB2 = api35,
            NAME_Y = "apicl_2000", NAME_Z = "apicl_1999",
            row_ID1 = 1, row_ID2 = 1,
            ordinal_DB1 = c(2:3, 8:12),
            ordinal_DB2 = c(2:3, 8:12))
```

As entry, the raw databases must be declared separately in the DB1 and DB2 arguments and the name of the related target variables of each database must be specified via the NAME_Y and NAME_Z for DB1 and DB2 respectively. In presence of row identifiers, the respective column indexes of each database must be set in the argument row_ID1 and row_ID2. The arguments ordinal_DB1 and ordinal_DB2 list the related column indexes of all the variables defined as ordinal in the two databases (including also the indexes of the target variables if necessary). Here, apicl_2000 is clearly an ordinal variable, and, by default, we suppose that the unknown encoding related to apicl_1999 is also ordinal: the corresponding indexes (2 and 3) are so added in these two arguments.

After running, the function informs users that no row was dropped from the databases during the merging because each target variable is fully completed in the two databases. Nine potential predictors are kept while only one variable is discarded because of discrepancies between the databases: its identity is consequently stored in output and informs user about the nature of the problem: the mobility factor has different levels from one database to the other.

```
summary(step1)
```

```
#>               Length Class      Mode
#> DB_READY      12     data.frame list
#> ID1_drop       0     -none-     character
#> ID2_drop       0     -none-     character
#> Y_LEVELS       3     -none-     character
#> Z_LEVELS       4     -none-     character
#> REMOVE1        0     -none-     NULL
#> REMOVE2        1     -none-     character
#> REMAINING_VAR  9     -none-     character
#> IMPUTE_TYPE    1     -none-     character
#> DB1_raw       12     data.frame list
#> DB2_raw       12     data.frame list
#> SEED           1     -none-     numeric
```

```
step1$REMOVE1   # List of removed variables because of type's problem
```

```
#> NULL
```

```
step1$REMOVE2   # List of removed factors because of levels' problem
```

```
#> [1] "mobility"
```

```
levels(api29$mobility)  # Verification
```

```
#> [1] "[0-20]"   "(20-100]"
```

```
levels(api29$mobility); levels(api35$mobility)
```

```
#> [1] "[0-20]"   "(20-100]"

#> [1] "1" "2"

step1$REMAINING_VAR

#> [1] "acs.core"  "acs.k3.20" "api.stu"   "awards"    "ell"       "full"
#> [7] "grad.sch"  "meals"     "stype"
```

The function returns a list object and notably `DB_READY`, a data.frame whose structure corresponds to the expected structure introduced in the previous subsection: a unique database, here result of the superimposition of `api29` on `api35` where the first column (`DB`) corresponds to the database identifier (1 for `api29` and 2 for `api35`), the second and third columns (`Y`, `Z` respectively) corresponds to the target variables of the two databases. Missing values are automatically assigned by the function to the unknown part of each target variable: in `Y` when the identifier equals to 2, in `Z` when the identifier equals to 1. The subset of shared variables whose information is homogeneous between the databases are now stored from the fourth column to the last one. Their identities are available in the output object called `REMAINING_VAR`.

The `merge_dbs` function can handle incomplete shared variables by using the `impute` argument. This option allows to keep the missing information unchanged (the choice by default, and also the case in this example), to work with complete cases only, to do fast multivariate imputations by chained equations approach (the function integrates the main functionalities of the `mice` function of the **mice** package), or to impute data using the dimensionality reduction method introduced in the **missMDA** package (see the pdf manual for details).

Selection of matching variables using `select_pred`

In data fusion projects, a selection of shared variables (also called matching variables) appears as an essential step for two main reasons. First, the proportion of shared variables X between the two databases can be important (much higher than three variables) and keeping a pointless part of variability between variables could strongly reduce the quality of the fusion. Second, this selection greatly influences the quality of the predictions regardless of the matching technique which is chosen a posteriori (Adamek 1994). The specific context of data fusion is subject to the following constraints:

1. Y, Z and X are never jointly observed in database A or B, so the relationships between Y and X, and between Z and X must be investigated separately.
2. matching variables need to be good predictors of both target variables (outcomes) Y and Z (Cohen 1991), in the sense that they explain relevant parts of variability of the targets.

These particularities suppose that the investigations have to be done separately but in the same manner in both databases. In this way, a predictor which appears at the same time as highly correlated to Y and Z will be automatically selected for the fusion. On the contrary, predictors whose effects on Y and Z seem not obvious will be discarded. Additional recommended rules also emerge from literature:

- concerning the subset of variables that would predict only one of the two targets Y or Z, D'Orazio, Di Zio, and Scanu (2006) suggests a moderate parsimonious selection remaining too many predictors could complicate the fusion procedure.
- Cibella (2010) and Scanu (2010) suggest to select quality predictors with no error and just a small amount of missing data.

The function of the package dedicated to this task is `select_pred`. From the `DB_READY` database generated in the previous subsection, studying the outputs related to each database and produced by the following `R` commands assist users in selecting the best predictors:

```
# For the dataset api29 --------------
step2a = select_pred(step1$DB_READY,
                     Y = "Y", Z = "Z", ID = 1, OUT = "Y",
                     quanti  = c(4,6), nominal = c(1,5,7),
                     ordinal = c(2,3,8:12), thresh_cat = 0.50,
                     thresh_num = 0.70, RF_SEED = 3011)

# For the dataset api35 --------------
step2b = select_pred(step1$DB_READY,
```

```
                    Y = "Y", Z = "Z", ID = 1, OUT = "Z",
                    quanti  = c(4,6), nominal = c(1,5,7),
                    ordinal = c(2,3,8:12), thresh_cat = 0.50,
                    thresh_num = 0.70, RF_SEED = 3011)
```

The `quanti`, `nominal`, and `ordinal` arguments requires vectors of column indexes related to the type of each variable of the input database. The `ID` argument specifies the column index of the database identifier. `Y` and `Z` expected the respective names of the target variables while `OUT` provides the target variable to predict (`Y` or `Z`).

To detect the subset of best predictors, `select_pred` studies the standard pairwise associations between each shared variable and the outcomes `Y` and `Z`, taken separately (by only varying the argument `OUT`), according to its type: for numeric and/or ordered factor variables, `select_pred` calculates the associations using rank correlation coefficients (Spearman) while the Cramer's V criterion (Bergsma 2013) is used for categorical variables and not ordered factors. The related ranking tables of top scoring predictors are available in two distinct output objects: `cor_OUTC_num` and `cor_OUTC_cat`. In our example, the corresponding results, for each database, are:

```
### ASSOCIATIONS BETWEEN TARGET VARIABLES AND SHARED VARIABLES

## Results for the api29 dataset -----

step2a$cor_OUTC_num   # Y versus numeric or ordinal predictors

#>   name1    name2 RANK_COR pv_COR_test   N
#> 7     Y    meals  -0.8030      0.0000 418
#> 4     Y      ell  -0.7514      0.0000 389
#> 5     Y     full   0.3919      0.0000 418
#> 6     Y grad.sch   0.3346      0.0000 418
#> 3     Y  api.stu  -0.1520      0.0018 418
#> 8     Y    stype  -0.1520      0.0018 418
#> 2     Y acs.core  -0.0556      0.2566 418

step2a$vcrm_OUTC_cat  # Y versus nominal or ordinal predictors

#>   name1     name2 V_Cramer CorrV_Cramer   N
#> 7     Y     meals   0.6871       0.6835 418
#> 4     Y       ell   0.6015       0.5966 389
#> 5     Y      full   0.4508       0.4459 418
#> 6     Y  grad.sch   0.3869       0.3816 418
#> 3     Y    awards   0.2188       0.2073 401
#> 2     Y acs.k3.20   0.1514       0.1349 418
#> 8     Y     stype   0.1370       0.1185 418

## Results for the api35 dataset -----

step2b$cor_OUTC_num   # Z versus numeric or ordinal predictors

#>   name1    name2 RANK_COR pv_COR_test   N
#> 4     Z      ell  -0.7511      0.0000 337
#> 7     Z    meals  -0.7291      0.0000 362
#> 6     Z grad.sch   0.6229      0.0000 362
#> 5     Z     full   0.3997      0.0000 362
#> 3     Z  api.stu  -0.0563      0.2851 362
#> 8     Z    stype   0.0359      0.4959 362
#> 2     Z acs.core   0.0119      0.8219 362

step2b$vcrm_OUTC_cat  # Z versus nominal or ordinal predictors

#>   name1     name2 V_Cramer CorrV_Cramer   N
#> 7     Z     meals   0.5181       0.5121 362
#> 6     Z  grad.sch   0.5131       0.5063 362
#> 4     Z       ell   0.4775       0.4701 337
#> 5     Z      full   0.4033       0.3934 362
```

```
#> 3     Z    awards   0.2040       0.1818 348
#> 8     Z     stype   0.1320       0.0957 362
#> 2     Z acs.k3.20   0.1223       0.0817 362
```

The two first tables related to `api29` highlights the strong associations between `Y` and the variables `meals`, `ell`, `full` and `grad.sch` in this order of importance, while the summary tables related to `api35` highlights the strong associations between `Z` and the variables `meals`, `grad.sch`, `ell` and `full`.

It is often not unusual to observe that one or more predictors are in fact linear combinations of others. In supervised learning areas, these risks of collinearity increase with the number of predictors, and must be detected beforehand to keep only the most parsimonious subset of predictors for fusion. To avoid collinearity situations, the result of a *Farrar and Glauber (FG) test* is provided (Farrar and Glauber 1967). This test is based on the determinant of the rank correlation matrix of the shared variables D (Spearman) and the corresponding test statistic is given by:

$$S_{FG} = -\left(n - 1 - \frac{1}{6}(2k+5) \times \ln(\det(D))\right)$$

where n is the number of rows and k is the number of covariates. The null hypothesis supposes that S_{FG} follows a chi square distribution with $k(k-1)/2$ degrees of freedom, and its acceptation indicates an absence of collinearity. In presence of a large number of numeric covariates and/or ordered factors, the approximate Farrar-Glauber test, based on the normal approximation of the null distribution (Kotz, Balakrishnan, and Johnson 2000) can be more adapted and the statistic test becomes:

$$\sqrt{2S_{FG}} - \sqrt{2k-1}$$

Users will note that these two tests can often appear highly sensitive in the sense that they tend to easily conclude in favor of multicollinearity. Thus, it is suggested to consider these results as indicators of collinearity between predictors rather than an essential condition of acceptability. The results related to this test are stored in the `FG_test` object of the `select_pred` output.

In presence of collinearity, `select_pred` tests the pairwise associations between all the potential predictors according to their types (Spearman or Cramér's V). The `thres_num` argument fixed the threshold beyond which two ranked predictors are considered as redundant while `thres_cat` fixed the threshold of acceptability for the Cramér's V criterion in the subgroup of factors. In output, the results stored in the `collinear_PB` object permit to identify the corresponding variables. In our example, we observe:

```
### DETECTION OF REDUNDANT PREDICTORS

## Results for the api29 dataset -----

## Results of the Farrar-Glauber test
step2a$FG_test

#>           DET_X      pv_FG_test pv_FG_test_appr
#>      0.04913067      0.00000000      0.00000000

## Identity of the redundant predictors

step2a$collinear_PB

#> $VCRAM
#>        name1 name2 V_Cramer CorrV_Cramer   N
#> 71 acs.k3.20 stype   0.6988       0.6971 418
#> 43       ell meals   0.6696       0.6664 389
#> 34      full meals   0.5215       0.5152 418
#>
#> $SPEARM
#>      name1 name2 RANK_COR pv_COR_test   N
#> 43     ell meals   0.9047           0 389
#> 62 api.stu stype   0.7069           0 418

#### Results for the api35 dataset -----

step2b$FG_test    # Significant result
```

```
#>           DET_X      pv_FG_test pv_FG_test_appr
#>       0.1479339       0.0000000       0.0000000

step2b$collinear_PB

#> $VCRAM
#>        name1 name2 V_Cramer CorrV_Cramer   N
#> 71 acs.k3.20 stype   0.6977       0.6957 362
#>
#> $SPEARM
#> [1] name1       name2       RANK_COR    pv_COR_test N
#> <0 rows> (or 0-length row.names)
```

The FG test warns the user against the risks of collinearity between predictors, and the function notably detects strong collinearities between the variables `meals`, `ell`, `full` in the `api29` (less strong trends in `api35`): an information that have to be taken into account during the selection. The part of predictors finally kept for the data fusion must be small to improve its quality. When the initial number of shared variables is not too important as in this example, choosing the best candidates between groups of redundant predictors can be made manually by selecting highest ranked predictors in the summary tables previously described. In this way, the variable `meals` could be preferred to `ell`, and `full`, while the variable `stype` could be dropped. Consequently, a possible final list of predictors could be only composed of the variables `meals` and `grad.sch`.

When the number of predictors is important, or when users prefer that an automatic process performs the variable selection, a random forest procedure can also be used via the `select_pred` function. Random forest approaches (Leo Breiman 2001) are here particularly convenient (Grajski et al. 1986) for multiple reasons: it works fine when the number of variables exceeds the number of rows, whatever the types of covariates, it allows to deal with non linearity, to consider correlated predictors, ordered or not ordered outcomes and to rank good candidate predictors through an inbuilt criterion: the variable importance measure.

In few words, random forest processes aggregates many CART models (L. Breiman et al. 1984) with `RF_ntree` bootstrap samples from the raw data source and averaging accuracies from each model permits to reduce the related variances and also the errors of prediction. A standard random forest process provides two distinct measures of importance of each variable for the prediction of the outcome, the Gini importance criterion and the permutation importance criterion, which depends on the appearance frequency of the predictor but also on its place taken up in each tree. For more details about random forest theory, user can consult Leo Breiman (2001) and/or the pdf manual of the **randomForest** (Liaw and Wiener 2002) package.

Strobl, Hothorn, and Zeileis (2009) suggests that the permutation importance criterion, which works with permuted samples (subsampling without replacements) instead of bootstrap ones, is particularly convenient with uncorrelated predictors, but must be replaced by a conditional permutation measurement in presence of strong correlations. `select_pred` provides these assessments by integrating the main functionalities of the `cforest` and `varimp` functions of the package **party**(Hothorn, Hornik, and Zeileis 2006; Zeileis and Hothorn 2008; Hothorn et al. 2005; Strobl et al. 2007, 2008). However, these measurements must be used with caution, by accounting the following constraints:

- the Gini importance criterion can produce bias in favor of continuous variables and variables with many categories. This criterion is thus not available in the function.
- the permutation importance criterion can overestimate the importance of highly correlated predictors and therefore redundant predictors will be discarded beforehand using the first steps of the process integrated in the function.

Possible scenarios	Correlation between predictors	State of the RF_condi argument	Incomplete information
Same type predictors	NO	FALSE	Allowed
Same type predictors	YES	TRUE	Not allowed
Different type predictors	NO	FALSE	Allowed
Different type predictors	YES	TRUE	Not Allowed

Table 4: Completing the `RF_condi` argument according to predictors

The `select_pred` function allows to proceed with different scenarios according to the type of predictors (Table 4 can help users to choose). The first one consists in boiling down to a set of

categorical variables (ordered or not) by categorizing all the continuous predictors using the dedicated argument (`convert_num` and `convert_clss`) and to work with the conditional importance assessments that directly provide unbiased estimations (by setting the `RF_condi` argument to `TRUE`). Users can consult (Hothorn, Hornik, and Zeileis 2006) for more details about the approach and consult the pdf manual of the package for details about the related arguments of the `select_pred` function. This approach does not take into account incomplete information, so that the method will be applied to complete data only (incomplete rows will be temporarily removed from the study). It is nevertheless possible to impute missing data beforehand by using dedicated pre-existing `R` packages like **mice** (van Buuren and Groothuis-Oudshoorn 2011) or by using the `imput_cov` function provided in the **OTrecod** package.

The second possible scenario (always usable in presence of mixed type predictors), consists in the execution of a standard random forest procedure after taking care to rule out collinearity issues by first selecting unique candidates between potential redundant predictors (in this case, the discarded predictors are stored in the `drop_var` output object). This is the default approach used by `select_pred` as soon as the `RF_condi` argument is set to `FALSE` while `RF` is set to `TRUE`. This scenario can work in presence of incomplete predictors. By constructing, note that results from random forest procedures stay dependent on successive random draws carried out for the constitution of trees, and it is so suggested to check this stability by testing different random seeds (`RF_SEED` argument) before concluding.

The `R` command previously written provides automatically the results related to the second approach as additional results. The results from the two datasets show here the permutation importance estimates of each variable ranked in order of importance and expressed as percentage, after resolving collinearity problems:

```
step2a$RF_PRED   # For the api29 dataset

#>    meals    stype grad.sch   awards acs.core
#>  71.5529  11.6282  11.1181   5.4674   0.2334

step2b$RF_PRED   # For the api35 dataset

#>    meals      ell grad.sch     full    stype  api.stu acs.core   awards
#>  35.9974  28.3051  22.2094   5.2143   4.0192   1.8965   1.5643   0.7937
```

The results confirm that the variable `meals` appeared as the best predictor of the target variables in `api29` and `api35` respectively. The variable `ell` is not present in the first list (see `RF_PRED` from `step2a`) because the variables `meals` and `ell` has been detected as strongly collinear (according to the initial chosen threshold) and so `select_pred` keep the best choice between the two predictors: `meals` (the reasoning is the same for `full` which disappeared from the list).

The `ell` variable has been kept in the second list (not found as collinear enough to be removed here) and appears moreover as a good predictor of `apicl_1999` in `api35`. Nevertheless its potential collinearity problem with `meals` encourages us not to keep it for the rest of the study. According to this discussion, we finally keep `meals`, `stype` and `grad.sch` which combines the advantages of being good predictors for the two target variables while not presenting major problems of collinearity between them.

The following synthetic database (called here `bdd_ex`) is now ready for the prediction of the missing *API scores* in `api29`, `api35` or both, using the function `OT_outcome` or `OT_joint`:

```
bdd_ex = step1$DB_READY[, c(1:3,10:12)]; head(bdd_ex,3)

#>      DB         Y    Z grad.sch    meals stype
#> 2850  1 (600-800] <NA>      >10   [0-25]     H
#> 2851  1 (600-800] <NA>      >10   [0-25]     H
#> 2852  1 [200-600] <NA>     1-10 (75-100]     H
```

Data fusion using `OT_outcome`

The `OT_outcome` function provides individual predictions of Z in A (and/or Y in B) by considering the recoding problem involving optimal transportation of outcomes. In a first step, the aim of the function is so to determine γ from (2) while this estimate is used in a second step to provide the predictions. The main input arguments of the function and the output values are respectively described in Tables 5 and 6.

Argument	OT_outcome	OT_joint	Description (default value)
datab	✓	✓	Data.frame in the expected structure
index_DB_Y_Z	✓	✓	Indexes of the ID, Y, and Z columns (1,2,3)
nominal	✓	✓	Column indexes of nominal variables (*)
ordinal	✓	✓	Col. indexes of ordinal variables (*)
logic	✓	✓	Col. indexes of boolean variables (*)
quanti	✓		Col. indexes of quantitative variables (*)
convert.num	✓	✓	Col. indexes of the quantitative variables to convert (* or =quanti in OT_joint)
convert.clss	✓	✓	Corresponding numbers of desired classes for conversion (*)
which.DB	✓	✓	Specify the target variables to complete: Both or only one (BOTH)
solvR	✓	✓	Choice of the solver to solve the optimization problem (glpk)
dist.choice	✓	✓	Distance function (Euclidean). See Table 7
percent.knn	✓	✓	Ratio of closest neighbors involved int the computations (1)
indiv.method	✓		Type of individual predictions (sequential) for OUTCOME and R-OUTCOME algorithms
maxrelax	✓	✓	Adding of a relaxation parameter (0)
lambda.reg		✓	Adding of regularization parameter (0)

Table 5: Main arguments of the OT_outcome and OT_joint functions. (*: NULL as default value)

The arguments datab, index_DB_Y_Z, quanti, nominal, ordinal, and logic are not optional and must be carefully completed before each execution. A unique synthetic data.frame made of two overlayed databases (called A and B for example) (see Table 3) is expected as datab argument. If this data.frame corresponds to the output objects DB_USED or DB_READY of the select_pred or merge_dbs functions respectively, then the expected object has the required structure. Otherwise users must be sure that their data.frames are made up of two overlayed databases with at least 4 variables as described in the subsection Optimal transportation of outcomes applied to data recoding. The order of the variables have no importance in the raw database but will have to be specified a posteriori in the index_DB_Y_Z argument if necessary.

The subset of remaining predictors used for fusion may requires prior transformations according to the distance function chosen in input by the user. This process is fixed by setting the dist.choice argument. The distances actually implemented in the function are: the standard Euclidean ("E") and Manhattan ("M") distances, the Hamming distance ("H", for binary covariates), and the Gower distance ("G" sometimes preferred with mixed variables). Automatic transformations prior to the use of each distance function are summarized in Table 7.

The first version of the OT algorithm described in Garès et al. (2020) was tested on numeric coordinates extracted from a factor analysis of mixed data (FAMD) fitted on mixed raw covariates (Pagès 2002). This transformation is here available by setting the FAMD.coord argument to "YES". The minimal percentage of information explained by the FAMD is also fixable using the FAMD.perc argument. The OT_outcome functions proposes four types of models for the prediction of Y in B (and/or) Z in A, according to the values of the method and maxrelax arguments:

- When maxrelax = 0 and indiv.method = "sequential" (default options), the fitted model corresponds to the OUTCOME algorithm described by $\hat{\mathcal{P}}_n^0$ in equation (6). Assuming that Y and Z in A follow the same distribution as Y and Z in B respectively (**assumption 1**), this related algorithm derives the joint distribution of Y and Z in A (respectively in B) in a first step, and uses in a second step, a nearest neighbor procedure to predict missing values of Z in A (resp. Y in B). This algorithm calculates averaged distances between each subject from A (resp. B) and subgroups of subjects from B (resp. A) having same levels of Z in B (resp. Y in A). These

Value	OT_outcome	OT_joint	Description
time_exe	✓	✓	Running time of the algorithm
gamma_A	✓	✓	Estimation of the joint distribution of (Y, Z) for the prediction of Z in A (*)
gamma_B	✓	✓	Estimation of the joint distribution of (Y, Z) for the prediction of Y in B (*)
profile	✓	✓	The list of detected profiles of covariates
res.prox	✓	✓	A list that provides all the information related to the estimated proximities between profiles and groups of profiles
estimator_ZA	✓	✓	Estimates of the probability distribution of Z conditional to X and Y in database A (*)
estimator_YB	✓	✓	Estimates of the probability distribution of Y conditional to X and Z in database B (*)
DATA1_OT	✓	✓	The database A fully completed (if required in input by the which.DB argument)
DATA2_OT	✓	✓	The database B fully completed (if required in input by the which.DB argument)

Table 6: Values of the OT_outcome and OT_joint functions. (*: NULL if not required)

Distance function	Euclidean	Manhattan	Gower	Hamming
Variable transformations				
Continuous	Standardized	Standardized	No	Not allowed
Boolean	Binary	Binary	No	Binary
Nominal	Disjunctive T	Disjunctive T	No	Disjunctive T
Ordinal	Discrete	Discrete	No	Disjunctive T
Incomplete information	Allowed*	Allowed*	Allowed*	Allowed*
dist.choice **argument**	"E"	"M"	"G"	"H"

Table 7: Internal variable transformations related to the choice of each distance function in OT_outcome and OT_joint. (*) If the number of covariates exceeds 1. T for table

calculations can be done using all subjects of each subgroups (by default, percent.knn = 1) or only restricted parts of them (percent.knn < 1).

- When maxrelax > 0 and indiv.method = "sequential", **assumption 1** is alleviated by relaxing the constraints on marginal distributions and an R-OUTCOME algorithm (with relaxation) is fitted. In this case, the maxrelax argument corresponds to the α_n parameter in (18).
- When maxrelax = 0 and indiv.method = "optimal", the second step of the original algorithm (nearest neighbor procedure) is replaced by a linear optimization problem: searching for the individual predictions of Z in A (resp. Y in B) by minimizing the total of the distances between each individual of A and individuals of each levels of Z in B.
- When maxrelax > 0 and indiv.method = "optimal", the constraints on marginal distributions of the previous model are also relaxed and the function fits an R-OUTCOME algorithm with relaxation. For these three last situations, the corresponding R-OUTCOME algorithm is so described by $\hat{\mathcal{P}}_n^{0-R}$ (18).

When maxrelax > 0, it is recommended to calibrate the maxrelax parameter by testing different values according to the stability of individual predictions. In output, the gamma_A and gamma_B} matrices correspond to the estimates of the joint distributions γ of (Y^A, Z^A) and (Y^B, Z^B) respectively (2). Moreover, a posteriori estimates of conditional distributions probabilities $(Z^A|Y^A, X^A)$ and $(Y^B|Z^B, X^B)$ (10) are also provided in two lists called estimatorZA, and estimatorYB respectively and the completed databases A and B are stored in the DATA1_OT and DATA2_OT objects. In particular, the individual predictions are stored in the OTpred column of these data.frames. Moreover, the profile object is a data.frame that stores the profile of covariates encountered in the two data sources while

the res_prox object stored all the distance computations that will be used in the validation step (users can consult details of the proxim_dist function of the pdf manual.

The computation of conditional probabilities implies to define the part of individuals considered as neighbors of each encountered profile of covariates. The argument prox.dist fixes this threshold for each profile, following the decision rule for A (and respectively for B): a subject i of A (or a statistical unit corresponding to a row of A) will be considered as neighbor of a given profile of shared variables C as soon as:

$$\text{dist}(subject_i, C) < \text{prox.dist} \times \max_{k=1,\ldots,n_A} \text{dist}(subject_k, C)$$

When the number of shared variables is small and all of them are categorical (optimal situation), it is suggested to set the prox.dist argument to 0. Finally, using the which.DB argument, users can choose to estimate the individual predictions of Y and Z in the two databases (which.DB = "BOTH") or only one of the both (which.DB = "A" for the predictions of Z in A or which.DB = "B" for the predictions Y in B).

From the data.frame bdd_ex built previously, we use the OT_outcome function to illustrate the prediction of the target variable apicl_2000 in the api35 dataset via an OUTCOME algorithm and using an Euclidean distance function:

```
outc1 = OT_outcome(bdd_ex, quanti = 1, ordinal = 2:6,
                   dist.choice = "E", indiv.method = "sequential",
                   which.DB = "B")

#---------------------------------------
# OT PROCEDURE in progress ...
#---------------------------------------
# Type                     = OUTCOME
# Distance                 = Euclidean
# Percent closest knn      = 100%
# Relaxation parameter     = NO
# Relaxation value         = 0
# Individual pred process  = Sequential
# DB imputed               = B
#---------------------------------------
```

In bdd_ex, all variables except the first one (DB: the database identifier) are or can be considered as ordinal factors, and the quanti and ordinal arguments are filled in accordingly. For the prediction of apicl_2000 in the api35 dataset (the steps would be the same for the prediction of apicl_1999 in api29 by setting which.DB = "A" or "BOTH"), the optimal transportation theory determines a map γ that pushes, in the distribution of apicl_1999 forward to the distribution of apicl_2000 in the database api35. In this case, γ^B is an estimator of the joint distribution of apicl_2000 and apicl_1999 in api35. The estimation of γ^B is available in the gamma_B output object while all the profiles of predictors met in the two databases are listed in the profile object:

```
outc1$gamma_B

#>                     G1        G2         G3        G4
#> [200-600]  0.22248804 0.0000000 0.00000000 0.0000000
#> (600-800]  0.02889318 0.2486188 0.15311005 0.0000000
#> (800-1000] 0.00000000 0.0000000 0.09550874 0.2513812

outc1$profile[1,]         # the first profile

#>       ID grad.sch meals stype
#> 2850 P_1        3     1     3
```

The output object estimatorYB is a list that corresponds to the estimations of the conditional probabilities of apicl_2000 in the api35 dataset for a given profile of predictors. For example, the conditional probabilities related to the first profile of predictors are:

```
outc1$estimatorYB[1,,]    # conditional probabilities (1st profile)

#>         [,1]      [,2]      [,3]
#> G1 0.3333333 0.3333333 0.3333333
#> G2 0.3333333 0.3333333 0.3333333
#> G3 0.0000000 0.0000000 1.0000000
#> G4 0.0000000 0.0000000 1.0000000
```

According to these results, we can conclude that: for a subject from api35 with a related profile of predictors P_1 and whose levels of apicl_1999 is 'G1', the probability for apicl_2000 to be predicted '[200,600]' is about 0.63. Finally, the individual predictions of apicl_2000 in api35 are available in the DATA2_OT object corresponding to the OTpred column:

```
head(outc1$DATA2_OT,3)  # The 1st 3 rows only

#>      DB    Y  Z grad.sch meals stype    OTpred
#> 3895  2 <NA> G1        2     4     1 [200-600]
#> 3896  2 <NA> G3        2     3     1 (600-800]
#> 3897  2 <NA> G2        2     3     2 (600-800]
```

The OUTCOME algorithm uses here a nearest neighbor procedure to assign the individual predictions from the estimation of γ which can be a drawback as described in (Garès et al. 2020). The R-OUTCOME algorithm proposes an enrichment that directly assigns the individual predictions of apicl_2000 from the estimation of γ, without using the nearest neighbor approach. In this case, a linear optimization problem is solved to determine the individual predictions that minimize the sum of the individual distances in api35 having the modalities of the target apicl_2000 in api29. The corresponding R command is:

```
### R-OUTCOME algorithm: optimal assignments + relaxation parameter = 0
R_outc3 = OT_outcome(bdd_ex, quanti = 1, ordinal = 2:6,
                     dist.choice = "E" , indiv.method = "optimal",
                     which.DB = "B")
```

Moreover, the OUTCOME algorithm assumes that the distribution of apicl_2000 is identically distributed in api29 and api35 (**assumption 1** from the subsection Optimal transportation of outcomes applied to data recoding) which can appear as a strong hypothesis to hold in many situations. To overcome this constraint, the R-OUTCOME algorithm also allows to relax the **assumption 1** by adding a relaxation parameter (18) in the optimization system. This relaxation can be done by varying the maxrelax argument of the function:

```
### R-OUTCOME algorithm: optimal assignments + relaxation parameter = 0.4
R_outc4 = OT_outcome(bdd_ex, quanti = 1, ordinal = 2:6,
                     dist.choice = "E",
                     indiv.method = "optimal",
                     maxrelax = 0.4, which.DB = "B")
```

The running times of these two models can take few minutes. The quality of these models will be compared in Tables 8, 9 and 10 (subsection Validation of the data fusion using verif_OT).

Data fusion using OT_joint

The OT_joint function provides individual predictions of Z in A (and/or Y in B) by considering the recoding problem as an optimal transportation problem of covariates and outcomes, that pushes the conditional distribution of $(Y^A|X^A)$ forward to the conditional distribution of $(Z^A|X^A)$ (and conversely $(Z^B|X^B)$ forward to the conditional distribution of $(Y|X)$). The aim is to determine γ from (7). Because joint distributions of outcomes and covariates are now mapped together (it was not the case with the OUTCOME family of algorithms), it is not required for target variables to be equally distributed (**assumption 1**).

The call of OT_joint was thought to propose globally the same structure as those of the function OT_outcome as described in Tables 5 and 6 with few differences described below. JOINT and R-JOINT algorithms are usable via OT_joint for solving the recoding problem depending on the values related to the maxrelax and lambda_reg arguments:

- When maxrelax = 0 and lambda.reg = 0 (default values), the fitted model corresponds to the JOINT algorithm described by $\hat{\mathcal{P}}_n$ in equation (7).
- When at least one of these two arguments differs from 0, the R-JOINT algorithm is called. Using R-JOINT, it is so possible to relax constraints on marginal distributions (maxrelax > 0) and/or add an eventual more or less strong *L1* regularisation term among the constraints of the algorithm by filling the lambda_reg argument. maxrelax parameter correspond to parameter α_n in (16) and lambda.reg parameter correspond to parameter λ in (17).

When `maxrelax > 0` and/or `lambda.reg > 0`, it is recommended to calibrate these two parameters by testing many different values and so studying the stability of the related individual predictions. Nevertheless, by default or as a starting point, (Garès et al. 2020) suggests the use of default values determined from simulated databases: 0.1 for the regularization parameter (`lambda.reg`) and 0.4 for the relaxation one (`maxrelax`). Finally, the output objects are similar to those previously described in the `OT_outcome` function.

The implementation of the function is very similar to that of `OT_outcome`, and applied to our example, the `R` code related to a `JOINT` algorithm is:

```
outj1 = OT_joint(bdd_ex, nominal = 1, ordinal = c(2:6),
                 dist.choice = "E"  , which.DB = "B")

#---------------------------------------
# OT JOINT PROCEDURE in progress ...
#---------------------------------------
# Type                  = JOINT
# Distance              = Euclidean
# Percent closest       = 100%
# Relaxation term       = 0
# Regularization term   = 0
# Aggregation tol cov   = 0.3
# DB imputed            = B
#---------------------------------------


### Extract individual predictions from the OTpred column
head(outj1$DATA2_OT,3)  # The 1st 3 rows only

#>      DB    Y  Z grad.sch meals stype    OTpred
#> 3895  2 <NA> G1        2     4     1 [200-600]
#> 3896  2 <NA> G3        2     3     1 (600-800]
#> 3897  2 <NA> G2        2     3     2 (600-800]
```

For relaxing the constraints stated on marginal distributions, we use the `maxrelax` argument that corresponds to varying the α parameter of a `R-JOINT` algorithm (see (17)) and a parameter of regularization λ can be simply added to the previous model as follows:

```
### R-JOINT algorithm (relaxation parameter = 0.4)
R_outj1 = OT_joint(bdd_ex, nominal = 1, ordinal = c(2:6),
                   dist.choice = "E",  maxrelax = 0.4,
                   which.DB = "B")

### R-JOINT algorithm (relaxation parameter = 0.4,
###                                   & regularization parameter = 0.1)
R_outj4 = OT_joint(bdd_ex, nominal = 1, ordinal = c(2:6),
                   dist.choice = "E",  maxrelax = 0.4,
                   lambda.reg = 0.1,
                   which.DB = "B")
```

These models are compared in the next subsection.

Validation of the data fusion using `verif_OT`

Assessing the quality of individual predictions obtained through statistical matching techniques can be a complex task notably because it is legitimate to wonder about the true reliability of a joint distribution estimated from two variables which are never jointly observed (Kadane 2001; Rodgers 1984). When the study aims to identify estimators of interests that improve the understanding of the relationships between variables in the different databases (called macro approach in D'Orazio, Di Zio, and Scanu (2006)) without going until the creation of a complete and unique dataset, uncertainty analyses are usually proposed to estimate the sensitivity of the assessments (Rässler 2002). On the contrary, if the objective is to create a synthetic dataset where the information is available for every unit, the use of auxiliary information or proxy variables must be privileged to assess the quality of the results (Paass 1986). In the absence of complementary information, Rubin (1986) suggests to study the preservation

of the relationships at best between the distributions of the target variables. In this way, the `verif_OT` function proposes tools to assess quality of the individual predictions provided by the algorithms previously introduced.

The first expected input argument of `verif_OT` is an `'otres'` object from the `OT_outcome` or `OT_joint` functions. In our example, this function is firstly applied to the `out_c1` model by following the R command:

```
### Quality criteria for outc1 (OUTCOME model)
verif_outc1   = verif_OT(outc1, group.class = TRUE, ordinal = FALSE,
                         stab.prob = TRUE, min.neigb = 5)

### First results related to outc1:
verif_outc1$nb.profil

#> [1] 27

verif_outc1$res.prox

#>        N   V_cram rank_cor
#>  362.000    0.860    0.892
```

In output, the `nb.profil` value gives the number of profiles of predictors globally detected in the two databases ($nb = 27$). In the example, a profile will be characterized by a combination of values related to the three predictors kept: `grad.sch`, `meals` and `stype`.

The first step of the function is dedicated to the study of the proximity between the two target variables. Therefore, standard criteria (Cramer's V and Spearman's rank correlation coefficient) are used to evaluate the pairwise association between Y and Z, globally or in one of the two databases only, according to the predictions provided by the output of `OT_outcome` or `OT_joint`. Only the database `api35` was completed in the example (because `which.DB = "B"` as argument of `OT_outcome`) and stored in the `DATA2_OT` object, therefore, the criteria compare here the proximity distribution between the predicted values of `apicl_2000` (`Y`) and the observed value of `apicl_1999` (`Z`) in the database `api35` (`B`). Regarding independence or a small correlation between Y^A and Z^A (or Y^B and Z^B) must question about the reliability of the predictions especially when Y and Z are supposed to summarize a same information. In the example, whatever the criteria used, we notice via the `res.prox` object, a strong association between the two target variables in `api35` which reassures about the consistency of the predictions. The related confusion matrix between the predicted values of `apicl_2000` (`Y`) and the observed value of `apicl_1999` (`Z`) is stored in the `conf.mat` object.

Second, the function proposes an optional tool (by setting `group.clss= TRUE`) which evaluates the impact of grouping levels of one factor on the association of Y and Z. When users have initially no information about one of the two encodings of interest, this approach can be particularly useful to detect ordinal from nominal ones and its principle is as follow. Assuming that $Y \in \mathcal{Y}$, and $Z \in \mathcal{Z}$ ($\mathcal{Y}$ and $\mathcal{Z}$ are the respective levels) and that $|\mathcal{Y}| \geq |\mathcal{Z}|$. From Y, successive new variables $Y' \in \mathcal{Y}'$ are built, as soon as $\mathcal{Y}'$ is a partition of $\mathcal{Y}$ such as $|\mathcal{Y}'| = |\mathcal{Z}|$ (and inversely for Z if the levels of Z is the greatest). The related associations between Z and Y' (with now equal number of levels) are then studied using: Cramer's V, rank correlation, Kappa coefficient and confusion matrix and the results are stored in a table called `res.grp`. The corresponding outputs of the example are:

```
verif_outc1$conf.mat

#>             Z
#> predY         G1  G2  G3  G4 Sum
#>   [200-600]   81   1   1   1  84
#>   (600-800]   10  89  55   1 155
#>   (800-1000]   0   0  34  89 123
#>   Sum         91  90  90  91 362

verif_outc1$res.grp

#>   grp levels Z to Y error_rate Kappa Vcramer RankCor
#> 4       G1/G2 G3/G4       13.3 0.794    0.83   0.877
#> 6       G1/G2/G3 G4       19.1 0.714    0.78   0.839
#> 2       G1 G3/G2/G4       28.2 0.593    0.68   0.624
#> 1       G1 G2/G3/G4       37.6 0.457    0.64   0.813
#> 3       G1 G4/G2/G3       43.4 0.374    0.59   0.115
#> 5       G1/G2 G4/G3       43.4 0.326    0.64   0.574
```

It appears from these results that grouping the levels G2 and G3 of apicl_1999 (Z) strongly improves the association of this variable with apicl_2000 (Y) (the error rate of the confusion matrix varies from 43.4 to 13.3). Moreover the structure of conf.mat confirms that the encoding of apicl_1999 seems to be ordinal.

The third step of the quality process integrated in the function corresponds to the study of the Hellinger distance (Liese and Miescke 2008). This distance function is used as a measure of the discrepancies between the observed and predicted distributions of Y ($\mathcal{L}(Y^A)$ versus $\mathcal{L}(\widehat{Y}^B)$) and/or ($\mathcal{L}(\widehat{Z}^A)$ versus $\mathcal{L}(Z^B)$). For Y and Z, the definition of the distance is respectively:

$$\text{dist}_{\text{hell}}(Y^A, \widehat{Y}^B) = \sqrt{\frac{1}{2}\sum_{y\in\mathcal{Y}}\left(\sqrt{\mu_{n,y}^{Y^A}} - \sqrt{\mu_{n,y}^{\widehat{Y}^B}}\right)^2)}$$

and

$$\text{dist}_{\text{hell}}(Z^B, \widehat{Z}^A) = \sqrt{\frac{1}{2}\sum_{z\in\mathcal{Z}}\left(\sqrt{\mu_{n,z}^{Z^B}} - \sqrt{\mu_{n,z}^{\widehat{Z}^A}}\right)^2)},$$

where $\mu_{n,y}^{\widehat{Y}^B}$ and $\mu_{n,z}^{\widehat{Z}^A}$ correspond to the empirical estimators of $\mu^{\widehat{Y}^B}$ and $\mu^{\widehat{Z}^A}$ respectively.

The Hellinger distance varies between 0 (identical) and 1 (strong dissimilarities) while 0.05 can be used as an acceptable threshold below which two distributions can be considered as similar. When OUTCOME models are applied on datasets, this criterion shows that predictions respect **assumption 1**. It can also be used to determine the best relaxation parameter of an R-OUTCOME model. On the contrary, there is no need to interpret this criterion when an R-JOINT model is applied, because in this case, the **assumption 1** is not required. Results related to this criterion are stored in the hell object:

```
verif_outc1$hell
```

```
#>                  YA_YB ZA_ZB
#> Hellinger dist. 0.008    NA
```

With a p-value equals to 0.008, the **assumption 1** hold here but it stays interesting to test other algorithms to eventually improve the current one by notably adding relaxation and/or regularization parameters.

Finally, the verif_OT function uses the mean and standard deviance of the conditional probabilities $\mathbb{P}(Z = \hat{z}_i|Y = y_i, X = x_i)$ estimated by each model, as indicators of the stability of the individual predictions (provided that stab.prob = TRUE). It is nevertheless possible that conditional probabilities are computed from too few individuals (according to the frequency of each profile of shared variables met), to be considered as a reliable estimate of the reality. To avoid this problem, trimmed means and standard deviances are suggested by removing these specific probabilities from the computation, using the min.neigb parameter. In output, the results related to this last study are stored in the res.stab object:

```
verif_outc1$eff.neig
```

```
#>    Nb.neighbor Nb.Prob
#> 1            1      14
#> 2            2      18
#> 3            3      18
#> 4            4      28
#> 5            5      20
#> 6            6       6
#> 7            7      14
#> 8            8      16
#> 9            9      27
#> 10          10      20
```

```
verif_outc1$res.stab
```

```
#>          N min.N  mean    sd
#> 2nd DB 284     5 0.968 0.122
```

The first result shows that 14 individual predictions among 362 (the total number of rows in `api35`) have been assigned to subjects that exhibits a unique combination of predictors and outcome. From these subjects, it would be obviously overkill to draw conclusions about the reliability of the predictions provided by the model. We therefore decide to fix here a threshold of 5 below which it is difficult to extract any information related to the prediction. From the remaining ones, we simply perform the average (0.968) which could be interpreted as follows: when the fitted model is confronted with a same profile of predictors and a same level of `apicl_1999`, more than 96 times of a hundred, it will return the same individual prediction for `apicl_2000`.

We run a total of 11 models to determine the optimal one for the prediction of `apicl_2000` in `api35`. Every arguments from the syntax of the `OT_outcome` and `OT_joint` functions stay unchanged with the exception of `indiv.method`, `maxrelax`, and `lambda.reg` which vary. The values linked to each criterion of the `verif_OT` function are summarized in Tables 8, 9 and 10. The syntax related to `verif_OT` stay unchanged for each model (notably same min.neigb arguments). Note that comparisons of stability predictions between `OUTCOME` and `JOINT` models impose that the `prox.dist` argument of the OT_outcome function is fixed to 0.

Model	Type	Method	Relax	Regul	N	V_cram	rank_cor
outc1	OUTCOME	SEQUENTIAL	0.0	-	362	0.86	0.892
R_outc1	R-OUTCOME	SEQUENTIAL	0.4	-	362	0.94	0.923
R_outc2	R-OUTCOME	SEQUENTIAL	0.6	-	362	0.91	0.917
R_outc3	R-OUTCOME	OPTIMAL	0.0	-	362	0.87	0.911
R_outc4	R-OUTCOME	OPTIMAL	0.4	-	362	0.95	0.939
R_outc5	R-OUTCOME	OPTIMAL	0.6	-	362	0.92	0.932
outj1	JOINT	-	0.0	0.0	362	0.74	0.834
R_outj1	R-JOINT	-	0.4	0.0	362	0.95	0.935
R_outj2	R-JOINT	-	0.6	0.0	362	0.91	0.927
R_outj3	R-JOINT	-	0.8	0.0	362	0.91	0.927
R_outj4	R-JOINT	-	0.4	0.1	362	0.95	0.931

Table 8: V Cramer criterion (`V_cram`) and rank correlation (`rank_cor`) between `apicl_1999` (the observed variable) and `apicl_2000` (the predicted variable) in the `api35` database ($N = 362$). Predictions comes from 11 models with various algorithms, relaxation and regularization parameters. They are very stable and reproductible here when the relaxation parameter differs from 0.

From these results, we can conclude that:

- whatever the algorithm used (`OUTCOME` or `JOINT`), adding a relaxation parameter improves here the association between the target variables (see Table 8).
- according to the results of Table 9, the `R_outc2` and `R_outc5` models seem not optimal because they are those for which the Hellinger criterion reflects the most clear violation of **assumption 1** (see Table 9). Therefore, it is here suggested to keep a relaxation parameter less than 0.6 in the final model.
- Table 8 confirms this trend because adding a too high relaxation parameter seems to potentially affect the quality of the association (the V Cramer and rank correlation decrease when the relaxation parameter increases from 0.4 to 0.6). Consequently, in this example, fixing 0.4 seems to be an acceptable compromise for the relaxation parameter whatever the `R-OUTCOME` algorithm used (0.3 could also have been tested here).
- among the remaining models (`R_outc1`, `R_outc4`, `R_outj1`, and `R_outj4`), `R_outj1` and `R_outj4` seem to be those with the most stable predictive potential (Table 10). Moreover, `R_outc4` appears here as the best model from the tested `R-OUTCOME` algorithms.
- adding a regularization parameter to the `R_outj1` model caused a decrease in the stability of the predictions (see the value of `R_outj4` compared to `R_outj1` in Table 10) and we thus conclude in favor of `R_outj1` as best model among those tested in the `JOINT` family of algorithms.
- Table 11 shows that the three remaining models (`R_outc4`, `R_outj1` and `R_outj4`) counted between 92 an 97% of common predictions and these last result also reassures the user about the quality of the provided predictions.

The confusion matrices related to `R_outc4` and `R_outj1` are described in Table 12 and seems to confirm that the encoding of `apicl_2000` in three groups, could simply correspond to the grouping of levels *G*2 and *G*3 of the `api_cl1999` variable.

Finally, note that the running time of each model took less than 15 seconds with `R` version 4.0.3 for Windows (10 Pro-64bits/ Process Intel 2.66 GHz).

Model	Type	Method	Relax	Hell(YA_YB)
outc1	OUTCOME	SEQUENTIAL	0.0	0.008
R_outc1	R-OUTCOME	SEQUENTIAL	0.4	0.085
R_outc2	R-OUTCOME	SEQUENTIAL	0.6	0.107
R_outc3	R-OUTCOME	OPTIMAL	0.0	0.002
R_outc4	R-OUTCOME	OPTIMAL	0.4	0.080
R_outc5	R-OUTCOME	OPTIMAL	0.6	0.102

Table 9: Hellinger distances related to the 6 models that used OUTCOME and R-OUTCOME algorithms. The values are relatively homogeneous from one model to another and do not indicate strong ditributional divergences (all values are very far from 1). Nevertheless, choosing relaxation parameters higher than 0.4 can increase the risk of distributional dissimilarities (the criterion moves away from 0.05 when the relaxation parameter increases).

Model	Type	Method	Relax	Regul	N	mean	sd
outc1	OUTCOME	SEQUENTIAL	0.0	-	284	0.968	0.122
R_outc1	R-OUTCOME	SEQUENTIAL	0.4	-	284	0.950	0.151
R_outc2	R-OUTCOME	SEQUENTIAL	0.6	-	284	0.954	0.145
R_outc3	R-OUTCOME	OPTIMAL	0.0	-	284	0.979	0.100
R_outc4	R-OUTCOME	OPTIMAL	0.4	-	284	0.987	0.080
R_outc5	R-OUTCOME	OPTIMAL	0.6	-	284	0.983	0.091
outj1	JOINT	-	0.0	0.0	284	0.911	0.116
R_outj1	R-JOINT	-	0.4	0.0	284	0.942	0.128
R_outj2	R-JOINT	-	0.6	0.0	284	0.953	0.171
R_outj3	R-JOINT	-	0.8	0.0	284	0.934	0.199
R_outj4	R-JOINT	-	0.4	0.1	284	0.926	0.097

Table 10: Stability of the predictions (`min.neigb = 5`). When the R_outj1 model will be confronted with a same profile of predictors and a same level of apicl_1999, more than 94 times of a hundred ($mean = 0.942$), it will be able to return the same individual prediction for apicl_2000. Among the OUTCOME family of algorithms, R_outc4 provides here the best stability of prediction while R_outj2 is the optimal one in the JOINT family of algorithms.

5 Conclusion and perspectives

To our knowledge, **OTrecod** is the first R package that takes advantage of the optimal transportation theory (Monge 1781) in the context of data fusion and the comparative study of methods described in (Garès and Omer 2022) underlines the promising performances of these approaches.

For more details and examples about the functions, users are invited to consult the ARTICLES section of the dedicated website.

6 Drawbacks

The functions of **OTrecod** only apply to a specific data fusion context where there is no overlapping part between the two raw data sources. This package does not deal with record linkage which is already the focus of extensive works (Sariyar and Borg 2010). This package is not adapted when the target variables (outcomes) of the two databases are continuous with an infinite number of values (like weights in grams with decimals for example). Moreover, if the data fusion requires only one shared variable to work, the quality of the fusion depends on the presence of a subset of good shared predictors in the raw databases. Finally, if the function OT_outcome allows all types of predictors, the current version of the function OT_joint imposes categorical matching variables only (scale variables are allowed): this restriction should be relaxed in a next version.

7 Perspectives

A number of more advanced research still need further investigation:

Model	outc1	R_outc1	R_outc4	outj1	R_outj1
R_outc1	0.84	-	-	-	-
R_outc4	0.83	0.95	-	-	-
outj1	0.72	0.81	0.83	-	-
R_outj1	0.83	0.91	0.94	0.84	-
R_outj4	0.84	0.96	0.97	0.84	0.92

Table 11: Ratio of common predictions between two models

(a)	**apicl_1999**				(b)	**apicl_1999**			
apicl_2000	*G*1	*G*2	*G*3	*G*4	**apicl_2000**	*G*1	*G*2	*G*3	*G*4
$[200-600]$	91	15	0	0	$[200-600]$	91	14	1	0
$(600-800]$	0	75	90	0	$(600-800]$	0	76	89	0
$(800-1000]$	0	0	0	91	$(800-1000]$	0	0	0	91

Table 12: Confusion matrices in the api35 dataset for the models (a) R_outc4 and (b) R_outj1

1. the possibility of extending the OT algorithm for recoding variables to multidimensional frameworks.
2. the stability of the algorithm when the matching variables are incomplete and the non-response processes are missing at random or not.
3. the contribution of calibration techniques in the quality process assessment (Deming and Stephan 1940)
4. the creation of a tuning function which defines a grid search to find the optimal combination of parameters related to the OT algorithms, could be added in the future versions of the package.

8 Acknowledgments

The authors would especially thank Pierre Navaro (CNRS UMR 6625) for their advices during the implementation of the **OTrecod** package. This research has received the help from *Region Occitanie* Grant RBIO-2015-14054319 and Mastodons-CNRS Grant.

9 Supplementary R code

```
### BASIC R CODE FOR SUMMARY TABLES  8, 9, 10, and 11 ---------

## Validation of each model: Repeat the following R command for each model by
## changing outc1:
verif_outc1   = verif_OT(outc1, group.class = TRUE, ordinal = FALSE,
                         stab.prob = TRUE, min.neigb = 5)

## Association between Y and Z: Summary Table 8
res.prx = rbind(
           outc1   = verif_outc1$res.prox    , R_outc1 = verif_R_outc1$res.prox,
           R_outc2 = verif_R_outc2$res.prox  , R_outc3 = verif_R_outc3$res.prox,
           R_outc4 = verif_R_outc4$res.prox  , R_outc5 = verif_R_outc5$res.prox,
           outj1   = verif_outj1$res.prox    , R_outj1 = verif_R_outj1$res.prox,
           R_outj2 = verif_R_outj2$res.prox  , R_outj3 = verif_R_outj3$res.prox,
           R_outj4 = verif_R_outj4$res.prox )

res.prx = data.frame(Model = c("outc1","R_outc2","R_outc3","R_outc4",
                               "R_outc5","R_outc6","outj1", "R_outj1",
                               "R_outj2","R_outj3","R_outj4"),
                     Type  = c("OUTCOME",rep("R-OUTCOME",5),"JOINT",
                               rep("R-JOINT",4)),
                     Relax = c(0,0.4,0.6,0,0.4,0.6,0,0.4,0.6,0.8,0.4),
                     Regul = c(rep(0,10),0.1), res.prx)

row.names(res.prx) = NULL; head(res.prx,3)

#    Name      Type Relax Regul   N V_cram rank_cor
#   outc1   OUTCOME   0.0   0.0 362   0.86    0.892
# R_outc1 R-OUTCOME   0.0   0.0 362   0.87    0.911
# R_outc2 R_OUTCOME   0.4   0.0 362   0.93    0.933
#-----

## Hellinger distance: Summary Table 9
res.helld = rbind(
             outc1   = verif_outc1$hell  , R_outc1 = verif_R_outc1$hell,
             R_outc2 = verif_R_outc2$hell, R_outc3 = verif_R_outc3$hell,
             R_outc4 = verif_R_outc4$hell, R_outc5 = verif_R_outc5$hell,
             outj1   = verif_outj1$hell  , R_outj1 = verif_R_outj1$hell,
             R_outj2 = verif_R_outj2$hell, R_outj3 = verif_R_outj3$hell,
             R_outj4 = verif_R_outj4$hell )

res.helld = data.frame(res.prx[,1:4], res.helld)
row.names(res.helld) = NULL; res.helld
#-----

## Stability of the prediction: Summary Table 10
# same R code as for Summary Table 8, changing res.prox by res.stab
#----

## Ratio of common predictions: Table 11

stoc = list(outc1$DATA2_OT$OTpred  , R_outc1$DATA2_OT$OTpred,
            R_outc4$DATA2_OT$OTpred, outj1$DATA2_OT$OTpred  ,
            R_outj1$DATA2_OT$OTpred, R_outj4$DATA2_OT$OTpred)

corpred = matrix(ncol = 6, nrow = 6)
for (i in 1:6){
      for (j in 1:6){
        corpred[i,j] = round(sum(diag(table(stoc[[i]],stoc[[j]])))/362,2)
      }
}; corpred
#-----
```

References

Adamek, James C. 1994. "Fusion: Combining Data from Separate Sources." *Marketing Research* 6 (3): 48.

Bergsma, Wicher. 2013. "A Bias-Correction for Cramér's v and Tschuprow's t." *Journal of the Korean Statistical Society* 42 (3): 323–28. https://doi.org/https://doi.org/10.1016/j.jkss.2012.10.002.

Breiman, Leo. 2001. "Random Forests." *Machine Learning* 45 (1): 5–32. https://doi.org/10.1023/A:1010933404324.

Breiman, L., J. Friedman, C. J. Stone, and R. A. Olshen. 1984. *Classification and Regression Trees*. Taylor & Francis. https://books.google.fr/books?id=JwQx-WOmSyQC.

Castanedo, Federico. 2013. "A Review of Data Fusion Techniques." *TheScientificWorldJournal* 2013 (January): 704504. https://doi.org/10.1155/2013/704504.

Cibella, N. 2010. "How to Choose the Matching Variables, Report WP2, ESS-Net." *Statistical Methodology Project on Integration of Surveys and Administrative Data, EUROSTAT*.

Cohen, ML. 1991. "Statistical Matching and Microsimulation Models, Improving Information for Social Policy Decisions, the Use of Microsimulation Modeling, Technical Papers, II." *Washington, DC: National Academy*.

Cuturi, Marco. 2013. "Sinkhorn Distances: Lightspeed Computation of Optimal Transport." In *Advances in Neural Information Processing Systems*, edited by C. J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger. Vol. 26. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2013/file/af21d0c97db2e27e13572cbf59eb343d-Paper.pdf.

D'Orazio, Marcello. 2022. *StatMatch: Statistical Matching or Data Fusion*. https://CRAN.R-project.org/package=StatMatch.

D'Orazio, Marcello, Marco Di Zio, and Mauro Scanu. 2006. *Statistical Matching: Theory and Practice*. John Wiley & Sons.

Das, Subhadeep, and Dr. Sucheta Tripathy. 2022. *OMICsPCA: An r Package for Quantitative Integration and Analysis of Multiple Omics Assays from Heterogeneous Samples*.

Deming, W Edwards, and Frederick F Stephan. 1940. "On a Least Squares Adjustment of a Sampled Frequency Table When the Expected Marginal Totals Are Known." *The Annals of Mathematical Statistics* 11 (4): 427–44.

F, Rohart, Gautier B, Singh A, and Le Cao K-A. 2017. "mixOmics: An r Package for 'Omics Feature Selection and Multiple Data Integration." *PLoS Computational Biology* 13 (11): e1005752. http://www.mixOmics.org.

Farrar, Donald E., and Robert R. Glauber. 1967. "Multicollinearity in Regression Analysis: The Problem Revisited." *The Review of Economics and Statistics* 49 (1): 92–107. http://www.jstor.org/stable/1937887.

Flamary, Rémi, Nicolas Courty, Alexandre Gramfort, Mokhtar Zahdi Alaya, Aurélie Boisbunon, Stanislas Chambon, Laetitia Chapel, et al. 2021. "POT : Python Optimal Transport." *Journal of Machine Learning Research*, April. https://hal.archives-ouvertes.fr/hal-03264013.

Forrest, John, David de la Nuez, and Robin Lougee-Heimer. 2004. "CLP User Guide." *IBM Research*.

Garès, Valérie, Chloé Dimeglio, Grégory Guernec, Romain Fantin, Benoit Lepage, Michael R Kosorok, and Nicolas Savy. 2020. "On the use of optimal transportation theory to recode variables and application to database merging." *International Journal of Biostatistics* 16 (1): article number : 20180106. https://doi.org/10.1515/ijb-2018-0106.

Garès, Valérie, and Jérémy Omer. 2022. "Regularized Optimal Transport of Covariates and Outcomes in Data Recoding." *Journal of the American Statistical Association* 117 (537): 320–33. https://doi.org/10.1080/01621459.2020.1775615.

Grajski, Kamil A., Leo Breiman, Gonzalo Viana Di Prisco, and Walter J. Freeman. 1986. "Classification of EEG Spatial Patterns with a Tree-Structured Methodology: CART." *IEEE Transactions on Biomedical Engineering* BME-33 (12): 1076–86. https://doi.org/10.1109/TBME.1986.325684.

Hall, David, and James Llinas. 1997. "An Introduction to Multisensor Data Fusion." *Proceedings of the IEEE* 85 (February): 6–23. https://doi.org/10.1109/5.554205.

Hastie, Trevor, and Rahul Mazumder. 2021. *softImpute: Matrix Completion via Iterative Soft-Thresholded SVD*. https://CRAN.R-project.org/package=softImpute.

Hernandez-Ferrer, Carles, Carlos Ruiz-Arenas, Alba Beltran-Gomila, and Juan R. González. 2017. "MultiDataSet: An r Package for Encapsulating Multiple Data Sets with Application to Omic Data Integration." *BMC Bioinformatics* 18 (1): 36. https://doi.org/10.1186/s12859-016-1455-1.

Hitchcock, F. L. 1941. "The Distribution of a Product from Several Sources to Numerous Localities." *Journal of Mathematics and Physics / Massachusetts Institute of Technology.* 20: 224–30. https://doi.org/10.1002/sapm1941201224.

Hothorn, Torsten, Peter Bühlmann, Sandrine Dudoit, Annette Molinaro, and Mark J. Van Der Laan. 2005. "Survival ensembles." *Biostatistics* 7 (3): 355–73. https://doi.org/10.1093/biostatistics/kxj011.

Hothorn, Torsten, Kurt Hornik, and Achim Zeileis. 2006. "Unbiased Recursive Partitioning: A

Conditional Inference Framework." *Journal of Computational and Graphical Statistics* 15 (3): 651–74. https://doi.org/10.1198/106186006X133933.

Josse, Julie, and François Husson. 2016. "missMDA: A Package for Handling Missing Values in Multivariate Data Analysis." *Journal of Statistical Software* 70 (1): 1–31. https://doi.org/10.18637/jss.v070.i01.

Kadane, Joseph B. 2001. "Some Statistical Problems in Merging Data Files." *Journal of Official Statistics* 17: 423–33.

Kantorovich, L. 1942. "On the transfer of masses." *Doklady Akademii Nauk SSSR* 37: 7–8.

Klein, Lawrence A. 2004. *Sensor and Data Fusion: A Tool for Information Assessment and Decision Making*. Vol. 138. SPIE press.

Kotz, Samuel, Narayanaswamy Balakrishnan, and Norman Johnson. 2000. *Continuous Multivariate Distributions: Models and Applications, Volume 1, Second Edition*. Vol. 1. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons. https://doi.org/10.1002/0471722065.

Liaw, Andy, and Matthew Wiener. 2002. "Classification and Regression by randomForest." *R News* 2 (3): 18–22. https://CRAN.R-project.org/doc/Rnews/.

Liese, Friedrich, and Klaus -J. Miescke. 2008. "Statistical Decision Theory." In *Statistical Decision Theory: Estimation, Testing, and Selection*, 1–52. New York, NY: Springer New York. https://doi.org/10.1007/978-0-387-73194-0_3.

Little, R. J. A., and D. B. Rubin. 2019. *Statistical Analysis with Missing Data, Third Edition*. Wiley Series in Probability and Mathematical Statistics. Probability and Mathematical Statistics. Wiley.

Makhorin, Andrew. 2011. "GNU Linear Programming Kit, Reference Manual." *Free Software Foundation* 4.

Mayer, Imke, Aude Sportisse, Julie Josse, Nicholas Tierney, and Nathalie Vialaneix. 2019. "R-Miss-Tastic: A Unified Platform for Missing Values Methods and Workflows." arXiv. https://doi.org/10.48550/ARXIV.1908.04822.

Monge, G. 1781. "Mémoire sur la Théorie des Déblais et des Remblais." *Histoire de l'Académie Royale Des Sciences de Paris*, 666–704.

Muzellec, Boris, Julie Josse, Claire Boyer, and Marco Cuturi. 2020. "Missing Data Imputation Using Optimal Transport." In *ICML*.

Paass, Gerhard. 1986. "Statistical Match: Evaluation of Existing Procedures and Improvements by Using Additional Information." *Microanalytic Simulation Models to Support Social and Financial Policy*, 401–20.

Pagès, J. 2002. "Analyse Factorielle Multiple Appliquée Aux Variables Qualitatives Et Aux Données Mixtes." *Revue de Statistique Appliquée* 50 (4): 5–37. http://eudml.org/doc/106525.

Rässler, Susanne. 2002. "Frequentist Theory of Statistical Matching." In *Statistical Matching: A Frequentist Theory, Practical Applications, and Alternative Bayesian Approaches*, 15–43. New York, NY: Springer New York. https://doi.org/10.1007/978-1-4613-0053-3_2.

Rodgers, Willard L. 1984. "An Evaluation of Statistical Matching." *Journal of Business & Economic Statistics* 2 (1): 91–102. http://www.jstor.org/stable/1391358.

Rubin, Donald B. 1986. "Statistical Matching Using File Concatenation with Adjusted Weights and Multiple Imputations." *Journal of Business & Economic Statistics* 4 (1): 87–94. http://www.jstor.org/stable/1391390.

Sariyar, Murat, and Andreas Borg. 2010. "The RecordLinkage Package: Detecting Errors in Data." *The R Journal* 2 (2): 61–67.

Scanu, M. 2010. "Recommendations on Statistical Matching, Report WP2, ESS-Net." *Statistical Methodology Project on Integration of Surveys and Administrative Data*.

Schuhmacher, Dominic, Björn Bähre, Carsten Gottschlich, Valentin Hartmann, Florian Heinemann, and Bernhard Schmitzer. 2022. *transport: Computation of Optimal Transport Plans and Wasserstein Distances*. https://cran.r-project.org/package=transport.

Stekhoven, Daniel J. 2022. *missForest: Nonparametric Missing Value Imputation Using Random Forest*.

Stekhoven, Daniel J., and Peter Bühlmann. 2012. "MissForest–Non-Parametric Missing Value Imputation for Mixed-Type Data." *Bioinformatics (Oxford, England)* 28 (January): 112–18.

Strobl, Carolin, Anne-Laure Boulesteix, Thomas Kneib, Thomas Augustin, and Achim Zeileis. 2008. "Conditional Variable Importance for Random Forests." *BMC Bioinformatics* 9 (1): 307. https://doi.org/10.1186/1471-2105-9-307.

Strobl, Carolin, Anne-Laure Boulesteix, Achim Zeileis, and Torsten Hothorn. 2007. "Bias in Random Forest Variable Importance Measures: Illustrations, Sources and a Solution." *BMC Bioinformatics* 8 (1): 25. https://doi.org/10.1186/1471-2105-8-25.

Strobl, Carolin, Torsten Hothorn, and Achim Zeileis. 2009. "Party on!" *The R Journal* 1 (2): 14–17. https://doi.org/10.32614/RJ-2009-013.

Theußl, Stefan, Florian Schwendinger, and Kurt Hornik. 2017. "ROI: The R Optimization Infrastructure Package." Research Report Series / Department of Statistics and Mathematics 133. Vienna: WU Vienna University of Economics; Business. http://epub.wu.ac.at/5858/.

———. 2020. "ROI: An Extensible R Optimization Infrastructure." *Journal of Statistical Software* 94 (15):

1–64. https://doi.org/10.18637/jss.v094.i15.

van Buuren, Stef, and Karin Groothuis-Oudshoorn. 2011. "mice: Multivariate Imputation by Chained Equations in R." *Journal of Statistical Software* 45 (3): 1–67. https://doi.org/10.18637/jss.v045.i03.

Van Rossum, Guido, and Fred L Drake Jr. 1995. *Python Reference Manual*. Centrum voor Wiskunde en Informatica Amsterdam.

Vantaggi, Barbara. 2008. "Statistical Matching of Multiple Sources: A Look Through Coherence." *Int. J. Approx. Reasoning* 49 (November): 701–11. https://doi.org/10.1016/j.ijar.2008.07.005.

Wickham, Hadley, Jim Hester, Winston Chang, and Jennifer Bryan. 2022. *Devtools: Tools to Make Developing R Packages Easier*. https://CRAN.R-project.org/package=devtools.

Zeileis, Achim, and Torsten Hothorn. 2008. "Model-Based Recursive Partitioning." *Journal of Computational and Graphical Statistics - J COMPUT GRAPH STAT* 17 (June): 492–514. https://doi.org/10.1198/106186008X319331.

Zhu, Ziwei, Tengyao Wang, and Richard J. Samworth. 2019. "High-Dimensional Principal Component Analysis with Heterogeneous Missingness." arXiv. https://doi.org/10.48550/ARXIV.1906.12125.

Gregory Guernec
Université de Toulouse, INSERM, UPS
INSERM, CERPOP, UMR 1295
Toulouse, France
ORCiD: 0000-0002-0668-8606
gregory.guernec@inserm.fr

Valerie Gares
INSA, Université de Rennes
CNRS, IRMAR, UMR 6625
Rennes, France
http://vgares.perso.math.cnrs.fr/contact.html
valerie.gares@insa-rennes.fr

Jeremy Omer
INSA, Université de Rennes
CNRS, IRMAR, UMR 6625
Rennes, France
https://jeremyomer.wixsite.com/recherche
jeremy.omer@insa-rennes.fr

Philippe Saint-Pierre
IMT, Université Paul Sabatier, Toulouse
CNRS UMR 5219
Toulouse, France
https://perso.math.univ-toulouse.fr/psaintpi/
philippe.saint-pierre@univ-tlse3.fr

Nicolas Savy
IMT, Université Paul Sabatier, Toulouse
CNRS UMR 5219
Toulouse, France
https://perso.math.univ-toulouse.fr/savy/
nicolas.savy@univ-tlse3.fr

populR: a Package for Population Downscaling in R

by Marios Batsaris and Dimitris Kavroudakis

Abstract Population data provision is usually framed by regulations and restrictions and hence spatially aggregated in predefined enumeration units such as city blocks and census tracts. Many applications require population data at finer scale, and therefore, one may use downscaling methods to transform population counts from coarse spatial units into smaller ones. Although numerous methods for downscaling of population data have been reported in the scientific literature, only a limited number of implementation tools exist. In this study, we introduce populR, an R package that responds to this need. populR provides two downscaling methods, namely Areal Weighted Interpolation and Volume Weighted Interpolation, which are illustrated and compared to alternative implementations in the sf and areal packages using a case study from Mytilini, Greece. The results provide evidence that the vwi approach outperforms the others, and thus, we believe R users may gain significant advantage by using populR for population downscaling.

1 Introduction

Information about the spatial distribution of the population is crucial in addressing a wide range of geographical information science problems (Dobson et al. 2000; Ahola et al. 2007; Freire and Aubrecht 2012; Bian and Wilmot 2015; Calka, Nowak Da Costa, and Bielecka 2017; Batsaris et al. 2019; Bahadori, Gonçalves, and Moura 2021; Wang et al. 2021; Han, Hu, and Wang 2020). One of the most common sources of population data is the Census of Population (Tenerelli, Gallego, and Ehrlich 2015; Liu, Kyriakidis, and Goodchild 2008). In the EU, censuses are carried out every ten years and are regulated by the EU 763/2008 law (European Union 2009). During the census, detailed information about the population is collected with a high level of granularity. This information is confidential, and due to personal data laws and privacy concerns, census data are de-identified via spatial aggregation into coarse administrative units.

Aggregated representations of census data may not reflect the spatial heterogeneity of the population within the reported administrative units, and therefore, one may need to use downscaling methods to transform population counts from the predefined coarse boundaries (source) into finer-scale sub-units (target). Areal interpolation is broadly used in population downscaling (Tenerelli, Gallego, and Ehrlich 2015).

Areal interpolation methods can be categorized according to usage of additional information such as land use data and night lights to guide the interpolation process. Ancillary information independent methods are further distinguished in: a) point-based and b) area-based (Wu, Qiu, and Wang 2005; Comber and Zeng 2019). Areal Weighted Interpolation (*AWI*) is one of the most common methods of areal interpolation of population without exploiting ancillary information [Comber and Zeng (2019); Kim and Yao (2010);] in a Geographic Information System (GIS). *AWI* guides the interpolation process on the basis of areal weights calculated by the area of intersection between the source and target features (Goodchild and Lam 1980). This method has four main advantages: a) It is easy and straightforward to implement; b) it does not require ancillary information; c) it preserves the initial volume (i.e., the total population within the source zone); and d) it may be implemented in both vector and raster environments (Comber and Zeng 2019; Fisher and Langford 1995; Kim and Yao 2010; Lam 1983; Qiu, Zhang, and Zhou 2012). On the other hand, one of the main disadvantages of *AWI* is its assumption of homogeneity within source zone features (Comber and Zeng 2019; Qiu, Zhang, and Zhou 2012).

sf (Pebesma 2018) and **areal** (Prener and Revord 2019) provide *AWI* functionality in R. **sf** comes up with a formula either for extensive (population) or intensive (population density) interpolation that calculates the areal weights based on the total area of the source features (*total weights*), which makes it suitable for completely overlapping data. **areal** extends sf functionality by providing an additional formula for weight calculation for data without complete overlap. In this case, areal weights are calculated using the sum of the remaining source areas after the intersection (*sum weights*) (Prener and Revord 2019).

When the case involves downscaling of urban population data (small scale applications) where the source features (such as city blocks or census tracts) are somehow larger than target features (such as buildings) in terms of footprint area, the **sf** functionality is unable to calculate the areal weights correctly. Additionally, **areal** may be confusing for novice R (or GIS) users as it is not obvious that the weight option should be set to *sum* to calculate areal weights properly.

To overcome such limitations, we introduce **populR** (Batsaris 2022), a package for downscaling of population data using areal interpolation methods. **populR** provides an *AWI* approach that matches **areal** functionality using *sum weights* and a *VWI* (Volume Weighted Interpolation) approach that uses the area of intersection between source and target features multiplied by the building height or number of floors (volume) to guide the interpolation process.

The remainder of the paper is structured as follows. The next section lists the methods included in the **populR** package, further explains and demonstrates using examples. The numerical experiments section illustrates and discusses the results obtained by **populR** and compares the results to other alternatives in R. Finally, the paper concludes with a summary, along with future improvements, in the conclusions section.

2 The populR package

populR is available to R users through the CRAN and Github repositories and may be installed as shown in the code block below.

```
# CRAN installation
install.packages("populR")

# Github installation
devtools::install_github("mbatsaris/populR")
```

The **populR** functionality focuses on three main pillars, namely downscaling, rounding, and comparing. Additionally, sample data (objects of class *sf* (Pebesma 2018)) representing city blocks, including population counts (*src*) and individual buildings attributed by the number of floors (*trg*) from a small part of the city of Mytilini, Greece, are also available for further experimentation with the package's functionality.

Downscaling

Population downscaling is carried out by the primary `pp_estimate` function.

Suppose a set of $j = 1, ..., J$ city block polygons (source) accompanied by population counts and an incongruent yet superimposed set of $i = 1, ..., B_j$ individual building polygons (target), along with their characteristics such as footprint area and/or number of floors (or height), the `pp_estimate`, aims to transform population counts from the source to the target using *AWI* and *VWI* approaches. The mathematics behind both approaches is presented in the next two subsections and graphically illustrated in figure 1.

AWI

AWI proportionately interpolates the population value of the source features based on areal weights calculated by the area of intersection between the source and target zones and algebraically explained by equations (1) and (2) (Goodchild and Lam 1980). In equation (1), areal weights w_{ij}^a are calculated by standardizing the measured footprint area of individual building i in block j (a_{ij}) over the sum of building footprint areas in the $j-th$ city block. Finally, building population values for building i in block j (p_{ij}) may be calculated by multiplying the areal weights of building i in block j with the population value of $j-th$ block (P_j).

$$w_{ij}^a = \frac{a_{ij}}{\sum_{i=1}^{B_j} a_{ij}}\,, \tag{1}$$

$$p_{ij}^a = w_{ij}^a \times P_j\,. \tag{2}$$

A demonstration of the *AWI* approach is presented in the code block below. In the first line of code, **populR** is attached to the script. The next two lines load the package's built-in data, and the last line demonstrates the usage of the `pp_estimate` function. As a result, `pp_estimate` returns an object of class sf including individual buildings (*target*) with population estimates.

```
# attach package
library(populR)
```

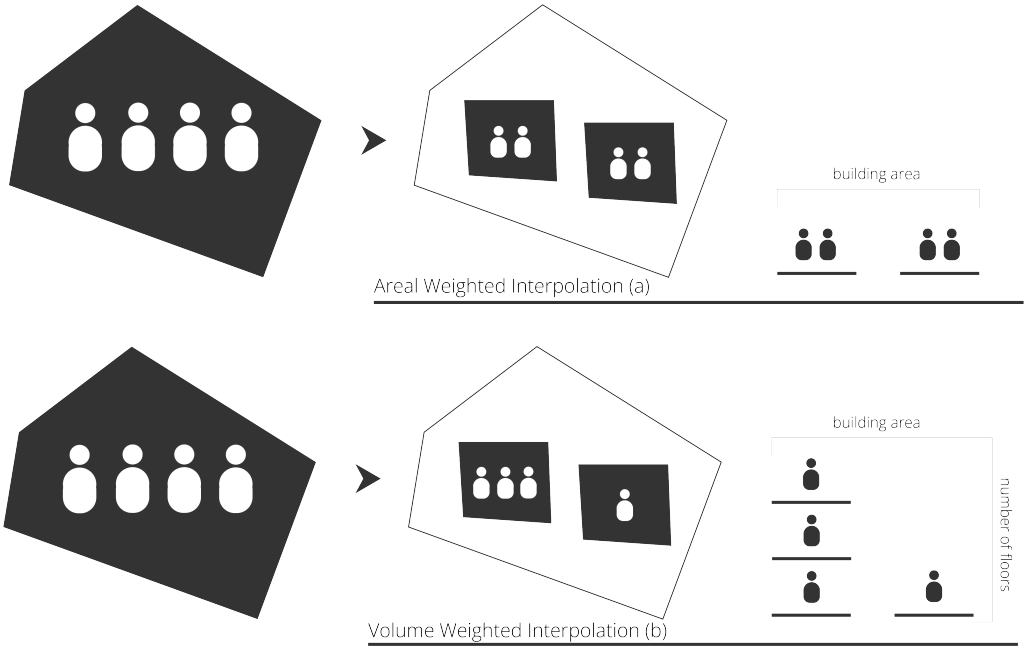


Figure 1: Illustration of the populR package's functionality. populR uses two areal interpolation methods to convert population values from city block polygons (left) to a set of individual building polygons (center). AWI (top) solely relies on the area of intersection between blocks and buildings (top-right). VWI (bottom) uses the area of intersection between blocks and buildings multiplied by the height or number of floors to aid the transformation process (bottom-right).

```
# load data
data('src')
data('trg')

# downscaling using awi
awi <- pp_estimate(target = trg, source = src, spop = pop, sid = sid,
method = awi)
```

Where:

1. *target*: An object of class *sf* that is used to interpolate data to. Usually, target may include polygon features representing building units
2. *source*: An object of class *sf* including data to be interpolated. Source may be a set of coarse polygon features, such as city blocks or census tracts
3. *sid*: Source identification number
4. *spop*: Source population values to be interpolated
5. *method*: Two methods provided: *AWI* (Areal Weighted Interpolation) and *VWI* (Volume Weighted Interpolation). *AWI* proportionately interpolates the population values based on areal weights calculated by the area of intersection between the source and target zones. *VWI* proportionately interpolates the population values based on volume weights calculated by the area of intersection between the source and target zones multiplied by the volume information (height or number of floors)

VWI

Given the number of floors (or height) of individual buildings, *VWI* applies the same logic as *AWI*. Instead of areal weights, *VWI* proportionately dis-aggregates source population counts using volume weights measured by the area of intersection multiplied either by individual building height (if available) or number of floors. *VWI* may be mathematically expressed by equations (3) to (5). First, the volume of building i in block j (v_{ij}) is measured by multiplying the footprint area (a_{ij}) of building i in block j with the number of floors or height (n_{ij}) as shown in equation (3). Next, volume weights (w_{ij}^{v})

are calculated by standardizing the volume of building i in block j (v_{ij}) by the sum of the total volume of buildings in the $j-th$ block (4). Then, building population values for building i in block j (p_{ij}^{v}) may be calculated by multiplying the volume weights (w_{ij}^{v}) of building i in block j with the population value of block j (P_J) as shown in (5).

$$v_{ij} = a_{ij} \times n_{ij} \,, \tag{3}$$

$$w_{ij}^{v} = \frac{v_{ij}}{\sum_{i=1}^{B_j} v_{ij}} \,, \tag{4}$$

$$p_{ij}^{v} = w_{ij}^{v} \times P_j \,. \tag{5}$$

The code block below provides an example of the *VWI* approach.

```
# downscaling using vwi
vwi <- pp_estimate(target = trg, source = src, spop = pop, sid = sid,
volume = floors, method = vwi)
```

Where:

1. *volume*: Target feature volume information (height or number of floors). Required when method is set to *vwi*

It is important to mention that when volume information (number of floors or building height) is not available for the entire target dataset, users may replace missing values with 1 if they want to include them as buildings with 1 floor; otherwise, users may replace missing values with 0 if they want to exclude them from the downscaling process.

Rounding

AWI is volume-preserving (Comber and Zeng 2019) (as *VWI*) and returns an object of class *sf* with decimal population counts for each target feature. To increase the readability of the results as well as to maintain the initial source population values, it is essential for many applications to provide integer values by rounding off to the closest integer. This transformation may result in a shortage or surplus in comparison to the initial values (source values), and therefore, to cope with this problem, the `pp_round` function is proposed.

First, estimated population values are converted into integer numbers. Next, the `pp_round` function calculates the differences between the initial population counts and the sum of the integer values for each city block; it is activated only if the quantified difference is either positive or negative. In either case, during this process, target-based differences are also calculated and used to refine the integer counts by adding or removing one population unit in order to preserve the initial source counts when summed up.

An example of the `pp_round` function is provided below.

```
# downscaling using awi
awi <- pp_round(x = awi, tpop = pp_est, spop = pop, sid = sid)

# downscaling using awi
vwi <- pp_round(x = vwi, tpop = pp_est, spop = pop, sid = sid)
```

Where:

1. *x*: An object of class sf obtained by the `pp_estimate` function
2. *tpop*: Target population estimates obtained by the `pp_estimate` function
3. *spop*: Initial source population values (included after the implementation of the `pp_estimate` function)
4. *sid*: Source identification number

As a result, `pp_round` returns an object of class *sf* with integer estimates of population values with respect to the initial source population values.

Table 1: Data used for numerical experiments

Title	Format	Source	Year
City block population counts	Tabular	Hellenic Statistical Authority	2011
City blocks	Spatial	Hellenic Statistical Authority	2011
Buildings	Spatial	Hellenic Statistical Authority	2011

Comparing

Volume-preserving means that estimated values should sum up to the initial population counts for each source unit, and therefore, one may need to compare target estimates to the initial source values. For this purpose, a function is introduced under the alias `pp_compare`. The `pp_compare` function measures the root mean squared error (RMSE), the mean absolute error (MAE), and finally, the statistical relationship between the initial source values and the estimated ones, which is calculated and depicted on a 2D scatter diagram along with the value of the correlation coefficient R^2.

A short example of the `pp_compare` function is presented in the code block below, where *src_awi* (and *src_vwi*) is a *data.frame* created by grouping the sum of the estimated values along with the initial population values for each source feature.

```
# attach library
library(dplyr)

# group estimated and actual values by source identification number - awi
src_awi <- awi %>%
  group_by(sid) %>%
  summarise(est = sum(pp_est), act = unique(pop))

# awi approach compare to source values
pp_compare(x = src_awi, estimated = est, actual = act, title = "awi vs source")

# group estimated and actual values by source identification number - vwi
src_vwi <- vwi %>%
  group_by(sid) %>%
  summarise(est = sum(pp_est), act = unique(pop))

# vwi approach compare to source values
pp_compare(x = src_vwi, estimated = est, actual = act, title = "vwi vs source")
```

Where:

1. *x*: An object of class *sf* or a *data.frame* including estimated and actual values
2. *estimated*: Target population estimates obtained by the `pp_estimate` function
3. *actual*: Actual population values
4. *title*: Scatterplot title (quoted)

`pp_compare` returns a list of elements with RMSE, MAE, linear regression, and R^2 coefficient information.

3 Numerical Experimentation

In this section, the results of the **populR** package are illustrated and further compared to **sf** and **areal**. The analysis focus on implementation, comparison to a reference data set, and performance.

Case study

To examine the efficacy and efficiency of the **populR** extension based on actual data, a small part of the city of Mytilini was chosen as the subject for a case study (Figure 2). The study area consists of 9 city blocks (*src*), 179 buildings (*trg*), and 911 residents.

Table 1 presents the data used in this case study. City block population counts were retrieved in tabular format while city blocks (source - *src*) and buildings (target - *trg*) were provided in spatial format. City block population counts and city blocks and buildings correspond to the 2011 Census of Housing and Population (Hellenic Statistical Authority 2014).

Figure 2: Part of the city of Mytilini used as the case study for numerical experiments. The study area is represented by 9 city blocks and 179 buildings (in orange). The study area counts 911 residents.

Implementation

In this section, a demonstration of the **sf**, **areal** and **populR** packages takes place. First, the packages are attached to the script and next, **populR** built-in data are loaded. Then, areal interpolation implementation follows for each one of the aforementioned packages.

For the reader's convenience, names were shortened as follows: a) *awi*: **populR** *AWI* approach, b) *VWI*: **populR** *vwi* approach, c) *aws*: **areal** using extensive interpolation and *sum weights*, d) *awt*: **areal** using extensive interpolation and *total weights*, and e) *sf*: **sf** using extensive interpolation and *total weights*.

```
# attach libraries
library(populR)
library(areal)
library(sf)

# load data
data("trg", package = "populR")
data("src", package = "populR")

# populR - awi
awi <- pp_estimate(target = trg, source = src, spop = pop, sid = sid,
method = awi)

# populR - vwi
vwi <- pp_estimate(target = trg, source = src, spop = pop, sid = sid,
volume = floors, method = vwi)

# areal - sum weights
aws <- aw_interpolate(trg, tid = tid, source = src, sid = 'sid',
weight = 'sum', output = 'sf', extensive = 'pop')

# areal - total weights
awt <- aw_interpolate(trg, tid = tid, source = src, sid = 'sid',
weight = 'total', output = 'sf', extensive = 'pop')
```

Table 2: Implementation results obtained by awi, vwi, awt, aws and sf for 10 buildings of the study area.

nr	awi	vwi	aws	awt	sf
1	1.36	0.37	1.36	0.58	0.58
2	1.42	0.39	1.42	0.60	0.60
3	16.12	15.47	16.12	5.99	5.99
4	7.47	4.30	7.47	2.77	2.77
5	2.22	0.61	2.22	0.94	0.94
6	21.08	28.32	21.08	7.83	7.83
7	1.34	0.44	1.34	0.62	0.62
8	2.57	1.45	2.57	1.36	1.36
9	6.54	5.03	6.54	2.75	2.75
10	6.68	3.84	6.68	2.48	2.48

```
# sf - total weights
sf <- st_interpolate_aw(src['pop'], trg, extensive = TRUE)
```

Evidently, **sf** requires less arguments than **populR** and **areal**, which makes it very easy to implement. **populR** requires at least 5 arguments, and **areal** at least 7, which may increase the implementation complexity.

The study area counts 911 residents, as already mentioned in previous section. *awi*, *vwi* and *aws* correctly estimated population values as they sum to 911, while *awt* and *sf* underestimated values. This is expected as both methods use the total area of the source features during the interpolation process and are useful when source and target features completely overlap.

Moreover, identical results were obtained by the *awi* and *aws* approaches, but somehow different results were obtained by the *vwi*, as shown in Table 2.

Finally, visual representations of the results are shown in Figure 3.

Comparison to reference data

Due to confidentiality concerns, population data at the granularity of building are not publicly available in Greece. Therefore, an alternate population distribution previously published in Batsaris et al. (2019) was used as reference data (*rf*). *rf* population values are also included in the build-in *trg* data set.

Using the `pp_compare` function as shown in the example below, we investigated the statistical relationship between *rf* and the results obtained by **populR**, **areal**, and **sf** packages.

```
# compare awi to rf
pp_compare(data, estimated = awi, actual = rf, title = "awi vs rf")

# compare vwi to rf
pp_compare(data, estimated = vwi, actual = rf, title = "vwi vs rf")

# compare sf to rf
pp_compare(data, estimated = sf, actual = rf, title = "sf vs rf")

# compare awt to rf
pp_compare(data, estimated = awt, actual = rf, title = "awt vs rf")

# compare aws to rf
pp_compare(data, estimated = aws, actual = rf, title = "aws vs rf")
```

Table 3 presents the results obtained for the first 10 individual buildings for each implementation in comparison to *rf* values. Additionally, *RMSE*, *MAE*, and R^2 values are measured and depicted in Table 4 and finally, scatter diagrams provided in Figure **??**.

Generally the scatter diagrams (Figure 4)) suggest strong and positive relationships in all cases. However, the *vwi* approach obtained the best results and achieved the smallest *RMSE* (1.44824) and *MAE* (0.9358159) values and the largest R^2 (0.9878) value as shown in Table and Figure. Moreover, *aws* and *awi* provided the same error and R^2 measurements (*RMSE*: 5.325914, *MAE*: 2.748126 and R^2: 0.8215). *sf* and *awt* provided the same results and performed poorly in comparison to *vwi* by obtaining the largest error measurements and the smallest R^2 (*RMSE*: 7.416329, *MAE*: 3.664695, and R^2: 0.80367).

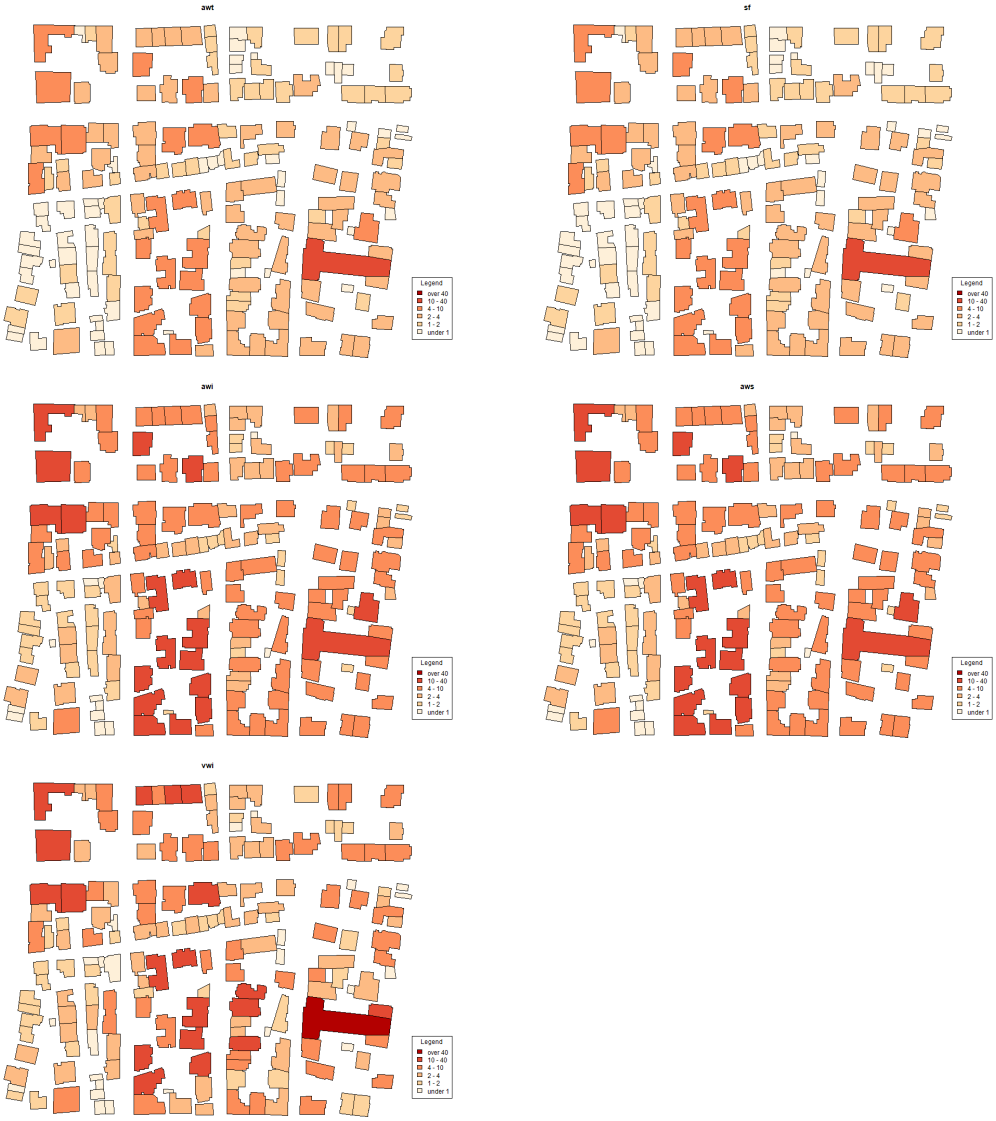


Figure 3: Cartographic representation of the downscaling results acquired by a) awt (top-left), b) sf (top-right), c) awi (center-left), d) aws (center-right), and e) vwi (bottom). awt and sf use the same formula therefore, they provide similar results. Additionally, the same results obtained by aws and awi as they are based on the same formula. vwi provide somehow different distribution because of the influence of the volume weights during the interpolation process.

Table 3: Implementation results obtained by awi, vwi, awt, aws and sf for 10 buildings of the study area in comparison to reference data (rf).

nr	rf	awi	vwi	aws	awt	sf
1	0.46	1.36	0.37	1.36	0.58	0.58
2	0.49	1.42	0.39	1.42	0.60	0.60
3	16.20	16.12	15.47	16.12	5.99	5.99
4	5.63	7.47	4.30	7.47	2.77	2.77
5	0.76	2.22	0.61	2.22	0.94	0.94
6	31.77	21.08	28.32	21.08	7.83	7.83
7	0.49	1.34	0.44	1.34	0.62	0.62
8	1.57	2.57	1.45	2.57	1.36	1.36
9	3.75	6.54	5.03	6.54	2.75	2.75
10	5.03	6.68	3.84	6.68	2.48	2.48

Table 4: RMSE, MAE and R^2 values were calculated to assess the estimation accuracy using rf as the control variable. vwi provide the best measurements.

Title	RMSE	MAE	R^2
rf vs vwi	1.45	0.94	0.988
rf vs awi	5.33	2.75	0.822
rf vs aws	5.33	2.75	0.822
rf vs sf	7.42	3.66	0.804
rf vs awt	7.42	3.66	0.804

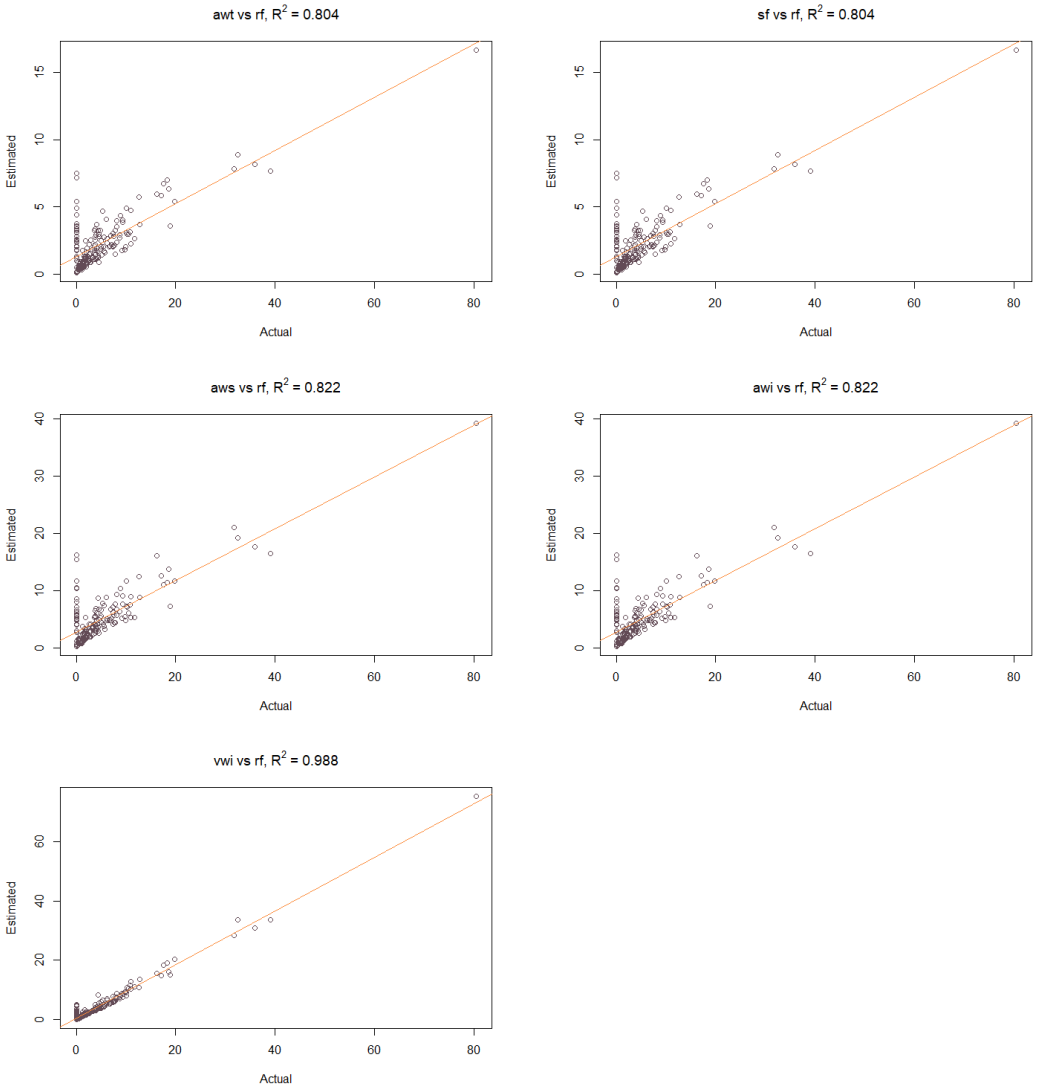


Figure 4: Investigation of the relationship of the outcomes and reference data (rf) as the control variable. a) rf vs awt, R^2 = 0.804 (top-left), b) rf vs sf, R^2 = 0.804 (top-right), c) rf vs aws, R^2 = 0.822 (center-left), d) rf vs awi, R^2 = 0.822 (center-right) and e) rf vs vwi, R^2 = 0.988.

Table 5: Execution time measurement comparisons (in milliseconds) using microbenchmark and sample data

Name	Min	Mean	Max
vwi	74.06	76.23	84.39
awi	74.31	75.91	99.70
aws	128.48	132.56	143.97
awt	172.08	176.27	185.86
sf	136.48	142.33	289.38

Table 6: Execution time measurement comparisons (in seconds) using microbenchmark and external data

Name	Min	Mean	Max
vwi	1.55	1.67	1.94
awi	1.55	1.68	1.95
aws	2.26	2.39	2.64
awt	2.27	2.42	2.62
sf	2.63	2.79	3.17

Performance

In this section, a performance comparison (execution times) takes place using the **microbenchmark** package. Performance tests are carried out using the sample data provided by **populR** as well as external data sets from a significantly larger study area (Chios, Greece) with 845 city blocks and 15,946 buildings.

Execution time measurements for both cases are shown in Tables 5 and 6 accordingly. In both cases, execution time measurements suggest that **populR** performs faster than **areal** and **sf**. Using the built-in data, *awi* and *vwi* scored the best mean execution time (about 76 milliseconds), which is about 54 millisecond faster than *aws*, 61 milliseconds faster than *sf*, and almost 70 milliseconds faster than *awt*.

4 Conclusions

This study is an attempt to contribute to the continuously growing scientific literature of population downscaling using areal interpolation methods. Despite the fact that there are so many downscaling methods developed, only a few implementation tools are available to the GIS and R community, and therefore, it may be challenging for non-expert GIS users to take advantage of these methods (Mennis 2009; Langford 2007). Due to this lack of implementation tools, this study attempts to fill this gap by introducing **populR**, an R package for population downscaling using areal interpolation methods.

This package implements two areal interpolation methods, namely *awi* and *vwi*. The *awi* approach guides the interpolation process using the area of intersection between source and target zones while *vwi* uses the number of floors as additional information to influence the downscaling process. The package is available to the R community though the CRAN and Github repositories.

In order to show the efficacy and efficiency of the introduced package on actual data, a subset area of the city of Mytilini, Greece, was used in the case study. Moreover, a comparative analysis between **populR**, **areal**, and **sf** was carried out, and the results showed that *vwi* outperformed others by achieving the smallest error measurements, easy implementation and faster execution times. Thus, we strongly believe that R users will gain significant advantage by using **populR** for population downscaling.

The evaluation of the results indicates the potential of **populR** in providing better and faster results in comparison to existing R packages. However, we believe that the results may be further improved by incorporating new forms of data, such as Volunteered Geographic Information (VGI), acquired from either social media or open spatial databases such as Open Street Map (Bakillah et al. 2014; Guo, Cao, and Wang 2017; Yang et al. 2019; Comber and Zeng 2019). By incorporating VGI as ancillary information, we can identify different types of buildings and therefore adjust the weight calculation accordingly. This will be an important improvement that would be helpful for R users interested in population downscaling.

References

Ahola, Terhi, Kirsi Virrantaus, Jukka Matthias Krisp, and Gary J. Hunter. 2007. "A Spatio-Temporal Population Model to Support Risk Assessment and Damage Analysis for Decision-Making." *International Journal of Geographical Information Science* 21 (8): 935–53. https://doi.org/10.1080/13658810701349078.

Bahadori, Mohammad Sadegh, Alexandre B. Gonçalves, and Filipe Moura. 2021. "A Systematic Review of Station Location Techniques for Bicycle-Sharing Systems Planning and Operation." *ISPRS International Journal of Geo-Information* 10 (8): 554. https://doi.org/10.3390/ijgi10080554.

Bakillah, Mohamed, Steve Liang, Amin Mobasheri, Jamal Jokar Arsanjani, and Alexander Zipf. 2014. "Fine-Resolution Population Mapping Using OpenStreetMap Points-of-Interest." *International Journal of Geographical Information Science* 28 (9): 1940–63. https://doi.org/10.1080/13658816.2014.909045.

Batsaris, Marios. 2022. *populR: Population Downscaling*. https://CRAN.R-project.org/package=populR.

Batsaris, Marios, Dimitris Kavroudakis, Nikolaos A. Soulakellis, and Themistoklis Kontos. 2019. "Location-Allocation Modeling for Emergency Evacuation Planning in a Smart City Context." *International Journal of Applied Geospatial Research* 10 (4): 28–43. https://doi.org/10.4018/ijagr.2019100103.

Bian, Ruijie, and Chester G. Wilmot. 2015. "Spatiotemporal Population Distribution Method for Emergency Evacuation." *Transportation Research Record: Journal of the Transportation Research Board* 2532 (1): 99–106. https://doi.org/10.3141/2532-12.

Calka, Beata, Joanna Nowak Da Costa, and Elzbieta Bielecka. 2017. "Fine Scale Population Density Data and Its Application in Risk Assessment." *Geomatics, Natural Hazards and Risk* 8 (2): 1440–55. https://doi.org/10.1080/19475705.2017.1345792.

Comber, Alexis, and Wen Zeng. 2019. "Spatial Interpolation Using Areal Features: A Review of Methods and Opportunities Using New Forms of Data with Coded Illustrations." *Geography Compass* 13 (10): 1–23. https://doi.org/10.1111/gec3.12465.

Dobson, J E, E A Bright, R G Durfee, and B A Worley. 2000. "LandScan: A Global Population Database for Estimating Population at Risk." *Photogrammetric Engineering and Remote Sensing* 66 (7): 849–57.

European Union, EU. 2009. "European Parliament and Council Regulation (EC) No 763/2008 on Population and Housing Censuses." https://ec.europa.eu/eurostat/web/population-demography/population-housing-censuses/legislation.

Fisher, F. Peter, and Mitchel Langford. 1995. "Modelling the Errors in Areal Interpolation Between Zonal Systems by Monte Carlo Simulation." *Environment and Planning A* 27: 211–24. https://doi.org/10.1068/a270211.

Freire, S., and C. Aubrecht. 2012. "Integrating Population Dynamics into Mapping Human Exposure to Seismic Hazard." *Natural Hazards and Earth System Science* 12 (11): 3533–43. https://doi.org/10.5194/nhess-12-3533-2012.

Goodchild, Michael F., and Siu-Ngan Nina Lam. 1980. "Areal Interpolation: A Variant of the Traditional Spatial Problem." *Geo-Processing* 1 (3): 297–312.

Guo, Hui, Kai Cao, and Peng Wang. 2017. "Population Estimation in Singapore Based on Remote Sensing and Open Data." *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives* 42: 1181–87. https://doi.org/10.5194/isprs-archives-XLII-2-W7-1181-2017.

Han, Bing, Mingxing Hu, and Jialing Wang. 2020. "Site Selection for Pre-Hospital Emergency Stations Based on the Actual Spatiotemporal Demand: A Case Study of Nanjing City, China." *ISPRS International Journal of Geo-Information* 9 (10): 542. https://doi.org/10.3390/ijgi9100559.

Hellenic Statistical Authority, HSA. 2014. "2011 Population and Housing Census of Greece." Hellenic Statistical Authority.

Kim, Hwahwan, and Xiaobai Yao. 2010. "Pycnophylactic Interpolation Revisited: Integration with the Dasymetric-Mapping Method." *International Journal of Remote Sensing* 31 (21): 5657–71. https://doi.org/10.1080/01431161.2010.496805.

Lam, Nina Siu-Ngan. 1983. "Spatial Interpolation Methods: A Review." *The American Cartographer* 10 (2): 129–50. https://doi.org/10.1559/152304083783914958.

Langford, Mitchel. 2007. "Rapid Facilitation of Dasymetric-Based Population Interpolation by Means of Raster Pixel Maps." *Computers, Environment and Urban Systems* 31 (1): 19–32. https://doi.org/10.1016/j.compenvurbsys.2005.07.005.

Liu, X. H., P. C. Kyriakidis, and M. F. Goodchild. 2008. "Population-Density Estimation Using Regression and Area-to-Point Residual Kriging." *International Journal of Geographical Information Science* 22 (4): 431–47. https://doi.org/10.1080/13658810701492225.

Mennis, Jeremy. 2009. "Dasymetric Mapping for Estimating Population in Small Areas." *Geography Compass* 3 (2): 727–45. https://doi.org/10.1111/j.1749-8198.2009.00220.x.

Pebesma, Edzer. 2018. "Simple Features for R: Standardized Support for Spatial Vector Data." *R*

Journal 10 (1): 439–46. https://doi.org/10.32614/rj-2018-009.

Prener, Christopher, and Charles Revord. 2019. "areal: An R Package for Areal Weighted Interpolation." *Journal of Open Source Software* 4 (37): 1221. https://doi.org/10.21105/joss.01221.

Qiu, Fang, Caiyun Zhang, and Yuhong Zhou. 2012. "The Development of an Areal Interpolation ArcGIS Extension and a Comparative Study." *GIScience and Remote Sensing* 49 (5): 644–63. https://doi.org/10.2747/1548-1603.49.5.644.

Tenerelli, Patrizia, Javier F. Gallego, and Daniele Ehrlich. 2015. "Population Density Modelling in Support of Disaster Risk Assessment." *International Journal of Disaster Risk Reduction* 13: 334–41. https://doi.org/10.1016/j.ijdrr.2015.07.015.

Wang, Wenda, Zhibang Xu, Dongqi Sun, and Ting Lan. 2021. "Spatial Optimization of Mega-City Fire Stations Based on Multi-Source Geospatial Data: A Case Study in Beijing." *ISPRS International Journal of Geo-Information* 10 (5): 282. https://doi.org/10.3390/ijgi10050282.

Wu, Shuo-Sheng, Xiaomin Qiu, and Le Wang. 2005. "Population Estimation Methods in GIS and Remote Sensing: A Review." *GIScience and Remote Sensing* 42 (1): 80–96. https://doi.org/10.2747/1548-1603.42.1.80.

Yang, Xuchao, Tingting Ye, Naizhuo Zhao, Qian Chen, Wenze Yue, Jiaguo Qi, Biao Zeng, and Peng Jia. 2019. "Population Mapping with Multisensor Remote Sensing Images and Point-of-Interest Data." *Remote Sensing* 11 (5): 574. https://doi.org/10.3390/rs11050574.

Marios Batsaris
University of the Aegean
Department of Geography
Mytilini, Greece
https://www.mbatsaris.gr
ORCiD: 0000-0002-1805-3528
m.batsaris@aegean.gr

Dimitris Kavroudakis
University of the Aegean
Department of Geography
Mytilini, Greece
https://www.britannica.com/animal/bilby
ORCiD: 0000-0001-5782-3049
dimitrisk@aegean.gr

dycdtools: an R Package for Assisting Calibration and Visualising Outputs of an Aquatic Ecosystem Model

by Songyan Yu, Christopher G. McBride, Marieke A. Frassl, Matthew R. Hipsey and David P. Hamilton

Abstract The high complexity of aquatic ecosystem models (AEMs) necessitates a large number of parameters that need calibration, and visualisation of their multifaceted and multi-layered simulation results is necessary for effective communication. Here we present an R package "dycdtools" that contains calibration and post-processing tools for a widely applied aquatic ecosystem model (DYRESM-CAEDYM). The calibration assistant function within the package automatically tests a large number of combinations of parameter values and returns corresponding values for goodness-of-fit, allowing users to narrow parameter ranges or optimise parameter values. The post-processing functions enable users to visualise modelling outputs in four ways: as contours, profiles, time series, and scatterplots. The "dycdtools" package is the first open-source calibration and post-processing tool for DYRESM-CAEDYM, and can also be adjusted for other AEMs with a similar structure. This package is useful to reduce the calibration burden for users and to effectively communicate model results with a broader community.

1 Introduction

Aquatic ecosystem models are important tools to understand the structure and function of aquatic ecosystems, fill observation gaps, and support scientific management of water quality of inland waters (Jakeman et al., 2006). Processed-based aquatic ecosystem models represent the major physical, chemical and biological processes as a series of mathematical equations, offering opportunities for exploratory or predictive management applications (Frassl et al., 2019; Özkundakci et al., 2011). These models are increasingly used to simulate observed data and forecast changes that may occur under scenarios in a changing climate (Elliott, 2012; Nielsen et al., 2017; Rousso et al., 2020). A typical example of such models is DYRESM-CAEDYM (DYnamic REservoir Simulation Model – Computational Aquatic Ecosystem Dynamics Model), which has been developed from a one-dimensional coupled hydrodynamic-ecological lake model (Hamilton and Schladow, 1997) and has been widely used to simulate water quality and ecosystem processes for a large number of lakes and reservoirs (Cui et al., 2016; Lewis et al., 2004; Takkouk and Casamitjana, 2016).

Aquatic ecosystem models, particularly those with a large number of physical and biogeochemical parameters, need calibration before they can be used for reliable simulation outputs (Luo et al., 2018). For example, water quality simulations with DYRESM-CAEDYM usually involve calibration of 20-30 parameters (e.g. 18 parameters in Luo et al. (2018); 28 parameters in Schladow and Hamilton (1997)). Parameter values are usually chosen via trial and error (Takkouk and Casamitjana, 2016), parameter ranges in the literature and modeller experience (Lehmann and Hamilton, 2018; Robson and Hamilton, 2004), laboratory experimental data (Robson and Hamilton, 2003), or small-scale field measurements (Burger et al., 2008). The calibration process of stepwise iterative manual adjustment of parameters is labour intensive and time consuming, partly due to the interdependent nature by which state variables respond to individual parameter adjustments (Lehmann and Hamilton, 2018). The success of calibration (i.e., ability to accurately reproduce a set of observed data) relies strongly on the skill and experience of the modeller, and the calibration process is often considered to be complete when the difference between observations and simulations is within an acceptable level of statistical compliance (Hipsey et al., 2020).

Numerous development efforts are underway to provide assistance in the calibration process. A promising approach is to take advantage of increasing computational power to overcome some shortcomings of traditional manual calibration methods and to automatically trial a large number of possible combinations of parameters in silico. For example, a model independent parameter estimator tool, PEST (Parameter ESTimation), is able to carry out nonlinear parameter estimation for most environmental simulation models (Doherty et al., 1994; Doherty, 2018). PEST has been widely used for parameter calibration and to quantify errors in surface water and groundwater models (Christensen and Doherty, 2008; Gallagher and Doherty, 2007; White et al., 2014), but has rarely been applied in numerical lake models. In addition, multiple calibration assistant tools have been developed for specific lake models, but many are not open-source or freely accessible to model users, limiting their potential application to a broad community. Developing tools in a more open way to facilitate

aquatic ecosystem model calibration and output processing can facilitate accessibility and create user-friendly tools, as evidenced in the lake modelling community (Frassl et al., 2019). Such tools have been developed for a range of lake models, including the Freshwater Lake Model (FLake), General Lake Model (GLM), General Ocean Turbulence Model (GOTM) and Simstrat in the **LakeEnsembleR** package (https://github.com/aemon-j/LakeEnsemblR). However, currently no such open-source tools have been developed to assist calibration for the widely applied lake model, DYRESM-CAEDYM.

Visualisation is fundamental to studying complex subject matter and supporting the whole information pipeline, from acquiring and exploring data and analysing models, to visual analytics, through to storytelling to communicate background information, results, and conclusions (McInerny et al., 2014). Simulation results from lake models are often multi-layered across different depths, multifaceted across different modelling variables, and dynamic over a simulation period, posing difficulties in effective presentation for many end users. Although some lake models have a default Graphical User Interface (GUI) to visualise modelling results for different layers and variables, the built-in plotting functions in these GUIs are often limited and inflexible. For example, users are not able to compare observations and simulations in the same figure using the current GUI in DYRESM, undermining the ability of visual checks to assess goodness-of-fit to inform parameter calibration.

To bridge this gap, we demonstrate in this study the development and application of an R package **dycdtools** for assisting calibration of DYRESM-CAEDYM and post-processing of simulation outputs. A calibration assistant tool has previously been developed for DYRESM-CAEDYM (Luo et al., 2018), but was coded in Fortran language and was not stored on an open platform, such as GitHub, to support further development. By contrast, the **dycdtools** package designed to support additional development can be freely downloaded and used by DYRESM-CAEDYM users. The package also includes advanced tools to support visual assessments and enable effective communication of modelling outcomes. This package not only works with DYRESM-CAEDYM but can be adjusted for other aquatic ecological models with a similar structure, such as the General Lake Model (Hipsey et al., 2019).

In the following, we first describe briefly the structure of DYRESM-CAEDYM and the shortcomings in its application that were the primary driver for development of **dycdtools**. We also present in detail the structure of the R package, including a demonstration of the package through application to a monomictic lake. We conclude with the development and application of calibration assistant and visualisation tools for aquatic ecosystem models.

2 DYRESM-CAEDYM

DYRESM-CAEDYM consists of a hydrodynamic model DYRESM coupled with an ecological model CAEDYM. DYRESM is one-dimensional and resolves vertical distributions of temperature, salinity and density in lakes and reservoirs based on a dynamic Lagrangian layer structure, which simulates the lake as horizontally uniform layers that expand and contract in response to heat, mass and momentum exchanges (Gal et al., 2003). CAEDYM is an aquatic ecological model designed to simulate processes affecting carbon, nitrogen, phosphorus, silicon and dissolved oxygen cycles, including several size classes of inorganic suspended solids, and phytoplankton dynamics.

DYRESM-CAEDYM takes a wide variety of forcing and morphometry data and configuration files as input to represent the major aquatic physical and biogeochemical processes. The required forcing and morphometry data include meteorology, morphometry, inflows and outflows. Configuration files allow users to adjust sensitive parameter values for a model simulation. It is the configuration files that the auto-calibration function in **dycdtools** deals with, assuming that the forcing (e.g., meteorology, inflows and outflows) and fixed data (morphometry) have been obtained from external sources (Figure 1). The scientific foundations of DYRESM-CAEDYM and information on data input preparation can be found in its scientific manual (Imerito, 2007) and in Robson and Hamilton (2004), Romero et al. (2004), Luo et al. (2018).

DYRESM-CAEDYM is supported by a proprietary GUI, but the GUI provides limited means to visualise simulation results. So far, only time series and contour plots are built in the GUI, and they require users to take complex steps to plot the observation and simulation in the same figure. The dycdtools package provides four different types of visualisation plots that could be used to complement the default GUI and to examine model simulation results in more detail (Figure 1).

3 Structure of the dycdtools package

There are two main function categories in the **dycdtools** package: calibration assistant and post-processing (Table 1). After the forcing and morphometry input data to DYRESM-CAEDYM have been prepared, the calibration assistant function can be used to 1) choose the optimal set of parameter

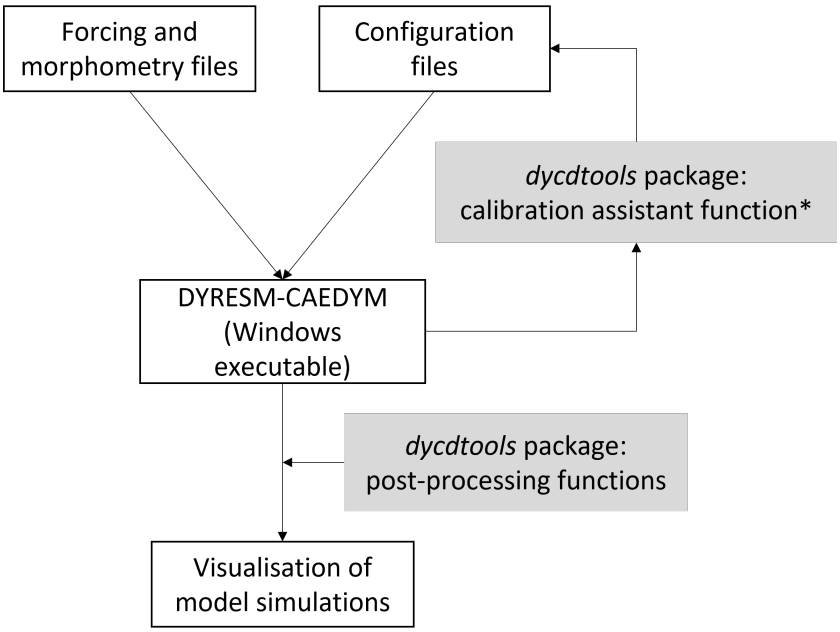


Figure 1: DYRESM-CAEDYM simulation process. The calibration assistant function in the **dycdtools** package deals with the model configuration files (the top shading rectangle), while the post-processing functions can be used to visualise model outputs (the bottom shading rectangle). *The calibration assistant function works only on the Windows platform, as it needs to call the Windows executable of DYRESM-CAEDYM to run model simulations, while the post-processing functions are cross-platform.

values for a model simulation application to a lake, or 2) conduct sensitivity analysis of parameters of interest including using a Monte-Carlo parameter perturbation function. Post-processing includes a suite of R functions to visualise DYRESM-CAEDYM output in multiple ways, such as contours, profiles, time series, and scatterplots, for different simulation and observation depths (Table 1).

Calibration assistant function

The calibration assistant function carries out simulations with a large number of possible combinations of parameter values that users regard as potentially suitable for their model calibration, and calculates the values of nominated objective functions (i.e. statistical measures of goodness-of-fit) for each combination. Based on the calculated objective function values, users can determine the optimal set(s) of parameter values or narrow the ranges of possible parameter values. Note that the calibration assistant function does not have the source codes of DYRESM-CAEDYM, but calls the Windows executable of the lake model to run separate simulations with various combinations of parameter values. For this reason, the calibration assistant function can only work on the Windows platform.

The calibration assistant function first requires that users prepare a list of sensitive parameters for their simulation (Figure 2). The selection of sensitive parameters can be made by referring to previous literature (Bruce et al., 2006; Robson and Hamilton, 2004; Schladow and Hamilton, 1997) or through expert opinion (Lehmann and Hamilton, 2018) as well as by using a parameter sensitivity analysis, which is a part of the calibration assistant function (Ofir et al., 2017). The list of selected sensitive parameters should be prepared as comma-separated values (.csv) in a file input to the function. The potential value range and the preferred number of values within the range for each parameter also need to be defined by users in the csv file. To facilitate this process, an example csv file of many common parameters has been included in the package data set, so that users can follow the given format and adjust values to their specific case study.

Based on the user-defined potential value range and the preferred number of values, the calibration assistant function forms all possible combinations of parameter values (Figure 2). For example, for three parameters that have three different values, there are 3^3=27 possible combinations. Users can choose to make the function try every possible combination for model calibration or only a subset of randomly selected combinations. To help users deal with complicated situations where much more parameter combinations are involved, the calibration assistant function is able to parallelise DYRESM-CAEDYM simulations with multiple cores, which could substantially reduce model running time.

Table 1: List of functions in the dycdtools package for calibration assistant and post-processing and their descriptions.

Function name	Description	Categories
calib.assist	Apply different combinations of selected parameter values and output corresponding values of goodness-of-fit by calculating objective functions. Users can choose the optimal set of parameter values or refine potential parameter value ranges based on the calculated values of goodness-of-fit.	calibration assistant
plot_cont	Contour plot (heat map) showing a depth- and time-resolved matrix of biogeochemical variables.	post-processing
plot_prof	Profile graph of simulations over time (only for dates when observations are available), with variable values shown in the x axis and depth in the y axis.	post-processing
plot_ts	Time series plot of simulations for one or multiple specific depths, with time as the x axis and variable values as the y axis.	post-processing
plot_scatter	Scatter plot with observations as the x axis and simulations as the y axis. Points in the scatter plot are colour coded by depth.	post-processing

For each of the combinations, the calibration assistant function makes changes to configuration files and then calls the DYRESM-CAEDYM model to run simulations (Figure 2). After each model run, the function extracts model outputs to compare against the observations and calculates objective function values. Five commonly used objective functions are currently available in the package: Nash-Sutcliffe Efficiency (NSE) coefficient, Root Mean Square Error (RMSE), Mean Absolute Error (MAE), Relative Absolute Error (RAE), and Pearson's r (Pearson), with more to be added in an update of the package by referring to the inventory of different statistical measures in Bennett et al. (2013). The output of the calibration assistant function is a table that outlines all combinations of parameters that were used and the corresponding objective function values. Based on the output, users can determine the optimal set(s) of parameter values (e.g., the set that gives the highest NSE value or highest weighted average values for NSE and RMSE) or narrow the range of suitable parameter values.

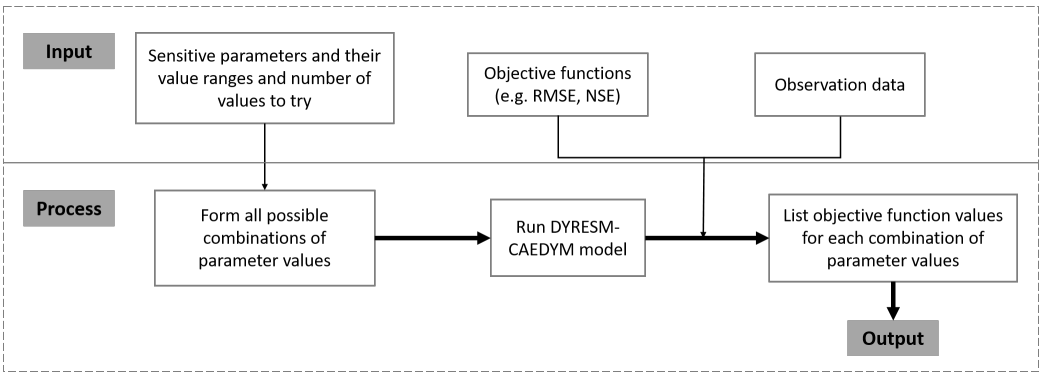


Figure 2: Structure of the calibration assistant function. Three key inputs to the calibration assistant function include a list of sensitive parameters and their assigned value ranges, objective functions, and observation data. The function forms scenarios, runs DYRESM-CAEDYM model on each of the scenarios, and outputs objective function values.

The calibration assistant function can also be used for parameter sensitivity analysis by automatically modifying parameter values. Two commonly used methods for sensitivity analysis, One At a Time (OAT) and All At a Time (AAT), can be carried out through this function by setting the "combination" argument as "all" and preparing specifically designed parameter combinations. An OAT analysis changes a single parameter at a time and is suitable for local sensitivity analysis, while ATT

modifies multiple parameters simultaneously and is suitable for global sensitivity analysis, notably when there are nonlinear relationships between parameter and model outputs. Increasing numbers of environmental modelling studies report model uncertainty for a variety of purposes, including assessment of model errors, model calibration communication and diagnostic evaluation (Couture et al., 2018; Dietzel and Reichert, 2014; Pianosi et al., 2016). This function is also flexible, allowing users to try a subset of randomly selected parameter combinations (e.g. Monte-Carlo parameter perturbation) or a carefully selected combination by setting the "combination" argument as "random".

Post-processing functions

Post-processing functions provide multiple ways to visualise DYRESM-CAEDYM outputs, as follows:

- Function "plot_cont" displays a heat map of variable values with depth within the water column and over time. This visualisation is particularly suitable for displaying temporal and depth dynamics of a variable at one lake site.
- Function "plot_prof" shows vertical profiles of the simulation and corresponding observations, for all dates where observations are available.
- Function "plot_ts" plots simulated values and observations for a specified variable and depth over time. It can be used to compare temporal changes of a variable for simulations and observations at specific depths.
- Function "plot_scatter" shows observations against simulated values for corresponding time and depth, with a colour scale representing measured depth. It can be used to demonstrate visually the goodness of fit for a variable across the water column.

All four types of graphs can be used to compare simulations and observations in the same figure. Examples of each type of graph are shown in the following section.

4 Minimal case study example

Lake Okareka is a medium size lake in the Bay of Plenty region of North Island, New Zealand. It has a surface area of 3.46 km^2, a land catchment area of 16.7 km^2 and a maximum depth of 33.5 meters. Water drains from the lake via an outlet canal towards a large downstream lake, Tarawera. The **dycdtools** package was used to calibrate three parameters for the lake temperature simulation and visualise the temperature simulation against observed data with the four aforementioned plotting functions. The three calibrated parameters were wind stirring efficiency, vertical mixing coefficient, and light extinction coefficient, as they have been found to be most sensitive in temperature simulations in other lake systems (Weinberger and Vetter, 2012).

The simulations were conducted with forcing inputs (e.g., meteorology, inflows and outflow) at a daily time step over the period of 2002-01-23 to 2016-12-31. The meteorology and outflow data were respectively collected from the New Zealand National Climate Database (`https://cliflo.niwa.co.nz`) and an environmental monitoring website (`https://www.boprc.govt.nz/environment/maps-and-data/environmental-data`). Discharge output from a SWAT model application (unpublished) was used as catchment surface inflow to the lake. All example data are provided in a public data repository (`https://doi.org/10.5281/zenodo.7431128`) for users to familiarise themselves with model runs, calibration and visualisation.

The calibration assistant function deals with the configuration files. In this simulation, both the wind stirring efficiency and vertical mixing coefficient are in the parameter (.par) file, while the light extinction coefficient is in the configuration (.cfg) file. We assigned a range and a preferred number of values to try for each of the three parameters to form a sequence of possible values (Table 2).

Table 2: Three parameters that were calibrated with assigned value ranges and preferred number of values to try (N_values). The columns of "Input file" and "Line No." describe the line of the input file where the parameter is located.

Parameter	Unit	Min	Max	N_values	Input file	Line No.
Wind stirring efficiency	-	0.2	0.8	4	Okareka.par	12
Vertical mixing coefficient	-	100	700	4	Okareka.par	15
Light extinction coefficient	m^{-1}	0.1	0.25	4	Okareka.cfg	7

The calibration assistant function tried all combinations of the three parameter values and calculated the objective function NSE for each model run. We chose the combination of parameter values that gave the highest NSE values as the optimal set and then used the post-processing tools to plot simulations against observations in four different ways. The R code for using the calibration assistant function in this example is as follows:

```
# Assisting calibration of DYRESM-CAEDYM using the dycdtools package
 library(dycdtools)
 calib.assist(cal.para = "calibration_data/Calibration_parameters.csv",
              combination = "all",
              model.var = "TEMP",
              obs.data = "calibration_data/Obs_data_template.csv",
              objective.function = "NSE",
              start.date = "2002-01-23",
              end.date = "2016-12-31",
              dycd.wd = "calibration_data/DYRESM_CAEDYM_Lake-Okareka/",
              dycd.output = "calibration_data/DYRESM_CAEDYM_Lake-Okareka/DYsim.nc",
              file.name = "calibration_data/Calibration_outputs.csv",
              write.out = TRUE,
              parallel = TRUE,
              verbose = TRUE)
```

Given each parameter had four assigned values, the calibration assistant function called DYRESM-CAEDYM to run a total of $4^3 = 64$ combinations for the three parameters. For each model run, the objective function NSE was calculated for temperature simulations. A heat map is a good way to visualise the variations in these NSE values under different combinations of parameter values. An example of R code for producing a heat map of the auto-calibration outcome is as follows:

```
 # Read in model calibration results
 calibration <- read.csv("calibration_data/Calibration_outputs.csv")

 # Heat map
 library(ggplot2)
 ggplot(calibration, aes(x = wse,y = vmc,fill = NSE.TEMP)) +
        geom_tile() +
        scale_fill_distiller(palette = "PuBu", direction = 1) +
        facet_grid(~lec, scales = "free") +
        xlab("Wind stirring efficiency") +
        ylab("Vertical mixing coefficient") +
        labs(title = "Light extinction coefficient", fill = "NSE") +
        theme_bw()  +
        theme(plot.title = element_text(size = 11, hjust = 0.5))

 ggsave(filename = "Figure_03.png", width = 8, height = 4)
```

The calculated NSE values varied significantly among the combinations, ranging from 0.03 to 0.95. The combination of wind stirring efficiency = 0.6, vertical mixing coefficient = 500, and light extinction coefficient = 0.25 m^{-1} achieved the highest NSE value of 0.95 (i.e., considered to be the optimal set of parameter values) (Figure 3).

A few other combinations could also achieve similar outcomes. For example, when the light extinction coefficient = 0.25 m^{-1}, the NSE values ranged between 0.90 and 0.95, regardless of the wind stirring efficiency and vertical mixing coefficient values shown in Table 2. Figure 3 also revealed that temperature simulations were generally poor when light extinction coefficient = 0.1 and 0.15 m^{-1}, suggesting that these values of light extinction coefficient are too low to achieve a suitably accurate temperature simulation for Lake Okareka.

Under the optimal set of parameter values identified from the calibration assistant function (Figure 3), we re-ran the DYRESM-CAEDYM model and visualised the temperature simulations with the four post-processing functions. An example R code for visualisation functions is shown preceding the corresponding figure.

The temperature simulations in Lake Okareka displayed a clear and consistent seasonal pattern. Temperature was up to 24 °C in the summer season and was below 10 °C in the winter (Figure 4). The simulation contour plot showed that the thermocline of Lake Okareka (the layer of most rapid temperature change) was around 15 m in summer, but the lake always mixed in winter (Figure 4), corresponding to isothermal conditions over the water depth, suggesting a monomictic lake mixing

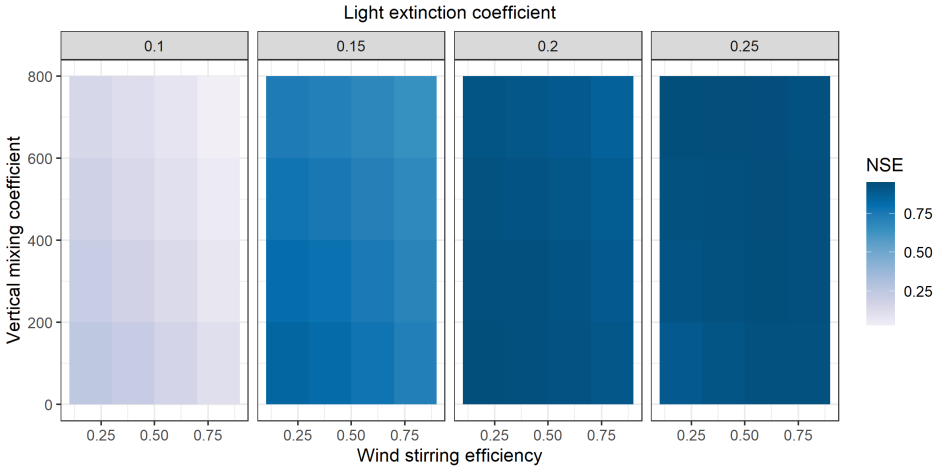


Figure 3: Heat map of Nash-Sutcliffe Efficiency (NSE) coefficient values for temperature simulations in Lake Okareka estimated for the combinations of three parameters given in Table 2 (wind stirring efficiency, vertical mixing coefficient, and light extinction coefficient [m^{-1}]). The darker the colour, the better the model performance (measured as NSE).

regime. Observations are shown as dots in Figure 4 and match well with simulations both spatially (i.e., vertically) and temporally (i.e., horizontally).

```
# Extract temperature simulations
var.values <- ext_output(dycd.output = "DYCD_Okareka/DYsim.nc",
                         var.extract = c("TEMP"))

# Interpolation of temperature across water column at an interval of 0.5 m
temp.interpolated < -interpol(layerHeights = var.values$dyresmLAYER_HTS_Var,
                              var = var.values$dyresmTEMPTURE_Var,
                              min.dept = 0, max.dept = 33, by.value = 0.5)

# Read in observed water quality data
obs.okareka <- read.csv("plotting_data/Obs_data_template.csv")
obs.okareka$Date <- as.Date(obs.okareka$Date,format="%d/%m/%Y")
# subset observed data to remain temperature observations
obs.temp <- obs.okareka[, c('Date','Depth','TEMP')]

# Contour plot
png(filename = 'Figure_04.png', width = 1200, height = 700)
plot_cont_com(sim = temp.interpolated,
              obs = obs.temp,
              plot.start = "2002-01-23",
              plot.end = "2006-12-31",
              sim.start = "2002-01-23",
              sim.end = "2016-12-31"
              legend.title = "T\n(\u00B0C)",
              min.depth = 0,
              max.depth = 33,
              by.value = 0.5,
              nlevels = 20)
dev.off()
```

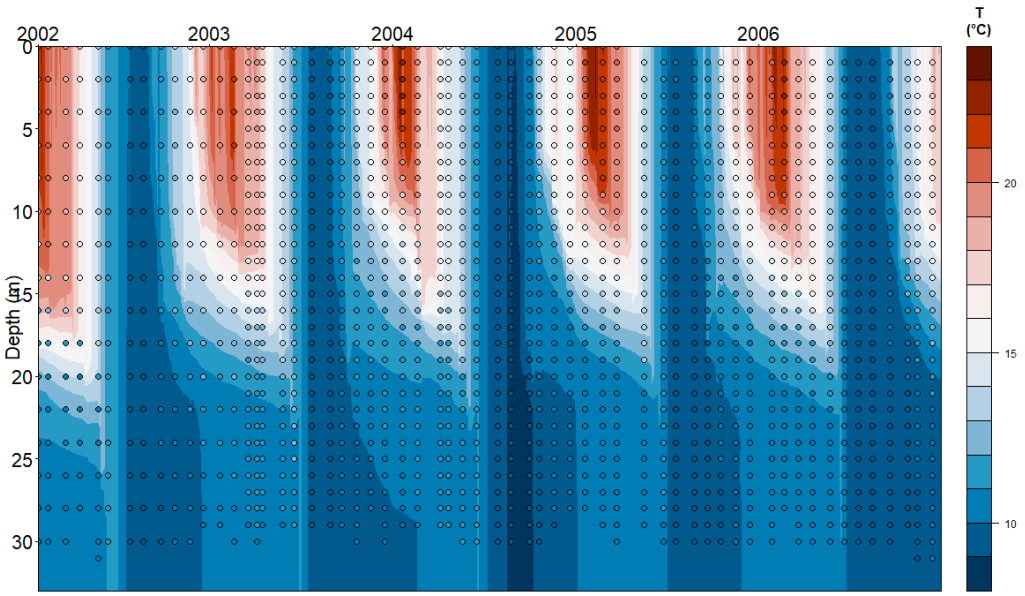


Figure 4: Example contour plot of temperature simulations and observations for Lake Okareka from 2002-01-23 to 2006-12-31. Each dot represents an observation at a specific depth on a specific date. Dots are colour coded on the same scale (right-hand side) as the simulation.

Simulations and observations of temperature with depth are shown in Figure 5 for the dates when observations were available. The vertical profiles provide a more discerning view of details such as the thermocline depth in Lake Okareka. The simulated temperature profile mostly aligned well with the observed profile but had a tendency to overestimate the depth of the thermocline in the first summer (early 2002), whilst capturing the timing of mixing (little or no temperature gradient) in winter.

```
# Profile plot
plot_prof(sim = temp.interpolated,
          obs = obs.temp,
          sim.start = "2002-01-23",
          sim.end = "2016-12-31",
          plot.start = "2002-01-23",
          plot.end = "2002-12-31",
          min.depth = 0,
          max.depth = 33,
          by.value = 0.5,
          xlabel = "Temperature \u00B0C")

ggsave(filename = "Figrue_05.png", height = 4, width = 7)
```

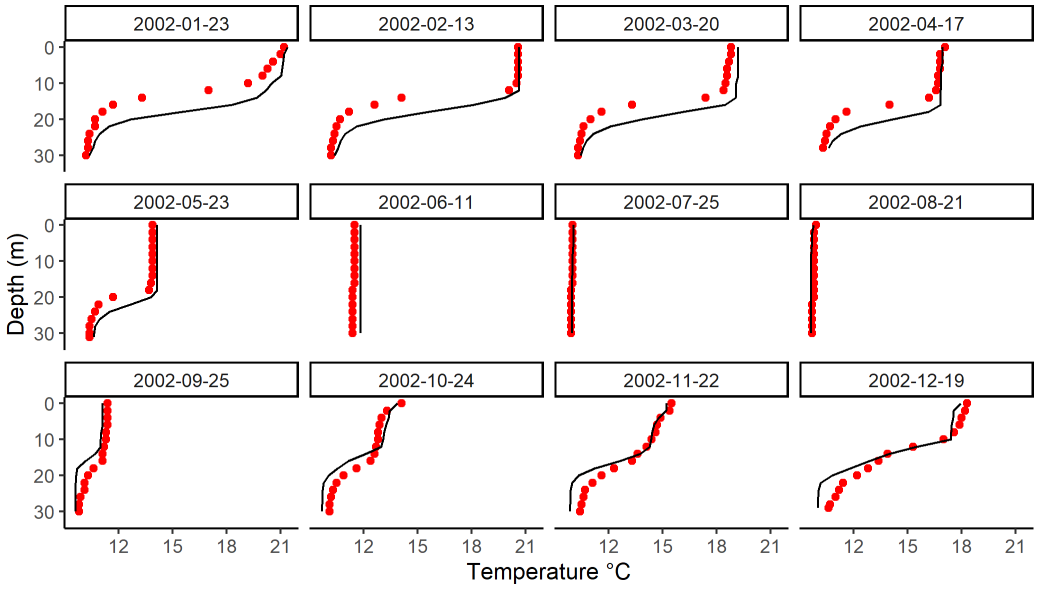


Figure 5: Example profile plots of temperature simulations (black line) and observations (brown dots) for Lake Okareka from 2002-01-23 to 2002-12-19, available for each observation occasion. The profile plots shows a seasonable pattern of stratification and mixing in the lake over the 12 months time, and the model was able to repeat the pattern.

The time series figure displays temperature simulations as a continuous line for a specific depth over the entire simulation period, with observations shown as dots (Figure 6). The temperature in the surface of the lake (1 m deep) varied strongly over the years, with a clear seasonal pattern, while at mid-depth (14 m deep) it varied to a lesser degree but showed a similar pattern, and in the near-bottom (30 m deep) layer it remained relatively unchanged.

```
# Time series plot
p <- plot_ts(sim = temp.interpolated,
             obs = obs.temp,
             target.depth = c(1, 14, 30),
             sim.start = "2002-01-23",
             sim.end = "2016-12-31",
             plot.start = "2002-01-23",
             plot.end = "2012-12-31",
             min.depth = 0,
             max.depth = 33,
             by.value = 0.5,
             ylabel = "Temperature \u00B0C")

rmse_dpt01 <- objective_fun(sim = temp.interpolated[2:4,],
                            obs = obs.temp[obs.temp$Depth == 1, ],
                            fun = c('RMSE'),
                            start.date = '2002-01-23',
                            end.date = '2016-12-31',
                            min.depth = 0.5,
                            max.depth = 1.5,
                            by.value = 0.5)

rmse_dpt14 <- objective_fun(sim = temp.interpolated[28:30,],
                            obs = obs.temp[obs.temp$Depth == 14, ],
                            fun = c('RMSE'),
                            start.date = '2002-01-23',
                            end.date = '2016-12-31',
                            min.depth = 13.5,
                            max.depth = 14.5,
                            by.value = 0.5)
```

```
rmse_dpt30 <- objective_fun(sim = temp.interpolated[60:62,],
                            obs = obs.temp[obs.temp$Depth == 30, ],
                            fun = c('RMSE'),
                            start.date = '2002-01-23',
                            end.date = '2016-12-31',
                            min.depth = 29.5,
                            max.depth = 30.5,
                            by.value = 0.5)

rmse_text <- data.frame(x = as.Date('2007-06-01'),
                        y = 25,
                        Depth = c(1, 14, 30),
                        label = c(paste0('RMSE = ', round(rmse_dpt01$RMSE,2), ' \u00B0C'),
                                  paste0('RMSE = ', round(rmse_dpt14$RMSE,2), ' \u00B0C'),
                                  paste0('RMSE = ', round(rmse_dpt30$RMSE,2), ' \u00B0C')))

p + ylim(8,26) +
    geom_text(data = rmse_text,
              mapping = aes(x = x, y = y, label = label))

ggsave(filename = 'Figure_06.png', height = 4, width = 7)
```

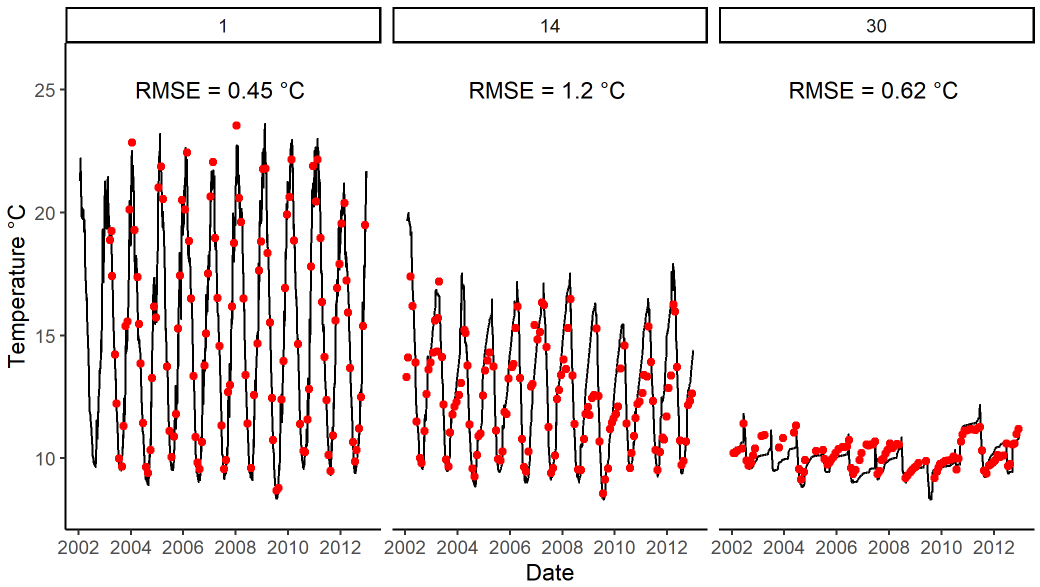


Figure 6: Example time series plot of temperature simulations (black line) and measurements (brown dots) for three depths (upper row box; 1 m – near-surface layer, 14 m – middle layer close to the thermocline, and 30 m – near-bottom layer) of Lake Okareka from 2002-01-23 to 2012-12-31. The RMSE values calculated from the 'objective_fun' function in the package are also shown for each depth profile.

The scatter plot compares all temperature simulation values against observations denoted as dots in a Cartesian coordinate system, with a reference line of y=x to indicate the ideal fit (1:1) of simulations to observations. Each point is colour coded by depth. The majority of dots were close to the reference line, indicating good performance of DYRESM-CAEDYM in the temperature simulation (Figure 7).

```
# Scatter plot
plot_scatter(sim=temp.interpolated,
             obs=obs.temp,
             sim.start="2002-01-23",
             sim.end="2016-12-31",
             plot.start = "2002-01-23",
             plot.end="2012-12-31",
             min.depth = 0,
             max.depth = 33,
             by.value = 0.5)

ggsave(filename = 'Figure_07.png', height = 4, width = 7)
```

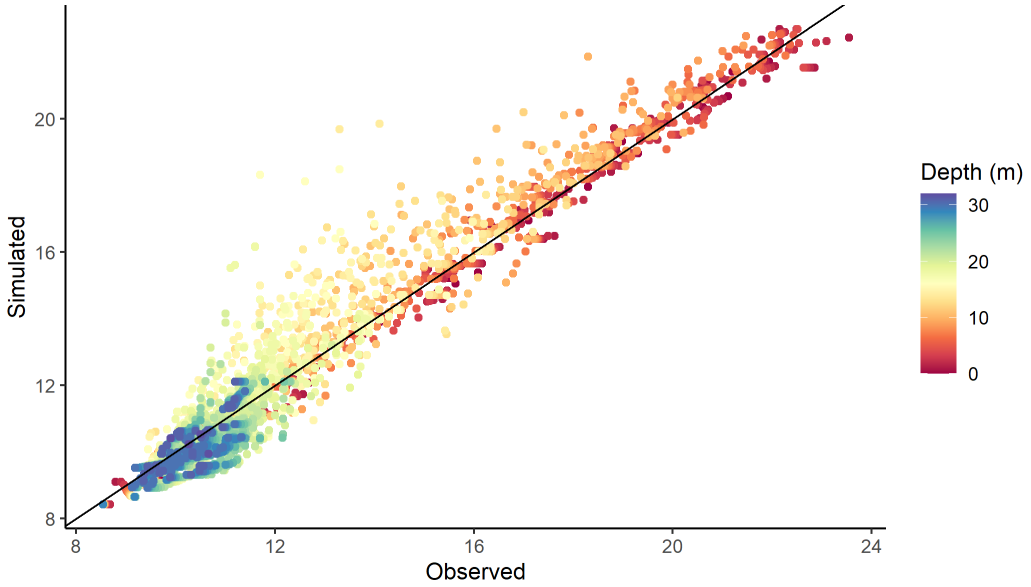


Figure 7: Example scatterplot of temperature simulations and observations for Lake Okareka from 2002-01-23 to 2012-12-31. Note that the dots are colour coded by depth. The red line is a 1:1 slope, i.e., simulated temperature = observed temperature. The plot indicates a good model performance as the majority of the dots are located around the black line.

5 Discussion

Aquatic ecosystem models have increasingly been used to shed light on lake ecosystem function (Scheffer et al., 2001) and inform policy and management decisions to improve water quality and control eutrophication (Wang et al., 2012). With progressive increases in the number and complexity of processes represented, and increasing computational power, these models have increased in complexity and have a large number of parameters. These types of models generally require extensive calibration to be suitable for lake-specific simulations, which is quite challenging, despite the development of parameter priors libraries (Robson et al., 2018) that can help with understanding possible ranges of parameters and synthesise values used in other studies. In addition, multifaceted and multi-layered modelling results need to be visualised in various ways to express model uncertainty and communicate effectively with a broader community that may be engaged in understanding and interpreting the implications of model scenarios. Here, we have presented an open-source R package **dycdtools** to help address these challenges in calibrating a widely used aquatic ecosystem model DYRESM-CAEDYM and visualising its simulation results. Importantly, this package has the potential to be adjusted for other aquatic ecological models with a similar structure.

Calibration assistant function

The output of the calibration assistant function in the **dycdtools** package is a list of combinations of parameter values with their objective function values, rather than a single optimal set of parameter values provided by many auto-calibration algorithms, such as the auto-calibration module in the

RAPID model developed in David et al. (2011), the auto-calibration algorithm proposed in Luo et al. (2018), and the Genetic Algorithm developed in Goldberg and Holland (1988). We chose not to provide a single parameter set partly due to the potential for parameter equifinality (Beven and Binley, 1992), a classical problem of optimisation where different sets of parameter values may provide similar simulation fit to observations. This problem is particularly pertinent for complex aquatic ecosystem models (e.g., DYRESM-CAEDYM) that include a large number of parameters. Many studies illustrated the difficulties of finding a single optimal set of parameter values in the high-dimensional parameter space, due to inter-correlation between parameters and autocorrelation of the residuals (Fernández Slezak et al., 2010; Gupta et al., 2006). Therefore, instead of using an auto-calibration algorithm to decide the optimal set of parameter values, we suggest users compare those combinations of parameter values that result in high objective values and manually choose the combination that they believe is the most physically meaningful. Such a process aligns with the concept of development of user expertise and knowledge that are critical for successful model applications (Hipsey et al., 2015).

It is worth noting that an objective function is a means to characterise the performance of environmental models, and is useful and often necessary (Bennett et al., 2013). A substantial body of work has been done to propose methods and criteria to judge the performance of environmental models, often for hydrological models (Jakeman et al., 2006; Krause et al., 2005) and less frequently for ecological models (Risbey et al., 1996). Many of the proposed objective functions have been summarised in Bennett et al. (2013) and can be applied to characterise performance of lake models. Five of the most commonly used objective functions have been included in the current package, and we plan to add functions to give users more choices of characterising model performance and to better inform parameter calibration.

Given that it is not realistic to expect any one set of parameter values to be truly representative of the actual parameter set (Beven and Binley, 1992), it is also important to convey the accuracy of the simulations and the associated uncertainties. In lake modelling, multi-parameter perturbation has been conducted using Monte-Carlo techniques to inform parameter uncertainty effects (Luo et al., 2018; Muraoka, 2019). The calibration assistant function developed here can be used to carry out such a sensitivity analysis by perturbing multiple parameters within a certain range when the "combination" argument is set to "random". User expert knowledge (Lehmann and Hamilton, 2018) and parameter priors libraries (Robson et al., 2018) are also valuable in narrowing parameter ranges which can be important in reducing computation times of calibration and sensitivity analysis, as well as reducing the incidence of common issues such as equifinality.

Suggestions on the use of calibration assistant function

The calibration assistant function developed in this study is not intended to replace manual calibration, rather, it is aimed to save time assigned to calibration and sensitivity analysis by automatically trialling a number of parameter values so as to bring a focus to the range of sensitive parameter values. We recommend manual calibration as a means to develop expert knowledge about model parameter values, in an iterative procedure that includes assistant calibration and sensitivity analysis, as well as literature review of parameters. We also recommend a combination of OAT and AAT calibration, with the former used to better understand individual responses of parameters and the latter used with care because exponentially large numbers of possible parameter value combinations are possible and computational times could become untenable even with parallel processing enabled. For example, in the case of 10 parameters where each parameter has three possible values, the total number of possible combinations is 3^{10} = 59,049. If each model run takes 2 minutes, then it would take 59,049 × 2 = 118,098 minutes (> 32 days) to run these combinations.

Manual calibration has also been used as a way to work around the difficulties with weighting and prioritising state variables using an auto-calibration procedure (Lehmann and Hamilton, 2018). Therefore, while acknowledging the benefits of assistant calibration in trialling a number of parameter values without the heavy time burden of manual adjustment, we encourage users to actively engage in the calibration process. More specifically, users may first define suitable value ranges of various parameters from the literature (including other modelling studies) and lab or field experiments (Robson et al., 2018), and then conduct both manual and automated sensitivity analysis to understand which parameters are sensitive and should be more strategically evaluated. Finally, users can apply both the calibration assistant function and manual calibration to optimise parameter values. This process can be repeated multiple times until users are satisfied with accuracy of the simulations, assessed using quantitative output statistics and qualitative comparisons with observed data.

Visualisation of model outputs

Visualisation of model outputs is an important tool for effectively engaging academic, the public and environmental managers in expressing model uncertainty, and as a prerequisite for improving confidence in scenario simulations. This is particularly true for multidimensional model outputs that are intangible, and visualisation has a fundamental role in exploring information and generating understanding (McInerny et al., 2014). The **dycdtools** package expands on existing lake model visualisation tools, such as contours, profiles and time series used by the **glmtools** and **LakeEnsemblR** packages, by providing additional functionality including scatterplots. Each of the plotting functions is designed to best suit different presentation purposes, as outlines in the "Post-processing functions" Section. We anticipate that the developed visualisation tools in **dycdtools** package can not only complement the existing associated GUI by providing more flexible ways of visualising DYRESM-CAEDYM model outputs, but can also be adapted for other lake ecosystem models, such as the General Lake Model (Hipsey et al., 2019) and the Fresh-water Lake Model for LM (Mironov et al., 2010).

In summary, dycdtools is a modular, flexible, and open-source tool that is designed to make the calibration process of the DYRESM-CAEDYM model substantially less time-consuming and to visualise complex modelling results in multiple ways for effective communication. It is suitable for users with various levels of expertise. This study places great emphasis on tools to evaluate the quality of model calibration as many of these models need to provide robust simulations that form the basis of scenario simulations, often designed to evaluate different management and environmental (e.g., climate change, land use change) scenarios.

6 Acknowledgements

We thank Gebiaw Ayele from Australian Rivers Institute, Griffith University for providing the catchment inflow simulations as part of the example data for the DYRESM-CAEDYM model application. We acknowledge Bay of Plenty Regional Council for provision of data for model runs. Author contributions: S.Y., M.F., and C.M. developed the source codes for the R package; S.Y. prepared the application example; and S.Y., D.H., C.M. and M.F. wrote the paper. This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

7 Data and software availability

The published dycdtools package can be downloaded from CRAN and the developer version is available from https://github.com/SongyanYu/dycdtools. Issues regarding the package can be submitted via https://github.com/SongyanYu/dycdtools/issues. The executables of DYRESM-CAEDYM and the example data supporting the case study can be found from https://doi.org/10.5281/zenodo.7431128. The source codes of DYRESM-CAEDYM are accessible from https://github.com/AquaticEcoDynamics/dy-cd

Bibliography

N. D. Bennett, B. F. Croke, G. Guariso, J. H. Guillaume, S. H. Hamilton, A. J. Jakeman, S. Marsili-Libelli, L. T. Newham, J. P. Norton, C. Perrin, et al. Characterising performance of environmental models. *Environmental Modelling & Software*, 40:1–20, 2013. URL https://doi.org/10.1016/j.envsoft.2012.09.011. [p238, 246]

K. Beven and A. Binley. The future of distributed models: model calibration and uncertainty prediction. *Hydrological Processes*, 6(3):279–298, 1992. URL https://doi.org/10.1002/hyp.3360060305. [p246]

L. C. Bruce, D. Hamilton, J. Imberger, G. Gal, M. Gophen, T. Zohary, and K. D. Hambright. A numerical simulation of the role of zooplankton in C, N and P cycling in Lake Kinneret, Israel. *Ecological Modelling*, 193(3-4):412–436, 2006. URL https://doi.org/10.1016/j.ecolmodel.2005.09.008. [p237]

D. F. Burger, D. P. Hamilton, and C. A. Pilditch. Modelling the relative importance of internal and external nutrient loads on water column nutrient concentrations and phytoplankton biomass in a shallow polymictic lake. *Ecological Modelling*, 211(3-4):411–423, 2008. URL https://doi.org/10.1016/j.ecolmodel.2007.09.028. [p235]

S. Christensen and J. Doherty. Predictive error dependencies when using pilot points and singular value decomposition in groundwater model calibration. *Advances in Water Resources*, 31(4):674–700, 2008. URL https://doi.org/10.1016/j.advwatres.2008.01.003. [p235]

R.-M. Couture, S. J. Moe, Y. Lin, Ø. Kaste, S. Haande, and A. L. Solheim. Simulating water quality and ecological status of Lake Vansjø, Norway, under land-use and climate change by linking process-oriented models with a Bayesian Network. *Science of the Total Environment*, 621:713–724, 2018. URL https://doi.org/10.1016/j.scitotenv.2017.11.303. [p239]

Y. Cui, G. Zhu, H. Li, L. Luo, X. Cheng, Y. Jin, and D. Trolle. Modeling the response of phytoplankton to reduced external nutrient load in a subtropical Chinese reservoir using DYRESM-CAEDYM. *Lake and Reservoir Management*, 32(2):146–157, 2016. URL https://doi.org/10.1080/10402381.2015.1136365. [p235]

C. H. David, D. R. Maidment, G.-Y. Niu, Z.-L. Yang, F. Habets, and V. Eijkhout. River network routing on the NHDPlus dataset. *Journal of Hydrometeorology*, 12(5):913–934, 2011. URL https://doi.org/10.1175/2011JHM1345.1. [p246]

A. Dietzel and P. Reichert. Bayesian inference of a lake water quality model by emulating its posterior density. *Water Resources Research*, 50(10):7626–7647, 2014. URL https://doi.org/10.1002/2012WR013086. [p239]

J. Doherty. Model-independent parameter estimation user manual part i: PEST, SENSAN and Global Optimisers. *Watermark Numerical Computing. Haettu*, 17(03):2019, 2018. URL https://pesthomepage.org/documentation. [p235]

J. Doherty et al. PEST: a unique computer program for model-independent parameter optimisation. In *National Conference Publication Institution of Engineers Australia NCP*, pages 551–554. Institution of Engineers, Australia, 1994. [p235]

J. A. Elliott. Is the future blue-green? a review of the current model predictions of how climate change could affect pelagic freshwater cyanobacteria. *Water Research*, 46(5):1364–1371, 2012. URL https://doi.org/10.1016/J.WATRES.2011.12.018. [p235]

D. Fernández Slezak, C. Suárez, G. A. Cecchi, G. Marshall, and G. Stolovitzky. When the optimal is not the best: parameter estimation in complex biological models. *PLOS ONE*, 5(10):e13283, 2010. URL https://doi.org/10.1371/journal.pone.0013283. [p246]

M. A. Frassl, J. M. Abell, D. A. Botelho, K. Cinque, B. R. Gibbes, K. D. Jöhnk, K. Muraoka, B. J. Robson, M. Wolski, M. Xiao, et al. A short review of contemporary developments in aquatic ecosystem modelling of lakes and reservoirs. *Environmental Modelling & Software*, 117:181–187, 2019. URL https://doi.org/10.1016/j.envsoft.2019.03.024. [p235, 236]

G. Gal, J. Imberger, T. Zohary, J. Antenucci, A. Anis, and T. Rosenberg. Simulating the thermal dynamics of Lake Kinneret. *Ecological Modelling*, 162(1-2):69–86, 2003. URL https://doi.org/10.1016/S0304-3800(02)00380-0. [p236]

M. Gallagher and J. Doherty. Predictive error analysis for a water resource management model. *Journal of Hydrology*, 334(3-4):513–533, 2007. URL https://doi.org/10.1016/j.jhydrol.2006.10.037. [p235]

D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3:95–99, 1988. URL https://doi.org/10.1023/A:1022602019183. [p246]

H. V. Gupta, K. J. Beven, and T. Wagener. Model calibration and uncertainty estimation. *Encyclopedia of Hydrological Sciences*, 2006. URL https://doi.org/10.1002/0470848944.hsa138. [p246]

D. P. Hamilton and S. G. Schladow. Prediction of water quality in lakes and reservoirs. Part I—Model description. *Ecological Modelling*, 96(1-3):91–110, 1997. URL https://doi.org/10.1016/S0304-3800(96)00062-2. [p235]

M. R. Hipsey, D. P. Hamilton, P. C. Hanson, C. C. Carey, J. Z. Coletti, J. S. Read, B. W. Ibelings, F. J. Valesini, and J. D. Brookes. Predicting the resilience and recovery of aquatic systems: A framework for model evolution within environmental observatories. *Water Resources Research*, 51(9):7023–7043, 2015. URL https://doi.org/10.1002/2015WR017175. [p246]

M. R. Hipsey, L. C. Bruce, C. Boon, B. Busch, C. C. Carey, D. P. Hamilton, P. C. Hanson, J. S. Read, E. de Sousa, M. Weber, et al. A General Lake Model (GLM 3.0) for linking with high-frequency sensor data from the Global Lake Ecological Observatory Network (GLEON). *Geoscientific Model Development*, 12(1):473–523, 2019. URL https://doi.org/10.5194/gmd-12-473-2019. [p236, 247]

M. R. Hipsey, G. Gal, G. B. Arhonditsis, C. C. Carey, J. A. Elliott, M. A. Frassl, J. H. Janse, L. de Mora, and B. J. Robson. A system of metrics for the assessment and improvement of aquatic ecosystem models. *Environmental Modelling & Software*, 128:104697, 2020. URL https://doi.org/10.1016/j.envsoft.2020.104697. [p235]

A. Imerito. Dynamic Reservoir Simulation Model DYRESM v4. 0 Science Manual. *Centre for Water Research, University of Western Australia*, 2007. [p236]

A. J. Jakeman, R. A. Letcher, and J. P. Norton. Ten iterative steps in development and evaluation of environmental models. *Environmental Modelling & Software*, 21(5):602–614, 2006. URL https://doi.org/10.1016/j.envsoft.2006.01.004. [p235, 246]

P. Krause, D. Boyle, and F. Bäse. Comparison of different efficiency criteria for hydrological model assessment. *Advances in Geosciences*, 5:89–97, 2005. URL https://doi.org/10.5194/adgeo-5-89-2005. [p246]

M. K. Lehmann and D. P. Hamilton. Modelling water quality to support lake restoration. In *Lake Restoration Handbook*, pages 67–105. Springer, 2018. [p235, 237, 246]

D. M. Lewis, J. D. Brookes, and M. F. Lambert. Numerical models for management of *Anabaena circinalis*. *Journal of Applied Phycology*, 16(6):457–468, 2004. URL https://doi.org/10.1007/s10811-004-5506-z. [p235]

L. Luo, D. Hamilton, J. Lan, C. McBride, and D. Trolle. Autocalibration of a one-dimensional hydrodynamic-ecological model (DYRESM 4.0-CAEDYM 3.1) using a Monte Carlo approach: simulations of hypoxic events in a polymictic lake. *Geoscientific Model Development*, 11(3):903–913, 2018. URL https://doi.org/10.5194/gmd-11-903-2018. [p235, 236, 246]

G. J. McInerny, M. Chen, R. Freeman, D. Gavaghan, M. Meyer, F. Rowland, D. J. Spiegelhalter, M. Stefaner, G. Tessarolo, and J. Hortal. Information visualisation for science and policy: engaging users and avoiding bias. *Trends in Ecology & Evolution*, 29(3):148–157, 2014. URL https://doi.org/10.1016/j.tree.2014.01.003. [p236, 247]

D. Mironov, E. Heise, E. Kourzeneva, B. Ritter, N. Schneider, and A. Terzhevik. Implementation of the lake parameterisation scheme FLake into the numerical weather prediction model COSMO. *Boreal Environment Research*, 15:218–230, 2010. [p247]

K. Muraoka. *Novel approaches for modelling changes in phytoplankton diversity and lake ecosystem function*. PhD thesis, The University of Waikato, 2019. [p246]

A. Nielsen, K. Bolding, F. Hu, and D. Trolle. An open source QGIS-based workflow for model application and experimentation with aquatic ecosystems. *Environmental Modelling & Software*, 95:358–364, 2017. URL https://doi.org/10.1016/j.envsoft.2017.06.032. [p235]

E. Ofir, J. Heymans, J. Shapiro, M. Goren, E. Spanier, and G. Gal. Predicting the impact of Lake Biomanipulation based on food-web modeling—Lake Kinneret as a case study. *Ecological Modelling*, 348:14–24, 2017. URL https://doi.org/10.1016/j.ecolmodel.2016.12.019. [p237]

D. Özkundakci, D. P. Hamilton, and D. Trolle. Modelling the response of a highly eutrophic lake to reductions in external and internal nutrient loading. *New Zealand Journal of Marine and Freshwater Research*, 45(2):165–185, 2011. URL https://doi.org/10.1080/00288330.2010.548072. [p235]

F. Pianosi, K. Beven, J. Freer, J. W. Hall, J. Rougier, D. B. Stephenson, and T. Wagener. Sensitivity analysis of environmental models: A systematic review with practical workflow. *Environmental Modelling & Software*, 79:214–232, 2016. URL https://doi.org/10.1016/j.envsoft.2016.02.008. [p239]

J. Risbey, M. Kandlikar, and A. Patwardhan. Assessing integrated assessments. *Climatic Change*, 34(3):369–395, 1996. URL https://doi.org/10.1007/BF00139298. [p246]

B. Robson and D. Hamilton. Three-dimensional modelling of a *Microcystis* bloom event in the Swan River estuary, Western Australia. *Ecological Modelling*, 174(1-2):203–222, 2004. URL https://doi.org/10.1016/j.ecolmodel.2004.01.006. [p235, 236, 237]

B. J. Robson and D. P. Hamilton. Summer flow event induces a cyanobacterial bloom in a seasonal Western Australian estuary. *Marine and Freshwater Research*, 54(2):139–151, 2003. URL https://doi.org/10.1071/MF02090. [p235]

B. J. Robson, G. B. Arhonditsis, M. E. Baird, J. Brebion, K. F. Edwards, L. Geoffroy, M.-P. Hébert, V. van Dongen-Vogels, E. M. Jones, C. Kruk, et al. Towards evidence-based parameter values and priors for aquatic ecosystem modelling. *Environmental Modelling & Software*, 100:74–81, 2018. URL https://doi.org/10.1016/j.envsoft.2017.11.018. [p245, 246]

J. Romero, J. Antenucci, and J. Imberger. One-and three-dimensional biogeochemical simulations of two differing reservoirs. *Ecological Modelling*, 174(1-2):143–160, 2004. URL https://doi.org/10.1016/j.ecolmodel.2004.01.005. [p236]

B. Z. Rousso, E. Bertone, R. Stewart, and D. P. Hamilton. A systematic literature review of forecasting and predictive models for cyanobacteria blooms in freshwater lakes. *Water Research*, 182:115959, 2020. URL https://doi.org/10.1016/j.watres.2020.115959. [p235]

M. Scheffer, S. Carpenter, J. A. Foley, C. Folke, and B. Walker. Catastrophic shifts in ecosystems. *Nature*, 413(6856):591–596, 2001. URL https://doi.org/10.1038/35098000. [p245]

S. G. Schladow and D. P. Hamilton. Prediction of water quality in lakes and reservoirs: Part II-Model calibration, sensitivity analysis and application. *Ecological Modelling*, 96(1-3):111–123, 1997. URL https://doi.org/10.1016/S0304-3800(96)00063-4. [p235, 237]

S. Takkouk and X. Casamitjana. Application of the DYRESM–CAEDYM model to the Sau Reservoir situated in Catalonia, Spain. *Desalination and Water Treatment*, 57(27):12453–12466, 2016. URL https://doi.org/10.1080/19443994.2015.1053530. [p235]

R. Wang, J. A. Dearing, P. G. Langdon, E. Zhang, X. Yang, V. Dakos, and M. Scheffer. Flickering gives early warning signals of a critical transition to a eutrophic lake state. *Nature*, 492(7429):419–422, 2012. URL https://doi.org/10.1038/nature11655. [p245]

S. Weinberger and M. Vetter. Using the hydrodynamic model DYRESM based on results of a regional climate model to estimate water temperature changes at Lake Ammersee. *Ecological Modelling*, 244: 38–48, 2012. URL https://doi.org/10.1016/j.ecolmodel.2012.06.016. [p239]

J. T. White, J. E. Doherty, and J. D. Hughes. Quantifying the predictive consequences of model error with linear subspace analysis. *Water Resources Research*, 50(2):1152–1173, 2014. URL https://doi.org/10.1002/2013WR014767. [p235]

Songyan Yu
Griffith University
Australian Rivers Institute and School of Environment and Science
Australia
ORCID: 0000-0001-5765-7060
sunny.yu@griffith.edu.au

Christopher G. McBride
University of Waikato
Environmental Research Institute
New Zealand
ORCID: 0000-0001-5784-9767
cmcbride@waikato.ac.nz

Marieke A. Frassl
Griffith University
Australian Rivers Institute and School of Environment and Science
Australia
ORCID: 0000-0001-8843-8477
marieke.frassl@gmail.com

Matthew R. Hipsey
The University of Western Australia
Aquatic Ecodynamics, UWA School of Agriculture and Environment
Australia
ORCID: 0000-0001-8386-4354
matt.hipsey@uwa.edu.au

David P. Hamilton
Griffith University
Australian Rivers Institute and School of Environment and Science
Australia
ORCID: 0000-0002-9341-8777
david.p.hamilton@griffith.edu.au

SurvMetrics: An R package for Predictive Evaluation Metrics in Survival Analysis

by Hanpu Zhou, Hong Wang, Sizheng Wang and Yi Zou

Abstract Recently, survival models have found vast applications in biostatistics, bioinformatics, reliability engineering, finance and related fields. But there are few R packages focusing on evaluating the predictive power of survival models. This lack of handy software on evaluating survival predictions hinders further applications of survival analysis for practitioners. In this research, we want to fill this gap by providing an "all-in-one" R package which implements most predictive evaluation metrics in survival analysis. In the proposed **SurvMetrics** R package, we implement concordance index for both untied and tied survival data; we give a new calculation process of Brier score and integrated Brier score; we also extend the applicability of integrated absolute error and integrated square error for real data. For models that can output survival time predictions, a simplified metric called mean absolute error is also implemented. In addition, we test the effectiveness of all these metrics on simulated and real survival data sets. The newly developed **SurvMetrics** R package is available on CRAN at https://CRAN.R-project.org/package=SurvMetrics and GitHub at https://github.com/skyee1/SurvMetrics.

1 Introduction

Survival analysis is an important part of biostatistics. It is frequently used to define prognostic indices for mortality or recurrence of a disease, and to study the outcome of treatment (Wang et al., 2019; Wijethilake et al., 2021; Fan et al., 2016; Wiens et al., 2016). Recent decades have witnessed many other applications of survival analysis in various disciplines such as finance, engineering and social sciences (Steyerberg et al., 2010; Bender et al., 2021; Wang et al., 2019). Evaluating the ability to predict future data is one of the most important considerations in the development of these survival models (Wang et al., 2019).

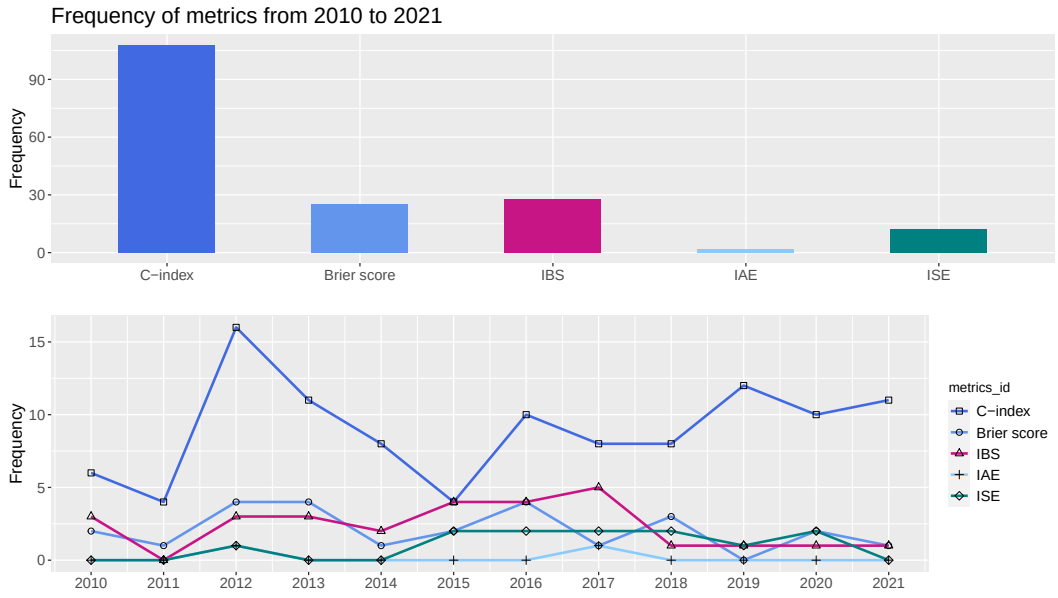


Figure 1: Frequency of survival model evaluation metrics appearing in journals such as "*Annals of Statistics*", "*Biometrika*", "*Journal of the American Statistical Association*", "*Journal of the Royal Statistical Society, Series B*", "*Statistics in Medicine*", "*Artificial Intelligence in Medicine*", "*Lifetime Data Analysis*" from 2010 to 2021.

When evaluating a prediction model, the predictive performance of the survival model is commonly addressed by appropriate measures which quantify the 'distance' or the agreement between the observed and predicted outcomes (Uno et al., 2011; Bylinskii et al., 2018). By investigating the predictive measures frequently used in the statistical literature (top or specialized statistical journals)

in the past decade (from 2010 to 2021) in Figure 1 and Figure 2, we have found that among these 136 research papers, the popular predictive metrics for survival models mainly include: concordance index (C-index), Brier score (BS), integrated Brier score (IBS), integrated absolute error (IAE), integrated square error (ISE) and mean absolute error (MAE) (Harrell et al., 1982; Brier, 1950; Graf et al., 1999; HooraMoradian et al., 2017; Schemper, 1992).

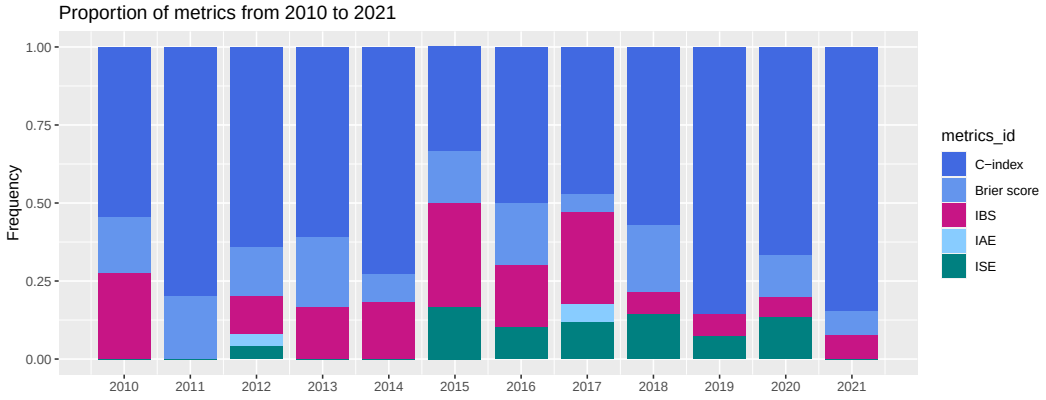


Figure 2: This graphic shows the percentage of metrics appearing from 2010 to 2021 in journals such as "*Annals of Statistics*", "*Biometrika*", "*Journal of the American Statistical Association*", "*Journal of the Royal Statistical Society, Series B*", "*Statistics in Medicine*", "*Artificial Intelligence in Medicine*", "*Lifetime Data Analysis*". It indicates that the use of C-index is steadily increasing and it has become a dominant predictive measure recently. Meanwhile, other metrics do not have an adequate seat in survival prediction practices.

From Figure 1, we can see that the two most frequently used metrics are C-index and IBS, while the other metrics are less used. This may be partly due to the fact that the available predictive metrics are scattered across several R packages, and differences in input data of these tools make it difficult for non-specialists to use or compare the survival models (Schröder et al., 2011; Peters and Hothorn, 2021). Hence, software which provides easy input data formats and includes most popular metrics is desirable (Schröder et al., 2011; Wang et al., 2019).

Figure 2 indicates that the use of C-index is steadily increasing and it has become a dominant predictive measure recently (Jing et al., 2019; Amico et al., 2021). On the one hand, this implies that the lack of handy evaluation metrics tools since other measures are also very important in survival analysis but do not have an adequate seat in survival prediction practices (Nemati et al., 2021; Li et al., 2021; Ensor et al., 2021). On the other hand, it reminds us that providing an accurate C-index measure which can take various situations into account is extremely important (H Ea Gerty and Zheng, 2005; Kang et al., 2015; Zadeh and Schmid, 2020; Zhang et al., 2021).

Different from common classification or regression problems where tools to evaluate predictive performance are abundant, there are few R packages focusing on evaluating the predictive power of survival models (Harrell Jr, 2021; Peters and Hothorn, 2021). To make things worse, most of these available packages have implemented only one or two evaluation metrics and/or some of them often throw errors in real survival problems (Peters and Hothorn, 2021). Take the C-index for an example, if the user wants to compare two samples with the same survival time, this scenario returns NA in **Hmisc**, **survivals**, and **survcomp** packages. However, as described by Ishwaran et al. (2008), "for sample pairs, where observed time $X_i = X_j$ and both are deaths, count 1 if predicted outcomes are tied." This error is mainly due to the fact that the above packages ignore this possible scenario in practice. Omitting such scenarios can result in the loss of information from survival data, especially when the sample size is small with high data collection costs. In **SurvMetrics**, we have implemented the method described by Ishwaran et al. (2008) to cover all tied scenarios. There are also functions available in the **ipred** package to calculate BS and IBS values, but a list of *survfit* objects is required in the calculation process (Peters and Hothorn, 2021). However, for most survival models, only a survival probability vector or matrix is provided, making calculations of such values a challenging problem (Ishwaran et al., 2008). It is not easy for non-specialists to get IBS using the **ipred** package directly on the survival model results.

We argue that an appealing survival evaluation metric tool should satisfy the following two properties. First, the evaluation metrics should not depend on any specific model and they can be applied to evaluating survival models without specifying a specific model form (Schröder et al., 2011). Second, the metrics should be computationally tractable and user has the choice to input only the

fitted model without any additional processing, especially for the non-specialist (Harrell et al., 1982; Graf et al., 1999; HooraMoradian et al., 2017; Ishwaran et al., 2008).

In this research, we are trying to develop a comprehensive and effective R package with the above two properties. In the proposed **SurvMetrics** package, the calculation process of most metrics does not depend on the specific model form and only predicted survival probability vector (or matrix) is needed. This feature is particularly desirable for non-statistical researchers such as clinical and financial practitioners (Ali et al., 2021). To illustrate the effectiveness of our tool, we choose two popular survival models, namely, a semi-parametric Cox model and a non-parametric random survival forest model (RSF) to check and test the provided functionalities (Cox, 1972; Ishwaran et al., 2008).

2 Predictive evaluation metrics in survival analysis

In this section, we will present the survival model evaluation metrics and some implementing details. The following Table 1 presents some popular R packages and the metrics they provide.

Packages	C-index	BS	IBS	IAE	ISE	MAE
survival	✓	✗	✗	✗	✗	✗
Hmisc	✓	✗	✗	✗	✗	✗
survcomp	✓	✓	✓	✗	✗	✗
ipred	✗	✓	✓	✗	✗	✗
mlr	✗	✗	✗	✗	✗	✓
SurvMetrics	✓	✓	✓	✓	✓	✓

Table 1: R packages that are commonly used to evaluate survival predictions.

As can be seen in Table 1, the proposed **SurvMetrics** package implements all popular evaluation metrics, while other packages mostly implement just one or two. With our **SurvMetrics** package, one can compare and evaluate different survival models in a convenient and comprehensive way.

C-index

The most popular evaluation metric in survival predictions is C-index (Harrell et al., 1982; Li et al., 2016; Lee et al., 2018; Devlin and Heller, 2021; Subramanian et al., 2020; Hsu and Lin, 2020; Zadeh and Schmid, 2020; Zhang et al., 2021; Wang and Zhou, 2017; Wang and Li, 2019), which is a generalization of the ROC curve (H Ea Gerty and Zheng, 2005; Obuchowski, 2006; Kang et al., 2015; Li et al., 2016). C-index intends to measure the proportion of predictions of the binary survival status that agree with the true survival status. If the study is concerned with the comparison of survival probability between different samples, for example, which component in the device shows damage earlier, the C-index can be used.

To calculate the values of C-index from different models, **Hmisc**, **survcomp** and **survival** R packages are frequently used in practice (Harrell et al., 1996; Schröder et al., 2011; Therneau, 2021; Harrell Jr, 2021). All the above-mentioned packages in their current versions do not consider the cases of tied survival data, i.e., samples with the same survival time. As shown in Table 3, they return 0 or NA for all cases of survival data tied. However, in practice, cancer patients are usually surveyed annually or monthly after surgery for survival status and with the advent of big data, tied samples are becoming commonplace. Simply ignoring the presence of tie data in calculating C-index will result in evaluation bias among survival models.

Meanwhile, in the calculation of the C-index, not all sample pairs are comparable, e.g., for those pairs whose shorter survival time is censored. So it is necessary to filter the sample pairs (i, j) that can be compared in the calculation, i and j denote any two samples in the data set, respectively. By defining the comparable sample pair in the following way, when $np_{ij} = 1$, all comparable sample pairs described by Ishwaran et al. (2008) can be covered.

In the proposed R package, we will take a similar strategy adopted in Ishwaran et al. (2008). δ_i denotes the survival status of sample i (0 means censoring, 1 means event), Y_i and X_i represent the predicted survival time and the observed survival time, respectively. By defining *csign* and *sign*, we can easily distinguish which sample pairs meet the concordance and which are not.

(a) The comparable sample pair is defined as:

$$np_{ij}\left(X_i, \delta_i, X_j, \delta_j\right) = \max(I\left(X_i \geqslant X_j\right)\delta_j, I\left(X_i \leqslant X_j\right)\delta_i). \tag{1}$$

(b) The complete concordance (CC) is defined as:

$$\mathrm{CC} = \sum_{i,j} I(\mathrm{sign}(Y_i, Y_j) = \mathrm{csign}(X_i, X_j) | np_{ij} = 1), \tag{2}$$

where

$$\mathrm{sign}(Y_i, Y_j) = I\left(Y_i \geqslant Y_j\right) - I\left(Y_i \leqslant Y_j\right) \tag{3}$$

and

$$\mathrm{csign}(X_i, \delta_i, X_j, \delta_j) = I(X_i \geqslant X_j)\delta_j - I\left(X_i \leqslant X_j\right)\delta_i. \tag{4}$$

(c) We derive the partial concordance (PC):

$$\begin{aligned} \mathrm{PC} = & \sum_{i,j} I\left(Y_i = Y_j \mid np_{ij} = 1, X_i \neq X_j\right) \\ & + I\left(Y_i \neq Y_j \mid np_{ij} = 1, X_i = X_j, \delta_i = \delta_j = 1\right) \\ & + I\left(Y_i \geq Y_j \mid np_{ij} = 1, X_i = X_j, \delta_i = 1, \delta_j = 0\right). \end{aligned} \tag{5}$$

(d) The C-index value can be given by:

$$C\text{-}index = \frac{\text{concordance}}{\text{permissible}} = \frac{CC + 0.5 \times PC}{\sum_{i,j} np_{ij}(X_i, \delta_i, X_j, \delta_j)}, \tag{6}$$

From the above, we know C-index lies between 0 and 1 and usually the larger the C-index, the more accurate prediction the model can get.

To further illustrate the proposed `Cindex()` function in **SurvMetrics** with other packages, we first give a toy example in Table 2.

sample	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
time	1	1	2	2	2	2	2	2
status	0	1	1	0	1	1	0	1
predicted	0.2	0.3	0.3	0.3	0.4	0.2	0.4	0.3

Table 2: A simulation data set which can include all the possible sample pairs, where time and status are the observed survival time and survival status of samples $S_1 ... S_8$, respectively, and predicted is the survival probability predicted by survival models.

Based on the above information, different R packages will take different ways to deal with these predictions. We summarize these calculation differences behind these R packages in Table 3.

sample pair	(S_1, S_2)	(S_3, S_5)	(S_3, S_6)	(S_3, S_8)	(S_4, S_5)	(S_5, S_6)	(S_5, S_8)	(S_6, S_8)
Hmisc	0	NA	NA	NA	0	NA	NA	NA
survival	0	NA	NA	NA	0	NA	NA	NA
survcomp	NA	NA	NA	NA	NA	NA	NA	NA
SurvMetrics	0.5	0.5	0.5	1	0.5	0.5	0.5	0.5

Table 3: Differences of C-index calculated by the **Hmisc** package, the **survival** package, the **survcomp** package and the **SurvMetrics** package.

From the above, we can find that when ignoring part or all of the information from tied sample pairs, the C-index values obtained from **Hmisc** and other two packages cannot reflect the predictive powers of survival models in a true manner. In this case, one can turn to the `Cindex()` function in **SurvMetrics** package for help.

BS

BS is another commonly used metric in survival analysis (Steyerberg et al., 2010; Chicco et al., 2021; Imani et al., 2019; Pakbin et al., 2018), which measures the mean squared error between the observed progression status and the predicted survival probability at time t^*. Thus, a lower BS usually indicates a better prediction model. BS focuses on the residuals between the predicted survival probability and the survival status at a fixed time point. If the study is concerned with the prediction probability of a model at a fixed time point, BS can be chosen as the metric. For example, the 10-year average survival probability of cancer patients.

In the **SurvMetrics** R package we will take the most classical method to calculate the BS by weighting the prediction residuals with inverse probability censoring weights (IPCW) proposed by Graf et al. (1999):

$$BS(t^*) = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{(\hat{S}(t^*|z_i))^2}{\hat{G}(X_i)}\cdot I(X_i < t^*, \delta_i = 1) + \frac{(1-\hat{S}(t^*|z_i))^2}{\hat{G}(t^*)}\cdot I(X_i \geq t^*)\right], \tag{7}$$

where t^* is the time point at which BS is to be calculated, N is the sample size, z_i is the covariates of instance i, $\hat{S}(\cdot)$ is the survival function predicted by the model, $\hat{G}(\cdot)$ denotes the weight for the instance which is estimated by the KM(Kaplan–Meier) estimator of the censoring distribution.

According to the definition of BS, its value depends on the selection of t^* provided by the user and different choices of t^* will always lead to different BS values (Chew et al., 2001; Li et al., 2021; Zhu and Kosorok, 2012; Ji et al., 2020). In the `Brier()` function of our **SurvMetrics** R package, t^* is set to median survival time as default.

IBS

As we can see above, BS depends partly on the choices of a user-specified time point. It makes the comparison between models somewhat difficult when we want to know an average prediction performance over all prediction times. In practice, the integral of BS or IBS, which does not depend on the selection of one time point, is more widely used when we are interested in the entire time interval (Zhu and Kosorok, 2012; Periáñez et al., 2016; Bertens et al., 2017). IBS is more concerned with the residuals at all observed time points. When the time of interest is no longer a specific time point, IBS can provide more information than BS. For example, the probability of survival of a cancer patient for each year after the disease.

The definition of IBS is straightforward:

$$IBS = \frac{1}{\max\limits_{i}(X_i)}\int_0^{\max\limits_{i}(X_i)} BS(t)dt. \tag{8}$$

But the calculation of IBS using `sbrier()` function from the **ipred** package does not always go smoothly (Peters and Hothorn, 2021) . If a list of `survfit` objects is incorrectly specified, then the package currently returns a particular error:

`'Error in switch(ptype,survfit = { : EXPR must be a length 1 vector} '`

In our experience, this kind of error is common, especially for the non-specialists (Peters and Hothorn, 2021).

In the **SurvMetrics** package, we will make life easier for non-specialists. Users only need to input survival time, survival status, the predicted survival probability matrix and the range of integration to the `IBS()` function, and our program will take care of all the rest work and give a correct output. Similar to BS, a smaller IBS value usually implies a more accurate survival model.

IAE and ISE

Another two evaluation metrics, namely, IAE and ISE are also occasionally used to compare the difference between the estimated survival function $\hat{S}(\cdot)$ and the true survival function $S(\cdot)$ in terms of L_1 and L_2 paradigms, respectively. When the purpose of the model is to approximate the theoretical distribution function, IAE and ISE can be selected as the metrics. HooraMoradian et al. (2017); Zou et al. (2021). The IAE and ISE can be defined as:

$$IAE = \int_t |S(t) - \hat{S}(t|X)|dt \tag{9}$$

and

$$ISE = \int_t (S(t) - \hat{S}(t|X))^2 dt, \tag{10}$$

where X is the covariate of the training set.

Since the true survival functions are usually unknown beforehand, traditional IAE and ISE methods are only applicable to simulation scenarios. Here, we propose to approximate $S(t)$ using the non-parametric KM estimator in the `IAEISE()` function and this makes IAE and ISE also suitable for real data study. Similar to IBS, a smaller IAE and ISE values lead to a more accurate model.

MAE

The last evaluation metric presented here is MAE (Schemper, 1992), which can be used to measure the residuals of observed survival time versus predicted survival time, for example, to predict the time to damage of the device. MAE is a better choice than BS if predicted survival time is a model's output. The definition of MAE is shown below:

$$MAE = \frac{1}{n}\sum_{i=1}^{N}\left(\delta_i \left|Y_i - X_i\right|\right), \tag{11}$$

where n is the event sample size.

MAE only estimates the average absolute error between the predicted survival time and the true survival time in the uncensored samples and is rarely used in practice. For consistence, we also provide this metric in the `MAE()` function. Similar to MSE, the lower the MAE, the higher the accuracy of prediction.

3 Simulations and examples

In this section, we will use some simulated and real survival data sets to illustrate the effectiveness of the metrics provided in the **SurvMetrics** package.

The performance of SurvMetrics on simulation data sets

First, Cox proportional hazard models will be fitted on three simulated scenarios which are very similar to Settings 1–4 by Steingrimsson et al. (2019). Scenario1 satisfies the proportional hazards assumption and others violate it.

Scenario1: This data set is created using N independent observations, where the covariate vector $(W_1, ..., W_p)$ is multivariate normal with mean zero and a covariance matrix having elements (i, j) equal to $0.9^{|i-j|}$. Survival times are simulated from an exponential distribution with mean $\mu = e^{0.1\sum_{i=[p/2]+1}^{p} W_i}$ (i.e., a proportional hazards model) and the censoring distribution is exponential with mean c_{mean} which is chosen to control the censoring rate. Here, $[x]$ denotes the largest integer no more than x.

Scenario2: This data set is created using N independent observations where the covariates $(W_1, ..., W_p)$ are multivariate normal with mean zero and a covariance matrix having elements (i, j) equal to $0.75^{|i-j|}$. Survival times are gamma distributed with shape parameter $\mu = 0.5 + 0.3|\sum_{i=[2p/5]}^{[3p/5]} W_i|$ and scale parameter 2. Censoring times are uniform on $[0, u_{max}]$, and u_{max} is chosen to control the censoring rate. Here, the proportional hazards assumption is violated.

Scenario3: This data set is created using N independent observations where the covariates $(W_1, ..., W_p)$ are multivariate normal with mean zero and a covariance matrix having elements (i, j) equal to $0.75^{|i-j|}$. Survival times are simulated according to a log-normal distribution with mean $\mu = 0.1|\sum_{i=1}^{[p/5]} W_i| + 0.1|\sum_{i=[4p/5]}^{p} W_i|$. Censoring times are log-normal with mean $\mu + c_{step}$ and scale parameter one, where c_{step} is chosen to control the censoring rate. Here, the underlying censoring distribution depends on covariates and the proportional hazards assumption is also violated.

From Figure 3, one may see that the prediction performance of the Cox model keeps decreasing in three scenarios as expected. However, the difference in terms of C-index and BS are not significant for Scenario2 and Scenario3 while in terms of IBS, IAE and ISE, a sharp difference and a clear trend can be observed.

As we know, C-index evaluates the model from a ranking perspective in a rough matter and BS only considers a fixed time point. Both metrics may not distinguish models described above or in other complex settings. In this case, evaluation metrics which consider model performance in a global way, such as IBS, IAE, and ISE, may be useful alternatives. This also justifies why a package such as **SurvMetrics** that can evaluate survival models from multiple perspectives is strongly needed in practice.

An example of SurvMetrics

In this section, a kidney dataset from the **survival** R package is used to illustrate the usage of the **SurvMetrics** package using popular survival models widely used in biostatics and biomedical study.

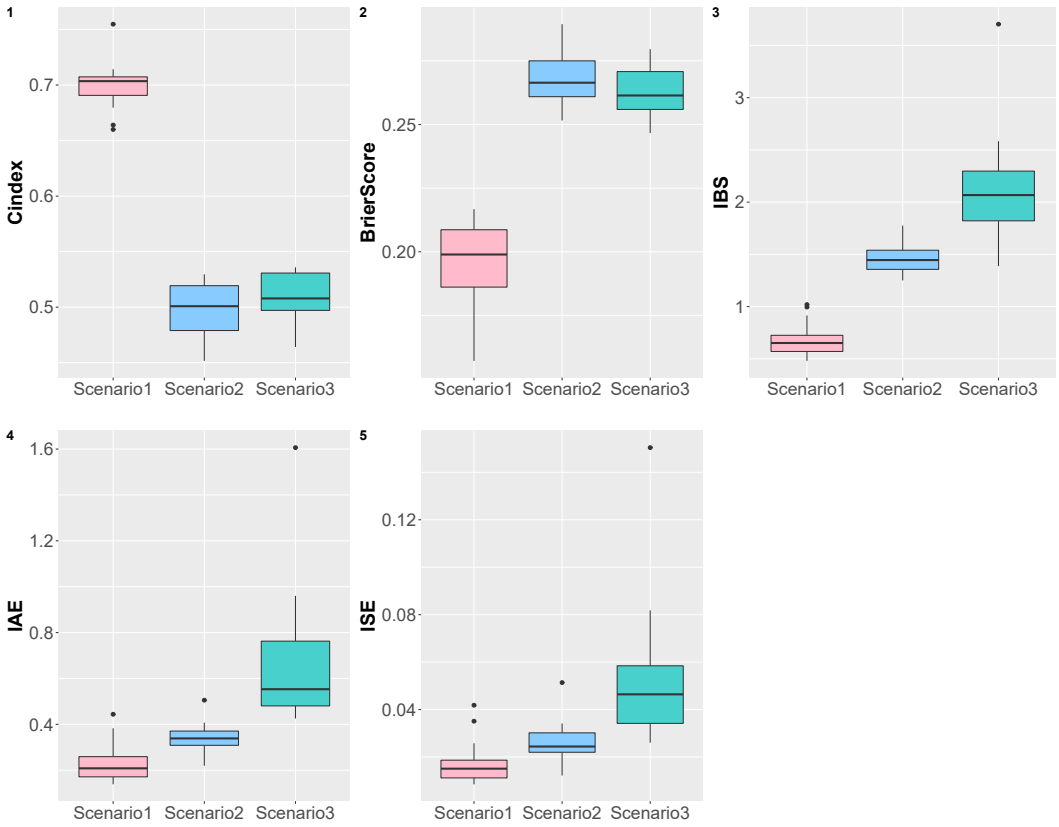


Figure 3: This graphic shows the Cox model prediction accuracy based on three different scenarios via 5 evaluation metrics by the **SurvMetrics** package. The sample size is 300, the dimension is 25, and control the censoring rate about 30%. The prediction performance of the Cox model keeps decreasing in three scenarios, which satisfies the data generated methods. However, in some complex situations where C-index and BS cannot distinguish models, IBS, IAE, and ISE, may be useful alternatives.

In the following, we first cut the kidney dataset into a training set and a testing set. Then, two popular models, namely, Cox model and random survival forest model are constructed based on the training set to show how different functions are used in **SurvMetrics**. Finally, we will show how to evaluate the predictive performance of these two models on the testing set using the proposed **SurvMetrics** R package. In the latest version of **SurvMetrics**, user has the choice to input only the standard survival models and testing sets. Meanwhile, it is still useful to deal with predicted survival probabilities from non-standard models. The following example will contain these two input forms.

The corresponding R code is provided here:

```
#1. data preparation
library(survival)        # to fit a Cox model
library(randomForestSRC) # to fit an RSF model
library(SurvMetrics) # to get all the metrics
library(pec)         # to make predictions based on Cox model
set.seed(1)
mydata <- kidney[, -1]
train_index <- sample(1:nrow(mydata), 0.7 * nrow(mydata))
train_data <- mydata[train_index, ]
test_data <- mydata[-train_index, ]

#2. fit the RSF model and Cox model to predict the testing set
#2.1 RSF model
fit_rsf <- rfsrc(Surv(time,status)~., data = train_data)  #fit the RSF model
distime <- fit_rsf$time.interest  #get the survival time of events
med_index <- median(1:length(distime))  #the index of median survival time of events
mat_rsf <- predict(fit_rsf, test_data)$survival  #get the survival probability matrix
vec_rsf <- mat_rsf[ ,med_index]  #median survival probability of all samples
```

```
#2.2 Cox model
fit_cox <- coxph(Surv(time,status)~., data = train_data, x = TRUE)  #fit the Cox model
mat_cox <- predictSurvProb(fit_cox, test_data, distime)  #get the survival probability matrix
vec_cox <- mat_cox[ ,med_index]

#3. get all the metrics by SurvMetrics
#3.1 CI BS IBS IAE ISE based on RSF model: standard model input methods
Cindex_rsf <- Cindex(fit_rsf, test_data)
BS_rsf <- Brier(fit_rsf, test_data, distime[med_index])
IBS_rsf <- IBS(fit_rsf, test_data)
IAE_rsf <- IAEISE(fit_rsf, test_data)[1]
ISE_rsf <- IAEISE(fit_rsf, test_data)[2]

#CI BS IBS IAE ISE based on Cox model: standard model input methods
Cindex_cox <- Cindex(fit_cox, test_data)
BS_cox <- Brier(fit_cox, test_data, distime[med_index])
IBS_cox <- IBS(fit_cox, test_data)
IAE_cox <- IAEISE(fit_cox, test_data)[1]
ISE_cox <- IAEISE(fit_cox, test_data)[2]

#3.2 CI BS IBS IAE ISE based on RSF model: Non-standard model input methods
times <- test_data$time
status <- test_data$status
Cindex_rsf <- Cindex(Surv(times, status), vec_rsf)
BS_rsf <- Brier(Surv(times, status), vec_rsf, distime[med_index])
IBS_rsf <- IBS(Surv(times, status), mat_rsf, distime)  # distime can be replaced by range(distime)
IAE_rsf <- IAEISE(Surv(times, status), mat_rsf, distime)[1]
ISE_rsf <- IAEISE(Surv(times, status), mat_rsf, distime)[2]

#CI BS IBS IAE ISE based on Cox model: Non-standard model input methods
Cindex_cox <- Cindex(Surv(times, status), vec_cox)
BS_cox <- Brier(Surv(times, status), vec_cox, distime[med_index])
IBS_cox <- IBS(Surv(times, status), mat_cox, distime)
IAE_cox <- IAEISE(Surv(times, status), mat_cox, distime)[1]
ISE_cox <- IAEISE(Surv(times, status), mat_cox, distime)[2]
```

models	C-index	BS	IBS	IAE	ISE
Cox	0.751185	0.18133	0.08842	77.90947	19.65131
RSF	0.729858	0.221523	0.10920	105.6936	28.80941

Table 4: This table shows the prediction accuracy by using **SurvMetrics** package to compare the Cox and RSF models based on the kidney data set from **survival** R package. In this case, the values of C-index are close, and the values of IAE and ISE are far different. When using C-index alone is hard to differentiate the predictive power of the two models, alternative measures provided in **SurvMetrics** may help give a more accurate result.

The results presented in Table 4 and Figure 4 show that using C-index alone is hard to differentiate the predictive power of the two models. In this case, alternative measures provided in the proposed package may help if further evaluation is needed. This result is also consistent with the simulated scenario mentioned above.

4 Summary

Assessing the predictive performance of survival models is complex due to the lack of standards regarding the best criterion to use in survival analysis. The available metrics are scatter across different R packages that use heterogeneous interfaces, which makes it difficult for the non-specialist to use or compare the performance of various survival models.

In this paper, we try to fill the gap by providing an "all-in-one" R package called **SurvMetrics** which provides a uniform interface to an extensive set of performance assessment and statistical comparison methods. Practitioners can easily implement comparative studies and identify the best model(s) using this package. In the current version of the **SurvMetrics** package, six evaluation metrics

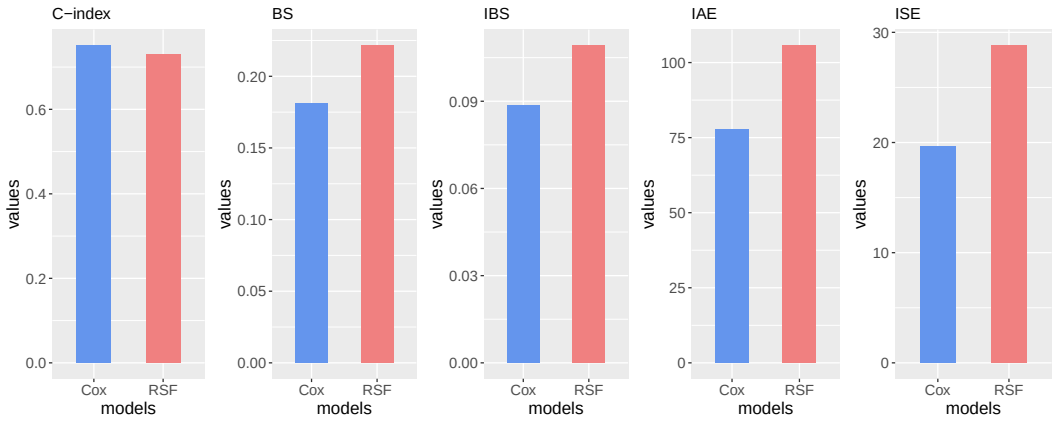


Figure 4: This graphic shows the prediction accuracy by using **SurvMetrics** package to compare the Cox and RSF models based on the kidney data set from **survival** R package. In this case, using IBS, IAE and ISE can give a more clear conclusion that Cox model performs better.

are present. Evaluation metrics used in more complicated settings such as time-dependent AUC for joint modelling of longitudinal and survival data, and concordance index for competitive risks are still being developed and will be added to the **SurvMetrics** package in the future. Meanwhile, we are also working to provide a more user-friendly interface to facilitate both statistical and non-statistical clinical research workers in evaluating survival models.

5 Acknowledgments

This research is support in part by National Statistical Scientific Research Project of China (No.2022LZ28), Changsha Municipal Natural Science Foundation (No.kq2202080), The Open Research Fund from the Guangdong Provincial Key Laboratory of Big Data Computing, The Chinese University of Hong Kong, Shenzhen(No. B10120210117-OF04) and the Postgraduate Scientific Research Innovation Project of Hunan Province, China (CX20210155).

Bibliography

M. Ali, M. Berrendorf, C. T. Hoyt, L. Vermue, S. Sharifzadeh, V. Tresp, and J. Lehmann. Pykeen 1.0: A python library for training and evaluating knowledge graph embeddings. *Journal of Machine Learning Research*, 22(82):1–6, 2021. [p3]

M. Amico, I. Van Keilegom, and B. Han. Assessing cure status prediction from survival data using receiver operating characteristic curves. *Biometrika*, 108(3):727–740, 2021. [p2]

A. Bender, D. Rügamer, F. Scheipl, and B. Bischl. A general machine learning framework for survival analysis. In F. Hutter, K. Kersting, J. Lijffijt, and I. Valera, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 158–173, Cham, 2021. Springer International Publishing. ISBN 978-3-030-67664-3. [p1]

P. Bertens, A. Guitart, and Á. Periáñez. Games and big data: A scalable multi-dimensional churn prediction model. In *2017 IEEE conference on computational intelligence and games (CIG)*, pages 33–36. IEEE, 2017. [p5]

G. W. Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78, 1950. [p2]

Z. Bylinskii, T. Judd, A. Oliva, A. Torralba, and F. Durand. What do different evaluation metrics tell us about saliency models? *IEEE transactions on pattern analysis and machine intelligence*, 41(3):740–757, 2018. [p1]

D. P. Chew, D. L. Bhatt, M. A. Robbins, M. S. Penn, J. P. Schneider, M. S. Lauer, E. J. Topol, and S. G. Ellis. Incremental prognostic value of elevated baseline c-reactive protein among established markers of risk in percutaneous coronary intervention. *Circulation*, 104(9):992–997, 2001. [p5]

D. Chicco, M. J. Warrens, and G. Jurman. The matthews correlation coefficient mcc is more informative than cohen's Kappa and Brier score in binary classification assessment. *IEEE Access*, 2021. [p4]

D. R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(2):187–202, 1972. [p3]

S. M. Devlin and G. Heller. Concordance probability as a meaningful contrast across disparate survival times. *Statistical methods in medical research*, 30(3):816–825, 2021. [p3]

J. Ensor, K. I. Snell, T. P. Debray, P. C. Lambert, M. P. Look, M. A. Mamas, K. G. Moons, and R. D. Riley. Individual participant data meta-analysis for external validation, recalibration, and updating of a flexible parametric prognostic model. *Statistics in Medicine*, 2021. [p2]

J. Fan, Y. Wu, M. Yuan, D. Page, J. Liu, I. M. Ong, P. Peissig, and E. Burnside. Structure-leveraged methods in breast cancer risk prediction. *Journal of Machine Learning Research*, 17(1):2956–2970, 2016. [p1]

E. Graf, C. Schmoor, W. Sauerbrei, and M. Schumacher. Assessment and comparison of prognostic classification schemes for survival data. *Statistics in Medicine*, 18(17-18):2529–2545, 1999. [p2, 3, 5]

P. J. H Ea Gerty and Y. Zheng. Survival model predictive accuracy and ROC curves. *Biometrics*, 61(1): 92–105, 2005. [p2, 3]

F. Harrell, K. Lee, and D. MARK. Multivariable prognostic models: Issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors. *Statistics in Medicine*, 15: 361–387, 02 1996. [p3]

F. E. J. Harrell, R. M. Califf, D. B. Pryor, K. L. Lee, and R. A. Rosati. Evaluating the yield of medical tests. *JAMA The Journal of the American Medical Association*, 247(18):2543–2546, 1982. [p2, 3]

F. E. Harrell Jr. *Hmisc: Harrell Miscellaneous*, 2021. URL https://CRAN.R-project.org/package=Hmisc. R package version 4.6-0. [p2, 3]

HooraMoradian, DenisLarocque, and FranoisBellavance. L1 splitting rules in survival forests. *Lifetime Data Analysis*, 23(4):671–691, 2017. [p2, 3, 5]

T.-C. Hsu and C. Lin. Generative adversarial networks for robust breast cancer prognosis prediction with limited data size. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 5669–5672. IEEE, 2020. [p3]

F. Imani, R. Chen, C. Tucker, and H. Yang. Random forest modeling for survival analysis of cancer recurrences. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 399–404. IEEE, 2019. [p4]

H. Ishwaran, U. B. Kogalur, E. H. Blackstone, and M. S. Lauer. Random survival forests. *The annals of applied statistics*, 2(3):841–860, 2008. [p2, 3]

G.-W. Ji, F.-P. Zhu, Q. Xu, K. Wang, M.-Y. Wu, W.-W. Tang, X.-C. Li, and X.-H. Wang. Radiomic features at contrast-enhanced ct predict recurrence in early stage hepatocellular carcinoma: a multi-institutional study. *Radiology*, 294(3):568–579, 2020. [p5]

B. Jing, T. Zhang, Z. Wang, Y. Jin, and C. Li. A deep survival analysis method based on ranking. *Artificial intelligence in medicine*, 98:1–9, 2019. [p2]

L. Kang, W. Chen, N. A. Petrick, and B. D. Gallas. Comparing two correlated c indices with right-censored survival outcome: a one-shot nonparametric approach. *Statistics in Medicine*, 34(4), 2015. [p2, 3]

C. Lee, W. Zame, J. Yoon, and M. van der Schaar. Deephit: A deep learning approach to survival analysis with competing risks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. [p3]

Y. Li, J. Wang, J. Ye, and C. K. Reddy. A multi-task learning formulation for survival analysis. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1715–1724, 2016. [p3]

Z. Li, L. Yuan, C. Zhang, J. Sun, Z. Wang, Y. Wang, X. Hao, F. Gao, and X. Jiang. A novel prognostic scoring system of intrahepatic cholangiocarcinoma with machine learning basing on real-world data. *Frontiers in Oncology*, 10:3146, 2021. ISSN 2234-943X. doi: 10.3389/fonc.2020.576901. [p2, 5]

M. Nemati, H. Zhang, M. Sloma, D. Bekbolsynov, H. Wang, S. Stepkowski, and K. S. Xu. Predicting kidney transplant survival using multiple feature representations for HLAS. In *International Conference on Artificial Intelligence in Medicine*, pages 51–60. Springer, 2021. [p2]

N. A. Obuchowski. An ROC-type measure of diagnostic accuracy when the gold standard is continuous-scale. *Statistics in Medicine*, 25(3):481–493, 2006. [p3]

A. Pakbin, P. Rafi, N. Hurley, W. Schulz, M. H. Krumholz, and J. B. Mortazavi. Prediction of icu readmissions using data at patient discharge. In *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 4932–4935. IEEE, 2018. [p4]

Á. Periáñez, A. Saas, A. Guitart, and C. Magne. Churn prediction in mobile social games: Towards a complete assessment using survival ensembles. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 564–573. IEEE, 2016. [p5]

A. Peters and T. Hothorn. *ipred: Improved Predictors*, 2021. URL https://CRAN.R-project.org/package=ipred. R package version 0.9-12. [p2, 5]

M. Schemper. The explained variation in proportional hazards regression. *Biometrika*, 79(1):202–204, 1992. [p2, 6]

M. S. Schröder, A. C. Culhane, J. Quackenbush, and B. Haibe-Kains. survcomp: an R/bioconductor package for performance assessment and comparison of survival models. *Bioinformatics*, 27(22): 3206–3208, 2011. [p2, 3]

J. A. Steingrimsson, L. Diao, and R. L. Strawderman. Censoring unbiased regression trees and ensembles. *Journal of the American Statistical Association*, 114, 2019. [p6]

E. W. Steyerberg, A. J. Vickers, N. R. Cook, T. A. Gerds, M. Gonen, N. Obuchowski, M. Pencina, and M. W. Kattan. Assessing the performance of prediction models: a framework for traditional and novel measures. *Epidemiology*, 21(1):128–138, 2010. [p1, 4]

V. Subramanian, M. N. Do, and T. Syeda-Mahmood. Multimodal fusion of imaging and genomics for lung cancer recurrence prediction. In *2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI)*, pages 804–808. IEEE, 2020. [p3]

T. M. Therneau. *A Package for Survival Analysis in R*, 2021. URL https://CRAN.R-project.org/package=survival. R package version 3.2-13. [p3]

H. Uno, T. Cai, M. Pencina, R. D'Agostino, and L. Wei. On the c-statistics for evaluating overall adequacy of risk prediction procedures with censored survival data. *Statistics in medicine*, 30: 1105–17, 05 2011. [p1]

H. Wang and G. Li. Extreme learning machine cox model for high-dimensional survival analysis. *Statistics in medicine*, 38(12):2139—2156, 2019. [p3]

H. Wang and L. Zhou. Random survival forest with space extensions for censored data. *Artificial Intelligence in Medicine*, 79:52–61, 2017. [p3]

P. Wang, Y. Li, and C. K. Reddy. Machine learning for survival analysis: A survey. *ACM Computing Surveys (CSUR)*, 51(6):1–36, 2019. [p1, 2]

J. Wiens, J. Guttag, and E. Horvitz. Patient risk stratification with time-varying parameters: a multitask learning approach. *Journal of Machine Learning Research*, 17(1):2797–2819, 2016. [p1]

N. Wijethilake, D. Meedeniya, C. Chitraranjan, I. Perera, and H. Ren. Glioma survival analysis empowered with data engineering—a survey. *IEEE Access*, 9:43168–43191, 2021. [p1]

S. G. Zadeh and M. Schmid. Bias in cross-entropy-based training of deep survival networks. *IEEE transactions on pattern analysis and machine intelligence*, 2020. [p2, 3]

L. Zhang, D. Dong, L. Zhong, C. Li, C. Hu, X. Yang, Z. Liu, R. Wang, J. Zhou, and J. Tian. Multi-focus network to decode imaging phenotype for overall survival prediction of gastric cancer patients. *IEEE Journal of Biomedical and Health Informatics*, 2021. [p2, 3]

R. Zhu and M. R. Kosorok. Recursively imputed survival trees. *Journal of the American Statistical Association*, 107(497):331–340, 2012. [p5]

Y. Zou, G. Fan, and R. Zhang. Integrated square error of hazard rate estimation for survival data with missing censoring indicators. *Journal of Systems Science and Complexity*, 34(2):735–758, 2021. [p5]

Hanpu Zhou
Central South University
School of Mathematics & Statistics
Changsha, Hunan Provinice,410075
China
zhouhanpu@csu.edu.cn

*Hong Wang**
Central South University
School of Mathematics & Statistics
Changsha, Hunan Provinice,410075
China
(Corresponding Author)
wh@csu.edu.cn

Sizheng Wang
Central South University
School of Mathematics & Statistics
Changsha, Hunan Provinice,410075
China
wszhchina@163.com

Yi Zou
Central South University
School of Mathematics & Statistics
Changsha, Hunan Provinice,410075
China
zy6868@csu.edu.cn

Limitations in Detecting Multicollinearity due to Scaling Issues in the mcvis Package

by Roman Salmeron Gomez, Catalina B. Garcia Garcia, Ainara Rodriguez Sanchez, and Claudia Garcia Garcia

Abstract Transformation of the observed data is a very common practice when a troubling degree of near multicollinearity is detected in a linear regression model. However, it is important to take into account that these transformations may affect the detection of this problem, so they should not be performed systematically. In this paper we analyze the transformation of the data when applying the R package mcvis, showing that it only detects essential near multicollinearity when the *studentise* transformation is performed.

1 Introduction

Given the model $\mathbf{Y} = \mathbf{X} \cdot \boldsymbol{\beta} + \mathbf{u}$ for n observations and p independent variables where $\mathbf{Y}$ is a vector that contains the observations of the dependent variables, $\mathbf{X} = [\mathbf{1}\, \mathbf{X}_2 \ldots \mathbf{X}_p]$ is a matrix whose columns contain the observations of the independent variables (where the first column is a vector of ones representing the intercept) and $\mathbf{u}$ represents the spherical random disturbance, the existence of linear relationships between the independent variables of a model is known as multicollinearity. It is well-known that a high degree of multicollinearity can affect the analysis of a linear regression model. In this case, it is said that the multicollinearity is troubling (Novales 1988; Ramanathan 2002; Wooldridge 2008; Gujarati 2010). It is also interesting to note the distinction made, for example, by Marquardt (1980) or Snee and Marquardt (1984), between essential (near-linear relationship between at least two independent variables excluding the intercept) and non-essential multicollinearity (near-linear relationship between the intercept and at least one of the remaining independent variables).

Note that the detection process is key to determining which tool is best suited to mitigation of the problem: for example, ridge regression of Hoerl and Kennard (1970) and Hoerl and Kennard (1970); LASSO of Tibshirani (1996) or elastic net of Zou and Hastie (2005), among others.

The most commonly applied measures to detect whether the degree of multicollinearity is troubling are the following:

- The Variation Inflation Factor (VIF) that is obtained with the following expression $VIF(j) = \frac{1}{1-R_j^2}$, $j = 2, \ldots, p$, where R_j^2 is the coefficient of determination of the auxiliary regression $\mathbf{X}_j = \mathbf{X}_{-j} \cdot \boldsymbol{\alpha} + \mathbf{w}$ with $\mathbf{X}_{-j}$ being the matrix obtained when the independent variable $\mathbf{X}_j$ is eliminated from matrix $\mathbf{X}$. It is considered that values of VIF higher than 10 indicate that the degree of multicollinearity is troubling (see, for example, Marquardt (1970) or O'Brien (2007)). R. Salmerón, García, and García (2018) and Román Salmerón, Rodríguez, and García (2020) showed that the VIF is not an appropriate measure to detect linear relations between the intercept and other explanatory variables (non-essential collinearity).
- The Condition Number (CN) that is obtained with the following expression $CN(\mathbf{X}) = \sqrt{\frac{\mu_{max}}{\mu_{min}}}$ where μ_{max} and μ_{min} are the maximum and minimum eigenvalue of matrix $\mathbf{X}^t\mathbf{X}$. To obtain the eigenvalues, the matrix $\mathbf{X}$ has to be previously transformed in order to ensure that all its columns present unit length (see R. Salmerón, García, and García (2018) for more details about this transformation). Values lower than 20 imply light collinearity, between 20 and 30 moderate collinearity, while values higher than 30 imply strong collinearity (see, for example, D. A. Belsley, Kuh, and Welsch (1980) and D. Belsley (1991)). Another alternative is to calculate the CN with and without the intercept with the goal of analyzing the contribution of the intercept.

Another set of measures to detect the existence of troubling multicollinearity are the matrix of simple linear correlations between the independent variables, $\mathbf{R} = (cor(X_l, X_m))_{l,m=2,\ldots,p}$ and its determinant, $|\mathbf{R}|$. C. García, Salmerón, and García (2019) show that values for the coefficient of simple correlation between the independent variables higher than $\sqrt{0.9}$ and determinant lower than $0.1013 + 0.00008626 \cdot n - 0.01384 \cdot p$ indicate a troubling degree of multicollinearity (see Román Salmerón, García, and García (2021a) or Román Salmerón, García, and García (2021b) for more details). The first value differs strongly from the threshold normally proposed equal to 0.7 to indicate a problem of near collinearity (see, for example, Halkos and Tsilika (2018)).

Also useful to use the coefficient of variation (CV), values less than 0.1002506 indicate the existence of troubling multicollinearity (see Román Salmerón, Rodríguez, and García (2020) for more details).

J. García et al. (2016) and R. Salmerón, García, and García (2020) showed that the VIF is invariant to origin and scale changes, which is the same as saying that model $\mathbf{Y} = \mathbf{X} \cdot \boldsymbol{\beta} + \mathbf{u}$ and the model $\mathbf{Y} = \mathbf{x} \cdot \boldsymbol{\beta} + \mathbf{u}$ present the same VIF, where $\mathbf{x} = [\mathbf{x}_1\ \mathbf{x}_2 \dots \mathbf{x}_p]$ with $\mathbf{x}_i = \frac{\mathbf{X}_i - a_i}{b_i}$ for $a_i \in \mathbb{R}$, $b_i > 0$ and $i = 1, \dots, p$. Note that if $a_i = \overline{\mathbf{X}}_i$, $\mathbf{x}_1$ is a vector of zeros, i.e. the intercept disappears from the model. Instead, R. Salmerón et al. (2018) showed that the CN is not invariant to origin and scale changes, meaning that the two previous models present different CNs. This fact implies that models $\mathbf{Y} = \mathbf{X} \cdot \boldsymbol{\beta} + \mathbf{u}$ and $\mathbf{Y} = \mathbf{x} \cdot \boldsymbol{\beta} + \mathbf{u}$ present different eigenvalues.

Consequently, transforming the data in a linear regression model may affect the detection of the multicollinearity problem depending on the diagnostic measure. Furthermore, note that this sensitivity to scaling is due to the fact that there are certain transformations (such as data centering) that mitigate the multicollinearity problem, so that the dependence or otherwise on scaling simply highlights the capacity/incapacity of each measure to detect this reduction of the degree of near multicollinearity.

Therefore, when transforming the data in a linear regression model and analyzing whether the degree of multicollinearity is of concern or not, it is necessary to be clear whether the measure used to detect it is affected by the transformation and whether it is capable of detecting the two types of near multicollinearity mentioned (essential and non-essential). Thus, in this paper we analyze the MC index recently presented in Lin, Wang, and Mueller (2020).

First, this paper will briefly review the MC index. In order to show that the MC index depends on the transformation of the data and its inability to detect non-essential multicollinearity, we present two simulations with a troubling degree of essential and non-essential multicollinearity, respectively, and a third simulation where the degree of multicollinearity is not troubling. For all these cases, we calculate the different measures to detect multicollinearity commented on the introduction together with the MC index. Two empirical applications recently applied in the scientific literature are also presented. After a discussion of the results we propose a scatter plot between the VIF and CV to detect which variables are the cause of the troubling degree of multicollinearity and the kind of multicollinearity (essential or non-essential) existing in the model. Finally, the main conclusions of the paper are summarized.

2 Background: MC index

The MC index presented in Lin, Wang, and Mueller (2020) is based on the existing relation between the VIFs ant the inverse of the eigenvalues of the matrix $\mathbf{Z}^t\mathbf{Z}$ where $\mathbf{Z}$ represents the standardized matrix of $\mathbf{X}$. This is to say, is the matrix $\mathbf{x}$ mentioned in the introduction when for all i it is obtained that $a_i = \overline{\mathbf{X}}_i$ and $b_i = \sqrt{n \cdot var\left(\mathbf{X}_i\right)}$ where $var\left(\mathbf{X}_i\right)$ represents the variance of $\mathbf{X}_i$. More precisely, taking into account the fact that in the main diagonal of $\left(\mathbf{Z}^t\mathbf{Z}\right)^{-1}$ we find the VIFs (for standardized data), it is possible to establish the following relation:

$$\begin{pmatrix} VIF(2) \\ \vdots \\ VIF(p) \end{pmatrix} = \mathbf{A} \cdot \begin{pmatrix} \frac{1}{\mu_2} \\ \vdots \\ \frac{1}{\mu_p} \end{pmatrix}, \tag{1}$$

where $\mathbf{A}$ is a matrix that depends on the eigenvalues of $\mathbf{Z}^t\mathbf{Z}$ and μ_p is the maximum eigenvalue of this matrix.

From this relationship, resampling and obtaining the regression of $1/\mu_p$ as a function of the VIFs, Lin, Wang, and Mueller (2020) proposed the use of the t-statistics to conclude which variable contributes the most to this relationship, thus identifying which variables are responsible for the degree of approximate multicollinearity in the model. These authors defined the MC index as *an index from zero to one, larger values indicating greater contribution of the variable i in explaining the observed severity of multicollinearity*.

Taking into account that the calculation of the MC index is based on the relation established between the VIFs and the inverse of the smallest eigenvalue, it seems logical to consider that the transformation of the data may affect the calculation of this measure. Thus, it is possible to conclude:

- If the MC index is calculated with a transformation of the data that leads to the elimination of the intercept, the non-essential multicollinearity will be ignored.
- Due to the fact that MC index is based on the VIF, it should inherit its inability to detect the non-essential multicollinearity.

In conclusion, regardless of whether the data is transformed or not, the MC index is not capable of detecting non-essential multicollinearity. It is expected that it will show its usefulness in the case of essential multicollinearity.

Therefore, when Lin, Wang, and Mueller (2020) commented that *there are different views on what centering technique is most appropriate in regression [. . .] To facilitate this diversity of opinion, in the software implementation of mcvis, we allow the option of passing matrices with different centering techniques as input. The role of scaling is not the focus of our work as our framework does not rely on any specific scaling method*, far from facilitating the use of their proposal, they consider scenarios for which the MC index is not designed, since in their theoretical development and step 2 of their method, the standardization of the data is performed. However, as shown in this paper, the MC index is capable of detecting multicollinearity of the essential type only when used with its default option *studentise*.

3 Simulations

In this section, different versions of the matrix $\mathbf{X} = [\mathbf{1}\ \mathbf{X}_2\ \mathbf{X}_3\ \mathbf{X}_4]$ will be simulated. The results on the correlation matrix, its determinant, condition number (with and without intercept), variance inflation factor and coefficient of variation are obtained using the **multiColl** package (see Román Salmerón, García, and García (2021a) and Román Salmerón, García, and García (2021b) for more details).

In all cases, are calculated the values for the MC index for each set of simulated data considering the two alternative transformation for the data: *Euclidean* (centered by mean and divided by Euclidean length) and *studentise* (centered by mean and divided by standard deviation). In each case (simulation and kind of transformation), the calculation of the MC index was performed 100 times.

Simulation 1

In this case, 100 observations are generated according to $\mathbf{X}_i \sim N(10, 100)$, $i = 2, 3$, and $\mathbf{X}_4 = \mathbf{X}_3 - \mathbf{p}$ where $\mathbf{p} \sim N(1, 0.5)$. The goal of this simulation is to ensure that the variables $\mathbf{X}_3$ and $\mathbf{X}_4$ will be highly correlated (essential multicollinearity). This fact is confirmed when taking into account the following results in relation to the correlation matrix, correlation matrix's determinant, the CN with and without the intercept (with its corresponding increasing), the VIFs and the coefficient of variation of the different variables.

```
  RdetR(X_S1)

#> $`Correlation matrix`
#>             X2_S1      X3_S1       X4_S1
#> X2_S1  1.00000000 -0.0875466 -0.09110579
#> X3_S1 -0.08754660  1.0000000  0.99881845
#> X4_S1 -0.09110579  0.9988184  1.00000000
#>
#> $`Correlation matrix's determinant`
#> [1] 0.002330192

  CNs(X_S1)

#> $`Condition Number without intercept`
#> [1] 38.67204
#>
#> $`Condition Number with intercept`
#> [1] 66.94135
#>
#> $`Increase (in percentage)`
#> [1] 42.22997

  VIF(X_S1)

#>      X2_S1      X3_S1      X4_S1
#>   1.013525 425.587184 425.860062

  CVs(X_S1)

#> [1] 1.239795 1.052896 1.166335
```

Table 1: MC index for the independent variables in Simulation 1. Three random iterations, the average value of the 100 times and the standard deviation for Euclidean and studentise transformations. The relationship between the second and third variables is detected when studentise transformation is performed.

	X2	X3	X4
Euclidean - Random 1	0.2846628	0.3670736	0.3482636
Euclidean - Random 2	0.1466484	0.4444270	0.4089246
Euclidean - Random 3	0.4026253	0.3140131	0.2833616
Euclidean - Average	0.2505292	0.3899482	0.3595226
Euclidean - Standard Deviation	0.1075164	0.0550333	0.0526817
Studentise - Random 1	0.0000338	0.4942761	0.5056901
Studentise - Random 2	0.0001294	0.4901233	0.5097473
Studentise - Random 3	0.0000307	0.4950536	0.5049157
Studentise - Average	0.0000519	0.4944290	0.5055191
Studentise - Standard Deviation	0.0000264	0.0023510	0.0023528

Table 1 shows three random iterations, the average value of the 100 times and the standard deviation. As expected in the case of essential multicollinearity, from the average values of Simulation 1 it is noted that (specially with the transformation *studentise*) the MC index correctly identified that the variables $\mathbf{X}_3$ and $\mathbf{X}_4$ are causing the troubling degree of essential multicollinearity. However, it is noted that in some cases the intercept or the variable $\mathbf{X}_2$ are identified as relevant in the existing linear relations when the *Euclidean* transformation is performed. This behavior is not observed when the *studentise* transformation is performed. This fact seems to indicate that the MC index depends on the transformation with the *studentise* transformation being the most appropriate.

Simulation 2

In this case, 100 observations are generated according to $\mathbf{X}_i \sim N(10, 100)$, $i = 2, 3$, and $\mathbf{X}_4 \sim N(10, 0.0001)$. The goal of this simulation is to ensure that the variable $\mathbf{X}_4$ will be highly correlated to the intercept (non-essential multicollinearity). This fact is confirmed from the following results taking into account that Román Salmerón, Rodríguez, and García (2020) showed that a value of the CV lower than 0.1002506 indicates a troubling degree of non-essential multicollinearity.

```
RdetR(X_S2)

#> $`Correlation matrix`
#>            X2_S2       X3_S2      X4_S2
#> X2_S2  1.0000000 -0.08754660 0.06676070
#> X3_S2 -0.0875466  1.00000000 0.09445547
#> X4_S2  0.0667607  0.09445547 1.00000000
#>
#> $`Correlation matrix's determinant`
#> [1] 0.9778526

CNs(X_S2)

#> $`Condition Number without intercept`
#> [1] 2.999836
#>
#> $`Condition Number with intercept`
#> [1] 2430.189
#>
#> $`Increase (in percentage)`
#> [1] 99.87656

VIF(X_S2)

#>    X2_S2    X3_S2    X4_S2
#> 1.013525 1.018091 1.014811
```

Table 2: MC index for the independent variables in Simulation 2. Three random iterations, the average value of the 100 times and the standard deviation for Euclidean and studentise transformations. The relationship between the four variable and the intercept is not detected.

	X2	X3	X4
Euclidean - Random 1	0.2671991	0.2097102	0.5230907
Euclidean - Random 2	0.1649092	0.5119198	0.3231711
Euclidean - Random 3	0.2126011	0.2678652	0.5195337
Euclidean - Average	0.1678676	0.3335266	0.4986058
Euclidean - Standard Deviation	0.0725673	0.0845660	0.1029678
Studentise - Random 1	0.3307490	0.3342851	0.3349658
Studentise - Random 2	0.3646487	0.2995420	0.3358093
Studentise - Random 3	0.3107573	0.3481032	0.3411395
Studentise - Average	0.3541923	0.3150319	0.3307758
Studentise - Standard Deviation	0.0201173	0.0203717	0.0187872

```
CVs(X_S2)

#> [1] 1.239794695 1.052896496 0.001022819
```

Table 2 presents three random iterations, the average value of the 100 times and the standard deviation for Simulation 2 for *Euclidean* and *studentise* transformations. Before commenting the results of Simulation 2, it is important to take into account the fact that with transformations that imply the elimination of the intercept it will not be possible to detect the non-essential multicollinearity. Note that in some occasions, when *Euclidean* transformation is performed, it is concluded that $\mathbf{X}_3$ and $\mathbf{X}_4$ are the most relevant while, when *studentise* transformation is performed, all variables seem to present the same relevance. In the first case, a higher stability is observed by considering the average values, although the conclusion is that there is a relation between $\mathbf{X}_3$ and $\mathbf{X}_4$ when the relationship is between the intercept and $\mathbf{X}_4$.

Simulation 3

Finally, in this case 100 observations are generated according to $\mathbf{X}_i \sim N(10, 100)$, $i = 2, 3, 4$. The goal of this simulation is to ensure that the degree of multicollinearity (essential and non-essential) will be not troubling. This fact is confirmed when taking into account the following results.

```
RdetR(X_S3)

#> $`Correlation matrix`
#>            X2_S3       X3_S3      X4_S3
#> X2_S3  1.0000000 -0.08754660 0.06676070
#> X3_S3 -0.0875466  1.00000000 0.09445547
#> X4_S3  0.0667607  0.09445547 1.00000000
#>
#> $`Correlation matrix's determinant`
#> [1] 0.9778526

CNs(X_S3)

#> $`Condition Number without intercept`
#> [1] 2.07584
#>
#> $`Condition Number with intercept`
#> [1] 3.60862
#>
#> $`Increase (in percentage)`
#> [1] 42.47552

VIF(X_S3)

#>    X2_S3    X3_S3    X4_S3
#> 1.013525 1.018091 1.014811
```

Table 3: MC index for the independent variables in Simulation 3. Three random iterations, the average value of the 100 times and the standard deviation for Euclidean and studentise transformations. The not troubling relationship between the variables is not detected.

	X2	X3	X4
Euclidean - Random 1	0.1318422	0.3832372	0.4849206
Euclidean - Random 2	0.1679453	0.4315253	0.4005294
Euclidean - Random 3	0.0728076	0.5693939	0.3577986
Euclidean - Average	0.1083812	0.4107279	0.4808910
Euclidean - Standard Deviation	0.0410296	0.0661189	0.0628212
Studentise - Random 1	0.3586837	0.3154482	0.3258681
Studentise - Random 2	0.3805084	0.3205751	0.2989166
Studentise - Random 3	0.3651891	0.3069920	0.3278189
Studentise - Average	0.3541923	0.3150319	0.3307758
Studentise - Standard Deviation	0.0201173	0.0203717	0.0187872

```
CVs(X_S3)
```

```
#> [1] 1.239795 1.052896 1.019006
```

Table 3 presents three random iterations, the average value of the 100 times and the standard deviation for Simulation 3 for *Euclidean* and *studentise* transformations. Simulation 3 shows different situations depending on the transformation: when the *Euclidean* transformation is performed, the variable $\mathbf{X}_3$ is also identified apart from variable $\mathbf{X}_4$; with the *studentise* transformation, all the variables seem to be relevant.

Interpretation of the obtained results

From the above results, it is concluded that the MC index applied individually is not able to detect if the degree of multicollinearity is troubling. This conclusion is in line with the comment presented by Lin, Wang, and Mueller (2020) where it is stated that *those classical collinearity measures are used together with mcvis for the better learning of how one or more variables display dominant behavior in explaining multicollinearity*. That is to say, it is recommended to use measures such as the VIF and the CN to detect whether the degree of multicollinearity is troubling and, if it is, then use the MC index to detect which variables are more relevant.

We should reiterate the fact that the results of the MC index depend on the transformation performed with the data.

Finally, it is worthy of note that the lowest dispersion is obtained when the *studentise* transformation is performed, which indicate that with this transformation a higher stability exists in the results obtained with the 100 iterations performed.

4 Examples

In this section we will analyze two examples applied recently to illustrate the multicollinearity problem. The first one focuses on the existence of non-essential approximate multicollinearity while the second one is focused on essential multicollinearity.

Example 1

Román Salmerón, Rodríguez, and García (2020) analyzed the Euribor as a function of the harmonized index of consumer prices (**HICP**), the balance of payments to net current account (**BC**) and the government deficit to net non-financial accounts (**GD**).

The following determinant of the matrix of correlations of the independent variables, the VIFs, condition number without intercept and with intercept and the coefficients of variation indicate that the degree of essential near multicollinearity is not troubling while the non-essential type (due to the variable **HIPC**) is.

Figure 1 shows a tour displayed with a scatterplot by using the **tourr** package. This package allows tours of multivariate data, see Wickham et al. (2011) for more details. From the tour on all

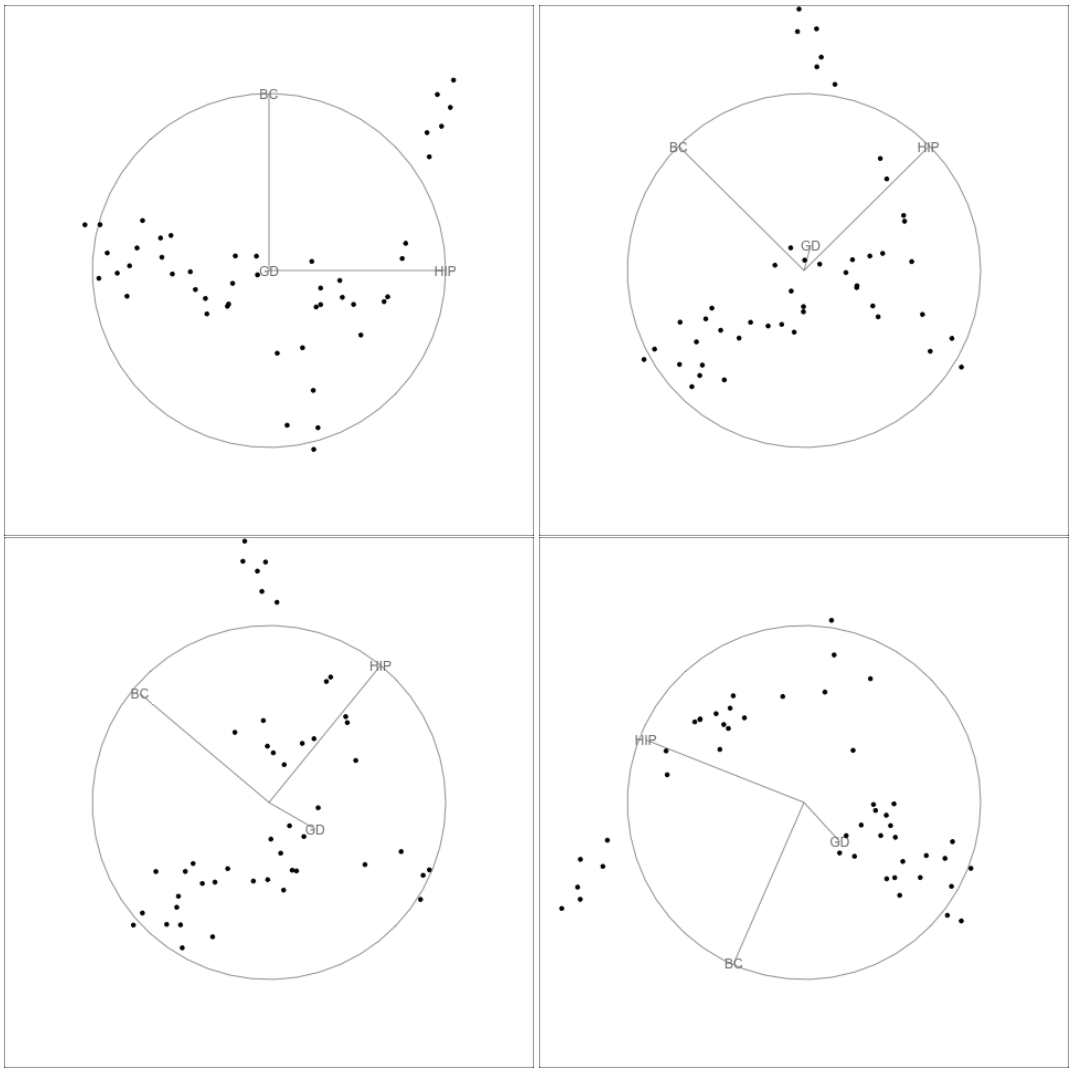


Figure 1: Representation of example 1 using the R tourr package. No linear relationship is observed between the explanatory variables in any of the four plots. See html version for an interactive 2-D tour.

the explanatory variables (it runs for 3.47 minutes in the html version), no linear relation is observed between the explanatory variables. Note that this package does not allow us to work with the intercept.

In relation to the application of the **mcvis** package in this example, it can be observed that if the *Euclidean* transformation is performed the method concludes that all the variables seem to be relevant:

```
#>      HIPC   BC   GD
#> tau3 0.29 0.39 0.32
```

For *studentise* transformation it concludes the establishing of a relationship between **HIPC** and **GD**:

```
#>      HIPC   BC   GD
#> tau3 0.49 0.13 0.38
```

Clearly, these conclusions are not consistent. Moreover, by eliminating the intercept when transforming the data, it is not feasible to detect the non-essential multicollinearity.

Example 2

James et al. (2013) illustrated multicollinearity with a data set which contains information about the dependent variable **balance** (average credit card debt for a number of individuals) and, among others, the following quantitative independent variables: **income**, **limit** (credit limit), **rating** (credit rating),

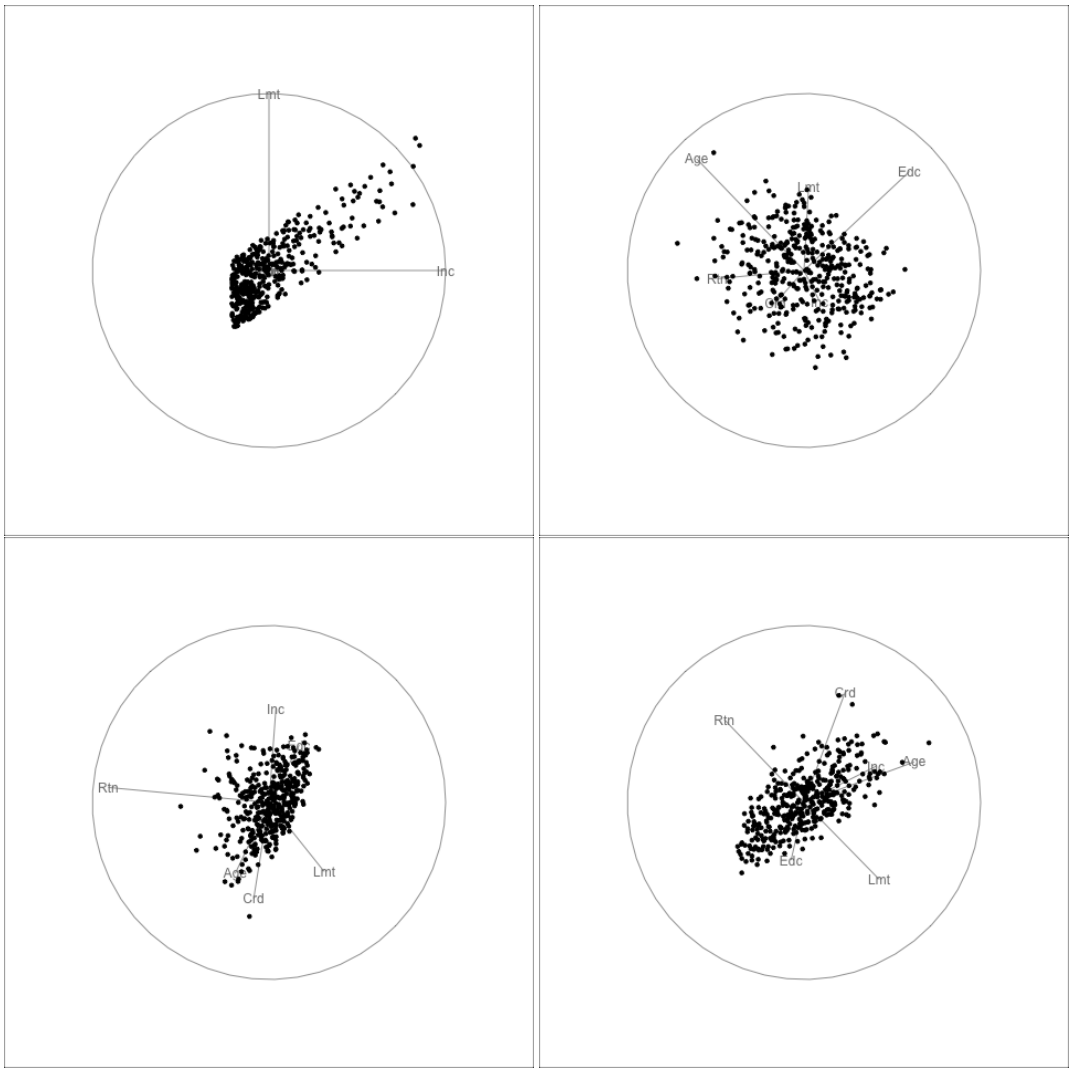


Figure 2: Representation of example 2 using the R tourr package. From the 3.47 minutes tour on all the explanatory variables, a certain linear relationship between independent variables is observed specially in plots one (top and left) and four (bottom and right). See html version for an interactive 2-D tour.

cards (number of credit cards), **age**, **education** (years of education). Data is available in the **ISLR** package (see James et al. (2021) for more details).

The following determinant of the matrix of correlations of the independent variables, the simple correlation between **limit** and **rating** (0.99687974), the VIFs, condition number without intercept and with intercept and CVs indicate that the degree of approximate multicollinearity of the non-essential type is not troubling while that of the essential type (due to the relationship between variables **limit** and **rating**) is troubling.

Figure 2 displays its tour by using again **tourr** package. Multicollinearity was checked using a tour on all the explanatory variables (it runs for 3.47 minutes in html version). In this case a certain linear relationship is observed, although it is difficult to determine which variables are related.

In relation to the application of the **mcvis** package in this example, it can be observed that if the *Euclidean* transformation is performed the variable with the highest value is **income**, with the variable **rating** being the second-lowest in value.

```
#>      Income Limit Rating Cards  Age Education
#> tau6   0.62  0.14   0.05  0.05 0.11      0.03
```

Finally, when the *studentise* transformation is applied, the method clearly indicates that variables **limit** and **rating** are the ones responsible for the multicollinearity problem.

```
#>      Income Limit Rating Cards Age Education
```

```
#> tau6      0  0.49   0.51     0   0         0
```

5 Discussion

The results shown in the previous sections indicate that, on the one hand, the calculation of the MC index depends on the transformation performed and, on the other hand, that to apply this index with guarantees, the *studentise* transformation is the most appropriate.

It was also showed that the MC index "inherits' ' the same limitations indicated by Lin, Wang, and Mueller (2020) when they state that *collinearity indices such as the variance inflation factor and the condition number have limitations and may not be effective in some applications*. In the case of VIF, these limitations are well summarized by Lin, Wang, and Mueller (2020): *The VIF can show how variables are correlated to each other, but as shown, low correlation does not guarantee low level of collinearity*. Note that the example applied by the authors in Section 1 (similar to the one presented in R. Salmerón, García, and García (2018) and a reduced version of the one presented by David A. Belsley (1984)) is a clear case in which the multicollinearity is non-essential. As was commented earlier, the VIF ((R. Salmerón, García, and García 2018) and (Román Salmerón, Rodríguez, and García 2020)) and the MC index are not able to detect this kind of multicollinearity, for this reason they are only recommended for detecting essential multicollinearity.

Another limitation of the VIF (and also of the MC index) worthy of note is that it is not adequate for calculating with dummy variables. Using these kinds of variables, the VIF is obtained from a coefficient of determination of an auxiliary regression whose dependent variable is a dummy variable. Although it is possible to apply ordinary least squares in this kind of regression, it is well known that it can present some problems: for example, the coefficient of determination is not representative since it measures the linear relation but the relation between the dummy variable and the rest of independent variables is not linear. For this reason, these kinds of regressions are estimated with non-linear models such as the logit/probit. Thus, we consider that if it is not adequate for calculating the VIF associated to a dummy variable, these kinds of variables should be avoided in the calculation of the MC index.

Finally, Simulation 3 shows that MC index is not able to detect whether the degree of essential multicollinearity is troubling. For this reason, it should only be used once this situation is determined by other measures (as set out in the introduction) and in order to determine which variables cause it.

6 Solution

The distinction between essential and non-essential multicollinearity and the limitations of each measure for detecting the different kinds of multicollinearity, can be very useful for detecting whether there is a troubling degree of multicollinearity, what kind of multicollinearity it is and which variables are causing the multicollinearity. Thus, taking into account the fact that the VIF is useful for detecting essential multicollinearity and the CV is useful for detecting non-essential multicollinearity, the scatter plot of both measures can provide interesting information in a joint way.

We present the scatter plot for the values of the VIF and the CV for the three simulations previously performed. Note that the figures include the lines corresponding to the established thresholds for each measure: red dashed vertical line for 0.1002506 (CV) and red dotted horizontal line for 10 (VIF). These lines determine four regions that can be interpreted as follows: A, existence of troubling non-essential and non-troubling essential multicollinearity; B, existence of troubling essential and non-essential multicollinearity; C, existence of non-troubling non-essential and troubling essential multicollinearity; D: non-troubling degree of existing multicollinearity (essential and non-essential).

Considering this classification, in Simulation 1 (Figure 3) it is noted that there is a troubling degree of essential multicollinearity due to the variables $\mathbf{X}_3$ and $\mathbf{X}_4$, while in Simulation 2 (Figure 4) it is noted that there is a troubling degree of non-essential multicollinearity due to the variable $\mathbf{X}_4$ and in Simulation 3 (Figure 5) it is noted that the degree of multicollinearity is not troubling.

Analogously, we present Figures 6 and 7 for Examples 1 and 2, respectively. For Example 1, it is concluded that the only type of troubling multicollinearity is non-essential due to the second variable (**HIPC**). On the other hand, for Example 2 it is concluded that the only type of troubling multicollinearity is that essentially due to the relationship existing between the third and fourth variables (**limit** and **rating**).

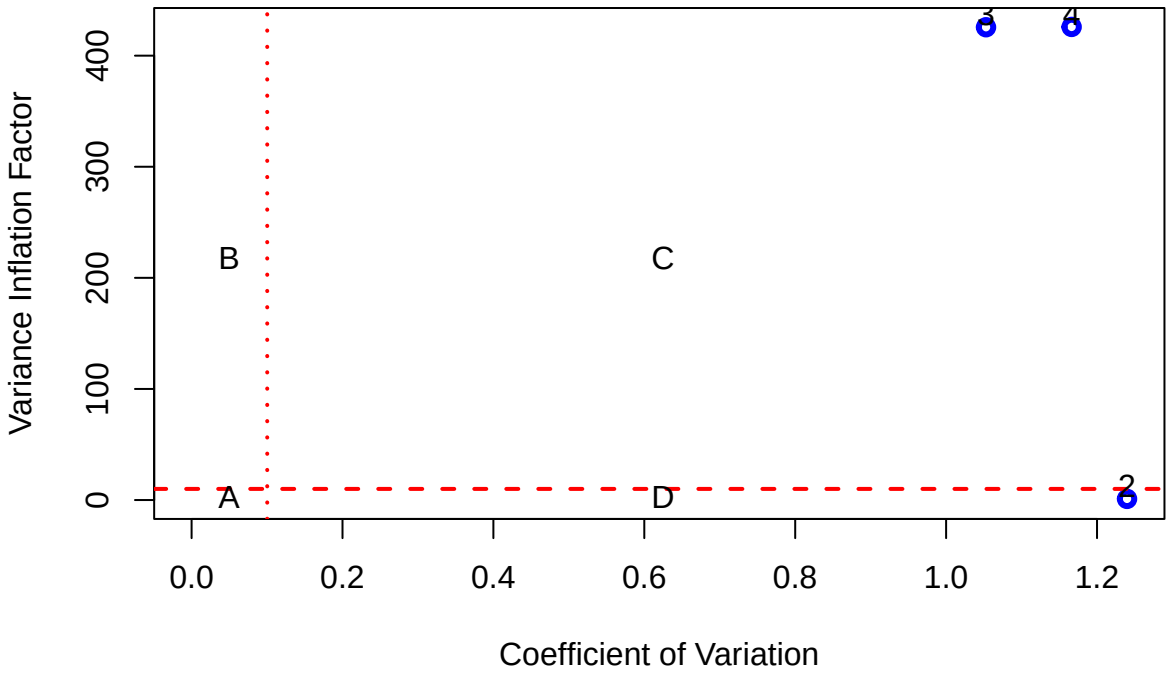


Figure 3: Representation of the VIFs and CVs of explanatory variables in Simulation 1. A troubling degree of essential multicollinearity due to variables third and four is detected by considering thresholds of 0.1002506 (CV) and 10 (VIF) highlighted with red lines. Note that VIFs of variables three and four are greater than 10 and the CV of variable 2 is lower than 0.1002506.

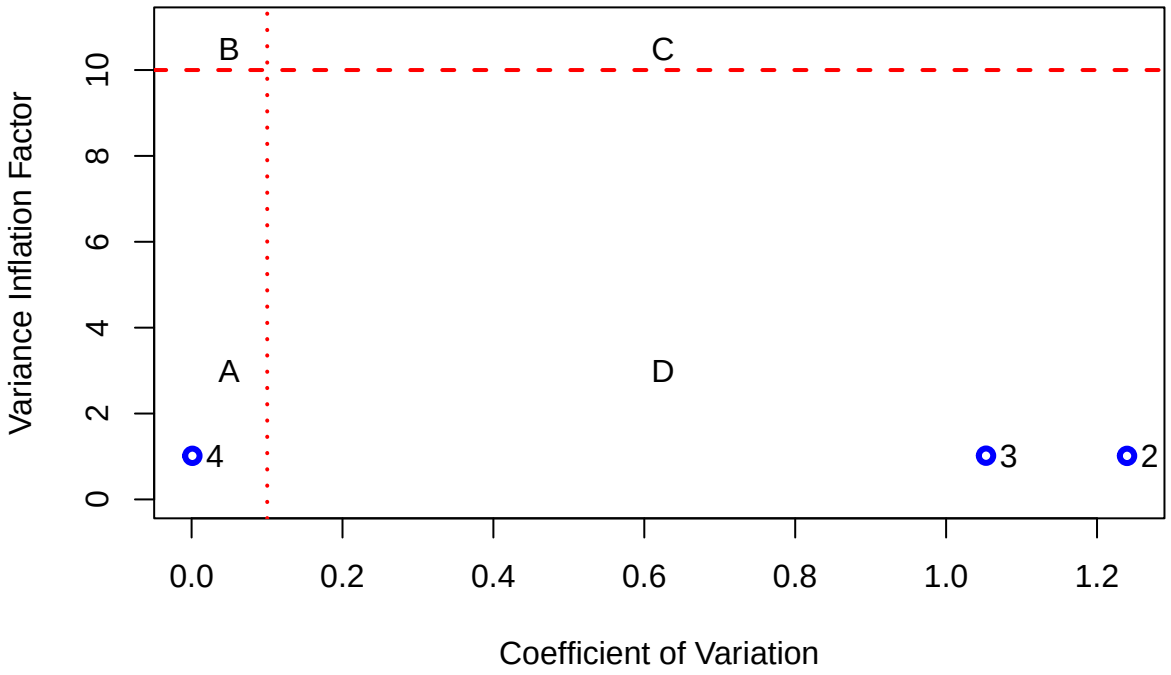


Figure 4: Representation of the VIFs and CVs of explanatory variables in Simulation 2. A troubling degree of non-essential multicollinearity due to variable four is detected by considering thresholds of 0.1002506 (CV) and 10 (VIF) highlighted with red lines. Note that the VIFs of variables three and four are less than 10 and the CV greater than 0.1002506; while the CV of second variable is less than 0.1002506.

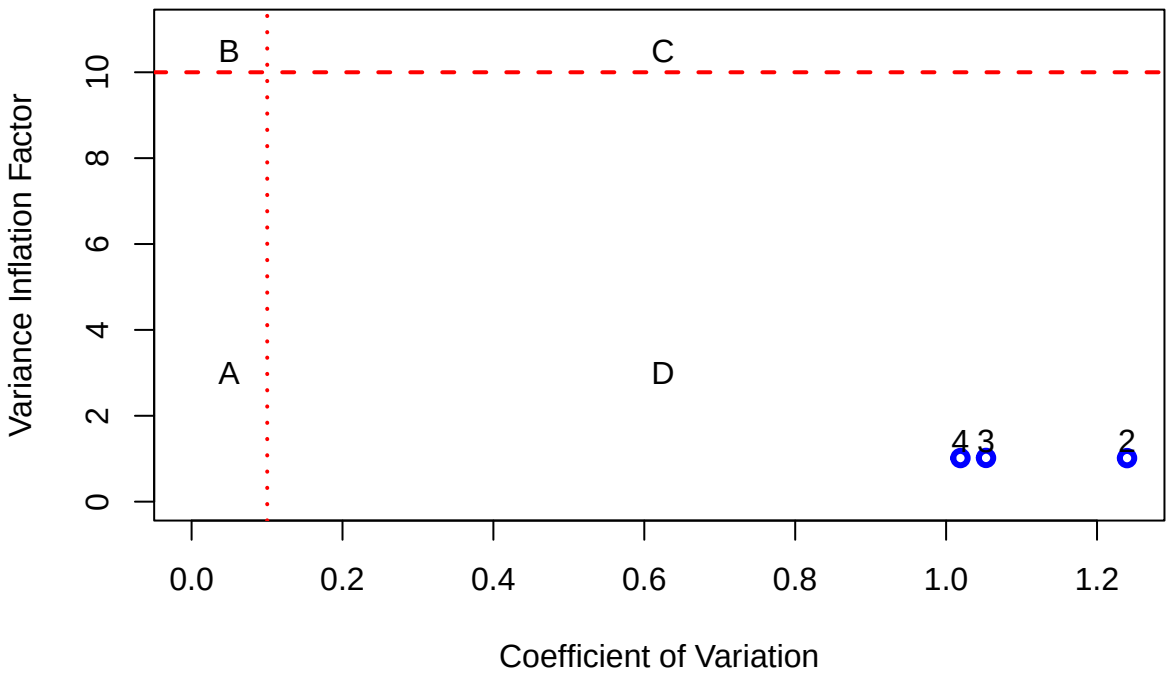


Figure 5: Representation of the VIFs and CVs of explanatory variables in Simulation 3. The degree of multicollinearity is not troubling by considering thresholds of 0.1002506 (CV) and 10 (VIF) highlighted with red lines. Note that the VIFs of all variables are lower than 10 and the CVs greater than 0.1002506.

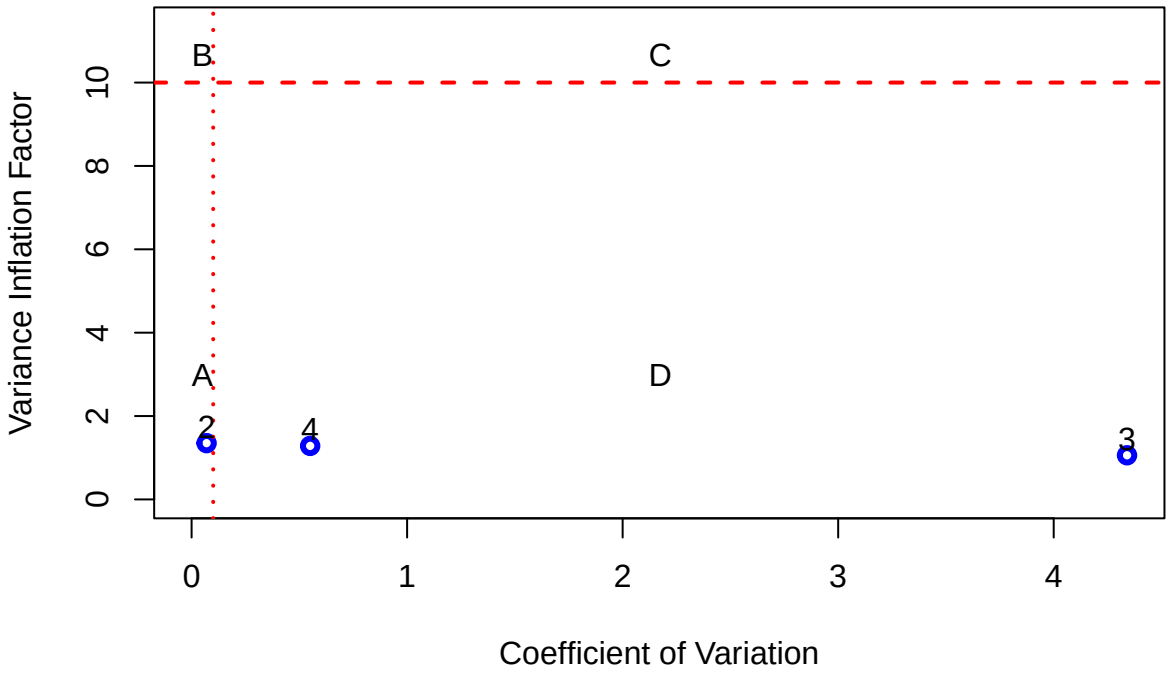


Figure 6: Representation of the VIFs and CVs of the variables of example 1. A troubling degree of non-essential multicollinearity generated by variable 2 (HIPC) is detected by considering thresholds of 0.1002506 (CV) and 10 (VIF) highlighted with red lines. Note that the VIFs of variables three and four are less than 10 and the CV greater than 0.1002506; while the CV of second variable is less than 0.1002506.

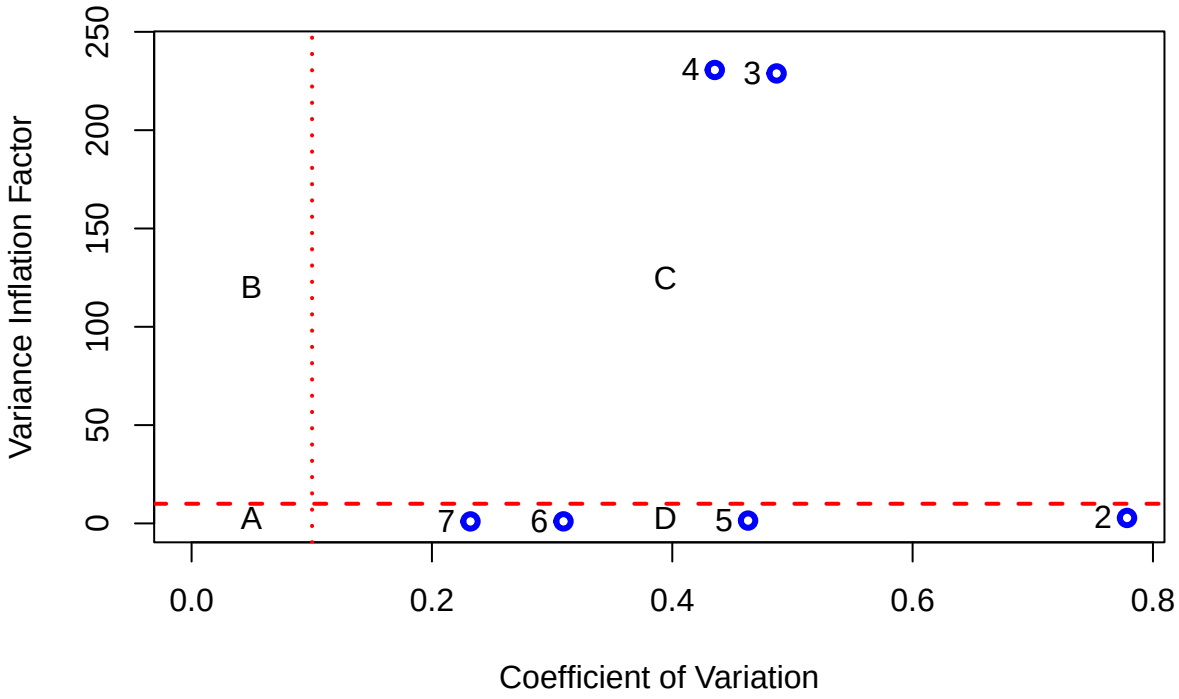


Figure 7: Representation of the VIFs and CVs of the variables of example 2. A troubling degree of essential multicollinearity generated by variables three and four (limit and rating) is detected by considering thresholds of 0.1002506 (CV) and 10 (VIF) highlighted with red lines. Note that the VIFs of variables three and four are greater than 10 while the VIFs and CVs of the rest of variables are, respectively, lower than 10 and greater than 0.1002506.

7 Summary

This paper analyses the limitations that may arise in detecting the problem of troubling multicollinearity in a linear regression model due to transformations performed on the data. The discussion is used to illustrate that the MC index presented by Lin, Wang, and Mueller (2020) depends on the way that the data are transformed.

It was shown that when the *studentise* transformation is performed, the measure is stable for *measuring what variable contributes the most to the linear relationship, and thus identifying what variables best explain the collinearity in the original data* (Lin, Wang, and Mueller 2020), as long as the multicollinearity is essential. This kind of transformation was taken as default by the authors and this paper contributes with a formal justification. Note that the MC index only provides interesting information if the troubling multicollinearity is previously detected using other measures (such as the VIF or the CN).

Summarizing, the achievement of the goal intended by the **mcvis** package is conditioned by the following limitations:

- It is not able to detect non-essential multicollinearity.
- Other measures (as the VIF or CN) should be applied previously to determine whether the degree of essential multicollinearity is troubling.
- The *studentise* transformation should be applied.
- It is not applicable for dummy variables.

Finally, it is proposed to use a scatter plot between the VIFs and the CVs, on the one hand, to detect whether the degree of multicollinearity (essential or not essential) is troubling and, on the other hand, to detect which variables are causing the multicollinearity. This would overcome the limitations of the **mcvis** package discussed above while achieving its overall purpose.

8 Acknowledgments

This work has been supported by project PP2019-EI-02 of the University of Granada (Spain), by project A-SEJ-496-UGR20 of the Andalusian Government's Counseling of Economic Transformation, Industry, Knowledge and Universities (Spain) and by project I+D+i PID2019-107767GA-I0 financed by MCIN/AEI/10.13039/501100011033.

References

Belsley, D. A., E. Kuh, and R. E. Welsch. 1980. *Regression diagnostics: Identifying influential data and sources of collinearity*. New York: John Wiley & Sons.

Belsley, David. 1991. "A Guide to Using the Collinearity Diagnostics." *Computational Science in Economics and Manegement* 4: 33–50.

Belsley, David A. 1984. "Demeaning Conditioning Diagnostics Through Centering." *The American Statistician* 38 (2): 73–77.

García, C., R. Salmerón, and C. B. García. 2019. "Choice of the Ridge Factor from the Correlation Matrix Determinant." *Journal of Statistical Computation and Simulation* 89 (2): 211–31.

García, J., R. Salmerón, C. García, and M. López. 2016. "Standardization of Variables and Collinearity Diagnostic in Ridge Regression." *International Statistical Review* 84: 245–66.

Gujarati, D. 2010. *Basic Econometrics*. 8th ed. McGraw Hill.

Halkos, G., and K. Tsilika. 2018. "Programming Correlation Criteria with Free Cas Software." *Computational Economics* 52 (1): 299–311.

Hoerl, A. E., and R. W. Kennard. 1970. "Ridge Regression: Biased Estimation for Nonorthogonal Problems." *Technometrics* 12 (1): 55–67.

James, Gareth, Daniela Witten, Trevor Hastie, and Rob Tibshirani. 2021. *ISLR: Data for an Introduction to Statistical Learning with Applications in r*. https://CRAN.R-project.org/package=ISLR.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning with Application in r*. Springer.

Lin, C., K. Wang, and S. Mueller. 2020. "MCVIS: A New Framework for Collinearity Discovery, Diagnostic and Visualization." *Journal of Computational and Graphical Statistics*. https://doi.org/10.1080/10618600.2020.1779729.

Marquardt, D. W. 1970. "Generalized inverses, ridge regression, biased linear estimation and nonlinear estimation." *Technometrics* 12 (3): 591–612.

———. 1980. "You Should Standardize the Predictor Variables in Your Regression Models." *J. Amer. Statist. Assoc.* 75 (369): 87–91.

Novales, A. 1988. *Econometría*. Madrid: McGraw-Hill.

O'Brien, R. M. 2007. "A caution regarding rules of thumb for variance inflation factors." *Quality & Quantity* 41: 673–90.

Ramanathan, Ramu. 2002. "Introductory Econometrics with Applications."

Salmerón, R., C. García, and J. García. 2018. "Variance Inflation Factor and Condition Number in Multiple Linear Regression." *Journal of Statistical Computation and Simulation* 88: 2365–84.

———. 2020. "Comment on 'a Note on Collinearity Diagnostics and Centering' by Velilla (2018)." *The American Statistician* 74 (1): 68–71.

Salmerón, R., J. García, C. García, and M. López. 2018. "Transformation of Variables and the Condition Number in Ridge Estimation." *Computational Statistics* 33: 1497–1524.

Salmerón, Román, Catalina García, and José García. 2021a. "A Guide to Using the r Package multiColl for Detecting Multicollinearity." *Computational Economics* 57: 529–36. https://doi.org/10.1007/s10614-019-09967-y.

———. 2021b. "The multiColl Package Versus Other Existing Packages in r to Detect Multicollinearity." *Computational Economics*. https://doi.org/10.1007/s10614-021-10154-1.

Salmerón, Román, Ainara Rodríguez, and Catalina García. 2020. "Diagnosis and Quantification of the Non-Essential Collinearity." *Computational Statistics* 35: 647–66.

Snee, R. D., and D. W. Marquardt. 1984. "Comment: Collinearity diagnostics depend on the domain of prediction, the model, and the data." *The American Statistician* 38 (2): 83–87.

Tibshirani, Robert. 1996. "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society. Series B (Methodological)*, 267–88.

Wickham, Hadley, Dianne Cook, Heike Hofmann, and Andreas Buja. 2011. "Tourr: A r Package for Exploring Multivariate Data with Projections." *Journal of Statistical Software* 40 (2): 1–18. https://doi.org/10.18637/jss.v040.i02.

Wooldridge, J. M. 2008. *Introducción a la econometría. Un enfoque moderno*. Second. Madrid: Thomson Paraninfo.

Zou, H., and T. Hastie. 2005. "Regularization and Variable Selection via the Elastic Net." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67: 301–20.

Roman Salmeron Gomez
University of Granada
Department of Quantitative Methods for Economics and Busines
Poligono La Cartuja sn, 18071, Granada, Spain.
http://metodoscuantitativos.ugr.es/pages/web/romansg
ORCiD: 0000-0003-2589-4058
romansg@ugr.es

Catalina B. Garcia Garcia
University of Granada
Department of Quantitative Methods for Economics and Busines
Poligono La Cartuja sn, 18071, Granada, Spain.
http://metodoscuantitativos.ugr.es/pages/web/cbgarcia
ORCiD: 0000-0003-1622-3877
cbgarcia@ugr.es

Ainara Rodriguez Sanchez
U.N.E.D.
Department of Applied Economics and Economic History
Madrid, Spain.
http://metodoscuantitativos.ugr.es/pages/web/cbgarcia
ORCiD: 0000-0003-1622-3877
arsanchez@cee.uned.es

Claudia Garcia Garcia
Complutense University of Madrid
Department of Applied Economics, Structure and History
Madrid, Spain.
http://metodoscuantitativos.ugr.es/pages/web/cbgarcia
ORCiD: 0000-0003-1622-3877
clgarc13@ucm.es

ppseq: An R Package for Sequential Predictive Probability Monitoring

by Emily C. Zabor, Brian P. Hobbs, and Michael J. Kane

Abstract Advances in drug discovery have produced numerous biomarker-guided therapeutic strategies for treating cancer. Yet the promise of precision medicine comes with the cost of increased complexity. Recent trials of targeted treatments have included expansion cohorts with sample sizes far exceeding those in traditional early phase trials of chemotherapeutic agents. The enlarged sample sizes raise ethical concerns for patients who enroll in clinical trials, and emphasize the need for rigorous statistical designs to ensure that trials can stop early for futility while maintaining traditional control of type I error and power. The R package **ppseq** provides a framework for designing early phase clinical trials of binary endpoints using sequential futility monitoring based on Bayesian predictive probability. Trial designs can be compared using interactive plots and selected based on measures of efficiency or accuracy.

1 Introduction

Statistical methods for early phase oncology trials were developed in the context of cytotoxic treatments. Most cytotoxic treatments exhibit increases in both efficacy and toxicity with increasing dose. As a result, phase I trials were centered on identifying the maximum tolerated dose (MTD), defined as the highest dose that did not exceed a pre-specified toxicity threshold. But advances in drug discovery have produced numerous biomarker-guided non-cytotoxic therapies such as small molecule inhibitors, antibody drug conjugates, immune checkpoint inhibitors, and monoclonal antibodies. These therapies typically do not exhibit the same pattern of increases in both efficacy and toxicity with increasing dose, so the MTD as traditionally defined may not exist. Instead, lower doses may have equivalent efficacy but lower toxicity as compared to higher doses. As a result, these therapies can be difficult to study with traditional dose-escalation designs such as the rule-based 3+3 and the model-based continual reassessment method, among others (Pestana et al. 2020). To address this issue, recent phase I trials have included large dose-expansion cohorts, in which additional patients are enrolled in phase I after the dose-escalation phase is complete. In this setup, the dose-escalation phase is considered phase Ia and used to assess the initial safety of multiple doses, then the dose-expansion phase is considered phase Ib and can have a variety of aims including to further refine the safety of one or more doses, to assess preliminary efficacy, to explore the treatment in various disease-specific subtypes, or to further characterize the pharmacokinetics and/or pharmacodynamics. The use of dose-expansion cohorts increased from 12% in 2006 to 38% in 2011 (Manji et al. 2013) and trials with dose-expansion cohorts led to higher response rates and more frequent success in phase II trials (Bugano et al. 2017).

But despite these successes, recent dose-expansion cohorts have not always been planned in advance, leading to uncertain statistical properties, and have at times included samples sizes that far exceed those typically seen in early phase trials. For example, the KEYNOTE-001 trial of pembrolizumab, initially designed as a 3+3 dose-escalation trial, included multiple protocol amendments and ultimately enrolled a total of 655 patients across five melanoma expansion cohorts and 550 patients across four non-small-cell lung cancer expansion cohorts (Khoja et al. 2015). In a basket trial of atezolizumab, an anti-PD-L1 treatment in patients with a variety of cancers and both with and without PD-L1 expression, an expansion cohort in metastatic urothelial carcinoma ultimately evaluated 95 patients, despite the fact that no expansion cohort in this disease subtype was originally planned in the trial protocol. The expansion cohort in metastatic urothelial carcinoma was added through a protocol amendment, and the sample size later was increased from what was initially planned (Petrylak et al. 2018; Powles et al. 2014). These enlarged sample sizes raise ethical concerns for patients who enroll in clinical trials, and emphasize the need for rigorous statistical designs to ensure that trials can stop early for futility while maintaining traditional control of type I error and power.

Bayesian predictive probability has been proposed as an approach for sequential monitoring in early phase oncology trials (Dmitrienko and Wang 2006; Lee and Liu 2008; Hobbs, Chen, and Lee 2018; Saville et al. 2014). However, in order to be useful to investigators designing such trials, software must be made available to calibrate the design for the desired statistical properties. To our knowledge no such software currently exists in the R programming language. This paper introduces the **ppseq** package for the R software language, which provides functions to design early phase clinical trials of binary endpoints using sequential predictive probability monitoring for futility. Static plots produced using the **ggplot2** package (Wickham 2016) and interactive plots produced using the **plotly** package (Sievert 2020) compare design options based on frequentist type I error and power.

Moreover, the **ppseq** package implements two criteria for selecting an ideal predictive probability monitoring design from among the options formed by combinations of posterior and predictive decision thresholds. While the **ppseq** package was developed with early phase oncology clinical trials in mind, the methodology is general and can be applied to any application of sequential futility monitoring in clinical trial design.

2 Predictive probability monitoring

Consider the setting of an expansion cohort with a binary outcome, such as tumor response as measured by the RECIST (Response Evaluation Criteria in Solid Tumors) criteria. Here we will focus on the one-sample setting, in which all patients in the trial are enrolled onto a single arm and given the experimental treatment of interest. Functionality is also available for the two-sample setting, in which patients are enrolled onto two different treatment arms, or a treatment and a control arm, for comparative purposes. Each patient, denoted by i, enrolled in the trial either has a response such that $x_i = 1$ or does not have a response such that $x_i = 0$. Then $X = \sum_{i=1}^{n} x_i$ is the number of responses out of n currently observed patients up to a maximum of N total patients. Let p represent the true probability of response. Fix p_0 as the threshold for unacceptable response rate and p_1 as the threshold for acceptable response rate. Most dose-expansion studies with an efficacy aim will wish to test the null hypothesis $H_0 : p \leq p_0$ versus the alternative hypothesis $H_1 : p \geq p_1$.

The Bayesian paradigm of statistics is founded on Bayes' theorem, which is a mathematical theory that specifies how to combine the prior distributions that define prior beliefs about parameters, such as the true response rate p, with the observed data, such as the total number of responses X, yielding a posterior distribution. Here the prior distribution of the response rate $\pi(p)$ has a beta distribution $\text{Beta}(a_0, b_0)$ and our data X have a binomial distribution $\text{Bin}(n, p)$. Combining the likelihood function for the observed data $L_x(p) \propto p^x(1-p)^{n-x}$ with the prior, we obtain the posterior distribution of the response rate, which follows the beta distribution $p|x \sim \text{Beta}(a_0 + x, b_0 + n - x)$. A posterior probability threshold θ would be pre-specified during the trial design stage. At the end of the trial, if the posterior probability exceeded the pre-specified threshold, i.e. if $\Pr(p > p_0|X) > \theta$, the trial would be declared a success.

The posterior predictive distribution of the number of future responses X^* in the remaining $n^* = N - n$ future patients follows a beta-binomial distribution $\text{Beta-binomial}(n^*, a_0 + x, b_0 + n - x)$. Then the posterior predictive probability (PPP), is calculated as $PPP = \sum_{x^*=0}^{n^*} \Pr(X^* = x^*|x) \times I(\Pr(p > p_0|X, X^* = x^*) > \theta)$. The posterior predictive probability represents the probability that, at any given interim monitoring point, the treatment will be declared efficacious at the end of the trial when full enrollment is reached, conditional on the currently observed data and the specified priors. We would stop the trial early for futility if the posterior predictive probability dropped below a pre-specified threshold θ^*, i.e. $PPP < \theta^*$. Predictive probability thresholds closer to 0 lead to less frequent stopping for futility whereas thresholds near 1 lead to frequent stopping unless there is almost certain probability of success. Predictive probability provides an intuitive interim monitoring strategy for clinical trials that tells the investigator what the chances are of declaring the treatment efficacious at the end of the trial if we were to continue enrolling to the maximum planned sample size, based on the data observed in the trial to date.

3 Package overview

The **ppseq** package facilitates the design of clinical trials utilizing sequential predictive probability monitoring for futility. The goal is to establish a set of decision rules at the trial planning phase that would be used for interim monitoring during the course of the trial. The main computational challenge in designing such a trial is joint calibration of the posterior probability and posterior predictive probability thresholds to be used in the trial in order to achieve the desired levels of frequentist type I error and power. The main function for achieving this aim is the `calibrate_thresholds()` function, which will evaluate a grid of posterior thresholds θ and predictive thresholds θ^* provided by the user as vector inputs specified with the arguments `pp_threshold` and `ppp_threshold`, respectively. Other required arguments include the unacceptable response rate p_0 specified by `p_null`, the acceptable response rate p_1 specified by `p_alt`, a vector of sample sizes at which interim analyses are to be performed `n`, and the maximum total sample size `N`. The direction of the alternative hypothesis is specified with the argument `direction` and defaults to `"greater"`, which corresponds to the alternative hypothesis $H_1 : p \geq p_1$. The hyperparameters of the prior beta distribution are specified with the argument `prior` and default to `c(0.5, 0.5)`, which denotes a $\text{Beta}(0.5, 0.5)$ distribution. The number of posterior samples is specified with the argument `S`, which defaults to `5000` samples, and the number of simulated trial datasets is specified with the argument `nsim`, which defaults to `1000`. The additional

argument `delta`, which defaults to `NULL` for the one-sample setting, can specify a clinically meaningful difference between groups $\delta = p_1 - p_0$ in the case of a two-sample trial design.

The `calibrate_thresholds()` function conducts the following algorithm, given the default arguments:

1. Generate `nsim` datasets, denoted by j, containing the cumulative number of responses x at each interim sample size n from a binomial distribution under the unacceptable (i.e. null) response rate p_0, specified by `p_null`, and under the acceptable (i.e. alternative) response rate p_1, specified by `p_alt`, for each interim look, denoted by l.
2. For dataset j and posterior threshold θ_k, draw `S` samples, denoted by s, from the posterior distribution $p|x_{ls} \sim \text{Beta}(a_0 + x_l, b_0 + n_l - x_l)$ where x_l is the number of responses at interim look l and n_l is the number of enrolled patients at interim look l. Use each $p|x_{ls}$ as the response probability in a binomial distribution to generate the number of future responses X^*_{ls} in the remaining $n^*_l = N - n_l$ future patients at interim look l.
    a. Then for each X^*_{ls}, generate `S` posterior probabilities, denoted by s', at the end of the trial: $PP^*_{lss'} \sim \text{Beta}(a_0 + X^*_{ls} + x_l, b_0 + n^*_l - (X^*_{ls} + x_l))$ and calculate $PP^*_{ls} = \Pr(p > p_0 | X^*_{ls}) = \frac{\sum_1^{S'} PP^*_{lss'} > p_0}{S'}$.
    b. Estimate the predictive probability at posterior threshold k as $PPP_{lk} = \frac{\sum_1^S PP^*_{ls} > \theta_k}{S}$.
    c. Stop the trial for dataset j at interim look l and predictive threshold m if $PPP_{lk} < \theta^*_m$. Otherwise continue enrolling.
3. Repeat (2) over all combinations of datasets j, posterior thresholds k, and predictive thresholds m.
4. If dataset j was stopped early for futility then we do not reject the null hypothesis. If dataset j reached full enrollment, we reject the null hypothesis $H_0 : p \leq p_0$ at posterior threshold k if $PPP_{lk} > \theta_k$.

The function returns a list, the first element of which is a `tibble` containing the posterior threshold θ, the predictive threshold θ^*, the mean sample size under the null and the alternative, the proportion of positive trials under the null and alternative, and the proportion of trials stopped early under the null and alternative. The proportion of trials simulated under the null hypothesis for which the null hypothesis was rejected is an estimate of the type I error, and the proportion of trials simulated under the alternative hypothesis for which the null hypothesis was rejected is an estimate of the power. The `print()` option will print the results summary for each combination of thresholds, filtered by an acceptable range of frequentist type I error and minimum power, if desired. Note that the results will be sensitive to the choice of `nsim` and `S`. We have set what we believe are reasonable defaults and would caution users against reducing these values without careful consideration.

Design selection

After obtaining results for all combinations of evaluated posterior and predictive thresholds, the next step is to select the ideal design from among the various options. The **ppseq** package introduces two criteria to assist users in making a selection. The first, called the "optimal accuracy" design, identifies the design that minimizes the Euclidean distance to 0 type I error probability and a power of 1. To accomplish this, the accuracy Euclidean distance (AED) for the design with posterior threshold k and predictive threshold m is calculated as $AED_{km} = w_\alpha * (\alpha_{km} - 0)^2 + w_{(1-\beta)} * ((1-\beta)_{km} - 1)^2$, where w_α and $w_{(1-\beta)}$ are optional weights on the type I error and power, respectively, and α_{km} denotes the estimated type I error and $(1-\beta)_{km}$ denotes the estimated power. The design with the smallest value of AED_{km} is selected as optimal. The second criteria, called the "optimal efficiency" design, identifies the design that minimizes the Euclidean distance to minimal average sample size under the null and maximal average sample size under the alternative. To accomplish this, the efficiency Euclidean distant (EED) for the design with posterior threshold k and predictive threshold m is calculated as $EED_{km} = w_{\bar{N}_{H_0}} * (\bar{N}_{H_0 km} - min(\bar{N}_{H_0}))^2 + w_{\bar{N}_{H_1}} * (\bar{N}_{H_1 km} - max(\bar{N}_{H_1}))^2$, where $w_{\bar{N}_{H_0}}$ and $w_{\bar{N}_{H_1}}$ are optional weights on the average sample size under the null and alternative, respectively, $\bar{N}_{H_0 km}$ and $\bar{N}_{H_1 km}$ denote the average sample sizes under the null and alternative, respectively, and $min(\bar{N}_{H_0})$ and $max(\bar{N}_{H_1})$ denote the minimum average sample size under the null and the maximum average sample size under the alternative alternative, respectively, across all combinations of k and m. The design with the smallest value of EED_{km} is selected as optimal. The `optimize_design()` function returns a list that contains the details of each of the two optimal designs.

Decision rules

To ease the implementation of clinical trials designed with sequential predictive probability monitoring, once a design has been selected, a table of decision rules can be produced using the `calc_decision_rules()` function. The function takes the sample sizes `n` at which interim analyses are to be performed as well as the maximum total sample size `N`, the null value to compare to in the one-sample case `p0` (set to `NULL` in the two-sample case), the posterior threshold of the selected design `theta`, and the predictive threshold of the selected design `ppp`. Arguments `direction`, `prior`, `S`, `delta` are as described in the Package Overview section, with the same defaults. The function results in a `tibble`. The trial would stop at a given look if the number of observed responses is less than or equal to r, otherwise the trial would continue enrolling if the number of observed responses is greater than r. At the end of the trial when the maximum planned sample size is reached, the treatment would be considered promising if the number of observed responses is greater than r. In the one-sample case, the resulting `tibble` includes a column for the sample size `n` at each interim look, r at each look, and a column for the associated posterior predictive probability `ppp`. In the two-sample case, the `tibble` includes columns for `n0` and `n1`, the sample size at each interim analysis in the control and experimental arms, respectively. There are also columns for `r0` and `r1`, the number of responses in the control arm and experimental arm, respectively, leading to the decision to stop or continue. Finally, there is a column for the posterior predictive probability associated with that decision `ppp`.

Visualizations

Finally, to assist users in comparing the results of the various design options, a `plot()` option is available for the results of `calibrate_thresholds` that allows creation of static plots using the **ggplot2** package (Wickham 2016) or interactive plots using the **plotly** package (Sievert 2020). Two plots are produced, one plotting type I error by power and indicating the optimal accuracy design, and one plotting the average sample size under the null by the average sample size under the alternative and indicating the optimal efficiency design. The motivation for including an interactive graphics option was the utility of the additional information available when hovering over each point. Instead of simply eyeballing where points fall along the axes, users can see the specific type I error, power, average sample size under the null, average sample size under the alternative, the posterior and predictive thresholds associated with the design, as well as the distance to the upper left point on the plot. A `plot()` option is also available for the results of `calc_decision_rules`. In the one-sample case it produces a single plot showing the sample size at each interim analysis on the x-axis and the possible number of responses at each interim analysis on the y-axis. In the two-sample case a grid of plots is produced, with one plot for each interim analysis. The x-axis shows the number of possible responses in the control group and the y-axis shows the number of possible responses in the experimental group. In both cases, the boxes are colored green for a "proceed" decision and red for a "stop" decision for each combination and the hover box produced by **plotly** provides the details.

4 Package demonstration

In this section I will present a basic example to demonstrate the functionality included in the **ppseq**. The following section will present a more realistic case study.

First we install and load the **ppseq** package.

```
install.packages("ppseq")
```

```
library(ppseq)
```

Consider the case where we are interested in designing a trial to investigate how a new treatment impacts tumor response measured as a binary outcome of response versus no response. We know the current standard of care treatment results in a tumor response rate of 10%, and we wish to improve this by 30%. So we wish to test $H_0 : p \leq 0.1$ versus $H_1 : p \geq 0.4$, so we set `p_null = 0.1` and `p_alt = 0.4`. This is a rare disease so our maximum sample size is 15, so we set `N = 15`, and we will do interim analyses after every 5 patients, so we set `n = seq(5, 15, 5)`. We wish to examine designs based on combinations of posterior thresholds $\theta = 0.85, 0.90$ and predictive thresholds $\theta^* = 0.1, 0.2$, so we set `pp_threshold = c(0.85, 0.9)` and `ppp_threshold = c(0.1, 0.2)`. Finally, for computational speed in this basic example, we set `S=50` and `nsim=50`, but in practice we would want to use much larger values, in the thousands.

```
set.seed(123)
```

```
cal_thresh <-
  calibrate_thresholds(
    p_null = 0.1,
    p_alt = 0.4,
    n = seq(5, 15, 5),
    N = 15,
    pp_threshold = c(0.85, 0.9),
    ppp_threshold = c(0.1, 0.2),
    S = 50,
    nsim = 50
    )
```

Since there are only four design options in this toy example, we print the entire results table using a call to `print()`. Each row represents a different combination of the considered posterior and predictive thresholds, denotes `pp_threshold` and `ppp_threshold`, respectively.

```
print(cal_thresh)
```

```
#> # A tibble: 4 x 8
#>   pp_threshold ppp_threshold mean_n1_n~1 prop_~2 prop_~3 mean_~4 prop_~5 prop_~6
#>          <dbl>         <dbl>       <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
#> 1         0.85           0.1        12.1    0.1     0.42    14.8    0.92    0.02
#> 2         0.85           0.2         9.5    0.06    0.64    14.8    0.92    0.02
#> 3         0.9            0.1        10.8    0.04    0.5     14.8    0.9     0.02
#> 4         0.9            0.2         8.9    0.04    0.74    14.7    0.88    0.04
#> # ... with abbreviated variable names 1: mean_n1_null, 2: prop_pos_null,
#> #   3: prop_stopped_null, 4: mean_n1_alt, 5: prop_pos_alt, 6: prop_stopped_alt
```

We use the `optimize_design()` function to identify the optimal accuracy and optimal efficiency designs. We constrain consideration to designs with type I error between 0.025 and 0.1, specified by `type1_range = c(0.025, 0.1)`, and power of at least 0.75, specified by `minimum_power = 0.75`.

```
optimize_design(cal_thresh, type1_range = c(0.025, 0.1), minimum_power = 0.75)
```

```
#> $`Optimal accuracy design:`
#> # A tibble: 1 x 6
#>   pp_threshold ppp_threshold `Type I error` Power `Average N under the null`
#>          <dbl>         <dbl>          <dbl> <dbl>                       <dbl>
#> 1         0.85           0.2           0.06  0.92                         9.5
#>   `Average N under the alternative`
#>                               <dbl>
#> 1                              14.8
#>
#> $`Optimal efficiency design:`
#> # A tibble: 1 x 6
#>   pp_threshold ppp_threshold `Type I error` Power `Average N under the null`
#>          <dbl>         <dbl>          <dbl> <dbl>                       <dbl>
#> 1          0.9           0.2           0.04  0.88                         8.9
#>   `Average N under the alternative`
#>                               <dbl>
#> 1                              14.7
```

To ease interim analysis during the course of the trial, we obtain the decision table using the `calc_decision_rules()` function, with the thresholds of our selected optimal efficiency design passed as `theta = 0.9` and `ppp_threshold = 0.2`. For computational speed in this basic example, we set `S=50`, but in practice we would want to draw many more posterior samples, and values of 5000 or more should be considered. If any `NA` value was returned in this table, it would indicate that at that interim look you would never stop, regardless of the number of responses.

```
calc_decision_rules(
  n = seq(5, 15, 5),
  N = 15,
```

```
  theta = 0.9,
  ppp = 0.2,
  p0 = 0.1,
  S = 50
)

#> # A tibble: 3 x 3
#>       n     r   ppp
#>   <dbl> <int> <dbl>
#> 1     5     0  0.06
#> 2    10     1  0.1
#> 3    15     2  0
```

This table tells us that at the first interim analysis after we have observed $n = 5$ patients for response, we would stop the trial if $\leq r = 0$ responses had been observed, and would continue enrolling if $> r = 0$ responses had been observed. At the second interim analysis after we have observed $n = 10$ patients for response, we would stop the trial if $\leq r = 1$ responses had been observed, and would continue enrolling if $> r = 1$ responses had been observed. At the end of the trial after all $n = N = 15$ patients have been observed for response, we would declare the new treatment promising if $> r = 2$ responses had been observed.

5 Case study

The case study focuses on a re-design of a phase II study of atezolizumab in metastatic urothelial carcinoma patients (mUC) using sequential predictive probability monitoring. Atezolizumab is a programmed death-ligand 1 (PD-L1) blocking monoclonal antibody that was given accelerated approval by the U.S. Food and Drug Administration in May 2016 for the treatment of patients with locally advanced or metastatic urothelial carcinoma who had disease progression following platinum-containing chemotherapy. The approval was based on the results of a single-arm phase II study in 310 patients (Rosenberg et al. 2016). The phase II study used a hierarchical fixed-sequence testing procedure to test increasingly broad subgroups of patients based on PD-L1 status, and found overall response rates of 26% (95% CI: 18-36), 18% (95% CI: 13-24), and 15% (95% CI 11-19) in patients with $\geq$ 5% PD-L1-positive immune cells (IC2/3 subgroup), in patients with $\geq$ 1% PD-L1-positive immune cells (IC1/2/3 subgroup), and in all patients, respectively (Rosenberg et al. 2016). All three rates exceeded the historical control rate of 10%. Then, in March 2021, the approval in this indication was voluntarily withdrawn by the sponsor following negative results from a randomized phase III study (Powles et al. 2018). In the phase III study, 931 patients were randomly assigned to receive atezolizumab or chemotherapy in a 1:1 ratio, and the same hierarchical fixed-sequence testing procedure as in the phase II study was used. The phase III study found that overall survival did not differ significantly between the atezolizumab and chemotherapy groups of the IC2/3 subgroup (median survival 11.1 months [95% CI: 8.6-15.5] versus 10.6 months [95% CI: 8.4-12.2]), so no further testing was conducted for the primary endpoint (Powles et al. 2018). Further analyses revealed that while the response rates to atezolizumab were comparable to those seen in the phase II study, the response rates to chemotherapy were much higher than the historical control rate of 10%. The overall response rates to chemotherapy were 21.6% (95% CI: 14.5-30.2), 14.7% (95% CI: 10.9-19.2), and 13.4% (95% CI: 10.5-16.9) for the IC2/3 subgroup, IC1/2/3 subgroup, and all patients, respectively. The overall response rates to atezolizumab were 23% (95% CI: 15.6-31.9), 14.1% (95% CI: 10.4-18.5), and 13.4% (95% CI: 10.5-16.9) for the IC2/3 subgroup, IC1/2/3 subgroup, and all patients, respectively. These results indicate that PD-L1 status is a prognostic biomarker for both standard of care chemotherapies that comprised the control arm as well as atezolizumab in this patient population.

We wish to re-design the phase II trial of atezolizumab using a two-arm randomized design with sequential predictive probability monitoring. We will use the `calibrate_thresholds()` function to obtain the operating characteristics of designs based on on each combination of posterior and predictive thresholds. We focus here on the main biomarker subgroup of interest, the IC2/3 subgroup. We design the study with a null response rate of 0.1 in both arms, and an alternative response rate of 0.25 in the atezolizumab arm. So we set `p_null = c(0.1, 0.1)` and `p_alt = c(0.1, 0.25)`. We plan the study with 100 participants, assuming that the total sample size available is similar to the 310 used in the actual single-arm phase II trial, and that a third of that patient population fall into our desired biomarker subgroup. So we set `N = c(50, 50)` to indicate equal allocation of the 100 patients to each of the two arms. We will check for futility after every 10 patients are enrolled on each arm, so we set `n = cbind(seq(10, 50, 10), seq(10, 50, 10))`, a matrix showing the interim analysis sample sizes in each of the two arms. To design the study, we need to calibrate the design over a range of posterior probability thresholds and predictive probability thresholds. In this example

we will consider posterior thresholds of 0.9, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, and 0.99, and predictive thresholds of 0.05, 0.1, 0.15, and 0.2. So we set `pp_threshold = seq(0.9, 0.99, 0.01)` and `ppp_threshold = seq(0.05, 0.2, 0.05)`. We selected these posterior thresholds because we want to have a posterior probability of at least 0.9 that our response rate exceeds the null. And we selected these predictive thresholds because we would not want to stop the trial early for futility if there was more than a 20% chance of success when full enrollment was reached. We use the defaults for all remaining arguments, including the defaults of `S=5000` and `nsim=1000`, which we believe are sufficient to obtain stable results. Because of the inherent computational intensity in these calculations, this function relies on the **future** (Bengtsson 2021) and **furrr** (Vaughan and Dancho 2021) packages to parallelize computations. The user will be responsible for setting up a call to `future::plan()` that is appropriate to their operating environment and simulation setting. Because the code takes some time to run, the results of the below example code are available as a dataset called `two_sample_cal_tbl` included in the **ppseq** package.

```
library(future)

set.seed(123)

future::plan(future::multicore(workers = 40))

two_sample_cal_tbl <-
  calibrate_thresholds(
    p_null = c(0.1, 0.1),
    p_alt = c(0.1, 0.25),
    n = cbind(seq(10, 50, 10), seq(10, 50, 10)),
    N = c(50, 50),
    pp_threshold = seq(0.9, 0.99, 0.01),
    ppp_threshold = seq(0.05, 0.2, 0.05),
    direction = "greater",
    delta = 0,
    prior = c(0.5, 0.5),
    S = 5000,
    nsim = 1000
  )
```

We can compare the design options that meet our desired range of type I error and minimum power by passing the results of our call to the `calibrate_thresholds()` function to the `plot()` function. The `plotly = TRUE` option produces interactive visualizations or the `plotly = FALSE` option produces static visualizations.

```
plot(
  two_sample_cal_tbl,
  type1_range = c(0.05, 0.1),
  minimum_power = 0.7,
  plotly = TRUE
)
```

We see that the optimal efficiency design is the one with posterior threshold 0.92 and predictive threshold 0.05. It has a type I error of 0.07, power of 0.7, average sample size under the null of 29 per arm, and average sample size under the alternative of 46 per arm. This design would allow us to stop early if the treatment were inefficacious, thus preserving valuable financial resources for use in studying more promising treatments and preventing our human subjects from continuing an ineffective treatment.

```
optimize_design(
  two_sample_cal_tbl,
  type1_range = c(0.05, 0.1),
  minimum_power = 0.7
)
```

```
#> $`Optimal accuracy design:`
#> # A tibble: 1 x 6
#>   pp_threshold ppp_threshold `Type I error` Power `Average N under the null`
#>          <dbl>         <dbl>          <dbl> <dbl>                      <dbl>
```

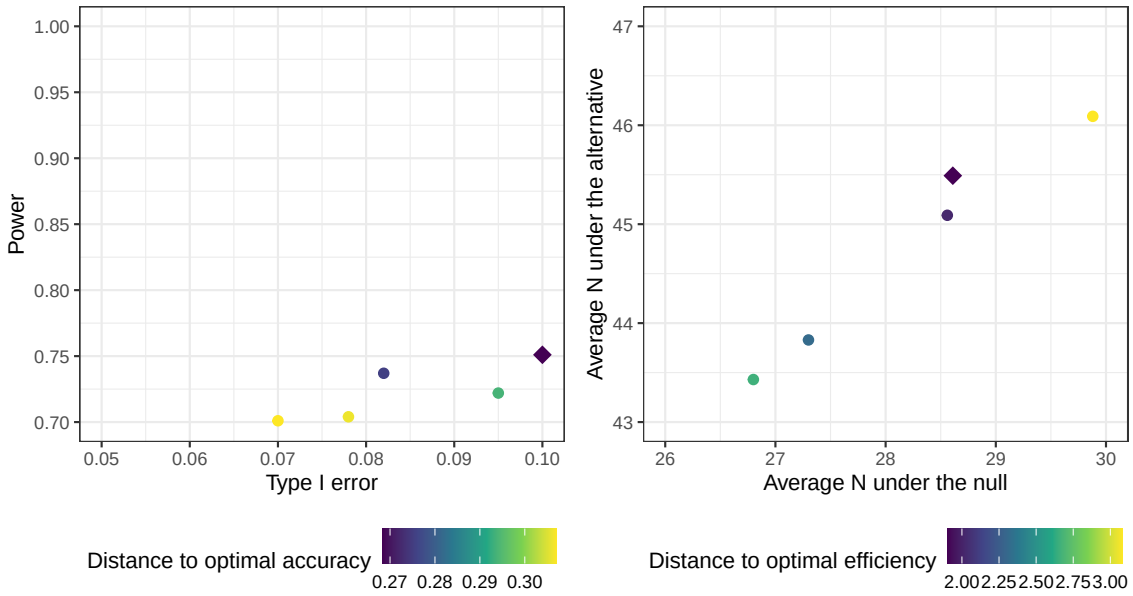


Figure 1: Plot of design options made with the ggplot2 package. The accuracy designs are on the left, with type I error on the x-axis and power on the y-axis, and the efficiency designs are on the right, with the average sample size under the null on the x-axis and the average sample size under the alternative on the y-axis. Each point represents one potential design based on a specific combination of posterior and predictive thresholds. The color represents the Euclidean distance to the top left point. The optimal design is indicated by a diamond. The optimal accuracy design is the one with posterior threshold 0.86 and predictive threshold 0.1. The optimal efficiency design is the one with posterior threshold 0.92 and predictive threshold 0.05.

```
#> 1         0.86          0.1           0.1 0.751                       28.6
#>   `Average N under the alternative`
#>                               <dbl>
#> 1                              45.1
#>
#> $`Optimal efficiency design:`
#> # A tibble: 1 x 6
#>   pp_threshold ppp_threshold `Type I error` Power `Average N under the null`
#>          <dbl>         <dbl>          <dbl> <dbl>                      <dbl>
#> 1         0.92          0.05           0.07 0.701                       28.6
#>   `Average N under the alternative`
#>                               <dbl>
#> 1                              45.5
```

Finally, we generate the decision table associated with the selected design for use in making decisions at each interim analysis during the conduct of the trial. Because of the computational time involved, the results of the below example code are available as a dataset called `two_sample_decision_tbl` included in the **ppseq** package. In the results table, we see that at the first interim futility look after just 5 patients, we would not stop the trial. After the first 10 patients we would stop the trial if there were 0 responses, and so on. At the end of the trial when all 95 patients have accrued, we would declare the treatment promising of further study if there were greater than or equal to 14 responses.

```
set.seed(123)

two_sample_decision_tbl <-
  calc_decision_rules(
    n = cbind(seq(10, 50, 10), seq(10, 50, 10)),
    N = c(50, 50),
    theta = 0.92,
    ppp = 0.05,
    p0 = NULL,
```

```
    direction = "greater",
    delta = 0,
    prior = c(0.5, 0.5),
    S = 5000
    )
```

The resulting table contains all possible combinations of responses in the two arms, so is quite long. We can visualize the resulting decision rules by passing the results of the call to the `calc_decision_rules()` function to the `plot()` function, which returns a faceted plot according to the interim look. The color denotes whether the trial should proceed (green) or stop (red) for a given combination of number of responses in the control arm (x-axis) and number of responses on the experimental arm (y-axis). For example, we see that at the second interim look when there are 20 patients enrolled on each arm, if there were 10 responses in the control arm, we would stop the trial if there were 8 or fewer responses on the experimental arm; otherwise we would proceed with enrollment.

```
dtp <- plot(two_sample_decision_tbl, plotly = FALSE)

dtp +
  theme(legend.position="bottom") +
  scale_fill_viridis_d()
```

6 Summary

With the focus of early stage clinical trial research in oncology shifting away from the study of cytotoxic treatments and toward immunotherapies and other non-cytotoxic treatments, new approaches to clinical trial design are needed that move beyond the traditional search for the maximum tolerated dose (Hobbs et al. 2019). Bayesian sequential predictive probability monitoring provides a natural and flexible way to expand the number of patients studied in phase 1 or to design phase 2 trials that allow for efficient early stopping for futility while maintaining control of type I error and power. The **ppseq** package implements functionality to evaluate a range of posterior and predictive thresholds for a given study design and identify the optimal design based on accuracy (i.e. type I error and power) or efficiency (i.e. average sample sizes under the null and alternative). Interactive visualization options are provided to ease comparison of the resulting design options. Once an ideal design is selected, a table of decision rules can be obtained to make trial conduct simple and straightforward.

References

Bengtsson, Henrik. 2021. "A Unifying Framework for Parallel and Distributed Processing in R using Futures." *The R Journal* 13 (2): 273–91. https://doi.org/10.32614/RJ-2021-048.

Bugano, D. D. G., K. Hess, D. L. F. Jardim, A. Zer, F. Meric-Bernstam, L. L. Siu, A. R. A. Razak, and D. S. Hong. 2017. "Use of Expansion Cohorts in Phase I Trials and Probability of Success in Phase II for 381 Anticancer Drugs." Journal Article. *Clin Cancer Res* 23 (15): 4020–26. https://doi.org/10.1158/1078-0432.Ccr-16-2354.

Dmitrienko, A., and M. D. Wang. 2006. "Bayesian Predictive Approach to Interim Monitoring in Clinical Trials." Journal Article. *Stat Med* 25 (13): 2178–95. https://doi.org/10.1002/sim.2204.

Hobbs, B. P., P. C. Barata, Y. Kanjanapan, C. J. Paller, J. Perlmutter, G. R. Pond, T. M. Prowell, et al. 2019. "Seamless Designs: Current Practice and Considerations for Early-Phase Drug Development in Oncology." *J Natl Cancer Inst* 111 (2): 118–28.

Hobbs, B. P., N. Chen, and J. J. Lee. 2018. "Controlled multi-arm platform design using predictive probability." *Stat Methods Med Res* 27 (1): 65–78.

Khoja, L., M. O. Butler, S. P. Kang, S. Ebbinghaus, and A. M. Joshua. 2015. "Pembrolizumab." *J Immunother Cancer* 3: 36.

Lee, J. J., and D. D. Liu. 2008. "A Predictive Probability Design for Phase II Cancer Clinical Trials." Journal Article. *Clin Trials* 5 (2): 93–106. https://doi.org/10.1177/1740774508089279.

Manji, A., I. Brana, E. Amir, G. Tomlinson, I. F. Tannock, P. L. Bedard, A. Oza, L. L. Siu, and A. R. Razak. 2013. "Evolution of Clinical Trial Design in Early Drug Development: Systematic Review of Expansion Cohort Use in Single-Agent Phase I Cancer Trials." Journal Article. *J Clin Oncol* 31 (33): 4260–67. https://doi.org/10.1200/jco.2012.47.4957.

Pestana, R. C., S. Sen, B. P. Hobbs, and D. S. Hong. 2020. "Histology-agnostic drug development - considering issues beyond the tissue." *Nat Rev Clin Oncol* 17 (9): 555–68.

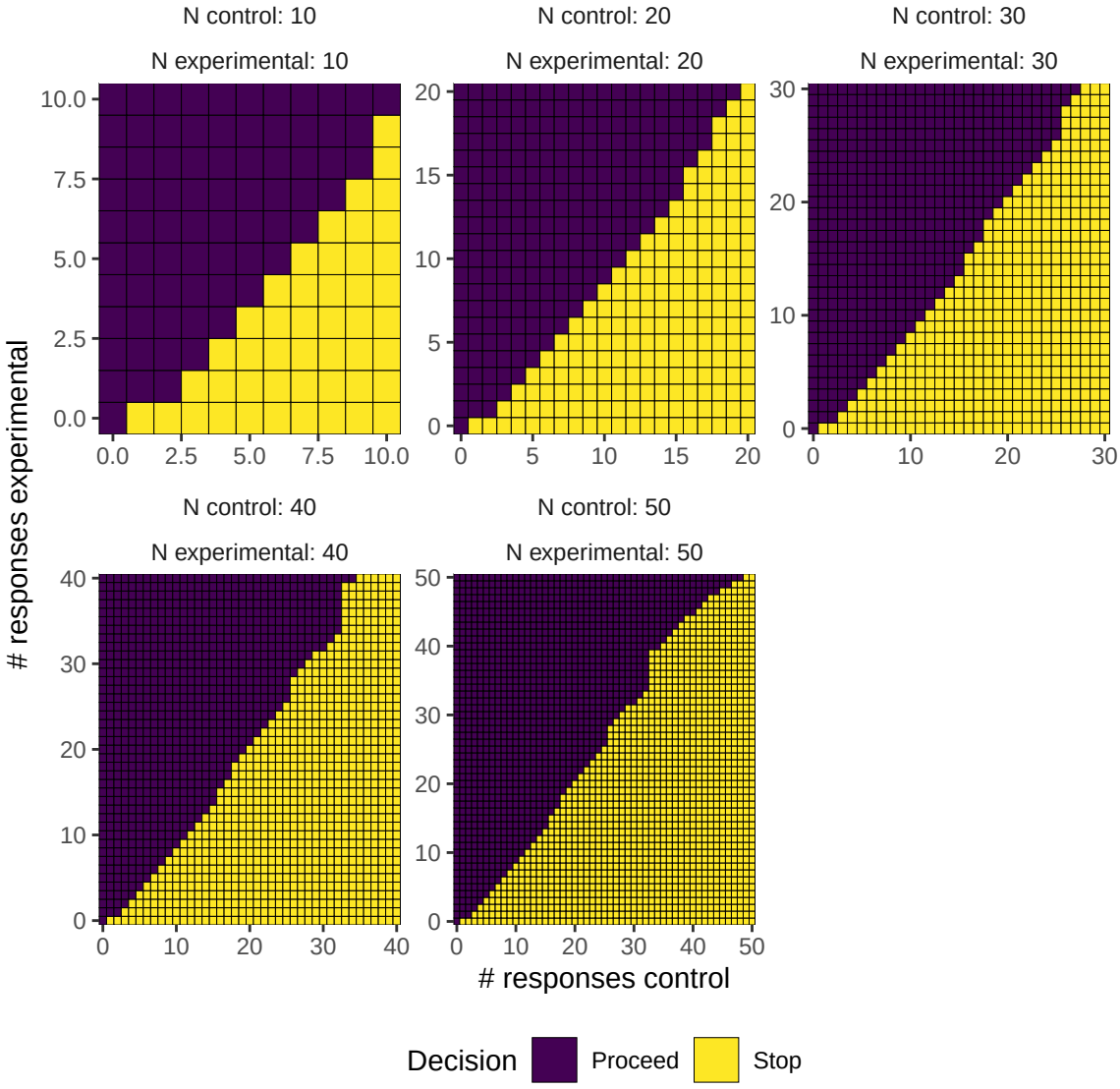


Figure 2: Plot of decision rules made with {ggplot2}. Each facet is for the total sample size at a pre-specified interim analysis. The number of responses in the control arm is on the x-axis and the number of responses in the experimental arm is on the y-axis. The fill color indicates whether the trial should stop (purple) or proceed (yellow) for a given combination of number of responses in the two arms at each interim analysis. At a given interim analysis, one can look up the number of observed responses to make a decision about whether the trial should stop or proceed enrollment.

Petrylak, D. P., T. Powles, J. Bellmunt, F. Braiteh, Y. Loriot, R. Morales-Barrera, H. A. Burris, et al. 2018. "Atezolizumab (MPDL3280A) Monotherapy for Patients with Metastatic Urothelial Cancer: Long-Term Outcomes from a Phase 1 Study." Journal Article. *JAMA Oncol* 4 (4): 537–44. https://doi.org/10.1001/jamaoncol.2017.5440.

Powles, T., I. Durán, M. S. van der Heijden, Y. Loriot, N. J. Vogelzang, U. De Giorgi, S. Oudard, et al. 2018. "Atezolizumab Versus Chemotherapy in Patients with Platinum-Treated Locally Advanced or Metastatic Urothelial Carcinoma (IMvigor211): A Multicentre, Open-Label, Phase 3 Randomised Controlled Trial." Journal Article. *Lancet* 391 (10122): 748–57. https://doi.org/10.1016/s0140-6736(17)33297-x.

Powles, T., J. P. Eder, G. D. Fine, F. S. Braiteh, Y. Loriot, C. Cruz, J. Bellmunt, et al. 2014. "MPDL3280A (Anti-PD-L1) Treatment Leads to Clinical Activity in Metastatic Bladder Cancer." Journal Article. *Nature* 515 (7528): 558–62. https://doi.org/10.1038/nature13904.

Rosenberg, J. E., J. Hoffman-Censits, T. Powles, M. S. van der Heijden, A. V. Balar, A. Necchi, N. Dawson, et al. 2016. "Atezolizumab in Patients with Locally Advanced and Metastatic Urothelial Carcinoma Who Have Progressed Following Treatment with Platinum-Based Chemotherapy: A Single-Arm, Multicentre, Phase 2 Trial." Journal Article. *Lancet* 387 (10031): 1909–20. https://doi.org/10.1016/s0140-6736(16)00561-4.

Saville, B. R., J. T. Connor, G. D. Ayers, and J. Alvarez. 2014. "The Utility of Bayesian Predictive Probabilities for Interim Monitoring of Clinical Trials." Journal Article. *Clin Trials* 11 (4): 485–93. https://doi.org/10.1177/1740774514531352.

Sievert, Carson. 2020. *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman; Hall/CRC. https://plotly-r.com.

Vaughan, Davis, and Matt Dancho. 2021. *furrr: Apply Mapping Functions in Parallel Using Futures*. https://CRAN.R-project.org/package=furrr.

Wickham, Hadley. 2016. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.

Emily C. Zabor
Department of Quantitative Health Sciences & Taussig Cancer Institute, Cleveland Clinic
9500 Euclid Ave. CA-60
Cleveland, OH 44195 USA
http://www.emilyzabor.com/
ORCiD: 0000-0002-1402-4498
zabore2@ccf.org

Brian P. Hobbs
Dell Medical School, The University of Texas at Austin
true
Austin, TX 78712
ORCiD: 0000-0003-2189-5846
brian.hobbs@austin.utexas.edu

Michael J. Kane
Department of Biostatistics, Yale University
60 College Street
New Haven, CT 06511
ORCiD: 0000-0003-1899-6662
michael.kane@yale.edu

pCODE: Estimating Parameters of ODE Models

by Haixu Wang and Jiguo Cao

Abstract The ordinary differential equation (ODE) models are prominent to characterize the mechanism of dynamical systems with various applications in biology, engineering, and many other areas. While the form of ODE models is often proposed based on the understanding or assumption of the dynamical systems, the values of ODE model parameters are often unknown. Hence, it is of great interest to estimate the ODE parameters once the observations of dynamic systems become available. The parameter cascade method initially proposed by Ramsay et al. (2007) is shown to provide an accurate estimation of ODE parameters from the noisy observations at a low computational cost. This method is further promoted with the implementation in the R package **CollocInfer** by Hooker et al. (2016). However, one bottleneck in using **CollocInfer** to implement the parameter cascade method is the tedious derivations and coding of the Jacobian and Hessian matrices required by the objective functions for doing estimation. We develop an R package **pCODE** to implement the parameter cascade method, which has the advantage that the users are not required to provide any Jacobian or Hessian matrices. Functions in the **pCODE** package accommodate users for estimating ODE parameters along with their variances and tuning the smoothing parameters. The package is demonstrated and assessed with four simulation examples with various settings. We show that **pCODE** offers a derivative-free procedure to estimate any ODE models where its functions are easy to understand and apply. Furthermore, the package has an online Shiny app at `https://pcode.shinyapps.io/pcode/`.

1 Introduction

The evolution of a dynamic system in time can be represented by ordinary differential equations (ODE) models consisting of a set of state variables and their derivatives. For example, the state variables can be the population of multiple species in an ecological system, the velocities, and locations of particles in a physics system, the concentration of chemicals in a reactor, or the energy of objects in a heating process. Usually, an ODE model is defined as a set of differential equations in the following form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t|\boldsymbol{\theta}), \tag{1}$$

where $\boldsymbol{x}(t)$ denotes the fully or partially observed states of the system at time t. The ODE model links the first derivative, the temporal change, of the states with $\boldsymbol{x}$ itself and other influential components $\boldsymbol{u}$ through some function $\boldsymbol{f}$ defined by the parameter vector $\boldsymbol{\theta}$.

Conventionally, parameter estimation of an ODE model combines numerically solving the system and updating the ODE parameters given the solution. The updating step complies with the nonlinear regression framework as suggested in Bard (1974) and Biegler et al. (1986). That is, the optimal $\boldsymbol{\theta}$ minimizes the sum of squared errors between the observations and the solutions of ODE models. It is not very typical for an ODE model to have analytical solutions unless the dynamic system is simple and linear. In more realistic situations where analytical solutions are not available, the ODE solutions must be numerically obtained prior to any estimations of the ODE parameters. The initial condition, which are values of state variables at the initial time, dictates the unique solution obtained by the numeric solver. However, the initial condition is often unavailable in real practices and must be estimated before obtaining the ODE solutions. If the initial conditions are unknown, then they are augmented to the parameter vector and estimated simultaneously with the ODE parameters. As a result, the complexity of optimization dramatically increases due to the fact that the solution is sensitive to ODE parameters and initial conditions. Also, the augmentation increases the dimension and complexity of the optimization surface hence hindering the convergence of optimization algorithms.

Being an improvement in the stability of estimation, the multiple shooting methods, presented in Bock (1981), Peifer and Timmer (2007), and Voss et al. (2004), expands the dimension of optimization surface by introducing additional initial conditions. The extra initial conditions are a result of partitioning the original time interval into multiple subintervals, and numeric solutions are obtained within each subinterval. Ideally, the introduction of these initial conditions will facilitate the optimization process for $\boldsymbol{\theta}$ to avoid local minima in the optimization surface and provide a faster convergence rate. Additionally, Huang et al. (2006) and Gelman et al. (1996) present Bayesian frameworks of estimating ODE parameters with a hierarchical model.

The aforementioned strategies depend on the numeric solutions of ODE models and can be

burdened with significant computational challenges given complex dynamic systems. That is, the system has to be numerically solved given each update or candidate of ODE parameters. To overcome these challenges, Varah (1982) proposes a two-step procedure to use nonparametric techniques for interpolating the time derivatives of state variables. Subsequently, the parameters are estimated through a nonlinear regression framework by treating the time derivatives as the response. This methodology is further developed in Chen and Wu (2008), Ramsay and Silverman (2005) , and Brunel (2008). Replacing numeric solutions with nonparametric estimations drastically reduces the computational cost, but estimation errors are introduced in the process of interpolating the time derivatives. The errors are due to the fact that there are no direct observations of the derivatives. Furthermore, Cao et al. (2012) shows that the estimation errors of derivatives will lead to bias in the estimation of ODE parameters.

As contrary to the two-step procedure, Ramsay et al. (2007) presents a new approximation strategy, called the parameter cascade method, in the family of collocation methods. The parameter cascade method integrates data smoothing and estimation of ODE parameters into a single estimation scheme. The parameter cascades method provides a much more efficient way of approximating the ODE solution without numerically solving the system. Additionally, it ensures that the ODE parameters are more accurately estimated than the two-step approach (Cao et al., 2012). Various well-known ODE models are examined in Ramsay et al. (2007). Qi and Zhao (2010) has discussed the asymptotic properties of the parameter cascades estimates for both ODE parameters and basis coefficients. The methodology has also been validated with many applications. Zhang et al. (2015) investigates the model selection problem in ODE models based on parameters estimated from the cascading procedure. When the system is measured with replications, mixed effects need to be incorporated into the model. This mixed-effect ODE model is explored in Wang et al. (2014).

The cascading estimation strategy is further promoted as in the R package **CollocInfer** by Hooker et al. (2016). **CollocInfer** offers a variety of functions regarding the estimation of ODE models. It has included the implementation of the parameter cascade method, the tuning criterion of smoothing parameter, and the sample variance of parameter estimates through the Newey-West method in Newey and West (1987). However, one bottleneck of using **CollocInfer** is the requirement of manually providing all the partial derivatives for constructing the Jacobian and Hessian matrices in the optimization process. For complex ODE models, the calculation and coding of those derivatives can be tedious and easy to make mistakes. As an improvement to the **CollocInfer** package, we propose a new package, named **pCODE**, which offers more user-friendly functions for estimating ODE models without specifying any derivatives. That is, an analytic Jacobian and Hessian matrix is required for the optimization routine in **CollocInfer**, whereas **pCODE** automatically calculates the numeric approximations to both matrices. Furthermore, **pCODE** also adds a bootstrap variance estimator besides the variance estimator obtained by the Delta method in Ramsay et al. (2007) and **CollocInfer**. **pCODE** uses the k-fold cross-validation for finding the optimal smoothing parameter while **CollocInfer** selects the smoothing parameters based on the forward prediction errors. We have also developed an online Shiny app at `https://pcode.shinyapps.io/pcode/`.

The remainder of this paper is organized as follows. Next section will introduce the parameter cascade methodology for simultaneous estimation of both ODE solutions $\boldsymbol{x}(t)$ and parameters $\boldsymbol{\theta}$. The third section explains the core functions of the package **pCODE** along with their usages and syntaxes. Subsequently, The fourth section illustrates the application of **pCODE** on several datasets with simulations for assessing the performance. At last, The fifth section gives a conclusion of the paper.

2 The parameter cascade method

An ODE model is build upon a set of differential equations $\dot{x}_i(t) = f_i(\boldsymbol{x}, t|\boldsymbol{\theta})$ for $i = 1, ..., I$, corresponding to the index of state variables. Each differential equation represents how the time derivative of one state variable depends on $\boldsymbol{x} = (x_1, ..., x_I)$. Furthermore, $f_i(\boldsymbol{x}, t|\boldsymbol{\theta})$'s are assumed to be parametrized by a vector of parameters $\boldsymbol{\theta}$. Let $y_{ij} = x_i(t_{ij}) + e_{ij}$ be the j-th noisy observation of i-th state variable at time t_{ij}, where e_{ij} is the measurement error or noise, $j = 1, ..., n_i$. We assume the vector of random errors $\boldsymbol{e}_i = (e_{i1}, ..., e_{in_i})$ has a distribution function $g(\boldsymbol{e}_i)$ with the distribution parameter $\boldsymbol{\sigma}_i$. Our goal is to estimate $\boldsymbol{\theta}$ from the noisy observations from the system.

The lower stream estimation of the parameter cascade method is to provide smooth interpolations of the observations. As suggested in Ramsay et al. (2007), the data interpolation part is done by the smoothing splines. The goal of smoothing is to diminish the random errors in the observations of all dimensions, hence the underlying solutions $x_i(t)$'s can be recovered from the noisy observations.

That is, each $x_i(t)$ is expressed as a linear combination of B-spline basis functions

$$x_i(t) = \sum_{i}^{K_i} c_{ik}\phi_{ik}(t) = \boldsymbol{c}_i'\boldsymbol{\phi}_i(t),$$

where K_i is the number of basis functions, and $\boldsymbol{\phi}_i(t)$ is the vector of basis functions evaluated at a time point t. Data smoothing is then equivalent to the estimation of these basis coefficients $\boldsymbol{c}_i$'s.

To comply the lower level estimation, a simple objective function $J(\boldsymbol{c})$, following the smoothing spline routine, is considered for obtaining basis coefficients $\boldsymbol{c} = (\boldsymbol{c}_1', \ldots, \boldsymbol{c}_I')'$. That is, the negative log-likelihood function is chosen for non-normal errors, whereas the least square criterion is naturally suitable for i.i.d. normal errors $e_{ij} \sim \mathrm{N}(0, \sigma_i)$. The summation $J(\boldsymbol{c}|\boldsymbol{\sigma}) = \sum_i J(\boldsymbol{c}_i|\boldsymbol{\sigma}_i)$ of individual fitting criterion over i defines a composite objective function for all I dimensions. The data interpolation will use a saturated number of basis functions which requires a penalty term in the objective function to prevent overfitting problems. Ramsay and Silverman (2005) suggests to use a linear differential operator for introducing a smoothness penalty in the objective functions. Subsequently, Poyton et al. (2006) utilizes the smoothness penalty in estimating parameters and solutions of ODE models. Following the same technique, the smoothness penalty is defined based on the discrepancy between the time derivatives and the ODE model, i.e.,

$$\int (\dot{x}_i(t) - f_i(\boldsymbol{x}, t|\boldsymbol{\theta}))^2 dt, \tag{2}$$

for each state variable x_i. Multiplying the penalty with a smoothing parameter λ_i and combining the penalties over all dimensions, the complete objective function is defined to be

$$J(\boldsymbol{c}|\boldsymbol{\theta}) = \sum_{i=1}^{I} \big[-\ln g(\boldsymbol{e}_i) + \lambda_i \int (\dot{x}_i(t) - f(\boldsymbol{x}, t|\boldsymbol{\theta}))^2 dt \big]. \tag{3}$$

For each given set of ODE parameters $\boldsymbol{\theta}$, the basis coefficients $\boldsymbol{c}$ is estimated by minimizing $J(\boldsymbol{c}|\boldsymbol{\theta})$. Hence, the estimate for the basis coefficients $\boldsymbol{c}$ become an implicit function of the ODE parameters $\boldsymbol{\theta}$ as $\boldsymbol{c}(\boldsymbol{\theta})$. The objective function (3), referred to as the inner objective function, needs to be optimized for estimating $\boldsymbol{c}$ whenever $\boldsymbol{\theta}$ is updated.

To clearly distinguish two sets of parameters $\boldsymbol{c}$ and $\boldsymbol{\theta}$, $\boldsymbol{c}$ will be referred to as the nuisance parameters since they are not essential for estimating the ODE model rather interpolate the observations. On the other hand, $\boldsymbol{\theta}$ is responsible for defining the structure of the ODE model and will be denoted as structural parameters. Often, $\boldsymbol{\theta}$ will be the primary concern given any ODE models.

The upper stream estimation of the parameter cascade method focuses on the structural parameter $\boldsymbol{\theta}$. To comply this estimation, the objective function $H(\boldsymbol{\theta})$, referred to as the outer objective function, is optimized with respect only to the structural parameters $\boldsymbol{\theta}$. It is usually defined to be the negative log-likelihood function or the sum of squared errors given the distributions of random errors, that is,

$$H(\boldsymbol{\theta}) = -\sum_{i=1}^{I} \ln\, g(\widehat{\boldsymbol{e}}_i|\boldsymbol{\theta}), \tag{4}$$

where

$$\widehat{\boldsymbol{e}}_i = \boldsymbol{y}_i - \hat{x}_i(\boldsymbol{t}_i|\boldsymbol{\theta}).$$

Here $\hat{x}_i(\boldsymbol{t}_i|\boldsymbol{\theta}) = \hat{\boldsymbol{c}}_i'(\boldsymbol{\theta})\boldsymbol{\phi}_i(\boldsymbol{t}_i)$, where $\hat{\boldsymbol{c}}_i'(\boldsymbol{\theta})$ is the optimizer of $J(\boldsymbol{c}|\boldsymbol{\theta})$ given current $\boldsymbol{\theta}$.

The parameter cascade method nests the estimation of basis coefficients in that of the structural parameters, where both estimations depend on the choice of smoothing parameters $\boldsymbol{\lambda}$. As a result, the interpolation of data points is not separated from estimating ODE parameters. The stream of dependencies defined a parameter cascade which offers great accuracy and efficiency in estimating ODE models. To take full advantage of the parameter cascade method, the **pCODE** package provides several R functions that are able to apply the aforementioned methodology for estimating ODE models.

3 Main functions in the pCODE package

Parameter estimation: `pcode`

The main function to perform the parameter cascade method is `pcode` in the package. This is a generic wrapper function for handling several scenarios given different objective functions, normal or non-normal errors, and missing state variables.

First, we focus on the situation when we have the normal random errors for all dimensions where $e_{ij} \sim \mathrm{N}(0, \sigma_i^2)$. In fact, both the inner optimization function (3) and the outer optimization function (4) are calculated based on a vector of residuals. The first part of (3) is easily recognized as the residuals from fitting the observation with basic functions. If the second part, the integral, is approximated by the composite Simpson's rule, then the approximation can be written in a quadratic form based on a vector of residual-like quantities. Stringing the aforementioned two vectors into one, and the concatenation of these vectors from all dimensions will produce a single vector of residuals. Hence, the optimization of the inner objective function adheres to the non-linear least square (NLS) framework. As a result, we can use the popular Levenberg-Marquart (LM) algorithm to obtain an estimate $\hat{\boldsymbol{c}}_i'(\boldsymbol{\theta})$. Our package `pcode` employs the function `lsqnonlin` from the package **PRACMA** (Borchers, 2019) for applying the LM algorithm.

The optimization of the outer objective function appears to be exactly an NLS problem. However, the parameter cascade strategy appends an extra layer of optimization to each update of the outer estimation, which characterizes a nested optimization problem. It is applicable to apply the Levenberg-Marquart algorithm again to the outer objective, however, the computational effort is much greater than that of inner optimization. The reason is that a Jacobian matrix needs to be calculated for each iteration of the update on the structural parameters, and each entry of the Jacobian for the outer objective involves a complete optimization of the inner one. This occurs to be the major delay in computation time.

Most commonly, the random errors e_{ij}'s are i.i.d from the normal distribution. However, some dynamic systems may produce non-normal errors which make the NLS approach not applicable. One solution is to perform a transformation on the observations and state variables, which forces the errors of transformed variables to have a normal distribution. As an alternative, `pcode` allows one to input a likelihood (or log-likelihood) function as a fitting criterion for both inner and outer objectives. In such a case, `optim` will be used for both levels of objective functions.

The following demonstrates the syntax of `pcode` with its argument list:

```
pcode(
  data, time, ode.model, par.names, state.names, likelihood.fun,
  par.initial, basis.list, lambda, controls
)
```

where `data`, a matrix, contains the observations of state variables as columns. The names of state variables are stored in `state.names` as a vector. `time` includes the observation time points in a vector. The ODE model to be estimated is provided to `pcode` as `ode.model` and defined in a similar way as that in package **deSolve** (Soetaert et al., 2010):

```
ode.model <- function(t, state, parameters) {
  with(as.list(c(state, parameters)), {
    dV <- c * (V - (V^3) / 3 + R)
    dR <- -(1 / c) * (V - a + b * R)
    return(list(c(dV, dR)))
  })
}
```

For this example, the state names are passed to the function `PCODE` as a vector `c('V','R')`, and the structural parameter names `c('a','b','c')` are given to `par.names` argument of the function. Optimization for the structural parameters requires a initial value input as `par.initial`, and there is no need for an input of `likelihood.fun` when the error distributions are Normal. `basis.list` contains the list of basis objects defined by the package **fda**, and the default `basis` has 9 interior knots with B-spline basis functions of order of 4. `lambda` corresponds to the penalty parameter $\boldsymbol{\lambda}$ which controls the fitness of estimated state variables to the differential equations. A scalar input of `lambda` means that all dimensions will subject to the same penalty, or a vector input differentiates the penalty calculated for each dimension. `controls` contains addition parameters to adjust optimizations for both inner and outer objective functions and obtain initial value for basis coefficients $\boldsymbol{c}$. Two scenarios are discussed depending on the different distribution of errors. All the mentioned scenarios will be demonstrated in the fourth section with examples.

The bootstrap variance estimator: `bootsvar`

All the functions of this package are derivative-free, hence the variance of structural parameters is numerically approximated. The first option is to use the bootstrap variance estimator. Given the estimation of both parameters $\boldsymbol{\theta}$ and $\boldsymbol{c}$, we are able to solve the ODE directly with estimated initial value $\hat{\boldsymbol{x}}(t_0)$. Hence, we can simulate bootstrap samples of the ODE model based on the estimated distributions of errors $\boldsymbol{e}_i$ for all I dimensions. The detailed algorithm for obtaining a bootstrap variance estimate of $\boldsymbol{\theta}$ is as follows

Algorithm 1 Bootstrap variance estimator

Initialization:

- $\boldsymbol{\theta}_0$: initial value for structural parameters
- B: the number of bootstrap samples
- $\boldsymbol{y}$: observations

function BOOTSVAR($\boldsymbol{\theta}_0, B, \boldsymbol{y}$...)
  $\boldsymbol{\theta}_\star, \boldsymbol{c}_\star \leftarrow$ results from `pcode(`$\boldsymbol{\theta}_0, \boldsymbol{y}$...`)` ▷ Estimation from the original data
  $\sigma_i^2 \leftarrow \text{var}(\boldsymbol{y}_i - \boldsymbol{x}_i)$ ▷ Estimate the variance of observation errors
  $\boldsymbol{x}_\star \leftarrow$ Solution to the ODE given $\boldsymbol{\theta}_\star$ and $\boldsymbol{x}_\star(t_0)$
  for $b \leftarrow 1$ to B **do**
    Obtain $\boldsymbol{z}_i^{(b)} = \boldsymbol{x}_{i,\star} + \boldsymbol{\epsilon}_i^{(b)}$. ▷ $\boldsymbol{\epsilon}_i^{(b)} \sim N(0, \sigma_i^2)$
    $(\boldsymbol{\theta}_\star^{(b)}, \boldsymbol{c}_\star^{(b)}) \leftarrow$ results from `pcode(`$\boldsymbol{\theta}_\star, \boldsymbol{z}_i^{(b)}$...`)`
  end for
  return $Var(\boldsymbol{\theta}_\star^{(b)}), Var(\boldsymbol{c}_\star^{(b)})$
end function

The syntax of `bootsvar` is illustrated as the following

```
bootsvar(
  data, time, ode.model, par.names, state.names, par.initial,
  lambda, basis.list, bootsrep, controls
)
```

Most of the arguments are the same as the function `pcode` with only one addition of argument `bootsrep` which indicates the number of bootstrap samples to be taken for obtaining the variance estimator.

The Delta variance estimator: `deltavar`

As an alternative to the bootstrap variance estimator, the package **PCODE** also offers another numeric estimator for the variance of structural parameters. Ramsay et al. (2007) has developed the approximation to $Var(\hat{\boldsymbol{\theta}}(\boldsymbol{y}))$ via the delta-method. The resulting approximation is of the form

$$Var(\hat{\boldsymbol{\theta}}(\boldsymbol{y})) \approx \left[\frac{d\hat{\boldsymbol{\theta}}}{d\boldsymbol{y}}\right] \boldsymbol{\Sigma} \left[\frac{d\hat{\boldsymbol{\theta}}}{d\boldsymbol{y}}\right]'$$

where $\frac{d\hat{\boldsymbol{\theta}}}{d\boldsymbol{y}}$ is obtained as

$$\frac{d\hat{\boldsymbol{\theta}}}{d\boldsymbol{y}} = -\left[\frac{\partial^2 H}{\partial \boldsymbol{\theta}^2}\bigg|_{\hat{\boldsymbol{\theta}}(\boldsymbol{y})}\right]^{-1} \left[\frac{\partial^2 H}{\partial \boldsymbol{\theta} \partial \boldsymbol{y}}\bigg|_{\hat{\boldsymbol{\theta}}(\boldsymbol{y})}\right] \tag{5}$$

and $\boldsymbol{\Sigma}$ is a $N \times N$ variance-covariance matrix for observations put into a vector. N is the total number of observation summing over all dimensions of $\boldsymbol{y}$. Then the estimation of variance relies on the computation of (5). The partial derivatives are approximated by the finite difference method, i.e.,

$$\left[\frac{\partial^2 H}{\partial \boldsymbol{\theta}_u^2}\bigg|_{\hat{\boldsymbol{\theta}}(\boldsymbol{y})}\right] \approx \frac{H(\hat{\boldsymbol{\theta}}_{u+}(\boldsymbol{y})|\boldsymbol{\lambda}) - 2H(\hat{\boldsymbol{\theta}}(\boldsymbol{y})|\boldsymbol{\lambda}) + H(\hat{\boldsymbol{\theta}}_{u-}(\boldsymbol{y})|\boldsymbol{\lambda})}{\Delta^2},$$

where $\hat{\boldsymbol{\theta}}_{u+}(\boldsymbol{y})$ and $\hat{\boldsymbol{\theta}}_{u-}(\boldsymbol{y})$ indicate the addition and subtraction of stepsize Δ to the u-th structural parameter estimate $\hat{\boldsymbol{\theta}}(\boldsymbol{y})$. The mixed partial derivatives are approximated as the following

$$\left[\left.\frac{\partial^2 H}{\partial \boldsymbol{\theta}_u \partial \boldsymbol{\theta}_v}\right|_{\hat{\boldsymbol{\theta}}(\boldsymbol{y})}\right] \approx \frac{H(\hat{\boldsymbol{\theta}}_{u+,v+}(\boldsymbol{y})) - H(\hat{\boldsymbol{\theta}}_{u-,v+}(\boldsymbol{y})) - H(\hat{\boldsymbol{\theta}}_{u+,v-}(\boldsymbol{y})) + H(\hat{\boldsymbol{\theta}}_{u-,v-}(\boldsymbol{y}))}{4\Delta^2}.$$

Given any fixed argument $\boldsymbol{\theta}$ for the outer objective function $H(\boldsymbol{\theta}, \boldsymbol{\sigma}|\boldsymbol{\lambda})$, its evaluation involves the profiled estimation of $\boldsymbol{c}(\boldsymbol{\theta}, \boldsymbol{\sigma}; |\boldsymbol{\lambda})$ and returns solely the likelihood or the sum of squared errors. Hence, individual evaluations of $H(\boldsymbol{\theta}, \boldsymbol{\sigma}|\boldsymbol{\lambda})$ used in numeric approximation is obtained in the following steps:

- Step 1: Optimize $J(\boldsymbol{c}|\boldsymbol{\theta}, \boldsymbol{\lambda})$ given $\boldsymbol{\theta}$.
- Step 2: Obtain $\hat{\boldsymbol{x}}(\boldsymbol{t})$ based on the estimated $\hat{\boldsymbol{c}}$.
- Step 3: Return $\sum_{i \in I} ||\boldsymbol{y}_i - \hat{x}_i(\boldsymbol{t}_i)||^2$ or $-\sum_{i \in I} g(\boldsymbol{y}_i - \hat{x}_i(\boldsymbol{t}_i)|\boldsymbol{\theta}, \boldsymbol{\lambda})$.

Approximation to the second term $\left.\frac{\partial^2 H}{\partial \boldsymbol{\theta} \partial \boldsymbol{y}}\right|_{\hat{\boldsymbol{\theta}}(\boldsymbol{y})}$ utilizes the finite difference method as well. After evaluating $H(\boldsymbol{\theta}, \boldsymbol{\sigma}|\boldsymbol{\lambda})$ given $\boldsymbol{\theta}$, the mixed partial derivative is calculated by moving the particular observation up or down by some stepsize Δ_y. That is,

$$\left[\left.\frac{\partial^2 H}{\partial \boldsymbol{\theta}_u \partial y_v}\right|_{\hat{\boldsymbol{\theta}}(\boldsymbol{y})}\right] \approx \frac{H(\hat{\boldsymbol{\theta}}_{u+}, \boldsymbol{y}_{v+}) - H(\hat{\boldsymbol{\theta}}_{u+}, \boldsymbol{y}_{v-}) - H(\hat{\boldsymbol{\theta}}_{u-}, \boldsymbol{y}_{v+}) + H(\hat{\boldsymbol{\theta}}_{u-}, \boldsymbol{y}_{v-})}{4\Delta\Delta_y},$$

where $\boldsymbol{y}_{v+}$ and $\boldsymbol{y}_{v-}$ represent moving up and down v-th observation by stepsize Δ_y in the last step of evaluating $H(\boldsymbol{\theta}, \boldsymbol{\sigma}|\boldsymbol{\lambda})$. The syntax of `numericvar` is based on that of `pcode`

```
numericvar(
  data, time, ode.model, par.names, state.names,
  par.initial, lambda, basis.list, stepsize, y_stepsize, controls
)
```

with addition of `stepsize` and `y_stepsize`. `stepsize` can be specified by a single number for which finite difference method will use it for all parameters or a vector where derivative of each parameter is estimated based on its own stepsize. `y_stepsize` allows an input of a vector where each element indicates the stepsize for each dimension of the ODE model.

4 Illustrations

A simple ODE model

A simple illustration uses an one-dimensional ODE model

$$\dot{X} = \theta X(1 - \frac{X}{10}). \tag{6}$$

The following code defines the aforementioned model that will be provided for `pcode` for estimating parameters:

```
model <- function(t, state, parameters) {
  with(as.list(c(state, parameters)), {
    dX <- theta * X * (1 - X / 10)
    return(list(dX))
  })
}
```

Given an observation period of $[0, 100]$, random noise errors are added to the ODE solution with a Normal distribution $\mathrm{N}(0, 0.5^2)$. Observations of the system are generated as follows:

```
times  <- seq(0, 100, length.out = 101)
mod    <- ode(y = state, times = times, func = ode.model, parms = model.par)
nobs   <- length(times)
scale  <- 0.5
noise  <- scale * rnorm(n = nobs, mean = 0, sd = 1)
observ <- mod[, 2] + noise
```

Subsequently, we can visualize the observations along the true solution of this simple ODE model in Figure 1.

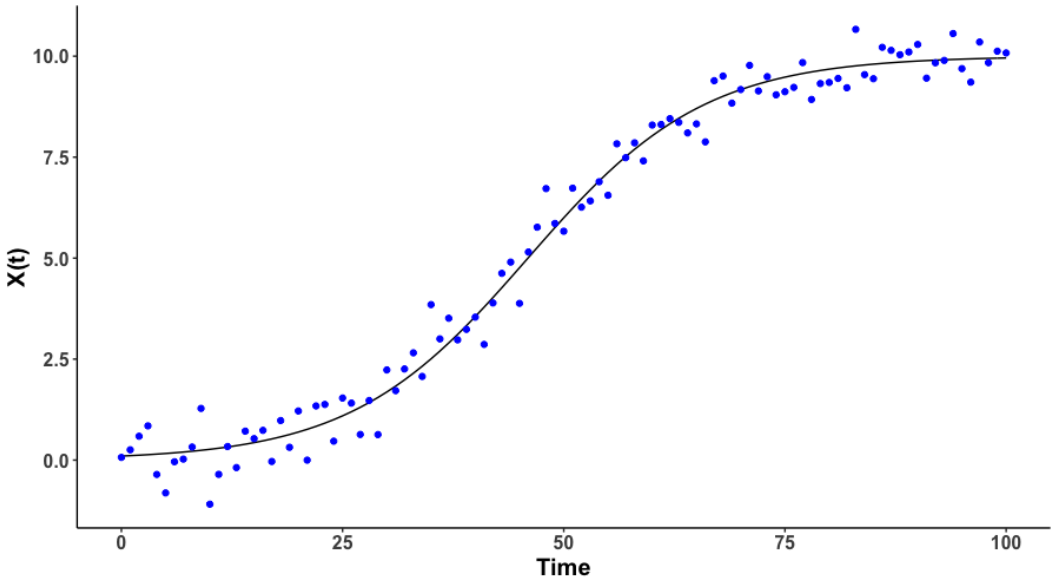


Figure 1: This plot demonstrates the solution of an ODE system. The black solid line represents the true ODE solution given a known initial value. The estimation of the solution and parameters of the ODE system usually depends on some noisy observations of the system. The noisy observations, the blue points, are used for estimation.

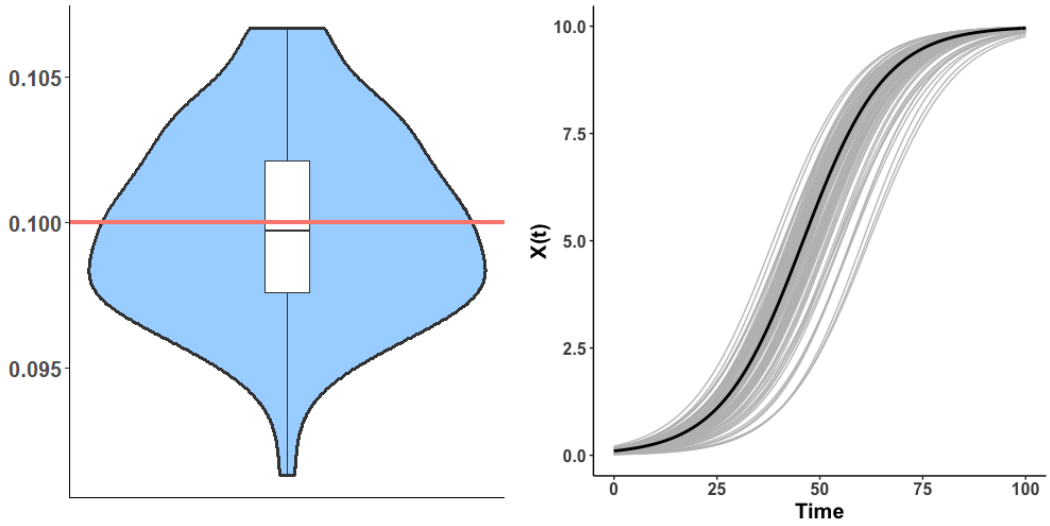
First, a basis object needs to be defined by `create.bspline.basis` from **fda** package

```
knots  <- seq(0, 100, length.out = 21)
norder <- 4
nbasis <- length(knots) + norder - 2
basis  <- create.bspline.basis(c(0, 100), nbasis, norder, breaks = knots)
```

A B-spline basis is created given 21 konts including both interior and boundary knots acorss observation interval $(0, 100)$. The order of basis functions is 4, so there is a total of 23 basis functions. To perform the parameter cascade method for estimating both structural and nuisance parameters, one can use `pcode` in the following way

```
pcode.result <- pcode(
  data = observ, time = times, ode.model = model,
  par.initial = 0.3, par.names = "theta", state.names = "X",
  basis.list = basis, lambda = 1e2
)
pcode.result["structural.par"]
theta
0.09995229
```

To validate the methodology of this function, data simulation and parameter estimation are repeated for 100 times.



(a) The violin and boxplot of the estimated θ over 100 replications.

(b) Solution of the ODE model with $\hat{\theta}$ and $\hat{X}(0)$.

Figure 2: The violin plot demonstrates the reliability of the estimation procedure. It summarizes the estimates of parameters over 100 replications. Each replication generates a entirely new set of noisy observations and gives a parameter estimate of the simple ODE model (6). The true parameter is indicated by the redline as $\theta = 0.1$. The right plot shows the estimation of the solution from the simple ODE model over 100 replications. Each grey line is obtained from solving the ODE system after obtaining the parameter estimate from one replication. The black curve is the ODE solution with the true value of θ.

Given that $\theta = 0.1$ is used for generating data, the parameter cascade method performs impressively well. The violin plot in Figure 2 shows that the distribution of estimates of θ covers 0.1 (indicated by the red horizontal line) and does not fall far away from the true value. Also, the state variable `X` (the bold black curve of the plot on the right) is well predicted by solving the ODE model with estimated initial value $\hat{x}(0)$ and structural parameters $\hat{\theta}$.

The true variance for data generating parameter θ is 1.003×10^{-5}. We can compare the performance of two functions `bootsvar` and `deltavar` for estimating $\text{var}(\theta)$.

```
bootsvar(
  data = observation, time = times, ode.model = ode.model,
  par.initial = 0.3, par.names = "theta", state.names = "X",
  basis.list = basis, lambda = 1e2, bootsrep = 20
)
theta
1.042763e-05

deltavar(
  data = observation, time = times, ode.model = ode.model,
  par.initial = 0.3, par.names = "theta", state.names = "X",
  basis.list = basis, lambda = 1e2,
  stepsize = 1e-5, y_stepsize = 1e-5
)
theta
9.923318e-06
```

Both variance estimator give excellent estimates of the true variance of structural parameter. Subsequently, the consistency and reliability of those estimator can be inspected by simulation. In 100 simulation replicates, the results of two variance estimators are summarized in Figure 3.

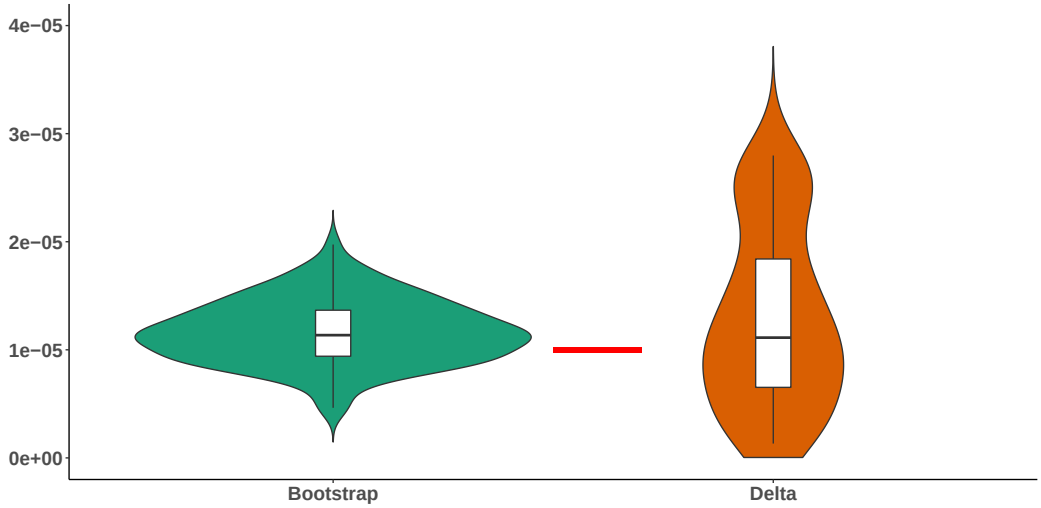


Figure 3: This plot demonstrates the estimation of variance of parameters over 100 replications. Two violin plots are results from two methods, the bootstrap and Delta method, introduced in the article. The red line segment indicates the true variance. Green plot includes the estimates from the bootstrap method, whereas the origin plot includes those from the Delta method.

The bootstrap variance estimator has a smaller variation but at the cost of higher computational effort in comparison with the Delta variance estimator. On the other hand, the Delta variance estimator is faster but sensitive to the choice of the step size in estimating all the derivatives and the subsequent var(θ).

The FitzHugh-Nagumo ODE model

The FitzHugh-Nagumo ODE model is well known for capturing the dynamic system of neuronal firings based on the membrane potential V and recovery variable R. This two-dimensional ODE contains 3 positive parameters $\boldsymbol{\theta} = (a, b, c)$, and it can be defined as

$$\dot{V} = c(V - \frac{V^3}{3} + R),$$
$$\dot{R} = -\frac{1}{c}(V - a + bR).$$

The parameters are assigned with values $(a, b, c) = (0.2, 0.2, 3)$ the same as in Ramsay et al. (2007). The state variables V and R should behave as the black solid line plotted in Figure 4. Following the routine of specifying an ODE model, the FitzHugh-Nagumo model is defined in the following function with simulated observations

```
ode.model <- function(t, state, parameters) {
  with(as.list(c(state, parameters)), {
    dV <- c * (V - (V^3) / 3 + R)
    dR <- -(1 / c) * (V - a + b * R)
    return(list(c(dV, dR)))
  })
}
model.par   <- c(a = 0.2, b = 0.2, c = 3)
desolve.mod <- ode(y = state, times = times, func = ode.model, parms = c(0.2, 0.2, 3))
nobs    <- length(times)
scale   <- 0.1
noise_v <- scale * rnorm(n = nobs, mean = 0, sd = 1)
noise_r <- scale * rnorm(n = nobs, mean = 0, sd = 1)
observ  <- matrix(NA, nrow = length(times), ncol = 3)
observ[, 1] <- times
observ[, 2] <- desolve.mod[, 2] + noise_v
observ[, 3] <- desolve.mod[, 3] + noise_r
```

The data generating functions and simulated data are illustrated in Figure 4.

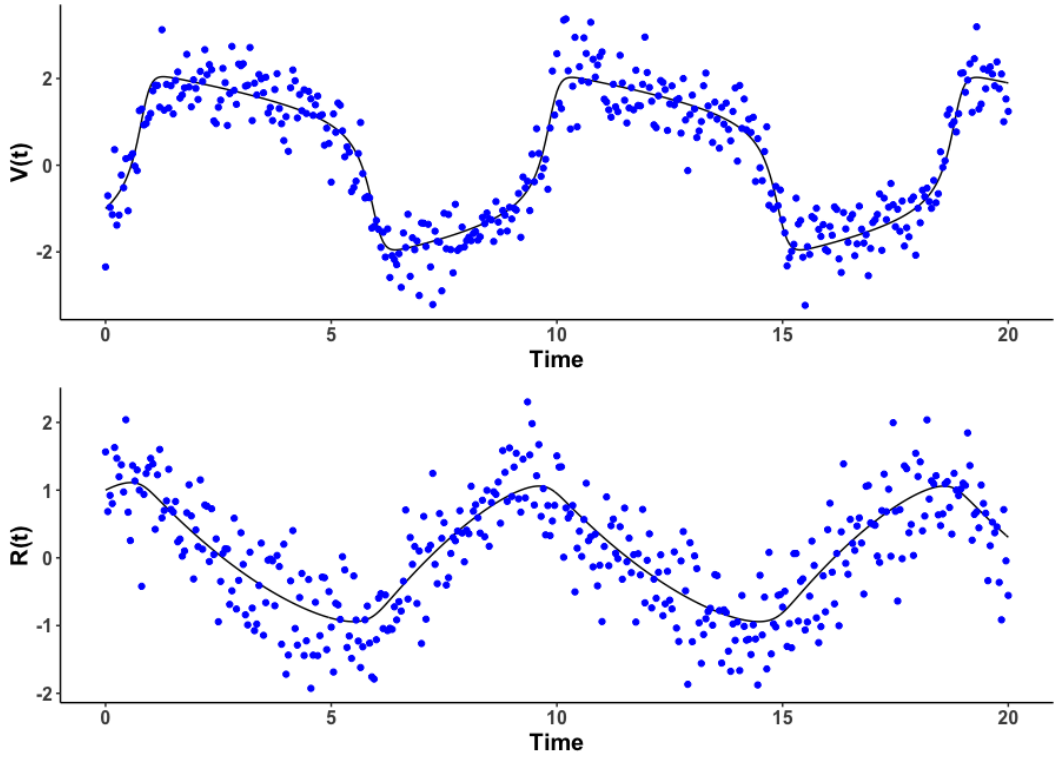


Figure 4: Given the known FitzHugh-Nagumo model, we are able to generate the true solutions. In this case, we set the parameters to be $(a, b, c) = (0.2, 0.2, 3)$ and are able to generate the true solution of the system, $V(t)$ and $R(t)$. The top graph demonstrates the solution $V(t)$, the black line, whereas the bottom graph shows the solution $R(t)$. For parameter estimations, we have added random noises to the true solution and use observations as the blue points in both graphs.

First step of estimating parameters is to declare basis object for each state variable. For observation period from time 0 to time 20, the basis is defined on 101 knots with B-spline basis function of order of 4. The total number of nuisance parameters is 103. For this example, the same basis will be used for both dimensions V and R:

```
knots  <- seq(0, 20, length.out = 101)
norder <- 4
nbasis <- length(knots) + norder - 2
basis  <- create.bspline.basis(c(0, 20), nbasis, norder, breaks = knots)
basis.list <- list(basis, basis)
```

Then, `pcode` can be executed as follows:

```
pcode.result <- pcode(
  data = observ[, 2:3], time = times, ode.model = Dmodel,
  par.names = c("a", "b", "c"), state.names = c("V", "R"),
  par.initial = rnorm(3), lambda = 1e2,
  basis.list = basis.list
)

pcode.result['structural.par']
0.1953324 0.2203495 2.9269980
```

In addition, we can also compare the estimation performance of the proposed package **pCODE** with the existing package **CollocInfer**. Figure 5 summarizes the comparison between two packages on the parameter estimation of the Fitz-Hugh Nagumo model.

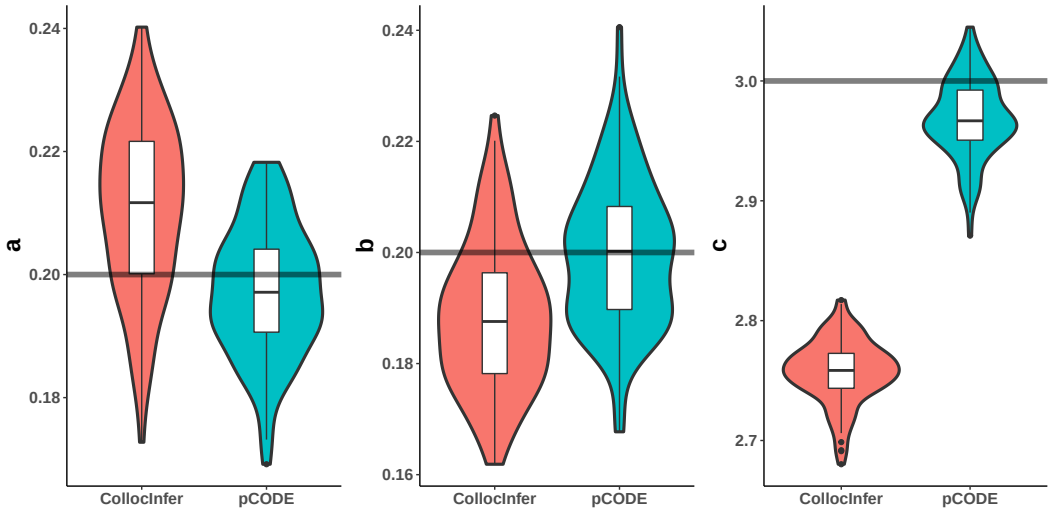


Figure 5: We have repeated the data generation and parameter estimation for 100 times given the FitzHugh-Nagumo model. The parameter estimates of (a, b, c) are summarized in the violin plots ordered from left to right. Within each replication, we also use the `CollocInfer` package for parameters estimation. In each plot, the blue violion plot contains estimates from `pCODE` whereas the red one contains those from `CollocInfer`. The black horizontal lines correspond to the true model parameters used for simulating data sets.

Package **CollocInfer** requires the Jacobian and Hessian matrices for estimating the parameters of the ODE models, whereas the proposed **pCODE** package is a derivative-free method. Both packages are estimating parameters a and b with satisfaction, and the estimates from both methods cover the true values. However, we can see that **pCODE** produced better estimations for the third parameter c in the model. Our package does not depend on users to provide tedious details but still performs equally well in parameter estimations.

As a part of the estimation routine, we can use two methods for estimating the variance of structural parameters. First, the usage of bootstrap variance estimator is

```
bootsvar.res <- bootsvar(
  data = data, time = observ[, 1], ode.model = ode.model,
  par.names = c("a", "b", "c"), state.names = c("V", "R"),
  par.initial = c(0.15, 0.25, 5), lambda = 1e2, basis.list = basis.list,
  controls = list(smooth.lambda = 10, verbal = 1, maxeval = 20), bootsrep = 20
)
```

based on a bootstrap sample with size of 20. Delta variance estimator is much faster in approximating the variance through the following execution

```
deltavar.res <- deltavar(
  data = observ[, 2:3], time = observ[, 1], ode.model = Dmodel,
  par.names = c("a", "b", "c"), state.names = c("V", "R"), par.initial = c(0.1, 0.3, 4),
  lambda = 1e2, basis.list = basis.list,
  stepsize = 0.001, y_stepsize = 0.001
)
```

Non-normal errors

The previous examples assume that the random errors are following Normal distributions, hence the optimization utilizes the routines of NLS problems. In some cases, if the state variables are bounded or the errors have a non-normal distribution, a likelihood (or log-likelihood) function can be passed to the function `pcode` in order to evaluate the objective functions. Subsequently, the function `pcode` will be using `optim` for optimizations of both inner and outter objective functions.

The first example would reuse the simple ODE model, i.e.,

$$\dot{X} = \theta X(1 - \frac{X}{10}).$$

In this case, the observations are simulated where log-normal errors are added to the solution of the ODE model in the following way:

$$log(y_i) = log(x_i) + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$. One simulated data is illustrated in the following Figure 6:

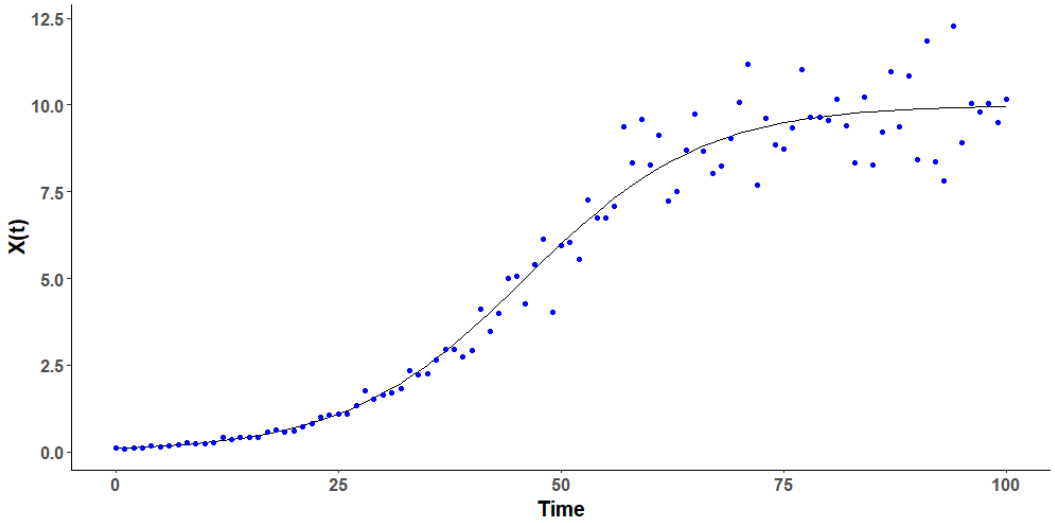


Figure 6: This plots illustrates that observations from ODE systems can follow a non-normal distribution. Given the previous simple ODE model, we generate random noises from a log-normal distribution and add them to the true solution of the system. Black line represents the solution of the simple ODE system, and simulated observations are indicated by blue points.

While keeping all the original settings for `pcode`, a log-likelihood function, as $g(\boldsymbol{e}_i|\cdot)$ in both 3 and 4, is additionally specified as follows

```
likfun <- function(x) {
  res <- lapply(x, function(t)
    dnorm(t,
      mean = 0,
      sd = 0.1, log = TRUE
    )
  )
  return(sum(unlist(res)))
}
```

Instead of defining the likelihood only as a function of residuals, `pcode` allows full flexibility in constructing a likelihood as long as it returns the evaluation. Then, the likelihood function can be passed to `pcode` through the argument `likelihood.fun` with specification of a basis object for interpolation

```
knots <- seq(0, 100, length.out = 21)
norder <- 4
nbasis <- length(knots) + norder - 2
basis <- create.bspline.basis(c(0, 100), nbasis, norder, breaks = knots)
lkh.result <- pcode(
  data = observ, time = times, likelihood.fun = likfun,
  par.initial = 0.1, ode.model = ode.model,
  basis.list = basis, par.names = "theta", state.names = "X", lambda = 1e2
)
lkh.result["structural.par"]
0.105
```

5 Summary

In this article, we have reviewed the parameter cascade method for estimating ordinary differential equations and introduced the new package **pCODE** for implementing this method. **pCODE** offers a

derivative-free procedure to estimate any ODE models where its functions are easily understood and to apply. Several examples of ODE models are considered for assessing the performance of functions from **pCODE** involving estimating parameters, producing variability measures, and tuning hyper-parameters. Subsequently, we can observe that the implemented functions provide satisfactory results presented with details in the fourth section. Type of examples differs in both model complexity and error distribution. Furthermore, a special case of predator-prey model is studied when some state variables are completely missing from observations. The package is able to simplifies the application procedure and reproduce the estimation results as in Cao et al. (2008).

One of the future works is to expand the flexibility of this package. Even though **pCODE** can provide satisfactory parameter estimates, current functions do not allow users to input the Jacobian and Hessian matrices to speed up the optimization process. In future package developments, we will implement the functionality to allow the input of these matrices. One limitation of the package is the time performance of procedures, especially for the calculation of the bootstrap variance estimator. For that matter, we are implementing this function with parallel computation ability to speed up the calculation. Moreover, we plan to expand the functionalities of the package to include data visualizations and generations of diagnostic plots and summaries.

Bibliography

Y. Bard. *Nonlinear parameter estimation*. Academic Press, New York, 1974. [p291]

L. T. Biegler, J. J. Damiano, and G. E. Blau. Nonlinear parameter estimation: A case study comparison. *AIChE Journal*, 32(1):29–45, 1986. [p291]

H. G. Bock. Numerical treatment of inverse problems in chemical reaction kinetics. In K. H. Ebert, P. Deuflhard, and W. Jäger, editors, *Modelling of Chemical Reaction Systems*, pages 102–125, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg. [p291]

H. W. Borchers. *pracma: Practical Numerical Math Functions*, 2019. URL https://CRAN.R-project.org/package=pracma. R package version 2.2.5. [p294]

N. J.-B. Brunel. Parameter estimation of ODE's via nonparametric estimators. *Electron. J. Statist.*, 2:1242–1267, 2008. doi: 10.1214/07-EJS132. URL https://doi.org/10.1214/07-EJS132. [p292]

J. Cao, G. F. Fussmann, and J. Ramsay. Estimating a predator-prey dynamical model with the parameter cascades method. *Biometrics*, 64(3):959–967, 2008. [p303]

J. Cao, J. Huang, and H. Wu. Penalized nonlinear least squares estimation of time-varying parameters in ordinary differential equations. *Journal of Computational and Graphical Statistics*, 21(1):42–56, 2012. doi: 10.1198/jcgs.2011.10021. PMID: 23155351. [p292]

J. Chen and H. Wu. Efficient local estimation for time-varying coefficients in deterministic dynamic models with applications to HIV-1 dynamics. *Journal of the American Statistical Association*, 103 (481):369–384, 2008. [p292]

A. Gelman, F. Bois, and J. Jiang. Physiological pharmacokinetic analysis using population modeling and informative prior distributions. *Journal of the American Statistical Association*, 91(436): 1400–1412, 1996. [p291]

G. Hooker, J. Ramsay, and L. Xiao. Collocinfer: Collocation inference in differential equation models. *Journal of Statistical Software, Articles*, 75(2):1–52, 2016. ISSN 1548-7660. doi: 10.18637/jss.v075.i02. URL https://www.jstatsoft.org/v075/i02. [p291, 292]

Y. Huang, D. Liu, and H. Wu. Hierarchical bayesian methods for estimation of parameters in a longitudinal HIV dynamic system. *Biometrics*, 62(2):413–423, 2006. [p291]

W. K. Newey and K. D. West. A Simple, Positive Semi-definite, Heteroskedasticity and Autocorrelation Consistent Covariance Matrix. *Econometrica*, 55(3):703–708, May 1987. [p292]

M. Peifer and J. Timmer. Parameter estimation in ordinary differential equations for biochemical processes using the method of multiple shooting. *IET systems biology*, 1(2):78–88, 2007. [p291]

A. Poyton, M. Varziri, K. Mcauley, P. Mclellan, and J. Ramsay. Parameter estimation in continuous-time dynamic models using principal differential analysis. *Computers & Chemical Engineering*, 30 (4):698–708, 2006. [p293]

X. Qi and H. Zhao. Asymptotic efficiency and finite-sample properties of the generalized profiling estimation of parameters in ordinary differential equations. *The Annals of Statistics*, 38(1):435–481, 02 2010. doi: 10.1214/09-AOS724. URL https://doi.org/10.1214/09-AOS724. [p292]

J. Ramsay and B. W. Silverman. *Functional data analysis*. New York: Springer, New York, 2nd edition, 2005. [p292, 293]

J. Ramsay, G. Hooker, D. Campbell, and J. Cao. Parameter estimation for differential equations: A generalized smoothing approach. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 69(5):741–796, 2007. [p291, 292, 295, 299]

K. Soetaert, T. Petzoldt, and R. W. Setzer. Solving differential equations in R: Package deSolve. *Journal of Statistical Software*, 33(9):1–25, 2010. ISSN 1548-7660. doi: 10.18637/jss.v033.i09. URL http://www.jstatsoft.org/v33/i09. [p294]

J. Varah. A spline least squares method for numerical parameter estimation in differential equations. *SIAM Journal on Scientific and Statistical Computing*, 3(1):28–46, 1982. doi: 10.1137/0903003. [p292]

H. U. Voss, J. Timmer, and J. Kurths. Nonlinear dynamic system identification from uncertain and indirect measurements. *International Journal of Bifurcation and Chaos*, 14(06):1905–1933, 2004. [p291]

L. Wang, J. Cao, J. Ramsay, D. M. Burger, C. J. Laporte, and J. K. Rockstroh. Estimating mixed-effects differential equation models. *Statistics and Computing*, 24(1):111–121, Jan. 2014. ISSN 0960-3174. doi: 10.1007/s11222-012-9357-1. URL http://dx.doi.org/10.1007/s11222-012-9357-1. [p292]

X. Zhang, J. Cao, and R. J. Carroll. On the selection of ordinary differential equation models with application to predator-prey dynamical models. *Biometrics*, 71(1):131–138, 2015. doi: 10.1111/biom.12243. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/biom.12243. [p292]

Haixu Wang
Simon Fraser University
8888 University Dr, Burnaby, BC V5A 1S6
Canada
haixuw@sfu.ca

Jiguo Cao
Simon Fraser University
8888 University Dr, Burnaby, BC V5A 1S6
Canada
jiguoc@sfu.ca

TreeSearch: Morphological Phylogenetic Analysis in R

by Martin R. Smith

Abstract TreeSearch is an R package for phylogenetic analysis, optimized for discrete character data. Tree search may be conducted using equal or implied step weights with an explicit (albeit inexact) allowance for inapplicable character entries, avoiding some of the pitfalls inherent in standard parsimony methods. Profile parsimony and user-specified optimality criteria are supported. A graphical interface, which requires no familiarity with R, is designed to help a user to improve the quality of datasets through critical review of underpinning character codings; and to obtain additional information from results by identifying and summarizing clusters of similar trees, mapping the distribution of trees, and removing 'rogue' taxa that obscure underlying relationships. Taken together, the package aims to support methodological rigour at each step of data collection, analysis, and the exploration of phylogenetic results.

1 Introduction

Even in the phylogenomic era, morphological data make an important contribution to phylogenetic questions. Discrete phenotypic data improve the accuracy and resolution of phylogenetic reconstruction even when outnumbered by molecular characters, and are the only way to incorporate the unique perspective on historical events that fossil taxa provide (Wiens 2004; Wortley and Scotland 2006; Koch and Parry 2020; Asher and Smith 2022).

One challenge with morphological analysis is the treatment of inapplicable character states: for example, 'tail colour' cannot logically be ascribed either of the states 'red' or 'blue' in a taxon that lacks a tail (W. P. Maddison 1993). This situation can profoundly mislead phylogenetic analysis, and is not handled appropriately by any standard Markov model or parsimony method.

Solutions to this issue have recently been proposed (De Laet 2005; Brazeau, Guillerme, and Smith 2019; Tarasov 2019, 2022; Goloboff et al. 2021; Hopkins and St. John 2021). Where a single 'principal' character (e.g. 'tail') exhibits n 'contingent' characters (e.g. 'tail colour', 'tail covering'), 'exact' solutions (Tarasov 2019, 2022; Goloboff et al. 2021) require the construction of multi-state hierarchies containing $O(2^n)$ entries, meaning that analysis is only computationally tractable for simple hierarchies with few contingent characters. Moreover, these approaches cannot accommodate characters that are contingent on more than one principal character: for example, characters describing appendages on a differentiated head may be contingent on the presence of the two characters 'appendages' and 'differentiated head'.

Such situations can be approached using the flexible parsimony approximation proposed by Brazeau, Guillerme, and Smith (2019). **TreeSearch** scores trees using the "Morphy" C implementation of this algorithm (Brazeau, Smith, and Guillerme 2017). Morphy implements tree search under equal step weights. TreeSearch additionally implements implied step weighting (Goloboff 1993), a method which consistently finds more accurate and precise trees than equal weights parsimony (Goloboff et al. 2008; Goloboff, Torres, and Arias 2018; M. R. Smith 2019a).

There has been lively discussion as to whether, with the rise of probabilistic approaches, parsimony remains a useful tool for morphological phylogenetics (e.g. O'Reilly et al. 2016; Puttick et al. 2017; Sansom et al. 2018; Goloboff, Torres Galvis, and Arias 2018; Brown et al. 2017). Notwithstanding scenarios that go beyond the limits of parsimony, such as the simultaneous incorporation of stratigraphic data and other prior knowledge (e.g. Guenser et al. 2021), neither parsimony nor probabilistic methods consistently recover 'better' trees when gains in accuracy are balanced against losses in precision (M. R. Smith 2019a). Even if probabilistic methods may eventually be improved through the creation of more sophisticated models that better reflect the nature of morphological data (Goloboff, Torres, and Arias 2018; Tarasov 2019, 2022), parsimony analysis remains a useful tool – not only because treatments of inapplicable character states are presently available, but also because it facilitates a deeper understanding of the underpinning data by emphasizing the reciprocal relationship between a tree and the synapomorphies that it implies.

Whatever method is used to find phylogenetic trees, a single consensus tree may fail to convey all the signal in a set of phylogenetic results (Wilkinson 1994, 1996, 2003). A set of optimal trees can be better interpreted by examining consensus trees generated from clusters of similar trees (Stockham, Wang, and Warnow 2002); by exploring tree space (Wright and Lloyd 2020; M. R. Smith 2022a) and by automatically identifying, annotating and removing 'wildcard' taxa (M. R. Smith 2022b) whose 'rogue' behaviour may reflect underlying character conflict or ambiguity (Kearney 2002). These methods are

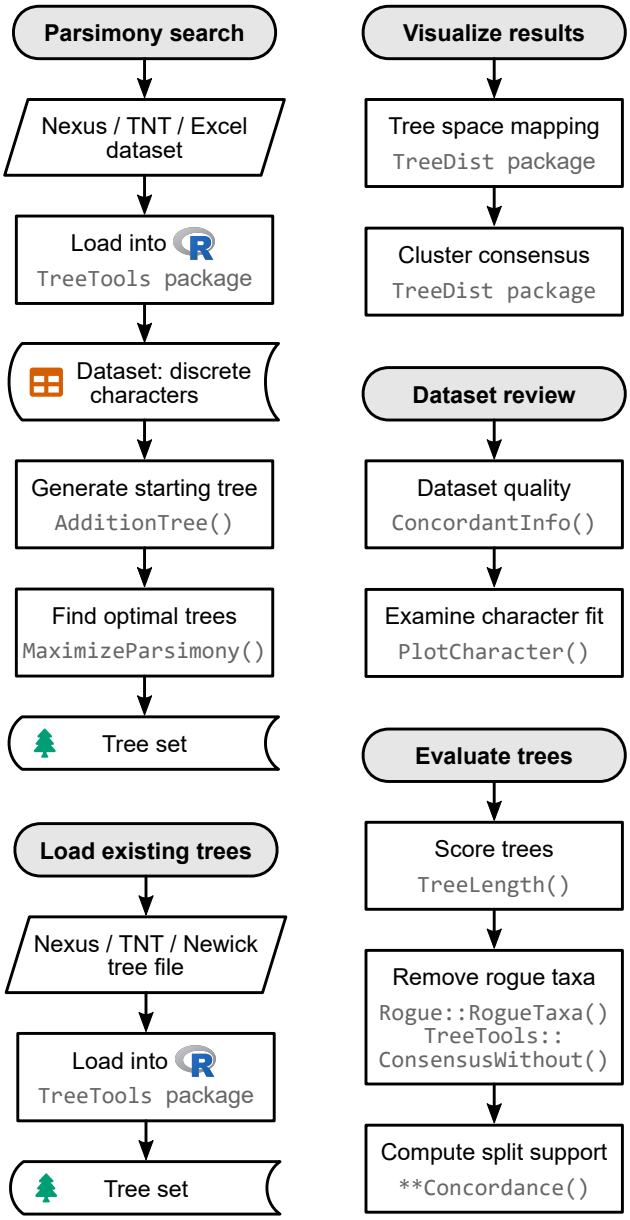


Figure 1: Flow charts summarizing key functions available in TreeSearch.

not always easy to integrate into phylogenetic workflows, so are not routinely included in empirical studies.

TreeSearch provides functions that allow researchers to engage with the three main aspects of morphological phylogenetic analysis: dataset construction and validation; phylogenetic search (including with inapplicable data); and the interrogation of optimal tree sets (Fig. 1). These functions can be accessed via the R command-line, as documented within the package and at ms609.github.io/TreeSearch, or through a graphical user interface (GUI). The GUI includes options to export a log of executed commands as a fully reproducible R script, and to save outputs in graphical, Nexus or Newick formats.

2 Implementation

Tree scoring

TreeSearch can score trees using equal weights, implied weighting (Goloboff 1993), or profile parsimony (Faith and Trueman 2001). The function `TreeLength()` calculates tree score using the "Morphy" phylogenetic library (Brazeau, Smith, and Guillerme 2017), which implements the Fitch (1971) and Brazeau, Guillerme, and Smith (2019) algorithms. Morphy returns the equal weights parsimony score

of a tree against a given dataset. Implied weights and profile parsimony scores are computed by first making a separate call to Morphy for each character in turn, passed as a single-character dataset; then passing this value to the appropriate weighting formula and summing the total score over all characters.

Implied weighting (Goloboff 1993) is an approximate method that treats each additional step (i.e. transition between tokens) in a character as less surprising – and thus requiring less penalty – than the previous step. Each additional step demonstrates that a character is less reliable for phylogenetic inference, and thus more likely to contain additional homoplasy. The score of a tree under implied weighting is $\sum \frac{e_i}{e_i+k}$, where e_i denotes the number of extra steps observed in character i, and is derived by subtracting the minimum score that the character can obtain on any tree from the score observed on the tree in question (Goloboff 1993). The minimum length of a tree is one less than the number of unique tokens (excluding the inapplicable token '-') that must be present.

Profile parsimony (Faith and Trueman 2001) represents an alternative formulation of how surprising each additional step in a character is (Arias and Miranda-Esquivel 2004): the penalty associated with each additional step in a character is a function of the probability that a character will fit at least as well as is observed on a uniformly selected tree. On this view, an additional step is less surprising if observed in a character where there are more opportunities to observe homoplasy, whether because a character contains fewer ambiguous codings (a motivation for the 'extended' implied weighting of Goloboff (2014)) or because states are distributed more evenly in a character, whose higher phylogenetic information content (Thorley, Wilkinson, and Charleston 1998) corresponds to a lower proportion of trees in which no additional steps are observed.

TreeSearch calculates the profile parsimony score by computing the logarithm of the number of trees onto which a character can be mapped using m steps, using theorem 1 of Carter et al. (1990). As computation for higher numbers of states (W. P. Maddison and Slatkin 1991) is more computationally complex, the present implementation is restricted to characters that contain two informative applicable states, and uses the Fitch (1971) algorithm.

Tree search

The TreeSearch GUI uses the routine `MaximizeParsimony()` to search for optimal trees using tree bisection and reconnection (TBR) searches and the parsimony ratchet (Nixon 1999). This goes beyond the heuristic tree search implementation in the R package **phangorn** (Schliep 2011) by using compiled C++ code to rearrange trees, dramatically accelerating computation, and thus increasing the scale of dataset that can be analysed in reasonable time; and in supporting TBR rearrangements, which explore larger neighbourhoods of tree space: TBR evaluates more trees than nearest-neighbour interchanges or subtree pruning and regrafting, leading to additional computational expense that is offset by a decreased likelihood that search will become trapped in a local optimum (Goeffon, Richer, and Jin-Kao Hao 2008; Whelan and Money 2010).

By default, search begins from a greedy addition tree generated by function `AdditionTree()`, which queues taxa in a random order, then attaches each taxon in turn to the growing tree at the most parsimonious location. Search may also be started from neighbour-joining trees, or the results of a previous search.

Search commences by conducting TBR rearrangements – a hill-climbing approach that locates a locally optimal tree from which no tree accessible by a single TBR rearrangement has a better score. A TBR iteration breaks a randomly selected edge in the focal tree, and reconnects each possible pair of edges in the resultant sub-trees to produce a list of candidate trees. Entries that are inconsistent with user-specified topological constraints are removed; remaining trees are inserted into a queue and scored in a random sequence. If the score of a candidate tree is at least as good as the best yet encountered (within the bounds of an optional tolerance parameter ϵ, which allows the retention of almost-optimal trees in order to improve accuracy – see e.g. M. R. Smith (2019a)), this tree is used as the starting point for a new TBR iteration. Otherwise, the next tree in the list is considered. TBR search continues until the best score is found a specified number of times; a specified number of TBR break points have been evaluated without any improvement to tree score; or a set amount of time has passed.

When TBR search is complete, iterations of the parsimony ratchet (Nixon 1999) are conducted in order to search areas of tree space that are separated from the best tree yet found by 'valleys' that cannot be traversed by TBR rearrangements without passing through trees whose optimality score is below the threshold for acceptance. Each ratchet iteration begins by resampling the original matrix. A round of TBR search is conducted using this resampled matrix, and the tree thus produced is used as a starting point for a new round of TBR search using the original data. After a specified number of ratchet iterations, an optional final round of TBR search allows a denser sampling of optimal trees from the final region of tree space.

A simple example search can be conducted using a morphological dataset included in the package, taken from Vinther, Van Roy, and Briggs (2008):

```
library("TreeSearch")
vinther <- inapplicable.phyData[["Vinther2008"]]
trees <- MaximizeParsimony(vinther, concavity = 10, tolerance = 0.05)
```

The `MaximizeParsimony()` command performs tree search under implied weights with a concavity value of 10 (`concavity = Inf` would select equal weights), retaining any tree whose score is within 0.05 of the best score.

The resulting trees can be summarised according to their scores (optionally, against a different `dataset` or under a different weighting strategy, as specified by `concavity`) and the iteration in which they were first hit:

```
TreeLength(trees, dataset = vinther, concavity = 10) |>
  signif() |>         # truncate non-significant digits
  table()             # tabulate by score

#>
#> 1.52814 1.54329  1.5641
#>       3      45       4

attr(trees, "firstHit")

#>   seed  start ratch1 ratch2 ratch3 ratch4 ratch5 ratch6 ratch7  final
#>      0     29      4      0     10      7      2      0      0      0
```

More flexible, if less computationally efficient, tree searches can be conducted at the command line using the `TreeSearch()`, `Ratchet()` and `Bootstrap()` commands, which support custom tree optimality criteria (e.g. Hopkins and St. John 2021).

Visualization

The distribution of optimal trees, however obtained, can be visualized through interactive mappings of tree space (Hillis, Heath, and St. John 2005; M. R. Smith 2022a). The TreeSearch GUI supports the use of information theoretic distances (M. R. Smith 2020a); the quartet distance (Estabrook, McMorris, and Meacham 1985); or the Robinson–Foulds distance (Robinson and Foulds 1981) to construct tree spaces, which are mapped into 2–12 dimensions using principal coordinates analysis (Gower 1966). The degree to which a mapping faithfully depicts original tree-to-tree distances is measured using the product of the trustworthiness and continuity metrics (Venna and Kaski 2001; Kaski et al. 2003; M. R. Smith 2022a), a composite score denoting the degree to which points that are nearby when mapped are truly close neighbours (trustworthiness), and the degree to which nearby points remain nearby when mapped (continuity). Plotting the minimum spanning tree – the shortest path that connects all trees (Gower and Ross 1969) – can highlight stress in a mapping (grey lines in Fig. 2): the spatial relationships of trees are distorted in regions where the minimum spanning tree takes a circuitous route to connect trees that are mapped close to one another (see fig. 1a–b in M. R. Smith 2022a).

To relate the geometry of tree space to the underlying trees, each point in tree space may be annotated according to the optimality score of its corresponding tree under a selected step weighting scheme; by the relationships between chosen taxa that are inferred by that tree; and by the search iteration in which the tree was first found by tree search (Fig. 2).

Annotating trees by the iteration in which they were first found allows a user to evaluate whether a continuation of tree search is likely to yield more optimal trees. For example, if the retained trees were only recently found, the search may not yet have located a global optimum. Alternatively, if certain regions of tree space are visited only by a single ratchet iteration, it is possible that further isolated 'islands' (Bastert et al. 2002) remain to be found; continuing tree search until subsequent ratchet iterations no longer locate new clusters of trees will reduce the chance that optimal regions of tree space remain unvisited.

As the identification of clusters from mappings of tree space can be misleading (M. R. Smith 2022a), TreeSearch identifies clusters of trees from tree-to-tree distances using K-means++ clustering, partitioning around medoids and hierarchical clustering with minimax linkage (Hartigan and Wong 1979; Arthur and Vassilvitskii 2007; Murtagh 1983; Bien and Tibshirani 2011; Maechler et al. 2019). Clusterings are evaluated using the silhouette coefficient, a measure of the extent of overlap between

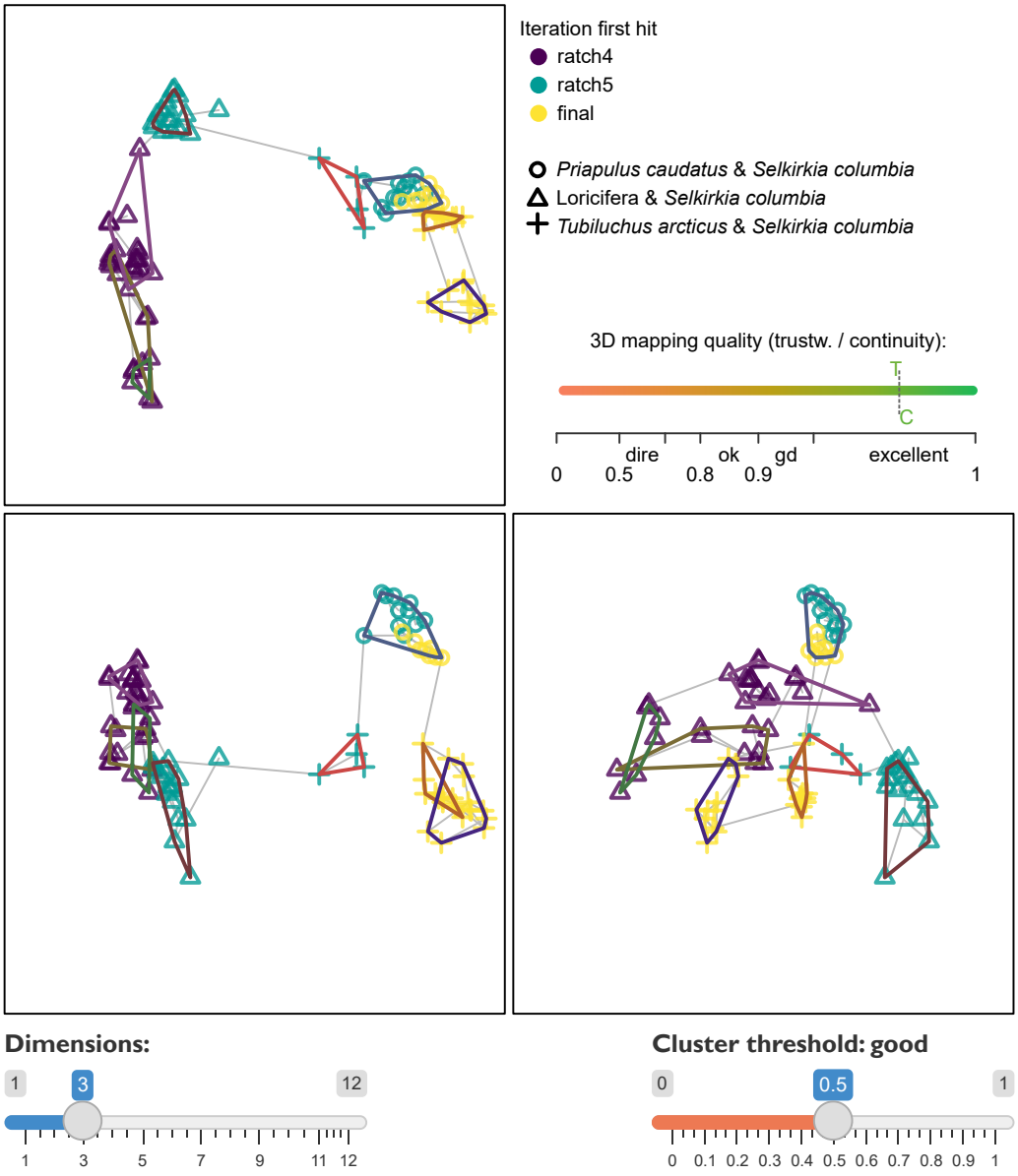


Figure 2: Three-dimensional map visualizing progress in a tree search in the TreeSearch GUI. Optimal trees belong to three statistically distinct clusters with good support (silhouette coefficient > 0.5), characterized by different relationships between certain taxa (plotting symbols). Although multiple ratchet iterations have visited each cluster, limited overlap between ratchet iterations suggests that a continuation of tree search may sample novel optimal trees. High trustworthiness and continuity values and a simple minimum spanning tree (grey) indicate that the mapping does not exhibit severe distortion. This figure depicts the tree space GUI display after loading the Wills et al. (2012) dataset; clearing previous trees from memory (sample *n* trees = 0); and starting a new search (Search→Configure) with equal step weighting and $10^{1.5}$ max hits. 93 trees were sampled, coloured by "When first found", with plotting symbols depicting "Relationships" between the specified taxa.

clusters (Kaufman and Rousseeuw 1990). The clustering with the highest silhouette coefficient is depicted if the silhouette coefficient exceeds a user-specified threshold; the interpretation of the chosen threshold according to Kaufman and Rousseeuw (1990) is displayed to the user. Plotting a separate consensus tree for each cluster often reveals phylogenetic information that is concealed by polytomies in the single 'plenary' consensus of all optimal trees (Stockham, Wang, and Warnow 2002).

Plenary consensus trees can also lack resolution because of wildcard or 'rogue' taxa, in which conflict or ambiguity in their character codings leads to an unsettled phylogenetic position (Wilkinson 1994, 2003; Kearney 2002). TreeSearch detects rogue taxa using a heuristic approach (M. R. Smith 2022b) that seeks to maximize the phylogenetic information content (*sensu* Thorley, Wilkinson, and Charleston 1998) of a consensus tree created after removing rogue taxa from input trees. The position of an excluded taxon is portrayed by shading each edge or node of the consensus according to the number of times the specified taxon occurs at that position on an underlying tree [Fig. 3; after Klopfstein and Spasojevic (2019)], equivalent to the 'branch attachment frequency' of Phyutility (S. A. Smith and Dunn 2008).

Identifying taxa with an unstable position, and splits with low support, can help an investigator to critically re-examine character codings; to this end, each edge of the resulting consensus can be annotated with the frequency of the split amongst the tree set, or with a concordance factor (Minh, Hahn, and Lanfear 2020) denoting the strength of support from the underlying dataset.

Dataset review

Ultimately, the quality of a dataset plays a central role in determining the reliability of phylogenetic results, with changes to a relatively small number of character codings potentially exhibiting an outsized impact on reconstructed topologies (Goloboff and Sereno 2021). Nevertheless, dataset quality does not always receive commensurate attention (Simões et al. 2017). One step towards improving the rigour of morphological datasets is to annotate each cell in a dataset with an explicit justification for each taxon's coding (Sereno 2009), which can be accomplished in Nexus-formatted data files (D. R. Maddison, Swofford, and Maddison 1997) using software such as MorphoBank (O'Leary and Kaufman 2011).

TreeSearch presents such annotations alongside a reconstruction of each character's states on a specified tree, with inapplicable states mapped according to the algorithm of Brazeau, Guillerme, and Smith (2019). Neomorphic (presence/absence) and transformational characters (Sereno 2007) are distinguished by reserving the token `0` to solely denote the absence of a neomorphic character, with tokens `1` … `n` used to denote the n states of a transformational character (Brazeau, Guillerme, and Smith 2019). In order to identify character codings that contribute to taxon instability, each leaf is coloured according to its mean contribution to tree length for the visualized character (Pol and Escapa 2009).

This visualization of reconstructed character transitions can help to identify cases where the formulation of characters has unintended consequences (Wilkinson 1995; Brazeau 2011); where inapplicable states have been inconsistently applied (Brazeau, Guillerme, and Smith 2019); where taphonomic absence is wrongly coded as biological absence (Donoghue and Purnell 2009); where previous datasets are uncritically recycled (Jenner 2001); or where taxa are coded with more confidence than a critical evaluation of available evidence can truly support. Insofar as the optimal tree and the underlying characters are reciprocally illuminating (Mooi and Gill 2016), successive cycles of phylogenetic analysis and character re-formulation can improve the integrity of morphological datasets, and thus increase their capacity to yield meaningful phylogenetic results (Hennig 1966).

3 Availability

TreeSearch can be installed through the Comprehensive R Archive Network (CRAN) using `install.packages("TreeSearch")`; the graphical user interface is launched with the command `TreeSearch::EasyTrees()`. The package has been tested on Windows 10, Mac OS X 10 and Ubuntu 20, and requires only packages available from the CRAN repository. Source code is available at https://github.com/ms609/TreeSearch/, and is permanently archived at Zenodo (https://dx.doi.org/10.5281/zenodo.1042590). Online documentation is available at https://ms609.github.io/TreeSearch/.

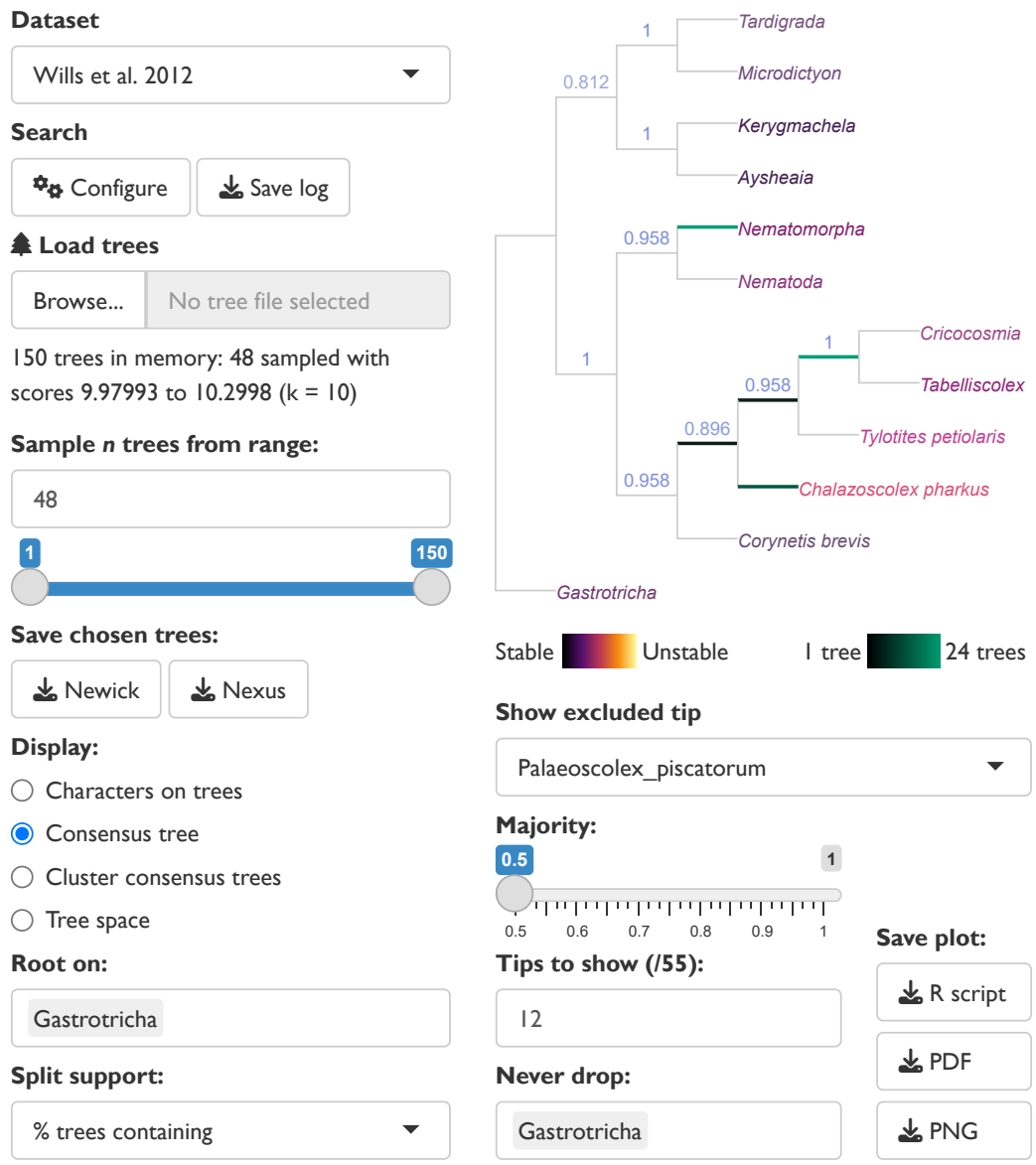


Figure 3: Reduced consensus of 48 cladograms generated by analysis of data from Wills et al. (2012) under different parsimony methods by Brazeau, Guillerme, and Smith (2019), as displayed in the TreeSearch graphical user interface. Removal of taxa reveals strong support for relationships that would otherwise be masked by rogues such as *Palaeoscolex*, whose position in optimal trees is marked by the highlighted edges. The GUI state can be reproduced by selecting the options displayed in the figure.

4 Acknowledgements

I thank Alavya Dhungana and Joe Moysiuk for feedback on preliminary versions of the software, and Martin Brazeau and anonymous referees for comments on the manuscript. Functionality in TreeSearch employs the underlying R packages **ape** (Paradis and Schliep 2019), **phangorn** (Schliep 2011), **Quartet** (Sand et al. 2014; M. R. Smith 2019b), **Rogue** (M. R. Smith 2022b), **shiny** (Chang et al. 2021), **shinyjs** (Attali 2020), **TreeDist** (M. R. Smith 2020b), and **TreeTools** (M. R. Smith 2019c). Icons from R used under GPL-3; Font Awesome, CC-BY-4.0.

References

Arias, J. Salvador, and Daniel Rafael Miranda-Esquivel. 2004. "Profile Parsimony (PP): An Analysis Under Implied Weights (IW)." *Cladistics* 20 (1): 56–63. https://doi.org/10.1111/j.1096-0031.2003.00001.x.

Arthur, David, and Sergei Vassilvitskii. 2007. "K-Means++: The Advantages of Careful Seeding." In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1027–35. SODA '07. USA: Society for Industrial and Applied Mathematics. https://dl.acm.org/doi/10.5555/1283383.1283494.

Asher, Robert J., and Martin R. Smith. 2022. "Phylogenetic Signal and Bias in Paleontology." *Systematic Biology* 71 (4): 986–1008. https://doi.org/10.1093/sysbio/syab072.

Attali, Dean. 2020. *Shinyjs: Easily Improve the User Experience of Your Shiny Apps in Seconds*. https://CRAN.R-project.org/package=shinyjs.

Bastert, Oliver, Dan Rockmore, Peter F. Stadler, and Gottfried Tinhofer. 2002. "Landscapes on Spaces of Trees." *Applied Mathematics and Computation* 131 (2-3): 439–59. https://doi.org/10.1016/S0096-3003(01)00164-3.

Bien, Jacob, and Robert Tibshirani. 2011. "Hierarchical Clustering with Prototypes via Minimax Linkage." *Journal of the American Statistical Association* 106 (495): 1075–84. https://doi.org/10.1198/jasa.2011.tm10183.

Brazeau, Martin D. 2011. "Problematic Character Coding Methods in Morphology and Their Effects." *Biological Journal of the Linnean Society* 104 (3): 489–98. https://doi.org/10.1111/j.1095-8312.2011.01755.x.

Brazeau, Martin D., Thomas Guillerme, and Martin R. Smith. 2019. "An Algorithm for Morphological Phylogenetic Analysis with Inapplicable Data." *Systematic Biology* 68 (4): 619–31. https://doi.org/10.1093/sysbio/syy083.

Brazeau, Martin D., Martin R. Smith, and Thomas Guillerme. 2017. "MorphyLib: A Library for Phylogenetic Analysis of Categorical Trait Data with Inapplicability." https://doi.org/10.5281/zenodo.815372.

Brown, Joseph W., Caroline Parins-Fukuchi, Gregory W. Stull, Oscar M. Vargas, and Stephen A. Smith. 2017. "Bayesian and Likelihood Phylogenetic Reconstructions of Morphological Traits Are Not Discordant When Taking Uncertainty into Consideration: A Comment on Puttick *Et Al.*" *Proceedings of the Royal Society B: Biological Sciences* 284 (1864): 20170986. https://doi.org/10.1098/rspb.2017.0986.

Carter, M., M. Hendy, D. Penny, L. A. Székely, and N. C. Wormald. 1990. "On the Distribution of Lengths of Evolutionary Trees." *SIAM Journal on Discrete Mathematics* 3 (1): 38–47. https://doi.org/10.1137/0403005.

Chang, Winston, Joe Cheng, J. J. Allaire, Carson Sievert, Barret Schloerke, Yi-Hui Xie, Jeff Allen, Jonathan McPherson, Alan Dipert, and Barbara Borges. 2021. *Shiny: Web Application Framework for R*. https://CRAN.R-project.org/package=shiny.

De Laet, Jan E. 2005. "Parsimony and the Problem of Inapplicables in Sequence Data." Edited by V. A. Albert. *Parsimony, Phylogeny, and Genomics*, 81–116. https://doi.org/10.1093/acprof:oso/9780199297306.003.0006.

Donoghue, Philip C. J., and Mark A. Purnell. 2009. "Distinguishing Heat from Light in Debate over Controversial Fossils." *BioEssays* 31 (2): 178–89. https://doi.org/10.1002/bies.200800128.

Estabrook, George F., F. R. McMorris, and Christopher A. Meacham. 1985. "Comparison of Undirected Phylogenetic Trees Based on Subtrees of Four Evolutionary Units." *Systematic Zoology* 34 (2): 193–200. https://doi.org/10.2307/sysbio/34.2.193.

Faith, Daniel P., and John W. H. Trueman. 2001. "Towards an Inclusive Philosophy for Phylogenetic Inference." *Systematic Biology* 50 (3): 331–50. https://doi.org/10.1080/10635150118627.

Fitch, Walter M. 1971. "Toward Defining the Course of Evolution: Minimum Change for a Specific Tree Topology." *Systematic Biology* 20 (4): 406–16. https://doi.org/10.1093/sysbio/20.4.406.

Goeffon, A., J.-M. Richer, and Jin-Kao Hao. 2008. "Progressive Tree Neighborhood Applied to the Maximum Parsimony Problem." *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 5 (1): 136–45. https://doi.org/10.1109/TCBB.2007.1065.

Goloboff, Pablo A. 1993. "Estimating Character Weights During Tree Search." *Cladistics* 9 (1): 83–91. https://doi.org/10.1111/j.1096-0031.1993.tb00209.x.

———. 2014. "Extended Implied Weighting." *Cladistics* 30 (3): 260–72. https://doi.org/10.1111/cla.12047.

Goloboff, Pablo A., James M. Carpenter, J. Salvador Arias, and Daniel Rafael Miranda Esquivel. 2008. "Weighting Against Homoplasy Improves Phylogenetic Analysis of Morphological Data Sets." *Cladistics* 24 (5): 758–73. https://doi.org/10.1111/j.1096-0031.2008.00209.x.

Goloboff, Pablo A., Jan De Laet, Duniesky Ríos-Tamayo, and Claudia A. Szumik. 2021. "A Reconsideration of Inapplicable Characters, and an Approximation with Step-Matrix Recoding." *Cladistics* 37 (5): 596–629. https://doi.org/10.1111/cla.12456.

Goloboff, Pablo A., and Paul C. Sereno. 2021. "Comparative Cladistics: Identifying the Sources for Differing Phylogenetic Results Between Competing Morphology-Based Datasets." *Journal of Systematic Palaeontology*, 1–26. https://doi.org/10.1080/14772019.2021.1970038.

Goloboff, Pablo A., Ambrosio Torres, and J. Salvador Arias. 2018. "Weighted Parsimony Outperforms Other Methods of Phylogenetic Inference Under Models Appropriate for Morphology." *Cladistics* 34 (4): 407–37. https://doi.org/10.1111/cla.12205.

Goloboff, Pablo A., Ambrosio Torres Galvis, and J. Salvatdor Arias. 2018. "Parsimony and Model-Based Phylogenetic Methods for Morphological Data: Comments on O'Reilly *Et Al.*" *Palaeontology* 61 (4): 625–30. https://doi.org/10.1111/pala.12353.

Gower, J. C. 1966. "Some Distance Properties of Latent Root and Vector Methods Used in Multivariate Analysis." *Biometrika* 53 (3/4): 325–38. https://doi.org/10.2307/2333639.

Gower, J. C., and G. J. S. Ross. 1969. "Minimum Spanning Trees and Single Linkage Cluster Analysis." *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 18 (1): 54–64. https://doi.org/10.2307/2346439.

Guenser, Pauline, Rachel C. M. Warnock, Walker Pett, Philip C. J. Donoghue, and Emilia Jarochowska. 2021. "Does Time Matter in Phylogeny? A Perspective from the Fossil Record." *bioRχiv*. https://doi.org/10.1101/2021.06.11.445746.

Hartigan, J. A., and M. A. Wong. 1979. "Algorithm AS 136: A *K*-Means Clustering Algorithm." *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28 (1): 100–108. https://doi.org/10.2307/2346830.

Hennig, Willi. 1966. *Phylogenetic Systematics*. Urbana: The University of Illinois Press.

Hillis, David M., Tracy A. Heath, and Katherine St. John. 2005. "Analysis and Visualization of Tree Space." *Systematic Biology* 54 (3): 471–82. https://doi.org/10.1080/10635150590946961.

Hopkins, Melanie J., and Katherine St. John. 2021. "Incorporating Hierarchical Characters into Phylogenetic Analysis." *Systematic Biology* 70 (6): 1163–80. https://doi.org/10.1093/sysbio/syab005.

Jenner, Ronald A. 2001. "Bilaterian Phylogeny and Uncritical Recycling of Morphological Data Sets." Edited by R. Olmstead. *Systematic Biology* 50 (5): 730–42. https://doi.org/10.1080/106351501753328857.

Kaski, Samuel, Janne Nikkilä, Merja Oja, Jarkko Venna, Petri Törönen, and Eero Castrén. 2003. "Trustworthiness and Metrics in Visualizing Similarity of Gene Expression." *BMC Bioinformatics* 4: 48. https://doi.org/10.1186/1471-2105-4-48.

Kaufman, Leonard, and Peter J. Rousseeuw. 1990. "Partitioning Around Medoids (Program PAM)." In *Finding Groups in Data: An Introduction to Cluster Analysis*, 68–125. Wiley Series in Probability and Statistics. John Wiley & Sons, Ltd.

Kearney, Maureen. 2002. "Fragmentary Taxa, Missing Data, and Ambiguity: Mistaken Assumptions and Conclusions." *Systematic Biology* 51 (2): 369–81. https://doi.org/10.1080/10635150252899824.

Klopfstein, Seraina, and Tamara Spasojevic. 2019. "Illustrating Phylogenetic Placement of Fossils Using RoguePlots: An Example from Ichneumonid Parasitoid Wasps (Hymenoptera, Ichneumonidae) and an Extensive Morphological Matrix." *PLoS ONE* 14 (4): e0212942. https://doi.org/10.1371/journal.pone.0212942.

Koch, Nicolás Mongiardino, and Luke A. Parry. 2020. "Death Is on Our Side: Paleontological Data Drastically Modify Phylogenetic Hypotheses." *Systematic Biology* 69 (6): 1052–67. https://doi.org/10.1093/sysbio/syaa023.

Maddison, David R., David L. Swofford, and Wayne P. Maddison. 1997. "NEXUS: An Extensible File Format for Systematic Information." *Systematic Biology* 46 (4): 590–621. https://doi.org/10.1093/sysbio/46.4.590.

Maddison, Wayne P. 1993. "Missing Data Versus Missing Characters in Phylogenetic Analysis." *Systematic Biology* 42 (4): 576–81. https://doi.org/10.1093/sysbio/42.4.576.

Maddison, Wayne P., and M. Slatkin. 1991. "Null Models for the Number of Evolutionary Steps in a Character on a Phylogenetic Tree." *Evolution* 45 (5): 1184–97. https://doi.org/10.1111/j.1558-5646.1991.tb04385.x.

Maechler, Martin, Peter Rousseeuw, Anja Struyf, Mia Hubert, and Kurt Hornik. 2019. "Cluster: Cluster Analysis Basics and Extensions." *Comprehensive R Archive Network* 2.1.0.

Minh, Bui Quang, Matthew W. Hahn, and Robert Lanfear. 2020. "New Methods to Calculate Concordance Factors for Phylogenomic Datasets." *Molecular Biology and Evolution* 37 (9): 2727–33. https://doi.org/10.1093/molbev/msaa106.

Mooi, Randall D., and Anthony C. Gill. 2016. "Hennig's Auxiliary Principle and Reciprocal Illumination Revisited." In *The Future of Phylogenetic Systematics*, edited by David Williams, Michael Schmitt, and Quentin Wheeler, 258–85. Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9781316338797.013.

Murtagh, F. 1983. "A Survey of Recent Advances in Hierarchical Clustering Algorithms." *The Computer Journal* 26 (4): 354–59. https://doi.org/10.1093/comjnl/26.4.354.

Nixon, Kevin C. 1999. "The Parsimony Ratchet, a New Method for Rapid Parsimony Analysis." *Cladistics* 15 (4): 407–14. https://doi.org/10.1111/j.1096-0031.1999.tb00277.x.

O'Leary, Maureen A., and Seth Kaufman. 2011. "MorphoBank: Phylophenomics in the 'Cloud'." *Cladistics* 27 (5): 529–37. https://doi.org/10.1111/j.1096-0031.2011.00355.x.

O'Reilly, Joseph E., Mark N. Puttick, Luke Parry, Alastair R. Tanner, James E. Tarver, James Fleming, Davide Pisani, and Philip C. J. Donoghue. 2016. "Bayesian Methods Outperform Parsimony but at the Expense of Precision in the Estimation of Phylogeny from Discrete Morphological Data." *Biology Letters* 12 (4): 20160081. https://doi.org/10.1098/rsbl.2016.0081.

Paradis, Emmanuel, and Klaus Schliep. 2019. "Ape 5.0: An Environment for Modern Phylogenetics and Evolutionary Analyses in R." *Bioinformatics* 35 (3): 526–28. https://doi.org/10.1093/bioinformatics/bty633.

Pol, Diego, and Ignacio H. Escapa. 2009. "Unstable Taxa in Cladistic Analysis: Identification and the Assessment of Relevant Characters." *Cladistics* 25 (5): 515–27. https://doi.org/10.1111/j.1096-0031.2009.00258.x.

Puttick, Mark N., Joseph E. O'Reilly, Alastair R. Tanner, James F. Fleming, James Clark, Lucy Holloway, Jesus Lozano-Fernandez, et al. 2017. "Uncertain-Tree: Discriminating Among Competing Approaches to the Phylogenetic Analysis of Phenotype Data." *Proceedings of the Royal Society B: Biological Sciences* 284 (1846): 20162290. https://doi.org/10.1098/rspb.2016.2290.

Robinson, David F., and Leslie R. Foulds. 1981. "Comparison of Phylogenetic Trees." *Mathematical Biosciences* 53 (1-2): 131–47. https://doi.org/10.1016/0025-5564(81)90043-2.

Sand, Andreas, Morten K. Holt, Jens Johansen, Gerth Stølting Brodal, Thomas Mailund, and Christian N. S. Pedersen. 2014. "tqDist: A Library for Computing the Quartet and Triplet Distances Between Binary or General Trees." *Bioinformatics* 30 (14): 2079–80. https://doi.org/10.1093/bioinformatics/btu157.

Sansom, Robert S., Peter G. Choate, Joseph N. Keating, and Emma Randle. 2018. "Parsimony, Not Bayesian Analysis, Recovers More Stratigraphically Congruent Phylogenetic Trees." *Biology Letters* 14 (6): 20180263. https://doi.org/10.1098/rsbl.2018.0263.

Schliep, Klaus Peter. 2011. "Phangorn: Phylogenetic Analysis in R." *Bioinformatics* 27 (4): 592–93. https://doi.org/10.1093/bioinformatics/btq706.

Sereno, Paul C. 2007. "Logical Basis for Morphological Characters in Phylogenetics." *Cladistics* 23 (6): 565–87. https://doi.org/10.1111/j.1096-0031.2007.00161.x.

———. 2009. "Comparative Cladistics." *Cladistics* 25 (6): 624–59. https://doi.org/10.1111/j.1096-0031.2009.00265.x.

Simões, Tiago R., Michael W. Caldwell, Alessandro Palci, and Randall L. Nydam. 2017. "Giant Taxon-Character Matrices: Quality of Character Constructions Remains Critical Regardless of Size." *Cladistics* 33 (2): 198–219. https://doi.org/10.1111/cla.12163.

Smith, Martin R. 2019a. "Bayesian and Parsimony Approaches Reconstruct Informative Trees from Simulated Morphological Datasets." *Biology Letters* 15 (2): 20180632. https://doi.org/10.1098/rsbl.2018.0632.

———. 2019b. "Quartet: Comparison of Phylogenetic Trees Using Quartet and Bipartition Measures." *Comprehensive R Archive Network*, doi:10.5281/zenodo.2536318. https://doi.org/10.5281/zenodo.2536318.

———. 2019c. "TreeTools: Create, Modify and Analyse Phylogenetic Trees." *Comprehensive R Archive Network*, doi:10.5281/zenodo.3522725. https://doi.org/10.5281/zenodo.3522725.

———. 2020a. "Information Theoretic Generalized Robinson–Foulds Metrics for Comparing Phylogenetic Trees." *Bioinformatics* 36 (20): 5007–13. https://doi.org/10.1093/bioinformatics/btaa614.

———. 2020b. "TreeDist: Calculate and Map Distances Between Phylogenetic Trees." *Comprehensive R Archive Network*, doi:10.5281/zenodo.3528123. https://doi.org/10.5281/zenodo.3528123.

———. 2022a. "Robust Analysis of Phylogenetic Tree Space." *Systematic Biology* 71 (5): 1255–70. https://doi.org/10.1093/sysbio/syab100.

———. 2022b. "Using Information Theory to Detect Rogue Taxa and Improve Consensus Trees." *Systematic Biology* 71 (5): 1088–94. https://doi.org/10.1093/sysbio/syab099.

Smith, Stephen A., and Casey W. Dunn. 2008. "Phyutility: A Phyloinformatics Tool for Trees, Alignments and Molecular Data." *Bioinformatics* 24 (5): 715–16. https://doi.org/10.1093/bioinformatics/btm619.

Stockham, C., L.-S. Wang, and T. Warnow. 2002. "Statistically Based Postprocessing of Phylogenetic Analysis by Clustering." *Bioinformatics* 18 (Suppl 1): S285–93. https://doi.org/10.1093/bioinformatics/18.suppl_1.S285.

Tarasov, Sergei. 2019. "Integration of Anatomy Ontologies and Evo-Devo Using Structured Markov Models Suggests a New Framework for Modeling Discrete Phenotypic Traits." *Systematic Biology* 68 (5): 698–716. https://doi.org/10.1093/sysbio/syz005.

———. 2022. "New Phylogenetic Markov Models for Inapplicable Morphological Characters." *bioRχiv*, 2021.04.26.441495. https://doi.org/10.1101/2021.04.26.441495.

Thorley, Joseph L., Mark Wilkinson, and Mike Charleston. 1998. "The Information Content of Consensus Trees." In *Advances in Data Science and Classification*, edited by Alfredo Rizzi, Maurizio Vichi, and Hans-Hermann Bock, 91–98. Berlin: Springer. https://doi.org/10.1007/978-3-642-72253-0_12.

Venna, Jarkko, and Samuel Kaski. 2001. "Neighborhood Preservation in Nonlinear Projection Methods: An Experimental Study." In *Artificial Neural Networks — ICANN 2001*, edited by Georg Dorffner, Horst Bischof, and Kurt Hornik, 485–91. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. https://doi.org/10.1007/3-540-44668-0_68.

Vinther, Jakob, Peter Van Roy, and Derek E. G. Briggs. 2008. "Machaeridians are Palaeozoic armoured annelids." *Nature* 451 (7175): 185–88. https://doi.org/10.1038/nature06474.

Whelan, Simon, and Daniel Money. 2010. "The Prevalence of Multifurcations in Tree-Space and Their Implications for Tree-Search." *Molecular Biology and Evolution* 27 (12): 2674–77. https://doi.org/10.1093/molbev/msq163.

Wiens, John J. 2004. "The Role of Morphological Data in Phylogeny Reconstruction." *Systematic Biology* 53 (4): 653–61. https://doi.org/10.1080/10635150490472959.

Wilkinson, Mark. 1994. "Common Cladistic Information and Its Consensus Representation: Reduced Adams and Reduced Cladistic Consensus Trees and Profiles." *Systematic Biology* 43 (3): 343–68. https://doi.org/10.2307/2413673.

———. 1995. "A Comparison of Two Methods of Character Construction." *Cladistics* 11 (3): 297–308. https://doi.org/10.1111/j.1096-0031.1995.tb00091.x.

———. 1996. "Majority-Rule Reduced Consensus Trees and Their Use in Bootstrapping." *Molecular Biology and Evolution* 13 (3): 437–44. https://doi.org/10.1093/oxfordjournals.molbev.a025604.

———. 2003. "Missing Entries and Multiple Trees: Instability, Relationships, and Support in Parsimony Analysis." *Journal of Vertebrate Paleontology* 23 (2): 311–23. https://doi.org/10.1671/0272-4634(2003)023%5B0311:MEAMTI%5D2.0.CO;2.

Wills, Matthew Albion, Sylvain Gerber, M. Ruta, and M. Hughes. 2012. "The Disparity of Priapulid, Archaeopriapulid and Palaeoscolecid Worms in the Light of New Data." *Journal of Evolutionary Biology* 25 (10): 2056–76. https://doi.org/10.1111/j.1420-9101.2012.02586.x.

Wortley, A. H., and Robert W. Scotland. 2006. "The Effect of Combining Molecular and Morphological Data in Published Phylogenetic Analyses." *Systematic Biology* 55 (4): 677–85. https://doi.org/10.1080/10635150600899798.

Wright, April M., and Graeme T. Lloyd. 2020. "Bayesian Analyses in Phylogenetic Palaeontology: Interpreting the Posterior Sample." *Palaeontology* 63 (6): 997–1006. https://doi.org/10.1111/pala.12500.

Martin R. Smith
University of Durham
Department of Earth Sciences
Durham, UK
DH1 3LE
https://smithlabdurham.github.io/
ORCiD: 0000-0001-5660-1727
martin.smith@durham.ac.uk

The openVA Toolkit for Verbal Autopsies

by Zehang Richard Li, Jason Thomas, Eungang Choi, Tyler H. McCormick, and Samuel J Clark

Abstract Verbal autopsy (VA) is a survey-based tool widely used to infer cause of death (COD) in regions without complete-coverage civil registration and vital statistics systems. In such settings, many deaths happen outside of medical facilities and are not officially documented by a medical professional. VA surveys, consisting of signs and symptoms reported by a person close to the decedent, are used to infer the COD for an individual, and to estimate and monitor the COD distribution in the population. Several classification algorithms have been developed and widely used to assign causes of death using VA data. However, the incompatibility between different idiosyncratic model implementations and required data structure makes it difficult to systematically apply and compare different methods. The **openVA** package provides the first standardized framework for analyzing VA data that is compatible with all openly available methods and data structure. It provides an open-source, R implementation of several most widely used VA methods. It supports different data input and output formats, and customizable information about the associations between causes and symptoms. The paper discusses the relevant algorithms, their implementations in R packages under the **openVA** suite, and demonstrates the pipeline of model fitting, summary, comparison, and visualization in the R environment.

1 Introduction

Verbal autopsy (VA) is a well-established approach to ascertaining the cause of death (COD) when medical certification and full autopsies are not feasible or practical (Garenne, 2014; Taylor et al., 1983). After a death is identified, a specially-trained fieldworker interviews the caregivers (usually family members) of the decedent. A typical VA interview includes a set of structured questions with categorical or quantitative responses and a narrative section that records the "story" of the death from the respondent's point of view (World Health Organization, 2012). Currently, there are multiple commonly used questionnaires with overlapping, but not identical questions. VAs are routinely used both by researchers in health and demographic surveillance systems (Sankoh and Byass, 2012; Maher et al., 2010) and multi-country research projects (Nkengasong et al., 2020; Breiman et al., 2021), and in national-scale surveys in many low- and middle-income countries (LMICs). For a more comprehensive overview of the current use of VA, we refer readers to Chandramohan et al. (2021).

The process of inferring a cause from VA data consists of two components. First, there must be some external information about the relationship between causes and symptoms. In supervised learning problems, the external information is typically derived from training datasets with known labels. In the context of VA, a common practice is to obtain training data using clinically trained, experienced physicians who read a fraction of the interviews and determine causes. To address the fact that physicians frequently do not agree on causes, VA interviews are often read by two physicians, and sometimes three, and the final causes are determined through a consensus mechanism (e.g., Kahn et al., 2012). This process can be extremely time- and resource-intensive. Another means of obtaining this information is directly through expert opinion. For example, one can ask groups of physicians to rate the likelihood of a symptoms occurring, given a particular COD, which can be converted into a set of probabilities of observing a symptom given a particular cause. Such expert knowledge can be highly useful in analyzing VA data when training datasets do not exist.

Secondly, an algorithmic or statistical model is needed to assign the causes of death by extrapolating the relationship between symptoms and causes to the target population. The cause-of-death assignment model is conceptually a separate construct from the symptom-cause information. The current state of the VA literature, however, often does not distinguish between the two. This is in part due to popular VA software that combines a source of symptom-cause information (e.g., a certain training data or a database of expert knowledge) with a specific algorithm, and requires a specific type of survey instrument to be used. This restriction prevents robust comparison between methods and contexts. A health agency in one region may, for example, want to analyze VA data using the same VA algorithm as a neighboring region to ensure estimates are comparable. Unless the agencies used the same survey format, however, this is not possible with existing tools.

The **openVA** package (Li et al., 2022b) addresses this issue through an open-source toolkit. The **openVA** suite comprises a collection of R packages for the analysis of verbal autopsy data. The goal of this package is to provide researchers with an open-source tool that supports flexible data input formats and allows easier data manipulation and model fitting. The **openVA** suite consists of four core packages that are on CRAN, **InterVA4** (Li et al., 2019), **InterVA5** (Thomas et al., 2021b), **InSilicoVA** (Li et al., 2022a), and **Tariff** (Li et al., 2018), and an optional package **nbc4va** (Wen et al., 2022). Each of these packages implements one coding algorithm. For three of these algorithms – namely, InterVA-4,

InterVA-5 and Tariff – there are also compiled software programs distributed by the original authors.

The main focus of this paper is to provide a general introduction to the implementation details of the included algorithms both in terms of the underlying methodology, and through case studies. The **openVA** package has four major contributions:

1. It provides a standardized and user-friendly interface for analysts to fit and evaluate each method on different types of data input using standard syntax. Previously, most of the methods were designed to be used specifically with their own input formats and are usually incompatible with others. The **openVA** package closes this gap by allowing easier and fair model comparison of multiple algorithms on the same data. This significantly facilitates further research on VA algorithms.
2. It provides a series of functionalities to summarize and visualize results from multiple algorithms, which is helpful for analysts not familiar with data manipulation and visualization in R.
3. It does not directly implement any algorithms for coding VA data, so that it is possible for a research group to maintain their own algorithm implementations callable from the **openVA** package, while also making it available to general users as a standalone piece of software. For example, the **nbc4va** was developed and maintained independently by researchers at the Center for Global Health Research in Toronto, but is designed so that it can be seamlessly integrated into the **openVA** package.
4. It is fully open source, and can be run on multiple platforms. The open-source nature of **openVA** significantly expands its potential for methodological research and its suitability for integration within a larger data analysis pipeline. Compared to the alternative implementations, the InterVA-4 and InterVA-5 software are distributed as compiled software that can only be run on Windows system. They provide the source codes as an additional code script, which are difficult to modify and re-compile. Tariff, as implemented through the SmartVA-Analyze application (Serina et al., 2015), was also primarily distributed as a compiled application that can only be run on Windows system (Institute for Health Metrics and Evaluation, 2021b). However, their source codes were recently made available under the open source MIT License on GitHub (Institute for Health Metrics and Evaluation, 2021a).

The rest of this paper is organized as follows: We first briefly introduce the main component packages and the underlying algorithms. We then demonstrate standard data structures and model fitting steps with the **openVA** package and functionalities to summarize results. We then discuss how additional information can be incorporated into modeling VA data, and we briefly survey additional packages and software developments built around the **openVA** package. We end with a discussion of remaining issues and limitations of the existing automated VA algorithms and propose new functionalities to be included in **openVA** package.

2 Structure of the openVA package

The **openVA** suite of packages currently consists of four standalone packages that are available on CRAN and one optional package hosted on GitHub. In this section, we first provide a brief introduction to these five packages, and we discuss the mathematical details behind each algorithm in the next subsection.

- **InterVA4** (Li et al., 2014, 2019) is an R package that implements the InterVA-4 model (Byass et al., 2012). It provides replication of the widely used InterVA software (Byass, 2015). The standard input of **InterVA4** is in the form of a pre-defined set of indicators, based on the 2012 World Health Organization (WHO) VA instrument (World Health Organization, 2012). The default InterVA-4 algorithm cannot be applied to other data input formats because its internal built-in prior information is specific to a fixed set of indicators and causes. The same restriction is also maintained in the **InterVA4** package. However, the mathematical formulation of the InterVA-4 model is completely generalizable to other binary input formats, as described in later sections, and is also implemented in the **openVA** package.
- **InterVA5** (Thomas et al., 2021b) is an R package that implements the InterVA-5 model (Byass et al., 2019). The InterVA-5 model updates the previous version in several ways. First, the input data must adhere to the format of the 2016 WHO VA instrument (D'Ambruoso et al., 2017). Second, changes have been made to the data processing steps. It is also worth noting that the model outputs have been expanded by the inclusion of the most likely Circumstances Of Mortality CATegory, or COMCAT, among the results – the categories include: culture, emergency, health systems, inevitable, knowledge, resources, or an indeterminant combination of multiple factors (for more details, see D'Ambruoso et al., 2017). Despite these changes, the mathematical formulation of InterVA-5 is identical to that of InterVA-4.

- **InSilicoVA** (Li et al., 2022a) is an R package that implements the InSilicoVA algorithm, a Bayesian hierarchical framework for cause-of-death assignment and cause-specific mortality fraction estimation proposed in McCormick et al. (2016). It is originally designed to work with the WHO VA instrument, i.e., the same input data used by the InterVA software, but is also generalizable to other data input formats. It is a fully probabilistic algorithm and can incorporate multiple sources of information, such as known sub-populations in the dataset, and physician coding when such data are available. The Markov Chain Monte Carlo (MCMC) sampler is implemented in Java for improved speed.
- **Tariff** (Li et al., 2018) is an R package that implements the Tariff algorithm (James et al., 2011). It most closely reflects the description of the Tariff 2.0 method (Serina et al., 2015). The Tariff algorithm is developed by the Institute for Health Metrics and Evaluation (IHME) and is officially implemented in the SmartVA-Analyze software (Institute for Health Metrics and Evaluation, 2021b). However, as the developers of this R package are not affiliated with the authors of the original algorithm, there are some discrepancies in implementation. The source code of the two versions of Tariff was not publicly available at the time when the **Tariff** package was created, so the package was developed based solely on the descriptions in the published work. Despite the difference in implementation, **Tariff** is able to achieve comparable results as the published work as demonstrated in McCormick et al. (2016). More detailed descriptions of the **Tariff** implementations are also discussed in the supplement of McCormick et al. (2016). The later released Python source codes of SmartVA-Analyze have been incorporated in the web application extension of the **openVA** package.
- **nbc4va** (Wen et al., 2022) is an R package that implements the Naive Bayes Classifier for VA encoding (Miasnikof et al., 2015). It calculates the conditional probabilities of symptoms given causes of death from a training dataset, instead of using physician-provided values. **nbc4va** is developed and maintained by researchers at the Center for Global Health Research in Toronto, but is designed so that it can be seamlessly integrated into **openVA**. It is an optional package that users can choose to load separately.

We note that there are additional methods and implementations to assign causes of death using VA data that are not included in the **openVA** implementation (e.g., Flaxman et al., 2011; Jeblee et al., 2019). These methods are not widely adopted by VA practitioners and most of these implementations are either not publicly available or require data processing steps that are specific to certain datasets, making them impractical for routine use in general. In addition, while the cause-of-death assignment process is closely related to the generic multi-class classification problem, naive application of off-the-shelf classification algorithms has been shown to perform poorly in the context of VA (Murray et al., 2014). Therefore, we focus on the methods that are currently adopted by practitioners. We briefly survey some more recent developments and their implementations at the end of this paper.

The **openVA** package is hosted on CRAN and can be installed with the following commands. Since posterior inference is carried out using MCMC in the InSilicoVA algorithm, we set the seed for the random number generator to make the paper reproducible. For the analysis in this paper, we also install the **nbc4va** package separately from GitHub. The versions of the supporting packages can be checked in R using the `openVA_status()` function.

```
set.seed(12345)
library(openVA)
openVA_status()
```

Overview of VA cause-of-death assignment methods

The common modeling framework for VA data consists of first converting the survey responses into a series of binary variables, and then assigning a COD to each death based on the binary input variables. Typically, the target of inference consists of two parts: the individual cause-of-death assignment, and the population-level cause-specific mortality fractions (CSMF), i.e., the fraction of deaths due to each cause. In this section, we formally compare the modeling approaches utilized by each algorithm for these two targets. We adopt the following notations. Consider N deaths, each with S binary indicators of symptoms. Let s_{ij} denote the indicator for the presence of j-th symptom in the i-th death, which can take values 0, 1, or NA (for missing data). We consider a pre-defined set of causes of size C. For the i-th death, denote the COD by $y_i \in \{1, ..., C\}$ and the probability of dying from cause k is denoted by P_{ik}. For the population, the CSMF of cause k is denoted as π_k, with $\sum_{k=1}^{C} \pi_k = 1$.

- **InterVA4** (Byass et al., 2012) and **InterVA5** (Byass et al., 2019) algorithms calculate the probabil-

ity of each COD given the observed symptoms using the following formula,

$$P_{ik} = \frac{\pi_k^{(0)} \prod_{j=1}^{S} P(s_{ij} = 1|y_i = k)\mathbf{1}_{s_{ij}=1}}{\sum_{k'=1}^{C} \pi_{k'}^{(0)} \prod_{j=1}^{S} P(s_{ij} = 1|y_i = k')\mathbf{1}_{s_{ij}=1}},$$

where both the prior distribution of each of the causes, $\pi_k^{(0)}$ and the conditional probabilities $P(s_{ij} = 1|y_i = k)$ are fixed values provided in the InterVA software. It is worth noting that the formula does not follow the standard Bayes' rule as it omits the probability that any symptom is absent. A detailed discussion of this modeling choice can be found in McCormick et al. (2016). The conditional probabilities, $P(s_{ij} = 1|y_i = k)$, used in InterVA algorithms are represented as rankings with letter grades instead of numerical values (Byass et al., 2012). For example, $P(s_{ij} = 1|y_i = k) = A+$ is translated into $P(s_{ij} = 1|y_i = k) = 0.8$, etc. The standard InterVA software only supports the fixed set of symptoms and causes where such prior information is provided. For a different data input format, this formulation can be easily generalized if training data are available. We include in the **openVA** package an extension of the algorithm that calculates $\hat{P}(s_{ij} = 1|y_i = k)$ from the empirical distribution in the training data and then maps to letter grades with different truncation rules. Details of the extended InterVA algorithm can be found in McCormick et al. (2016).

After the individual COD distributions are calculated, InterVA-4 utilizes a series of pre-defined rules to identify up to the top three most likely COD assignments, and truncates the probabilities for the rest of the CODs to 0 and adds an 'undetermined' category so that the probabilities sum up to 1 (See the user guide of Byass (2015)). Then the population-level CSMFs are calculated as the aggregation of individual COD distributions, such that

$$\pi_k = \sum_{i=1}^{N} P_{ik}^*,$$

where P_{ik}^* denotes the individual COD distribution after introducing the undetermined category.

- **Naive Bayes Classifier** (Miasnikof et al., 2015) is very similar to the InterVA algorithm with two major differences. First, instead of considering only symptoms that present, the NBC algorithm also considers symptoms that are absent. Second, the conditional probabilities of symptoms given causes are calculated from training data instead of given by physicians, which is similar to our extension of InterVA discussed above. Similar to InterVA, the NBC method can be written as

$$P_{ik} = \frac{\pi_k^{(0)} \prod_{j=1}^{S} (P(s_{ij} = 1|y_i = k)\mathbf{1}_{s_{ij}=1} + P(s_{ij} \neq 1|y_i = k)\mathbf{1}_{s_{ij}\neq 1})}{\sum_{k'=1}^{C} \pi_{k'}^{(0)} \prod_{j=1}^{S} (P(s_{ij} = 1|y_i = k')\mathbf{1}_{s_{ij}=1} + P(s_{ij} \neq 1|y_i = k')\mathbf{1}_{s_{ij}\neq 1})},$$

and the CSMFs are calculated by $\pi_k = \sum_{i=1}^{N} P_{ik}$.

- **InSilicoVA** algorithm (McCormick et al., 2016) assumes a generative model that characterizes both the CSMF at the population level, and the COD distributions at the individual level. In short, the core generative process assumes

$$\begin{aligned} s_{ij}|y_i = k &\propto \text{Bernoulli}(P(s_{ij}|y_i = k)), && (1)\\ y_i|\pi_1, ..., \pi_C &\propto \text{Categorical}(\pi_1, ..., \pi_C), && (2)\\ \pi_k &= \exp\theta_k / \sum_{k=1}^{C} \exp\theta_k, && (3)\\ \theta_k &\propto \text{Normal}(\mu, \sigma^2). && (4) \end{aligned}$$

Hyperpriors are also placed on $P(s_{ij}|y_i = k)$, μ, and σ^2. The priors for $P(s_{ij}|y_i = k)$ are set by the rankings used in InterVA-4 if the data are prepared into InterVA format, otherwise they are learned from training data. The priors on μ and σ^2 are diffuse uniform priors. Parameter estimation is performed using MCMC, so that a sample of posterior distributions of π_k can be obtained after the sampler converges.

- **Tariff** algorithm (James et al., 2011) differs from the other three methods in that it does not calculate an explicit probability distribution of the COD for each death. Instead, for each death i, a Tariff score is calculated for each COD k so that

$$Score_{ik} = \sum_{j=1}^{S} \text{Tariff}_{kj}\mathbf{1}_{s_{ij}=1},$$

where the symptom-specific Tariff score Tariff_{kj} is defined as

$$\text{Tariff}_{kj} = \frac{n_{kj} - median(n_{1j}, n_{2j}, ..., n_{Cj})}{IQR(n_{1j}, n_{2j}, ..., n_{Cj})},$$

where n_{kj} is the count of how many deaths from cause k contain symptom j in the training data. The Tariff scores are then turned into rankings by comparing them to a reference distribution of scores calculated from re-sampling the training dataset to obtain a uniform COD distribution. It is worth noting that the Tariff algorithm produces the COD distribution for each death in terms of their rankings instead of the probability distributions. And thus the CSMF for each cause k is calculated by the fraction of deaths with cause k being the highest ranked cause, i.e.,

$$\pi_k = \frac{\sum_{i=1}^{N} \mathbf{1}_{y_i=k}}{N}.$$

In addition to the different model specifications underlying each algorithm, there is also a major conceptual difference in handling missing symptoms across the algorithms. Missing symptoms could arise from different stages of the data collection process. For example, the respondent may not know whether certain symptoms existed or may refuse to answer a question. From a statistical point of view, knowing that a symptom does not exist provides some information to the possible cause assignment, while a missing symptom does not. Although in theory, most of the VA algorithms could benefit from distinguishing 'missing' from 'absence', InSilicoVA is the only algorithm that has been implemented to acknowledge missing data. Missing indicators are assumed to be equivalent to 'absence' in InterVA, NBC, and Tariff.

3 Data preparation

In the **openVA** package, we consider two main types of standardized questionnaire: the WHO instrument and the IHME questionnaire. In this section, we focus on describing these two data formats and tools to clean and convert data. Pre-processing the raw data collected from the survey instrument (usually with Open Data Toolkit) is usually performed with additional packages and software outside of the analysis pipeline in R. We briefly mention software for data pre-processing towards the end of this paper.

The WHO standard format

For users familiar with InterVA software and the associated data processing steps, the standard input format from the WHO 2012 and 2016 instruments is usually well understood. For the 2012 instrument, the data expected by the InterVA-4 software are organized into a data frame where each row represents one death and the corresponding VA information is contained in 246 fields, starting from the first item being the ID of the death. The 245 items following the ID each represent one binary variable of symptom/indicator, where 'presence' is coded by 'Y', and 'absence' is coded by an empty cell.

To accommodate updates for the WHO 2016 instrument (D'Ambruoso et al., 2017), the InterVA-5 software accepts a data frame with 354 columns that include 353 columns of symptom/indicators followed by an additional column for the record ID. It should be noted that the R package **InterVA5** retains the format with the record ID residing in the first column. Another important update with InterVA-5 is that it acknowledges the difference between "Yes" and "No" (or "Y/y" and "N/n", which is different from the coding scheme in InterVA-4), both of which are processed as relevant responses, while all other responses are treated as missing values and ignored. With respect to the list of causes of death, InterVA-5 utilizes the WHO 2016 COD categories, which is nearly identical to the WHO 2012 COD categories (used by InterVA-4) except that hemorrhagic fever and dengue fever are two separate categories in the 2016 COD categories.

The same input format is inherited by the **openVA** package, except for one modification. We further distinguish 'missing' and 'absence' in the input data frame explicitly. We highly recommend that users pre-process all the raw data so that a 'missing' value in the data spreadsheet is coded as a '.' (following the Stata practice familiar to many VA practitioners), and an 'absence' value is indicated by an empty cell, as in the standard InterVA-4 software. For WHO 2016 data, both '.' and '-' (following the default coding scheme of InterVA-5 software) are interpreted as missing values. For methods other than InSilicoVA, 'missing' and 'absence' will be considered the same internally and thus will not introduce a compatibility problem.

The PHMRC format

The Population Health Metrics Research Consortium (PHMRC) gold standard VA data (Murray et al., 2011) consist of three datasets corresponding to adult, child, and neonatal deaths, respectively. All deaths occurred in health facilities and gold-standard causes are determined based on laboratory, pathology and medical imaging findings. These datasets can be downloaded directly using the link returned by the function `getPHMRC_url()`. For example, we can read the adult VA dataset using the following command.

```
PHMRC_adult <- read.csv(getPHMRC_url("adult"))
```

Although the data are publicly accessible, a major practical challenge for studies involving the PHMRC gold standard dataset is that the pre-processing steps described from the relevant publications are not clear enough nor easy to implement. The **openVA** package internally cleans up the PHMRC gold standard data when calling the `codeVA()` function on the PHMRC data. The procedure follows the steps described in the supplement material of McCormick et al. (2016). Notice that the original PHMRC data are useful for comparing and validating new methods, as well as for using as training data, but the cleaning functions only require that the columns are exactly the same as the PHMRC gold standard datasets, so they could also be used for new data that are pre-processed into the same format.

Customized format

In addition to the two standard questionnaires discussed previously, researchers might also be interested in including customized dichotomous symptoms in their analysis. The **openVA** package also supports customized inputs as long as they are dichotomous. In such case, neither the built-in conditional probability matrix of InterVA nor the PHMRC gold standard dataset could be used to learn the relationship between training and testing data, thus different training data with known causes of death are necessary for all three algorithms. The `ConvertData()` function can be used to convert data with customized coding schemes into the format recognized by the **openVA** package.

Finally, we note that the **openVA** package currently does not reformat data from one standardized questionnaire to another. This is because mapping the symptoms collected from one questionnaire to those collected by another questionnaire inevitably creates loss of information. Such mapping tasks can be useful for some applications. For example, a full mapping of a PHMRC dataset into the WHO format enables the use of physician provided conditional probabilities included in the InterVA software on data collected by PHMRC questionnaires. This remains as an important feature to be added to the package in the future.

4 Fitting VA cause-of-death assignment models

In this section, we demonstrate the model fitting process in the **openVA** package using two datasets: (1) a random sample of 1,000 deaths from the ALPHA network without cause-of-death labels collected with the WHO 2012 instrument, and (2) the adult VA records in the PHMRC gold standard data, with the 1,554 records from Andhra Pradesh, India used as a testing set and the rest used as a training dataset. In the first case without gold standard training data, only InterVA and InSilicoVA can be fitted. All four methods can be fitted in the second case.

Modeling data collected with WHO 2012 questionnaire

The randomly sampled VA records from the ALPHA network sites are already included in the openVA package as a dataset `RandomVA1` and can be loaded directly.

```
data(RandomVA1)
dim(RandomVA1)

#> [1] 1000  246

head(RandomVA1[, 1:10])

#>   ID elder midage adult child under5 infant neonate male female
#> 1 d1     Y                                            Y
#> 2 d2     Y                                                   Y
#> 3 d3            Y                                     Y
```

```
#> 4 d4                  Y                                        Y
#> 5 d5                  Y                                  Y
#> 6 d6                  Y                                        Y
```

The codeVA() function provides a standardized syntax to fit different VA models. Internally, the codeVA() function organizes the input data according to the specified data type, checks for incompatibility of the data and specified model, and calls the corresponding model fitting functions. It returns a classed object of the specified model class. In this example, we use version 4.03 of the InterVA software, which is the latest release of the original software compatible with the WHO 2012 instrument. Any additional model-specific parameters can be passed through the arguments of codeVA(). Here we specify the HIV and malaria prevalence levels required by the InterVA model to be 'high'. Guidelines on how to set these parameters can be found in Byass et al. (2012).

```
fit_inter_who <- codeVA(data = RandomVA1, data.type = "WHO2012",
                        model = "InterVA", version = "4.03",
                        HIV = "h", Malaria = "h")

summary(fit_inter_who)

#> InterVA-4 fitted on 1000 deaths
#> CSMF calculated using reported causes by InterVA-4 only
#> The remaining probabilities are assigned to 'Undetermined'
#>
#> Top 5 CSMFs:
#>  cause                     likelihood
#>  Undetermined              0.154
#>  HIV/AIDS related death    0.122
#>  Stroke                    0.072
#>  Reproductive neoplasms MF 0.058
#>  Pulmonary tuberculosis    0.055
```

We can implement InSilicoVA method with similar syntax. We use the default parameters and run the MCMC for 10,000 iterations. Setting the auto.length argument to FALSE specifies that the algorithm does not automatically increase the length of the chain when convergence failed. In practice, we recommend setting this argument to TRUE if the algorithm displays warnings concerning MCMC convergence. The InSilicoVA algorithm is implemented using a Metropolis-Hastings within Gibbs sampler. The acceptance rate is printed as part of the message as the model samples from the posterior distribution.

```
fit_ins_who <- codeVA(RandomVA1, data.type = "WHO2012", model = "InSilicoVA",
                      Nsim = 10000, auto.length = FALSE)

summary(fit_ins_who)

#> InSilicoVA Call:
#> 1000 death processed
#> 10000 iterations performed, with first 5000 iterations discarded
#>  250 iterations saved after thinning
#> Fitted with re-estimated conditional probability level table
#> Data consistency check performed as in InterVA4
#>
#> Top 10 CSMFs:
#>                                  Mean Std.Error Lower Median Upper
#> Other and unspecified infect dis 0.266    0.0168 0.235  0.265 0.301
#> HIV/AIDS related death           0.102    0.0091 0.085  0.102 0.119
#> Renal failure                    0.101    0.0108 0.084  0.101 0.123
#> Other and unspecified neoplasms  0.062    0.0089 0.046  0.061 0.080
#> Other and unspecified cardiac dis 0.058   0.0076 0.044  0.058 0.075
#> Digestive neoplasms              0.050    0.0077 0.033  0.050 0.065
#> Acute resp infect incl pneumonia 0.048    0.0073 0.034  0.049 0.063
#> Pulmonary tuberculosis           0.039    0.0068 0.025  0.039 0.054
#> Stroke                           0.038    0.0061 0.027  0.038 0.052
#> Other and unspecified NCD        0.034    0.0089 0.018  0.034 0.052
```

Modeling the PHMRC data

In the second example, we consider a prediction task using the PHMRC adult dataset. We first load the complete PHMRC adult dataset from its on-line repository, and organize it into training and test datasets. We treat all deaths from Andhra Pradesh, India as the test dataset.

```
PHMRC_adult <- read.csv(getPHMRC_url("adult"))
is.test <- which(PHMRC_adult$site == "AP")
test <- PHMRC_adult[is.test, ]
train <- PHMRC_adult[-is.test, ]
dim(test)

#> [1] 1554  946

dim(train)

#> [1] 6287  946
```

In order to fit the models on the PHMRC data, we specify `data.type = "PHMRC"` and `phmrc.type = "adult"` to indicate the data input is collected using the PHMRC adult questionnaire. We also specify the column of the causes-of-death label in the training data. The rest of the syntax is similar to the previous example.

When the input consists of both training and testing data, the InterVA and InSilicoVA algorithms estimate the conditional probabilities of symptoms using the training data, instead of using the built-in values. In such case, the `version` argument for the InterVA algorithm is suppressed. There are several ways to map the conditional probabilities of symptoms given causes in the training dataset to a letter grade system, specified by the `convert.type` argument. The `convert.type = "quantile"` performs the mapping so that the percentile of each rank stays the same as the original $P_{s|c}$ matrix in InterVA software. Alternatively we can also use the original fixed values of translation, and assign letter grades closest to each entry in $\hat{P}_{s|c}$. This conversion is specified by `convert.type = "fixed"`, and is more closely aligned to the original InterVA and InSilicoVA setting. Finally, we can also directly use the values in the $\hat{P}_{s|c}$ without converting them to ranks and re-estimating the values associated with each rank. This can be specified by `convert.type = "empirical"`. In this demonstration, we assume the fixed value conversion.

```
fit_inter <- codeVA(data = test, data.type = "PHMRC", model = "InterVA",
                    data.train = train, causes.train = "gs_text34",
                    phmrc.type = "adult", convert.type = "fixed")

fit_ins <- codeVA(data = test, data.type = "PHMRC", model = "InSilicoVA",
                  data.train = train, causes.train = "gs_text34",
                  phmrc.type = "adult", convert.type = "fixed",
                  Nsim=10000, auto.length = FALSE)
```

The NBC and Tariff method can be fit using similar syntax.

```
fit_nbc <- codeVA(data = test, data.type = "PHMRC", model = "NBC",
                  data.train = train, causes.train = "gs_text34",
                  phmrc.type = "adult")

fit_tariff <- codeVA(data = test, data.type = "PHMRC", model = "Tariff",
                     data.train = train, causes.train = "gs_text34",
                     phmrc.type = "adult")
```

Notice that we do not need to transform the PHMRC data manually. Data transformations are performed automatically within the `codeVA()` function.

5 Summarizing results

In this section we demonstrate how to summarize results, extract output, and visualize and compare fitted results. All the fitted object returned by `codeVA()` are S3 objects, for which a readable summary of model results can be obtained with the `summary()` function as shown in the previous section. In addition, several other metrics are commonly used to evaluate and compare VA algorithms at either the population or individual levels. In the rest of this section, we show how to easily calculate and visualize some of these metrics with the **openVA** package.

CSMF accuracy

We can extract the CSMFs directly using the `getCSMF()` function. The function returns a vector of the point estimates of the CSMFs, or a matrix of posterior summaries of the CSMF for the InSilicoVA algorithm.

```
csmf_inter <- getCSMF(fit_inter)
csmf_ins <- getCSMF(fit_ins)
csmf_nbc <- getCSMF(fit_nbc)
csmf_tariff <- getCSMF(fit_tariff)
```

One commonly used metric to evaluate the CSMF estimates is the so-called CSMF accuracy, defined as

$$CSMF_{acc} = 1 - \frac{\sum_j^C CSMF_i - CSMF_j^{(true)}}{2(1 - \min CSMF^{(true)})}$$

The CSMF accuracy can be readily computed using functions in **openVA** as the codes below shows.

```
csmf_true <- table(c(test$gs_text34, unique(PHMRC_adult$gs_text34))) - 1
csmf_true <- csmf_true / sum(csmf_true)
c(getCSMF_accuracy(csmf_inter, csmf_true, undet = "Undetermined"),
  getCSMF_accuracy(csmf_ins[, "Mean"], csmf_true),
  getCSMF_accuracy(csmf_nbc, csmf_true),
  getCSMF_accuracy(csmf_tariff, csmf_true))
```

```
#> [1] 0.53 0.74 0.77 0.68
```

We use the empirical distribution in the test data to calculate the true CSMF distribution, i.e., $CSMF_j^{(true)} = \frac{1}{n}\sum_{i=1}^n \mathbf{1}_{y_i=j}$. Then we evaluate the CSMF accuracy using the `getCSMF_accuracy()` function. As discussed previously, the default CSMF calculation is slightly different for diCfferent methods. For example, the InterVA algorithm creates the additional category of `Undetermined` by default, which is not in the true CSMF categories and needs to be specified. The creation of the undetermined category can also be suppressed by `interVA.rule = FALSE` in the `getCSMF()` function call. For the InSilicoVA algorithm, we use the posterior mean to calculate the point estimates of the CSMF accuracy.

Individual COD summary

At the individual level, we can extract the most likely cause-of-death assignment from the fitted object using the `getTopCOD()` function.

```
cod_inter <- getTopCOD(fit_inter)
cod_ins <- getTopCOD(fit_ins)
cod_nbc <- getTopCOD(fit_nbc)
cod_tariff <- getTopCOD(fit_tariff)
```

With the most likely COD assignment, other types of metrics based on individual COD assignment accuracy can be similarly constructed by users. The summary methods can also be called for each death ID. For example, using the Tariff method, we can extract the fitted rankings of causes for the death with ID 6288 by

```
summary(fit_inter, id = "6288")
```

```
#> InterVA-4 fitted top 5 causes for death ID: 6288
#>
#>  Cause                     Likelihood
#>  Stroke                    0.509
#>  Pneumonia                 0.318
#>  COPD                      0.081
#>  Other Infectious Diseases 0.064
#>  Renal Failure             0.013
```

The `summary()` function for InSilcoVA does not provide uncertainty estimates for individual COD assignments by default. This is because, in practice, the calculation of individual posterior probabilities of COD distribution can be memory-intensive when the dataset is large. To obtain individual-level uncertainty measurements, we can either run the MCMC chain with the additional argument `indiv.CI = 0.95` when calling `codeVA()`, or update the fitted object directly with the saved posterior draws.

```
fit_ins <- updateIndiv(fit_ins, CI = 0.95)
summary(fit_ins, id = "6288")

#> InSilicoVA fitted top  causes for death ID: 6288
#> Credible intervals shown: 95%
#>                               Mean  Lower Median  Upper
#> Stroke                      0.5043 0.3485 0.5083 0.6361
#> Pneumonia                   0.4116 0.2615 0.4083 0.5834
#> Other Infectious Diseases   0.0660 0.0411 0.0642 0.0966
#> Epilepsy                    0.0099 0.0064 0.0097 0.0142
#> COPD                        0.0053 0.0031 0.0052 0.0079
#> Malaria                     0.0007 0.0005 0.0007 0.0011
#> Diabetes                    0.0005 0.0003 0.0005 0.0009
#> Acute Myocardial Infarction 0.0004 0.0003 0.0004 0.0006
#> Falls                       0.0004 0.0001 0.0004 0.0013
#> Renal Failure               0.0004 0.0002 0.0003 0.0005
```

For N deaths, C causes, the posterior mean of individual COD distributions returned by the InSilicoVA model, along with median and with credible intervals can be represented by a $(N \times C \times 4)$-dimensional array. The function `getIndivProb()` extracts this summary in the form of a list of 4 matrices of dimension N by C, which can then be saved to other formats to facilitate further analysis. For other methods, the point estimates of individual COD distribution are returned as the N by C matrix.

```
fit_prob <- getIndivProb(fit_inter)
dim(fit_prob)

#> [1] 1554   34
```

Visualization

The previous sections discuss how results could be extracted and examined in R. In this subsection, we show some visualization tools provided in the **openVA** package for presenting these results. The fitted CSMFs for the top causes can be easily visualized by the `plotVA()` function. The default graph components are specific to each algorithm and individual package implementations, with options for further customization. For example, Figure1 shows the estimated CSMF from the InterVA algorithm in the PHMRC data example.

```
plotVA(fit_inter, title = "InterVA")
```

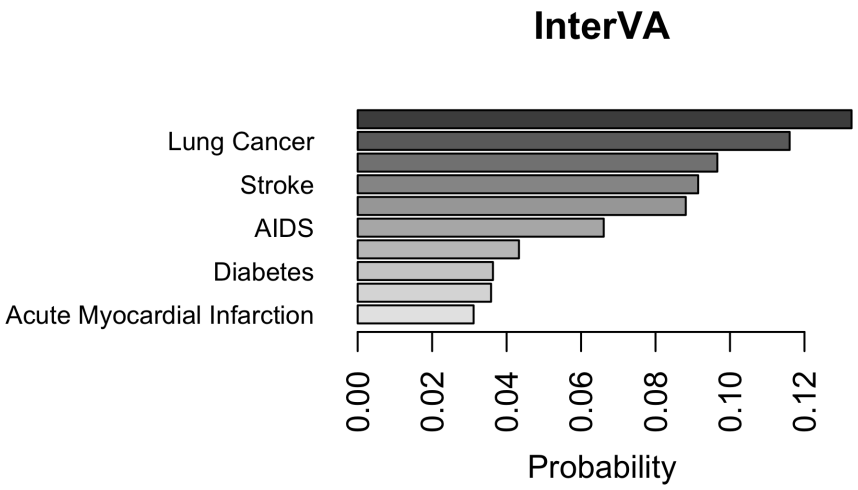


Figure 1: Bar plot of the top CSMFs estimated by InterVA. The five causes of death with the highes estimated mortality fraction predicted by InterVA are plotted. Colors of the bar indicate the corresponding fractions.

The CSMFs can also be aggregated for easier visualization of groups of causes. For the InterVA-4 cause list, we included an example grouping built into the package, so the aggregated CSMFs can be compared directly. In practice, the grouping of causes of deaths often needs to be determined according

to context and the research question of interest. Changing the grouping can be easily achieved by modifying the `grouping` argument in the `stackplotVA()` function. For example, to facilitate the new category of `Undetermined` returned by InterVA, we first modify the grouping matrix to include it as a new cause and visualize the aggregated CSMF estimates in Figure 2.

```
data(SampleCategory)
grouping <- SampleCategory
grouping[,1] <- as.character(grouping[,1])
grouping <- rbind(grouping, c("Undetermined", "Undetermined"))
compare <- list(InterVA4 = fit_inter_who,
                InSilicoVA = fit_ins_who)
stackplotVA(compare, xlab = "", angle = 0,  grouping = grouping)
```

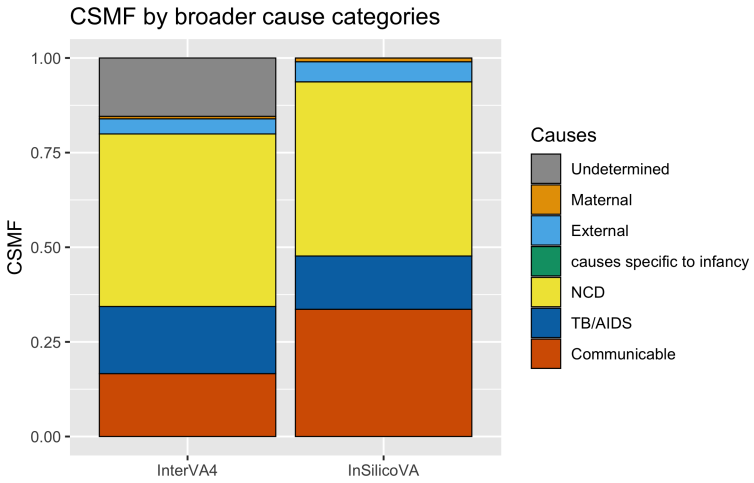


Figure 2: Estimated aggregated CSMFs for InterVA-4 and InSilicoVA, adding undetermined category. The causes of death are aggregated to seven broad categories. Color indicate the category and the height of bar indicate the fraction of deaths due to the cause category. Overall, InterVA-4 and InSilicoVA estimate similar aggregated CSMFs, except in the communicable disease (excluding TA/AIDS) category, InSilicoVA estimates a larger fraction.

The ordering of the stacked bars can also be changed to reflect the structures within the aggregated causes, as demonstrated in Figure 3.

```
group_order <- c("TB/AIDS",  "Communicable", "NCD", "External", "Maternal",
            "causes specific to infancy", "Undetermined")
stackplotVA(compare, xlab = "", angle = 0, grouping = grouping,
            group_order = group_order)
```

6 Incorporating additional information

Among the VA methods discussed in this paper, the InSilicoVA algorithm (McCormick et al., 2016) allows for more flexible modifications to the Bayesian hierarchical model structure when additional information is available. In this section, we illustrate two features unique to the InSilicoVA method: jointly estimating CSMFs from multiple populations, and incorporating partial and potentially noisy physician coding into the algorithm.

Sub-population specific CSMFs

In practice researchers may want to estimate and compare CSMFs for different regions, time periods, or demographic groups in the population. Running separate models on subsets of data can be inefficient and does not allow parameter estimation to borrow information across different groups. The generative framework adopted by InSilicoVA allows the specification of sub-populations in analyzing VA data. Consider an input dataset with G different sub-populations. We can estimate different CSMFs $\pi^{(g)}$ for $g = 1, ..., G$ for each sub-population, while assuming the same conditional probability matrix, $P_{s|c}$ and other hyperpriors. As an example, we show how to estimate different CSMFs for sub-populations specified by sex and age groups, using a randomly sampled ALPHA dataset with additional columns specifying the sub-population each death belongs to.

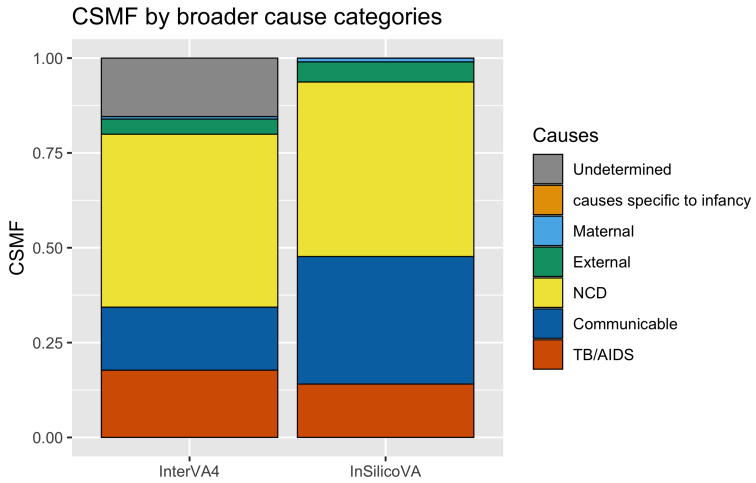


Figure 3: Estimated aggregated CSMFs for InterVA-4 and InSilicoVA, with the causes reordered. The causes of death are aggregated to seven broad categories. Color indicate the category and the height of bar indicate the fraction of deaths due to the cause category. The order of the bars are arranged according to user input.

```
data(RandomVA2)
head(RandomVA2[, 244:248])

#>   stradm smobph scosts   sex age
#> 1      .      .      .   Men 60+
#> 2      .      .      . Women 60-
#> 3      .      .      . Women 60-
#> 4      .      .      . Women 60+
#> 5      .      .      . Women 60-
#> 6      .      .      . Women 60-
```

Then we can fit the model with one or multiple additional columns specifying sub-population membership for each observation.

```
fit_sub <- codeVA(RandomVA2, model = "InSilicoVA",
              subpop = list("sex", "age"),  indiv.CI = 0.95,
              Nsim = 10000, auto.length = FALSE)
```

Functions discussed in the previous sections work in the same way for the fitted object with multiple sub-populations. Additional visualization tools are also available. Figure 4 plots the CSMFs for two sub-populations on the same plot by specify `type = "compare"`.

```
plotVA(fit_sub, type = "compare", title = "Comparing CSMFs", top = 3)
```

By default, the comparison plots will select all the CODs that are included in the top causes (specified by the `top` argument) for each of the sub-populations. We can also plot only subsets of them by specifying the causes of interest, as shown in Figure 5.

```
plotVA(fit_sub, type = "compare", title = "Comparing CSMFs",
               causelist = c("HIV/AIDS related death",
                             "Pulmonary tuberculosis",
                             "Other and unspecified infect dis",
                             "Other and unspecified NCD"))
```

Figure 6 shows the visualization of the top CSMFs for a chosen sub-population using the `which.sub` argument.

```
plotVA(fit_sub, which.sub = "Women 60-", title = "Women 60-")
```

Similar to before, the `stackplotVA()` function can also be used to compare different sub-populations in aggregated cause groups, as shown in Figure 7.

```
stackplotVA(fit_sub)
stackplotVA(fit_sub, type = "dodge")
```

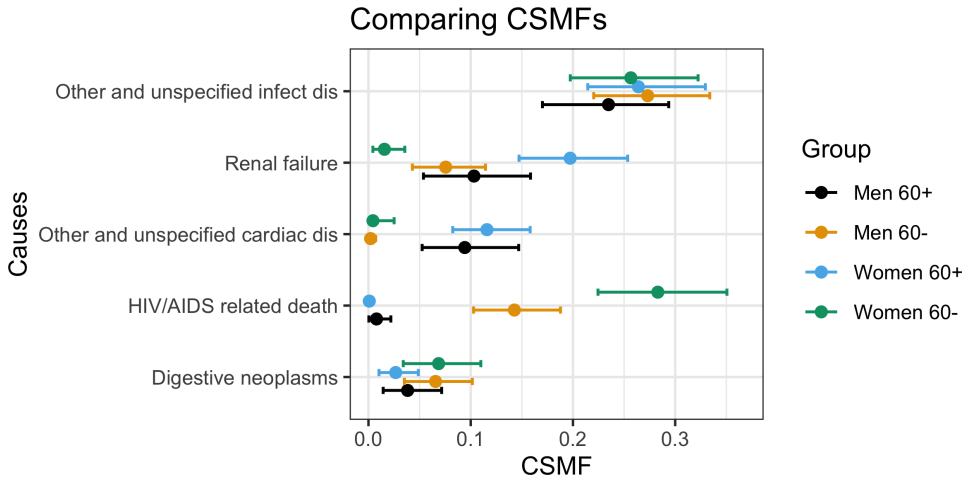


Figure 4: Estimated CSMFs for different sub-populations. The points indicate posterior means of the CSMF and the error bars indicate 95% credible intervals of the CSMF.

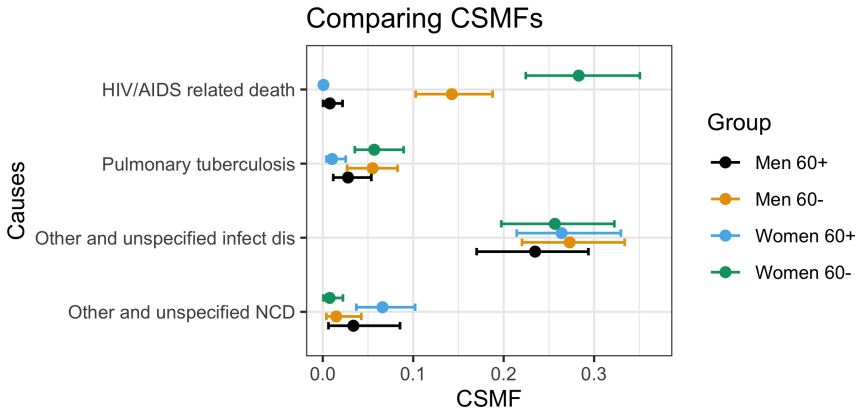


Figure 5: Estimated fraction of deaths due to selected CODs for different sub-populations. The points indicate posterior means of the CSMF and the error bars indicate 95% credible intervals of the CSMF.

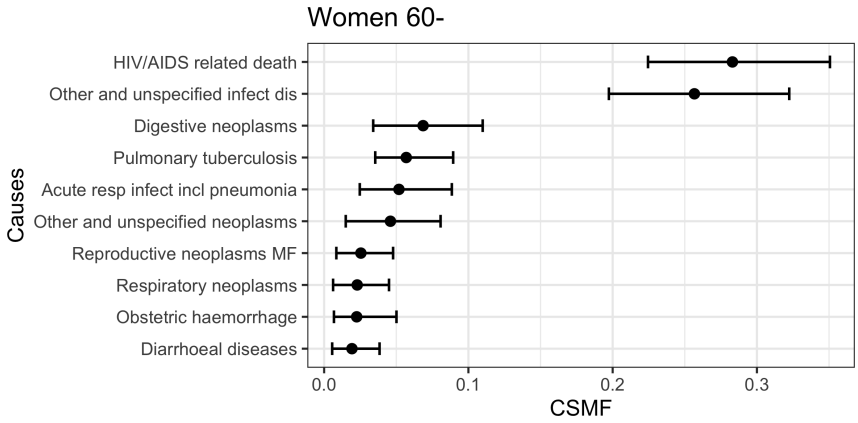


Figure 6: Top 10 CSMFs for a specified sub-population (women under 60 years old).

Physician coding

When physician-coded causes of death are available for all or a subset of the deaths, we can incorporate such information in the InSilicoVA model. The physician-coded causes can be either the same as the CODs used for the final algorithm, or consist of causes at a higher level of aggregation. When there is more than one physician code for each death and the physician identity is known, we can first de-bias the multiple codes provided from different physicians using the process described in McCormick et al. (2016). For the purpose of implementation, we only need to specify which columns are physician IDs, and which are their coded causes, respectively.

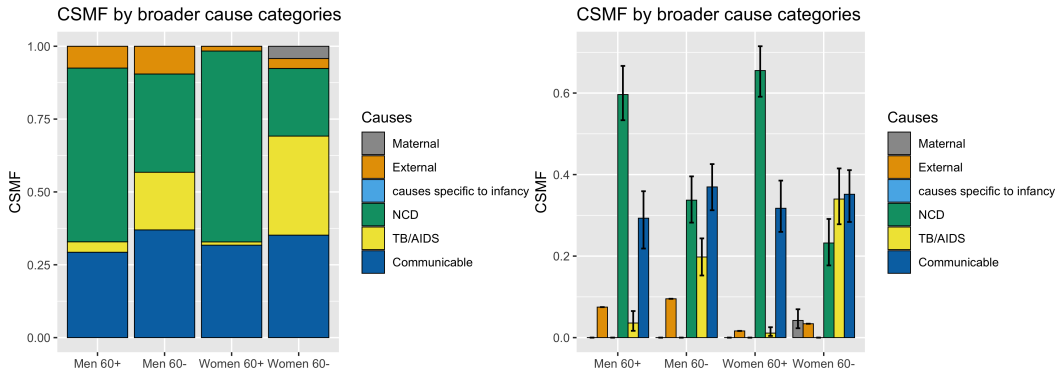


Figure 7: Aggregated CSMFs for four different sub-populations. Left: comparison using the stacked bar chart. Right: comparison using the bar chart arranged side to side. The height of the bars indicate posterior means of the CSMF and the error bars indicate 95% credible intervals of the CSMF. Sub-populations corresponding to 60+ age groups are estimated to have a larger fraction of deaths due to NCD compared to 60- age groups.

```
data(SampleCategory)
data(RandomPhysician)
head(RandomPhysician[, 245:250])
```

```
#>   smobph scosts        code1 rev1        code2 rev2
#> 1      .       .          NCD doc9          NCD doc6
#> 2      .       .          NCD doc4          NCD doc3
#> 3      .       .          NCD doc1          NCD doc5
#> 4      .       .      TB/AIDS doc4      TB/AIDS doc7
#> 5      .       .      TB/AIDS doc5      TB/AIDS doc9
#> 6      .       . Communicable doc9 Communicable <NA>
```

```
doctors <- paste0("doc", c(1:15))
causelist <- c("Communicable", "TB/AIDS", "Maternal",
               "NCD", "External", "Unknown")
phydebias <- physician_debias(RandomPhysician,
          phy.id = c("rev1", "rev2"), phy.code = c("code1", "code2"),
          phylist = doctors, causelist = causelist, tol = 0.0001, max.itr = 100)
```

The de-biased step essentially creates a prior probability distribution for each death over the broader categories of causes. Then to run InSilicoVA with the de-biased physician coding, we can simply pass the fitted object from the previous step to the model. Additional arguments are needed to specify both the external cause category, since external causes are handled by separate heuristics, and the unknown category, which is equivalent to a uniform probability distribution over all other categories, i.e., the same as the case where no physician coding exists.

```
fit_ins_phy <- codeVA(RandomVA1, model = "InSilicoVA",
             phy.debias = phydebias, phy.cat = SampleCategory,
             phy.external = "External", phy.unknown = "Unknown",
             Nsim = 10000, auto.length = FALSE)
```

Figure 8 compares the previous results without including physicians codes.

```
plotVA(fit_ins_who, title = "Without physician coding")
plotVA(fit_ins_phy, title = "With physician coding")
```

Removal of physically impossible causes

The originally proposed InSilicoVA assumes all causes of death are possible for each observation. The impact from such an assumption is mild when data are abundant, but could be problematic when either the sample size is small or the proportion of missing data is high. In both cases, physically impossible causes might get assigned with non-ignorable posterior mass. Since version 1.1.5 of **InSilicoVA**, when the input is in the WHO format, the algorithm automatically checks and removes impossible causes before fitting the model. The k-th cause is defined as physically impossible for the i-th death if $P(s_{ij} = 1|y_i = k) = 0$ where s_{ij} is an indicator that the decedent belongs to a particular sex

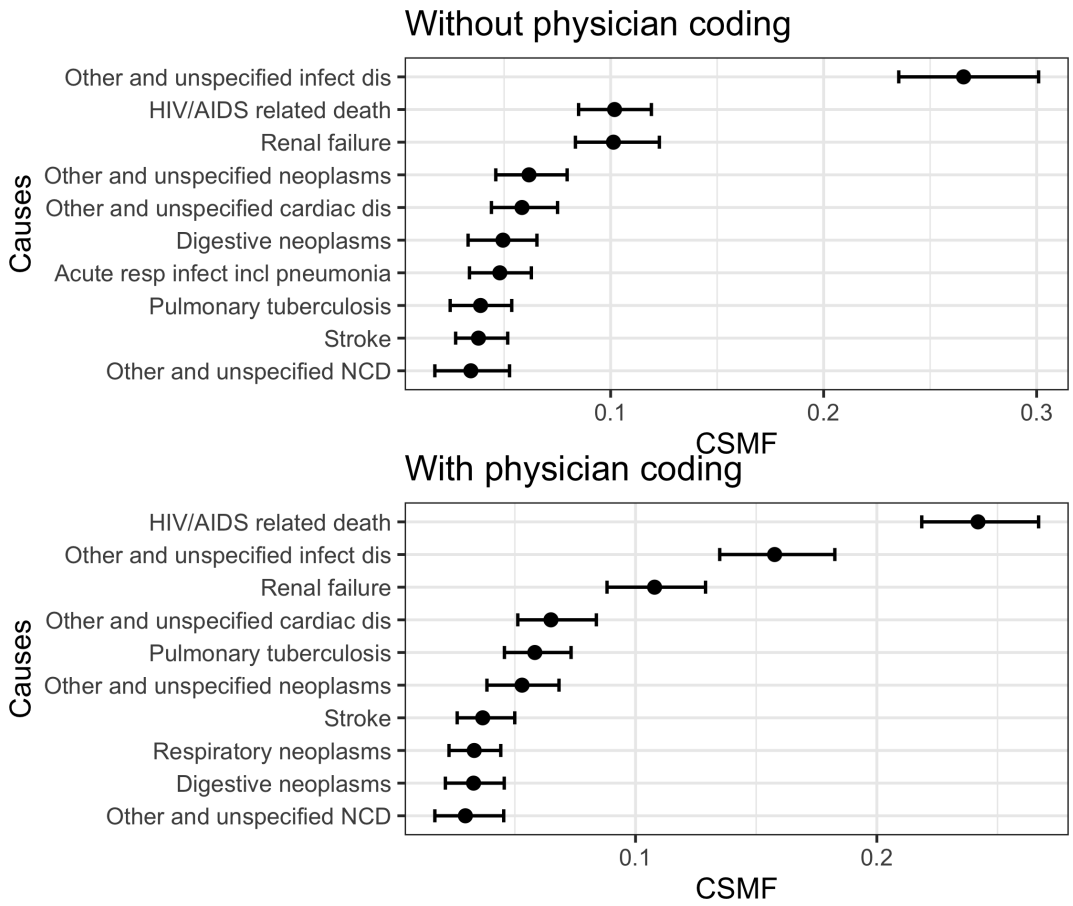


Figure 8: Comparing fitted CSMF with and without physicians. Top: top 10 CSMFs and the 95% credible intervals estimated by InSilicoVA without using physician coding. Bottom: top 10 CSMFs and the 95% credible intervals estimated by InSilicoVA taking physician coding into account. More deaths were assigned from 'Other and unspecified infectious disease' to 'HIV/AIDS related death' after accounting for physician coding.

or age group. We then consider a cause to be physically impossible for the underlying population if it is impossible for all the observations in the input data. For example, with the new implementation, CSMF for pregnancy-related causes will not be estimated if the input data consist of only male deaths.

7 Other related software packages

Since the release of the **openVA** package on CRAN, there have been several new developments in both methodology and software that build on the **openVA** suite of packages and further extend its functionalities. Here we briefly survey some of the related methods, software packages, and ongoing work in the VA community that are related to **openVA**.

First, the ability to easily fit and compare existing methods using the **openVA** package facilitated the development of several new VA methods in the last several years. Most of the development focuses on building statistical models to relax the conditional independence assumption of symptoms given a cause of death (e.g., Li et al., 2020; Kunihama et al., 2020; Moran et al., 2021; Li et al., 2021; Wu et al., 2021). These methods tend to be computationally more demanding compared to the algorithms currently included in the **openVA** package, but usually provide improved inference. It is future work to include some of these latest developments in the **openVA** package for routine use. Most of these methods have publicly available implementations, such as the **farva** package (Moran, 2020) on GitHub. In another direction of research, a class of transfer learning methods focuses on building models to correct for bias in existing methods when predicting out of domain. These methods take the predicted cause of death assignments and distributions obtained with the **openVA** package and learn an ensemble of the predictions calibrated to the target population (Datta et al., 2020; Fiksel et al., 2021). The **calibratedVA** package (Fiksel and Datta, 2018) is available to implement these models.

Outside of research community that develops new VA algorithms, **openVA** has also been used

extensively by governments and health care organizations, particularly in locations that lack a strong vital registration system and use VA data to identify the leading causes of death. To facilitate the work of these users, **openVA** has been wrapped into a web application, the openVA App (Thomas, 2021), using the **shiny** package (Chang et al., 2021). The open source openVA App is available on GitHub and provides an intuitive interface to **openVA** that does not require one to learn R, but provides visual and tabular output produced by the different VA algorithms. It also runs the official Tariff algorithm by calling the Python source code of SmartVA-Analyze (Institute for Health Metrics and Evaluation, 2021a) and processing the output to be consistent with the other algorithms. The **openVA** R package has also been a key component in a larger data analysis pipeline that pulls VA data from an Open Data Kit (ODK) Collect server and deposits the assigned causes of death to another server running the District Health Information Software 2 (DHIS2), which is a common information management system used in low and middle-income countries. This open-source software is implemented as a Python package, openva-pipeline, and is available on GitHub and the Python Package Index (Thomas et al., 2021a). Finally, the R package **CrossVA** (Thomas et al., 2020) and the Python package pyCrossVA (Choi et al., 2021) provide additional toolkits to convert raw VA data from its original format from the ODK server to the standardized formats discussed before. Both packages are open source and available on GitHub. The pyCrossVA package is also available on the Python Package Index.

8 Conclusion

In this paper, we introduce the **openVA** package. This is the first open-source software that implements and compares the major VA methods. The **openVA** package allows researchers to easily access the most recent tools that were previously difficult to work with or unavailable. It also enables the compatibility of multiple data input formats and significantly reduces the tedious work to pre-process different data formats specific to each algorithm. The software framework of the **openVA** package allows for the integration of new methods developed in the future. The **openVA** package makes all the steps involved in analyzing VA data – i.e., data processing, model tuning and fitting, summarizing results, and evaluation metrics – transparent and reproducible. This contributes significantly to the public health community using VA.

Finally, we make note of several features that will be helpful for future development. First, many users of the **openVA** package may not be familiar with the command line tools or do not have access to R on their local machines. A well designed graphical user interface can be very useful in such settings. The work on the shiny web application, openVA app, is a first step towards making the package more accessible. The authors intend to extend it to a better front end hosted on secure centralized servers. Second, although we aim to provide users with all the available methods for assigning causes of death, there is a lack of tools for comparing the accuracy and robustness between algorithms. Thus, much future work is needed to systematically assess, compare, and combine these methods in a better analytic framework. Finally, the development of VA algorithms is still an active area of research and it would be possible to extend the **openVA** suite to incorporate better VA algorithms and new types of data such as free-text narratives.

9 Acknowledgement

This work was supported by grants K01HD078452, R01HD086227, and R21HD095451 from the Eunice Kennedy Shriver National Institute of Child Health and Human Development (NICHD). Funding was also received from the Data for Health Initiative, a joint project of Vital Strategies, the CDC Foundation and Bloomberg Philanthropies, through Vital Strategies. The views expressed are not necessarily those of the initiative.

Bibliography

R. F. Breiman, D. M. Blau, P. Mutevedzi, V. Akelo, I. Mandomando, I. U. Ogbuanu, S. O. Sow, L. Madrid, S. El Arifeen, M. Garel, et al. Postmortem investigations and identification of multiple causes of child deaths: An analysis of findings from the Child Health and Mortality Prevention Surveillance (CHAMPS) network. *PLoS medicine*, 18(9):e1003814, 2021. [p1]

P. Byass. InterVA-4 Software [Windows Executable]. *www.interva.net*, 2015. [p2, 4]

P. Byass, D. Chandramohan, S. J. Clark, L. D'Ambruoso, E. Fottrell, W. J. Graham, A. J. Herbst, A. Hodgson, S. Hounton, K. Kahn, et al. Strengthening standardised interpretation of verbal autopsy data: The new InterVA-4 tool. *Global Health Action*, 5, 2012. [p2, 3, 4, 7]

P. Byass, L. Hussain-Alkhateeb, L. D'Ambruoso, S. Clark, J. Davies, E. Fottrell, J. Bird, C. Kabudula, S. Tollman, K. Kahn, L. Schiöler, and M. Petzold. An integrated approach to processing WHO-2016 verbal autopsy data: the interva-5 model. *BMC Medicine*, 17(1):1–12, 2019. [p2, 3]

D. Chandramohan, E. Fottrell, J. Leitao, E. Nichols, S. J. Clark, C. Alsokhn, D. C. Munoz, C. AbouZahr, A. D. Pasquale, R. Mswia, E. Choi, F. Baiden, J. Thomas, I. Lyatuu, Z. Li, P. Larbi-Debrah, Y. Chu, S. Cheburet, O. Sankoh, A. M. Badr, D. M. Fat, P. Setel, R. Jakob, and D. de Savigny. Estimating causes of death where there is no medical certification: evolution and state of the art of verbal autopsy. *Global Health Action*, 14(sup1):1982486, 2021. doi: 10.1080/16549716.2021.1982486. URL https://doi.org/10.1080/16549716.2021.1982486. PMID: 35377290. [p1]

W. Chang, J. Cheng, J. Allaire, C. Sievert, B. Schloerke, Y. Xie, J. Allen, J. McPherson, A. Dipert, and B. Borges. *shiny: Web Application Framework for R*, 2021. URL https://CRAN.R-project.org/package=shiny. R package version 1.7.1. [p16]

E. Choi, J. Thomas, and E. Karpinski. Prepare data from who and phrmc instruments for verbal autopsy algorithms. https://pypi.org/project/pycrossva/; Accessed July 21, 2021., 2021. [p16]

L. D'Ambruoso, T. Boerma, P. Byass, E. Fottrell, K. Herbst, K. Kallander, and Z. Mullan. The case for verbal autopsy in health systems strengthening. *LANCET GLOBAL HEALTH*, 5(1):E20–E21, JAN 2017. ISSN 2214-109X. [p2, 5]

A. Datta, J. Fiksel, A. Amouzou, and S. L. Zeger. Regularized bayesian transfer learning for population-level etiological distributions. *Biostatistics*, page ISSN 1465–4644, 2020. [p15]

J. Fiksel and A. Datta. *calibratedVA: Locally calibrated cause specific mortality fractions using verbal autopsy data*, 2018. URL https://github.com/jfiksel/CalibratedVA. [p15]

J. Fiksel, A. Datta, A. Amouzou, and S. Zeger. Generalized Bayesian quantification learning. *Journal of the American Statistical Association*, pages 1–19, 2021. [p15]

A. D. Flaxman, A. Vahdatpour, S. Green, S. L. James, and C. J. Murray. Random forests for verbal autopsy analysis: multisite validation study using clinical diagnostic gold standards. *Population health metrics*, 9(1):1–11, 2011. [p3]

M. Garenne. Prospects for automated diagnosis of verbal autopsies. *BMC Medicine*, 12(1):18, 2014. [p1]

Institute for Health Metrics and Evaluation. SmartVA-Analyze desktop application. https://github.com/ihmeuw/SmartVA-Analyze; Accessed July 21, 2021., 2021a. [p2, 16]

Institute for Health Metrics and Evaluation. Verbal autopsy tools. https://www.healthdata.org/data-tools-practices/verbal-autopsy; Accessed July 21, 2021., 2021b. [p2, 3]

S. L. James, A. D. Flaxman, C. J. Murray, and Consortium Population Health Metrics Research. Performance of the tariff method: validation of a simple additive algorithm for analysis of verbal autopsies. *Population Health Metrics*, 9(31), 2011. [p3, 4]

S. Jeblee, M. Gomes, P. Jha, F. Rudzicz, and G. Hirst. Automatically determining cause of death from verbal autopsy narratives. *BMC medical informatics and decision making*, 19(1):1–13, 2019. [p3]

K. Kahn, M. A. Collinson, F. X. Gómez-Olivé, O. Mokoena, R. Twine, P. Mee, S. A. Afolabi, B. D. Clark, C. W. Kabudula, A. Khosa, S. Khoza, M. G. Shabangu, B. Silaule, J. B. Tibane, R. G. Wagner, M. L. Garenne, S. J. Clark, and S. M. Tollman. Profile: Agincourt health and socio-demographic surveillance system. *International Journal of Epidemiology*, 41(4):988–1001, 2012. [p1]

T. Kunihama, Z. R. Li, S. J. Clark, and T. H. McCormick. Bayesian factor models for probabilistic cause of death assessment with verbal autopsies. *The Annals of Applied Statistics*, 14(1):241–256, 03 2020. [p15]

Z. R. Li, T. H. McCormick, and S. J. Clark. Interva4: An R package to analyze verbal autopsy data. *Center for Statistics and the Social Sciences Working Paper, No.146*, 2014. [p2]

Z. R. Li, T. H. McCormick, and S. J. Clark. **Tariff***: Replicate Tariff Method for Verbal Autopsy*, 2018. URL https://CRAN.R-project.org/package=Tariff. R package version 1.0.5. [p1, 3]

Z. R. Li, T. H. McCormick, S. J. Clark, and P. Byass. **InterVA4***: Replicate and Analyse 'InterVA4'*, 2019. URL https://CRAN.R-project.org/package=InterVA4. R package version 1.7.6. [p1, 2]

Z. R. Li, T. H. McCormick, and S. J. Clark. Using Bayesian latent Gaussian graphical models to infer symptom associations in verbal autopsies. *Bayesian Analysis*, 15(3):781, 2020. [p15]

Z. R. Li, Z. Wu, I. Chen, and S. J. Clark. Bayesian nested latent class models for cause-of-death assignment using verbal autopsies across multiple domains. *arXiv preprint arXiv:2112.12186*, 2021. [p15]

Z. R. Li, T. H. McCormick, and S. J. Clark. **InSilicoVA***: Probabilistic Verbal Autopsy Coding with 'InSilicoVA' Algorithm*, 2022a. URL https://CRAN.R-project.org/package=InSilicoVA. R package version 1.4.0. [p1, 3]

Z. R. Li, J. Thomas, T. McCormick, and S. Clark. **openVA***: Automated Method for Verbal Autopsy*, 2022b. URL https://CRAN.R-project.org/package=openVA. R package version 1.1.0. [p1]

D. Maher, S. Biraro, V. Hosegood, R. Isingo, T. Lutalo, P. Mushati, B. Ngwira, M. Nyirenda, J. Todd, and B. Zaba. Translating global health research aims into action: the example of the alpha network. *Tropical Medicine & International Health*, 15(3):321–328, 2010. [p1]

T. H. McCormick, Z. R. Li, C. Calvert, A. C. Crampin, K. Kahn, and S. J. Clark. Probabilistic cause-of-death assignment using verbal autopsies. *Journal of the American Statistical Association*, 111(515): 1036–1049, 2016. [p3, 4, 6, 11, 13]

P. Miasnikof, V. Giannakeas, M. Gomes, L. Aleksandrowicz, A. Y. Shestopaloff, D. Alam, S. Tollman, A. Samarikhalaj, and P. Jha. Naive bayes classifiers for verbal autopsies: comparison to physician-based classification for 21,000 child and adult deaths. *BMC Medicine*, 13(1):1, 2015. [p3, 4]

K. R. Moran. Factor regression for verbal autopsy. https://github.com/kelrenmor/farva; Accessed July 21, 2021., 2020. [p15]

K. R. Moran, E. L. Turner, D. Dunson, and A. H. Herring. Bayesian hierarchical factor regression models to infer cause of death from verbal autopsy data. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 2021. [p15]

C. J. Murray, A. D. Lopez, R. Black, R. Ahuja, S. M. Ali, A. Baqui, L. Dandona, E. Dantzer, V. Das, U. Dhingra, et al. Population Health Metrics Research Consortium gold standard verbal autopsy validation study: design, implementation, and development of analysis datasets. *Population Health Metrics*, 9(1):27, 2011. [p6]

C. J. Murray, R. Lozano, A. D. Flaxman, P. Serina, D. Phillips, A. Stewart, S. L. James, A. Vahdatpour, C. Atkinson, M. K. Freeman, et al. Using verbal autopsy to measure causes of death: the comparative performance of existing methods. *BMC medicine*, 12(1):1–19, 2014. [p3]

J. Nkengasong, E. Gudo, I. Macicame, X. Maunze, A. Amouzou, K. Banke, S. Dowell, and I. Jani. Improving birth and death data for African decision making. *The Lancet Global Health*, 8(1):e35–e36, 2020. [p1]

O. Sankoh and P. Byass. The INDEPTH network: filling vital gaps in global epidemiology. *International Journal of Epidemiology*, 41(3), 2012. [p1]

P. Serina, I. Riley, A. Stewart, S. L. James, A. D. Flaxman, R. Lozano, B. Hernandez, M. D. Mooney, R. Luning, R. Black, et al. Improving performance of the tariff method for assigning causes of death to verbal autopsies. *BMC Medicine*, 13(1):1, 2015. [p2, 3]

C. E. Taylor, R. Parker, W. Reinke, and R. Faruquee. *Child and maternal health services in rural India. The Narangwal experiment. 2. Integrated family planning and health care.* Johns Hopkins University Press, 1983. [p1]

J. Thomas. Software for automating the processing of verbal autopsy data. https://github.com/verbal-autopsy-software/openVA_App; Accessed July 21, 2021., 2021. [p16]

J. Thomas, E. Choi, Z. Li, N. Maire, T. McCormick, P. Byass, and S. Clark. **CrossVA***: Verbal Autopsy Data Transformation for InSilicoVA and InterVA5 Algorithms*, 2020. URL https://CRAN.R-project.org/package=CrossVA. R package version 1.0.0. [p16]

J. Thomas, S. J. Clark, and M. W. Bratschi. Software for automating the processing of verbal autopsy data. https://openva-pipeline.readthedocs.io/; Accessed July 21, 2021., 2021a. [p16]

J. Thomas, Z. Li, P. Byass, T. McCormick, M. Boyas, and S. Clark. *InterVA5: Replicate and Analyse 'InterVA5'*, 2021b. URL https://CRAN.R-project.org/package=InterVA5. R package version 1.1.3. [p1, 2]

R. Wen, P. Miasnikof, V. Giannakeas, and M. Gomes. **nbc4va***: Bayes Classifier for Verbal Autopsy Data*, 2022. URL https://CRAN.R-project.org/package=nbc4va. R package version 1.2. [p1, 3]

World Health Organization. Verbal autopsy standards: The 2012 who verbal autopsy instrument release candidate 1. Technical report, World Health Organization (WHO), Geneva, accessed 2015-01. 2012. [p1, 2]

Z. Wu, Z. R. Li, I. Chen, and M. Li. Tree-informed Bayesian multi-source domain adaptation: cross-population probabilistic cause-of-death assignment using verbal autopsy. *arXiv preprint arXiv:2112.10978*, 2021. [p15]

Zehang Richard Li
University of California, Santa Cruz
Santa Cruz CA, USA
ORCiD: 0000-0001-9551-9638
lizehang@ucsc.edu

Jason Thomas
The Ohio State University
Columbus, OH, USA
ORCiD: 0000-0002-7694-3262
thomas.3912@osu.edu

Eungang Choi
The Ohio State University
Columbus, OH, USA
ORCiD: 0000-0003-1333-8126
choi.1443@osu.edu

Tyler H. McCormick
University of Washington
Seattle, WA, USA
ORCiD: 0000-0002-6490-1129
tylermc@u.washington.edu

Samuel J Clark
The Ohio State University
Columbus, OH, USA
ORCiD: 0000-0002-4929-6231
work@samclark.net

BayesPPD: An R Package for Bayesian Sample Size Determination Using the Power and Normalized Power Prior for Generalized Linear Models

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

Abstract The R package **BayesPPD** (Bayesian Power Prior Design) supports Bayesian power and type I error calculation and model fitting after incorporating historical data with the power prior and the normalized power prior for generalized linear models (GLM). The package accommodates summary level data or subject level data with covariate information. It supports use of multiple historical datasets as well as design without historical data. Supported distributions for responses include normal, binary (Bernoulli/binomial), Poisson and exponential. The power parameter can be fixed or modeled as random using a normalized power prior for each of these distributions. In addition, the package supports the use of arbitrary sampling priors for computing Bayesian power and type I error rates. In addition to describing the statistical methodology and functions implemented in the package to enable sample size determination (SSD), we also demonstrate the use of **BayesPPD** in two comprehensive case studies.

Introduction: BayesPPD

There has been increasing interest over the past few decades in incorporating historical data in clinical trials, particularly on controls (Pocock, 1976; Neuenschwander et al., 2010; Viele et al., 2014). Use of historical data can increase effective sample size, potentially leading to more accurate point estimates and increased power (Neuenschwander et al., 2010; Viele et al., 2014). Bayesian methods provide a natural mechanism for information borrowing through the use of informative priors. Some popular informative priors for Bayesian clinical trial design include the power prior (Chen and Ibrahim, 2000), the normalized power prior (Duan et al., 2006), the commensurate power prior (Hobbs et al., 2011), and the robust meta-analytic-predictive prior (Schmidli et al., 2014).

Some advantages of the power prior include its easy construction, its natural way of incorporating historical data, its intuitive interpretation, and its desirable theoretical properties (Ibrahim et al., 2015). For example, Ibrahim et al. (2003) show that the power prior is an optimal class of informative priors in the sense that it minimizes a convex sum of the Kullback–Leibler (KL) divergences between two posterior densities, in which one density is based on no incorporation of historical data, and the other density is based on pooling the historical and current data. Duan et al. (2006) propose a modification of the power prior, the normalized power prior, which adds a normalizing constant component when the power parameter is modeled as random. The normalizing constant poses computational challenges in the presence of covariates, because it is analytically intractable except in the case of the normal linear model (Carvalho and Ibrahim, 2021). We address this challenge by utilizing the PWK estimator (Wang et al., 2018) to approximate the normalizing constant for use with generalized linear models. We also develop a novel way of incorporating the approximation of the normalizing constant into the Markov chain Monte Carlo (MCMC) algorithm.

There is a growing literature on Bayesian sample size determination, including the works of Rahme and Joseph (1998), Simon (1999), Wang and Gelfand (2002), De Santis (2007), M'Lan et al. (2006) and Joseph et al. (2008). We consider the simulation-based method developed in Chen et al. (2011) and Psioda and Ibrahim (2019), which extends the the fitting and sampling priors of Wang and Gelfand (2002) with a focus on controlling the type I error rate and calculating power. In addition, our package supports the use of arbitrary sampling priors for computing Bayesian power and type I error rates, and has specific features for GLMs that semi-automatically generate sampling priors from historical data.

The R package **BayesPPD** (Bayesian Power Prior Design) (Shen et al., 2022) supports Bayesian clinical trial design after incorporating historical data with the power prior and the normalized power prior. **BayesPPD** has two categories of functions: functions for model fitting and functions for Bayesian power and type I error rate estimation. The package accommodates summary level data or subject level data with covariate information for normal, binary (Bernoulli/binomial), Poisson and exponential models. It supports use of multiple historical datasets and design without historical data.

Several Bayesian clinical trial design packages are available on the Comprehensive R Archive Network (CRAN), such as **BDP2**, **ph2bayes** and **gsbDesign** (Kopp-Schneider et al., 2018; Nagashima, 2018; Gerber and Gsponer, 2016). However, these packages do not accommodate the incorporation of historical data and are limited to normal and binary endpoints. The **RBesT** package (Weber, 2021) accounts for historical data using the meta-analytic-predictive prior. Commercial software for clinical trial design such as **FACTS**, **East** and **ADDPLAN** (LLC Consultants, 2014; Cytel Software Corporation, 2014; Wassmer and Eisebitt, 2005) do not implement the power prior, to our knowledge. The **BayesCTDesign** (Eggleston et al., 2019) package supports two-arm randomized Bayesian trial design using historical control data with the power prior, but it does not allow covariates, nor does it allow the power parameter to be treated as random. The **NPP** (Han et al., 2021) package implements the normalized power prior for two group cases for Bernoulli, normal, multinomial and Poisson models, as well as for the normal linear model. It does not support generalized linear models, nor does it include functions for sample size determination. The **bayesDP** (Balcome et al., 2021) package implements the power prior where the power parameter is determined by a discounting function estimated based on a measure of prior-data conflict. Thus, this approach is not fully Bayesian, and the package must be used in conjunction with the package **bayesCT** (Chandereng et al., 2020) for trial design. While **bayesDP** supports two-arm trials for binomial, normal and survival models as well as linear and logistic regression models, **BayesPPD** allows covariates for Bernoulli/binomial, normal, Poisson and exponential models with several choices of link functions. The **BayesPPD** package is a comprehensive resource that supports Bayesian analysis and design using the power prior and normalized power prior.

Another advantage of **BayesPPD** is its computational speed. **BayesPPD** implements MCMC algorithms with **Rcpp** (Eddelbuettel and Francois, 2011) without recourse to asymptotics. For most sample sizes, functions for analysis take only a few seconds to run. Functions for design for two group cases run in seconds for fixed a_0, and generally run in less an hour for random a_0, depending on the desired level of precision (e.g., number of simulated datasets). In the presence of covariates, functions for design are more computation-intensive; an approximation method based on asymptotic theory has been implemented to help users obtain a rough estimate of the desired sample size before fine-tuning using the MCMC-based method.

This article is organized as follows. We first describe the methods implemented by the package. We then provide details on how to use **BayesPPD** for different data scenarios and model needs. We also present two case studies with example code, one with covariates and one without. The article is concluded with a brief discussion.

Theoretical framework

Basic formulation of the power prior

Let D denote data from the current study and D_0 denote data from a historical study. Let θ denote model parameters and $L(\theta|D)$ denote a general likelihood function associated with a given outcome model, such as a linear model, generalized linear model (GLM), survival model, or random effects model. Following Chen and Ibrahim (2000), the power prior is formulated as

$$\pi(\theta|D_0, a_0) \propto L(\theta|D_0)^{a_0}\pi_0(\theta).$$

where $0 \leq a_0 \leq 1$ is a discounting parameter for the historical data likelihood, and $\pi_0(\theta)$ is the initial prior for θ. The parameter a_0 allows researchers to control the influence of the historical data on the posterior distribution. When $a_0 = 0$, historical information is discarded and the power prior becomes equivalent to the initial prior $\pi_0(\theta)$. When $a_0 = 1$, the power prior corresponds to the posterior distribution of θ given the historical data and the initial prior. When a_0 is treated as fixed, sensitivity analysis can be performed to determine an appropriate a_0 value. When a_0 is treated as random, priors such as the beta distribution can be specified. The choice of a_0 is discussed in, for example, Ibrahim et al. (2015) and Psioda and Ibrahim (2018).

The power prior can easily accommodate multiple historical datasets. Suppose there are K historical datasets denoted by D_{0k} for $k = 1, \cdots, K$ and let $D_0 = (D_{01}, \cdots, D_{0K})$. The power prior becomes

$$\pi(\theta|D_0, a_0) \propto \prod_{k=1}^{K} L(\theta|D_{0k})^{a_{0k}}\pi_0(\theta),$$

where $a_0 = (a_{01}, \cdots, a_{0K})'$ and $0 \leq a_{0k} \leq 1$ for $k = 1, \cdots, K$.

The normalized power prior

Modeling a_0 as random allows one to represent uncertainty in how much the historical data should be discounted. The simplest power prior that allows this is the *joint power prior* (Chen and Ibrahim, 2000) which is given by

$$\pi(\theta, a_0|D_0) \propto L(\theta|D_0)^{a_0}\pi_0(\theta)\pi_0(a_0).$$

Neuenschwander et al. (2009) point out that this formulation is not ideal because the normalizing constant,

$$c(a_0) = \int L(\theta|D_0)^{a_0}\pi_0(\theta)d\theta,$$

for $L(\theta|D_0)^{a_0}\pi_0(\theta)$ is not incorporated and thus $\pi_0(a_0)$ is not actually the marginal prior for a_0. In fact, Duan et al. (2006) point out that this formulation of the power prior does not obey the likelihood principle. Duan et al. (2006) proposed a modification of the power prior, the *normalized power prior*, which is given by

$$\pi(\theta, a_0|D_0) = \pi(\theta|D_0, a_0)\pi(a_0) = \frac{L(\theta|D_0)^{a_0}\pi_0(\theta)}{c(a_0)}\pi_0(a_0),$$

where $\pi_0(a_0)$ is the initial prior for a_0. The normalized power prior specifies a conditional prior for θ given a_0 and a marginal prior for a_0. The normalizing constant,

$$c(a_0) = \int L(\theta|D_0)^{a_0}\pi_0(\theta)d\theta,$$

is often analytically intractable and requires Monte Carlo methods for estimation. When a_0 is modeled as random, the normalized power prior is implemented in **BayesPPD** using a beta initial prior on a_0, for which the user must specify values of the two shape parameters that define the beta density. The package supports the inclusion of multiple historical datasets when a_0 is modeled as random.

The power prior for generalized linear models

The power prior can easily accommodate covariates. Let y_i denote the response variable and x_i denote a p-dimensional vector of covariates for subject $i = 1, \cdots, n$. Denote $\tilde{\beta} = (\beta_0, \beta)$, where β_0 is the intercept and $\beta = (\beta_1, \cdots, \beta_p)'$ is a p-dimensional vector of regression coefficients. We assume the GLM of $y_i|x_i$ is given by

$$f(y_i|x_i, \tilde{\beta}, \tau) = \exp\{\alpha_i^{-1}(\tau)(y_i g(\beta_0 + x_i'\beta) - \psi(g(\beta_0 + x_i'\beta))) + \phi(y_i, \tau)\},$$

where τ is a scale parameter and g is a monotone differentiable link function. In particular, **BayesPPD** allows the distribution of $y_i|x_i$ to be normal, Bernoulli, binomial, Poisson or exponential. Note that for Bernoulli, binomial, Poisson and exponential regression models, τ is equal to 1.

Let $D_{0k} = \{(y_{0ki}, x_{0ki}), i = 1, \cdots, n_{0k}\}$ denote the k-th historical dataset, where y_{0ki} is the response variable for historical subject i and x_{0ki} is the p-dimensional vector of covariates for historical subject i. By default, **BayesPPD** assumes the historical data consists of control group subjects only. Therefore, the historical covariate matrix does not have the treatment indicator variable, while the current covariate matrix does. The package also allows the historical data to be used to inform the treatment effect parameter; then the historical and current covariate matrices will both have the treatment indicator.

The GLM for $y_{0ki}|x_{0ki}$ is

$$f(y_{0ki}|x_{0ki}, \tilde{\beta}, \tau_{0k}) = \exp\{\alpha_{0i}^{-1}(\tau_{0k})(y_{0ki} g(\beta_0 + x_{0ki}'\beta) - \psi(g(\beta_0 + x_{0ki}'\beta))) + \phi(y_{0ki}, \tau_{0k})\},$$

where τ_{0k} is the scale parameter for the k-th historical dataset. Note that the precision parameter is assumed to be unshared. The historical data likelihood for K historical datasets is $L(\tilde{\beta}, \tau_{01}, \cdots, \tau_{0K}|D_0) \propto \prod_{k=1}^{K}\prod_{i=1}^{n_{0k}} f(y_{0ki}|x_{0ki}, \tilde{\beta}, \tau_{0k})$. The power prior for GLMs with fixed $a_0 = (a_{01}, \cdots, a_{0K})'$ is

$$\pi(\tilde{\beta}, \tau_{01}, \cdots, \tau_{0K}|D_0, a_0) \propto \prod_{k=1}^{K}\left\{L(\tilde{\beta}, \tau_{0k}|D_{0k})^{a_{0k}}\pi_0(\tau_{0k})\right\}\pi_0(\tilde{\beta}).$$

When a_0 is modeled as random, we assume $\tau_{01}, \cdots, \tau_{0K} = \tau$ for computational simplicity. The

normalized power prior for GLMs with a random a_0 vector is given by

$$\pi(\tilde{\beta}, \tau, a_0|D_0) = \frac{\prod_{k=1}^{K} L(\tilde{\beta}, \tau|D_{0k})^{a_{0k}} \pi_0(\tilde{\beta})\pi_0(\tau)}{\int_0^\infty \int_{\mathbb{R}^p} \prod_{k=1}^{K} L(\tilde{\beta}, \tau|D_{0k})^{a_{0k}} \pi_0(\tilde{\beta})\pi_0(\tau) d\tilde{\beta} d\tau} \pi_0(a_0).$$

Estimating the normalizing constant for GLMs

The normalizing constant $c(a_0)$ in the normalized power prior for GLMs is analytically intractable except for normal linear regression models. For other types of regression models, we approximate the normalizing constant with the partition weighted kernel (PWK) estimator proposed by Wang et al. (2018). The PWK estimator requires MCMC samples from the posterior distribution (based on a discounted historical data likelihood with fixed a_0 value), which we obtain using the slice sampler (Neal, 2003), and the known kernel function for computing the normalizing constant. The authors first impose a working parameter space, defined as the space where the kernel value is bounded away from zero. As stated in Wang et al. (2018), the PWK estimator is constructed by first partitioning the working parameter space and then estimating the marginal likelihood by a weighted average of the kernel values evaluated at a MCMC sample for each partition, where the weights are assigned locally using a representative kernel value in each partitioned subset. The PWK estimator has been shown to have desirable properties, including being consistent and having finite variance (Wang et al., 2018).

The function `normalizing.constant` in our package computes a vector of coefficients that defines a function $f(a_0)$ that approximates the normalizing constant for GLMs with random a_0. Suppose there are K historical datasets. Basic usage of the `normalizing.constant` function entails the following steps:

1. The user inputs a grid of M rows and K columns of potential values for a_0.
2. For each row of a_0 values in the grid, the function obtains M samples for β from the power prior associated with the current values of a_0 using the slice sampler. Note that τ is not applicable here because the models implemented using the PWK estimator do not have scale parameters.
3. For each of the M sets of posterior samples, the PWK algorithm (Wang et al., 2018) is used to estimate the log of the normalizing constant $d_1, \cdots, d_M$ for the normalized power prior.
4. At this point, one has a dataset with outcomes $d_1, \cdots, d_M$ and predictors corresponding to the rows of the a_0 grid matrix. A polynomial regression is employed to estimate a function $d = f(a_0)$ based on these quantities. The degree of the polynomial regression is determined by the algorithm to ensure $R^2 > 0.99$.
5. The `normalizing.constant` function returns the vector of coefficients from the polynomial regression model, which the user must input into the analysis or design function for GLMs with a_0 modeled as random (`glm.random.a0` and `power.glm.random.a0`).

In the Examples section below, we demonstrate computing the normalizing constant for one historical dataset with three covariates. Due to computational intensity, the `normalizing.constant` function has not been evaluated for accuracy for high dimensional β (e.g., dimension > 10) or high dimensional a_0 (e.g., dimension > 5).

Sample size determination

Hypotheses for two group models

Following Chen et al. (2011), for two group models (i.e., treatment and control group with no covariates), denote the parameter for the treatment group by μ_t and the parameter for the control group by μ_c. For example, for binomial models, μ_t and μ_c are the probability of having some outcome (e.g., tumor response) for the treatment and control group, respectively. Let τ_c denote the nuisance parameters for the control group in the model. For normal models, τ_c is a vector of precision parameters. For K historical datasets $D_0 = (D_{01}, \cdots, D_{0K})'$ with fixed a_0, we assume each historical dataset D_{0k} has a precision parameter τ_{c0k}. When a_0 is modeled as random, the historical and current datasets are assumed to have the same precision parameter, in which case τ_c reduces to a scalar. The precision parameter of the treatment group is denoted by τ_t.

We consider the following power prior for (μ_c, τ_c) given multiple historical datasets D_0

$$\pi(\mu_c, \tau_c|D_0, a_0) \propto \prod_{k=1}^{K} \left[L(\mu_c|D_{0k}, \tau_c)^{a_{0k}}\right] \pi_0(\mu_c)\pi_0(\tau_c),$$

where $a_0 = (a_{01}, \cdots, a_{0K})'$, $0 \leq a_{0k} \leq 1$ for $k = 1, \cdots, K$, $L(\mu_c|D_{0k}, \tau_c)$ is the historical data likelihood, and $\pi_0(\mu_c)$ and $\pi_0(\tau_c)$ are the initial priors. To model a_0 as random, we consider the normalized power prior

$$\pi(\mu_c, \tau_c, a_0|D_0) \propto \frac{\prod_{k=1}^{K} [L(\mu_c|D_{0k}, \tau_c)^{a_{0k}}]\, \pi_0(\mu_c)\pi_0(\tau_c)}{c(a_0)} \pi_0(a_0),$$

where

$$c(a_0) = \int_0^\infty \int_{-\infty}^\infty \prod_{k=1}^{K} [L(\mu_c|D_{0k}, \tau_c)^{a_{0k}}] \pi_0(\mu_c)\pi_0(\tau_c) d\mu_c d\tau_c.$$

For models other than the exponential model, the power / type I error calculation algorithm assumes the null and alternative hypotheses are given by

$$H_0 : \mu_t - \mu_c \geq \delta$$

and

$$H_1 : \mu_t - \mu_c < \delta,$$

where δ is a prespecified constant. To test hypotheses of the opposite direction, i.e., $H_0 : \mu_t - \mu_c \leq \delta$ and $H_1 : \mu_t - \mu_c > \delta$, one can set the parameter `nullspace.ineq` to "<".

For positive continuous data assumed to follow exponential distribution, the hypotheses are given by

$$H_0 : \mu_t / \mu_c \geq \delta$$

and

$$H_1 : \mu_t / \mu_c < \delta,$$

where μ_t and μ_c are the hazards for the treatment and the control group, respectively.

Definition of Bayesian type I error rate and power

Let Θ_0 and Θ_1 denote the parameter spaces corresponding to H_0 and H_1. Let $y^{(n)}$ denote the simulated current data associated with a sample size of n and let $\theta = (\mu_t, \mu_c, \tau_c)$ denote the model parameters. Let $\pi^{(s)}(\theta)$ denote the sampling prior and let $\pi^{(f)}(\theta)$ denote the fitting prior. The sampling prior is used to generate the hypothetical data while the fitting prior is used to fit the model after the data is generated. Let $\pi_0^{(s)}(\theta)$ denote a sampling prior that only puts mass in the null region, i.e., $\theta \subset \Theta_0$. Let $\pi_1^{(s)}(\theta)$ denote a sampling prior that only puts mass in the alternative region, i.e., $\theta \subset \Theta_1$. To determine Bayesian sample size, we estimate the quantity

$$\beta_{sj}^{(n)} = E_s[I\{P(\mu_t - \mu_c < \delta|y^{(n)}, \pi^{(f)}) \geq \gamma\}], \tag{1}$$

where $j = 0$ or 1, corresponding to the expectation taken with respect to $\pi_0^{(s)}(\theta)$ or $\pi_1^{(s)}(\theta)$. The constant $\gamma > 0$ is a prespecified posterior probability threshold for rejecting the null hypothesis (e.g., 0.975). The probability is computed with respect to the posterior distribution given the simulated data $y^{(n)}$ and the fitting prior $\pi^{(f)}(\theta)$, and the expectation is taken with respect to the marginal distribution of $y^{(n)}$ defined based on the sampling prior $\pi^{(s)}(\theta)$. Then $\beta_{s0}^{(n)}$ corresponding to $\pi^{(s)}(\theta) = \pi_0^{(s)}(\theta)$ is the Bayesian type I error rate, while $\beta_{s1}^{(n)}$ corresponding to $\pi^{(s)}(\theta) = \pi_1^{(s)}(\theta)$ is the Bayesian power. Note that Bayesian type I error rate and power can be equivalently defined as weighted averages of the quantities based on fixed values of θ with weights determined by the sampling priors (Psioda and Ibrahim, 2018). For given $\alpha_0 > 0$ and $\alpha_1 > 0$, we can compute $n_{\alpha_0} = \min\{n : \beta_{s0}^{(n)} \leq \alpha_0\}$ and $n_{\alpha_1} = \min\{n : \beta_{s1}^{(n)} \geq 1 - \alpha_1\}$. Then, the sample size is taken to be $\max\{n_{\alpha_0}, n_{\alpha_1}\}$. Common choices of α_0 and α_1 include $\alpha_0 = 0.05$ and $\alpha_1 = 0.2$. These choices guarantee that the Bayesian type I error rate is at most 0.05 and the Bayesian power is at least 0.8.

Estimation of Bayesian type I error rate and power

In this section, we discuss the simulation-based procedure used to estimate the Bayesian type I error rate and power. Let N denote the number of simulated trials. To compute $\beta_{sj}^{(n)}$, the following algorithm is used for each simulated trial b:

- Step 1: Generate $\theta^{(b)} \sim \pi_j^{(s)}(\theta)$.
- Step 2: Generate $y^{(b)} \sim f(y^{(b)}|\theta^{(b)})$.

- Step 3: Estimate the posterior distribution $\pi(\theta|y^{(b)}, D_0, a_0)$ and the posterior probability $P(\mu_t - \mu_c < \delta|y^{(b)}, \pi^{(f)}, D_0, a_0)$.
- Step 4: Compute the indicator $r^{(b)} = I\{P(\mu_t - \mu_c < \delta|y^{(b)}, \pi^{(f)}, D_0, a_0) \geq \gamma\}$.

Then the estimate of $\beta_{sj}^{(n)}$ is $\frac{1}{N}\sum_{b=1}^{N} r^{(b)}$.

Specification for regression models

For regression models, we assume the first column of the covariate matrix is the treatment indicator, and the corresponding parameter is β_1, which, for example, corresponds to a difference in means for the linear regression model and a log hazard ratio for the exponential regression model. The hypotheses are given by

$$H_0 : \beta_1 \geq \delta$$

and

$$H_1 : \beta_1 < \delta.$$

The definition of $\beta_{sj}^{(n)}$ and the algorithm change accordingly.

Prior distributions

Two group cases

For two group models, continuous responses of the control group are assumed to follow $N(\mu_c, \tau_c^{-1})$. Each historical dataset D_{0k} is assumed to have a different precision parameter τ_{c0k}. The initial prior for the μ_c is the uniform improper prior. The initial prior for τ_c is the Jeffery's prior, τ_c^{-1}, and the initial prior for τ_{c0k} is τ_{c0k}^{-1}. Posterior samples of μ_c, τ_c and τ_{c0k}'s (if historical data is given) are obtained through Gibbs sampling. When a_0 is modeled as random, the historical datasets are assumed to have the same precision parameter τ_c as the current dataset for computational simplicity. The initial prior for τ_c is the Jeffery's prior, τ_c^{-1}. Posterior samples of a_0 are obtained through slice sampling.

For binary, count or positive continuous data, a single response from the control group is assumed to follow Bernoulli(μ_c), Poisson(μ_c) or exponential(rate=μ_c), respectively. A beta initial prior is used for μ_c for Bernoulli data, and a gamma prior is used for Poisson and exponential data. The user can specify the hyperparameters. When a_0 is modeled as random, posterior samples of a_0 are obtained through slice sampling. The conditional posterior distributions of μ_c given a_0 have closed form solutions.

When computing the power or the type I error rate, treatment group data are simulated and posterior samples of μ_t (and τ_t for normal data) are obtained using basic Bayesian models. The priors used for μ_t are the same as the initial priors used for μ_c. For normal data, the prior for τ_t is the Jeffery's prior, τ_t^{-1}.

GLM cases

For GLMs, a continuous response y_i is assumed to follow $N(\beta_0 + x_i'\beta, \tau^{-1})$. Each historical dataset D_{0k} is assumed to have a different precision parameter τ_k. The initial prior for τ is the Jeffery's prior, τ^{-1}, and the initial prior for τ_k is τ_k^{-1}. Posterior samples of β_0 and β are obtained through Gibbs sampling. For all other types of data, a link function must be applied. Posterior samples of β_0 and β are obtained through slice sampling. When a_0 is fixed, the initial prior for β_0 and β is the uniform improper prior. When a_0 is modeled as random, the historical datasets are assumed to have the same precision parameter τ as the current dataset. The initial prior for τ is the Jeffery's prior, τ^{-1}. Independent normal priors with mean zero and a user-specified variance are used for β. Here we use a proper initial prior for β to ensure the propriety of the normalized power prior. Posterior samples of a_0 are obtained through slice sampling. The normalizing constant of the normalized power prior is estimated using the PWK estimator.

	Two groups, fixed a_0	Two groups, random a_0	GLM, fixed a_0*	GLM, random a_0
Bernoulli/ Binomial	Numerical integration	Slice	Slice	Slice & PWK
Normal	Gibbs	Gibbs & Slice	Gibbs	Gibbs & Slice
Poisson	Numerical integration	Slice	Slice	Slice & PWK
Exponential	Numerical integration	Slice	Slice	Slice & PWK

Table 1: Estimation method used for each model and data type. Each column contains a type of model, and each row contains an outcome distribution. Gibbs sampling is used for normally distributed outcomes. Slice sampling is used when a_0 is modeled as random.
*Approximation method is available for sample size determination for fast implementation.

Using BayesPPD

Package overview

The **BayesPPD** package accommodates summary level data or subject level data with covariate information. It supports SSD for design applications with multiple historical datasets as well as with no historical data. Functions with names containing `"two.grp"` assume that the input data are sufficient statistics (e.g., sample mean) for independent and identically distributed treatment and control group data. Simulated control group data are analyzed using the power or normalized power prior and posterior samples of μ_c are returned. By default, functions with names containing `"glm"` assume that the historical control data include a covariate matrix X_0 and the current data include the same set of covariates with an additional column (the first column) of treatment indicator. The package assumes the historical data is composed of control group subjects only by default. If the user wants to use the historical data to inform the treatment effect, one can set `borrow.treat=TRUE` and include the treatment indicator in the historical covariate matrix. Simulated data are analyzed using the power or normalized power prior and posterior samples of the regression coefficients are returned. For each of two cases, the power parameter a_0 can be fixed or modeled as random, resulting in four model fitting functions, `two.grp.fixed.a0`, `two.grp,random.a0`, `glm.fixed.a0` and `glm.random.a0`. For each of the four model fitting functions, a companion function prefixed with `"power"` calculates power or type I error rate, given historical data and current data sample size. Supported distributions of responses include normal, binary (Bernoulli/binomial), Poisson and exponential. Since functions for sample size determination for GLMs are computationally intensive, an approximation method based on asymptotic theory has been implemented for the model with fixed a_0.

Table 1 shows the sampling methods used for each model and data distribution. Gibbs sampling is used for normally distributed data. Slice sampling (Neal, 2003) is used for all other data distributions, and for obtaining posterior samples of a_0 when a_0 is considered random. For two group models with fixed a_0, numerical integration is performed using the **RcppNumerical** package (Qiu et al., 2019). For GLMs with random a_0, the PWK estimator (Wang et al., 2018) is used to estimate the normalizing constant. The functions return S3 objects with `summary` methods implemented.

Two group cases

If one has current and/or historical control data for an application with no covariates and would like to obtain posterior samples of μ_c (and τ_c for normal data), one uses the function `two.grp.fixed.a0` or `two.grp.random.a0`. The user must specify the `data.type` (`"Normal"`, `"Bernoulli"`, `"Poisson"` or `"Exponential"`), the sum of responses `y.c`, the sample size `n.c` and the sample variance `v.c` (for normal data only) of the current control data. The optional `historical` argument is a matrix where the columns contain the sufficient statistics and each row represents a historical dataset. For `two.grp.fixed.a0`, `historical` must contain a column of a_0 values, one a_0 value for each historical dataset. For non-normal data, the user can specify `prior.mu.c.shape1` and `prior.mu.c.shape2`, the hyperparameters of the initial prior for μ_c.

When $a_0 = (a_{01}, \cdots, a_{0K})'$ is modeled as random, a beta prior is specified for a_0 with hyperparameters `prior.a0.shape1` and `prior.a0.shape2`. Posterior samples of a_0 are obtained through

slice sampling. The optional tuning parameters for the slice sampler include `lower.limits` and `upper.limits` which control the upper and lower limits of the parameters being sampled, as well as `slice.widths` which controls the width of each slice. The length of `lower.limits`, `upper.limits` and `slice.widths` should be at least equal to the number of parameters, i.e., the dimension of a_0. Their default values are 0, 1 and 0.1, respectively, for each a_{0k}.

For sample size determination, `power.two.grp.fixed.a0` and `power.two.grp.random.a0` compute the power or the type I error rate given the sample sizes of the treatment and control groups for the new study and other inputs. If a sampling prior with support in the null space is used, the value returned is a Bayesian type I error rate. If a sampling prior with support in the alternative space is used, the value returned is a Bayesian power. The arguments `samp.prior.mu.t` and `samp.prior.mu.c` contain vectors of samples for μ_t and μ_c, which are discrete approximations of the sampling priors. For normal data, arguments `samp.prior.var.t` and `samp.prior.var.c`, which contain samples for τ_t^{-1} and τ_c^{-1}, must also be provided. The argument `delta` specifies the constant that defines the boundary of the null hypothesis. The default value is zero. The argument `gamma` specifies the posterior probability threshold for rejecting the null hypothesis. The default value is 0.95.

GLM cases

If one has current and historical data for an application with covariates and would like to obtain posterior samples of β (and τ for normal data), one uses the function `glm.fixed.a0` or `glm.random.a0`. It is recommended that the covariates be transformed or standardized so that the estimation of β will be stable. The user must specify the `data.type`, the `data.link` (except for normal data), the vector of responses `y` and the matrix of covariates `x` where the first column should be the treatment indicator. Supported link functions include logit, probit, log, identity-positive, identity-probability and complementary log-log. If the data is binary and all covariates are discrete, the user can collapse the Bernoulli data into a binomial structure, which may result in a much faster slice sampler. In this case, the user needs to provide `n`, a vector of integers specifying the number of subjects who have a particular value of the covariate vector. The optional `historical` argument is a list of lists where each list contains information about a historical dataset with named elements `y0`, `x0` and `a0` (only for `glm.fixed.a0`). If `borrow.treat=FALSE` (the default), the historical covariate matrix `x0` should not have the treatment indicator. Apart from missing the treatment indicator, `x0` should have the same set of covariates in the same order as `x`. If `borrow.treat=TRUE`, `x0` should have the same set of covariates in the same order as `x`, where the first column of `x0` must be the treatment indicator. For non-normal data, slice sampling is used to obtain posterior samples of β, and the user can specify the `lower.limits`, `upper.limits` and `slice.widths` of the sampler. The length of `lower.limits`, `upper.limits` and `slice.widths` should be at least equal to the number of parameters, i.e., the dimension of β. A matrix of posterior samples of β is returned, where the first column contains posterior samples of the intercept and the second column contains posterior samples of β_1, the parameter for the treatment indicator.

When a_0 is modeled as random for non-normal data, the user must first use the function `normalizing.constant` to obtain the value of `a0.coefficients`, a vector of coefficients for a_0 necessary for estimating the normalizing constant for the normalized power prior. For the `grid` argument of `normalizing.constant`, the user inputs a grid of M rows and K columns of potential values for a_0 for K historical datasets. For example, one can choose the vector `v = c(0.1,0.25,0.5,0.75,1)` and use `expand.grid(a0_1=v,a0_2=v,a0_3=v)` when $K = 3$ to get a grid with $M = 5^3 = 125$ rows and three columns. If there are more than three historical datasets, the dimension of v can be reduced to limit the size of the grid. A large grid will increase runtime. If some of the coefficients are not estimable in the polynomial regression, the algorithm will product the error message, "some coefficients not defined because of singularities." To resolve the issue, the user can try increasing or decreasing the number of rows in the grid. Other possible causes include insufficient sample size of the historical data, insufficient number of iterations for the slice sampler, and near-zero grid values.

When a_0 is modeled as random, slice sampling is used for a_0 only for normal data, and the length of `lower.limits`, `upper.limits` and `slice.widths` should be equal to the dimension of a_0. For all other data types, slice sampling is used for β and a_0, and the length of those vectors should be equal to the dimension of β plus the dimension of a_0.

For sample size determination, `power.glm.fixed.a0` and `power.glm.random.a0` compute the power or the type I error given the total sample size (`data.size`) for the new study and other inputs. If historical datasets are provided, the algorithm samples with replacement from the historical covariates to construct the simulated datasets. Otherwise, the algorithm samples with replacement from `x.samples`. One of the arguments `historical` and `x.samples` must be provided. The argument `samp.prior.beta` contains a matrix of samples for β, which is a discrete approximation

of the sampling prior. For normal data, the argument `samp.prior.var` containing samples for τ^{-1} must also be provided. The average posterior means of the parameters are also returned.

Sampling priors

Our implementation in **BayesPPD** does not assume any particular distribution for the sampling priors. The user specifies discrete approximations of the sampling priors by providing a vector or a matrix of sample values and the algorithm samples with replacement from the vector or the matrix as the first step of data generation. For two group cases, the user simply specifies `samp.prior.mu.t` and `samp.prior.mu.c` which are vectors of samples for μ_t and μ_c. For normal data, arguments `samp.prior.var.t` and `samp.prior.var.c`, which contain samples for τ_t^{-1} and τ_c^{-1}, must also be provided. The second application example below demonstrates the use of point mass sampling priors for binary data.

For GLM cases, the user specifies `samp.prior.beta`, a matrix of samples for β. For normal data, the argument `samp.prior.var` containing samples for τ^{-1} must also be provided. For example, suppose one wants to compute the power for the hypotheses

$$H_0 : \beta_1 \geq 0$$

and

$$H_1 : \beta_1 < 0.$$

To approximate the sampling prior for β_1, one can simply sample from a truncated normal distribution with negative mean, so that the mass of the prior falls in the alternative space. Conversely, to compute the type I error rate, one can sample from a truncated normal distribution with positive mean, so that the mass of the prior falls in the null space. Next, to generate the sampling prior for the other parameters $(\beta_0, \beta_2, \cdots, \beta_p)$, one can use the posterior samples given the historical data as the discrete approximation to the sampling prior. The function `glm.fixed.a0` generates such posterior samples if the `current` argument is set to `FALSE` and $a_{0k} = 1$ for $k = 1, \cdots, K$. The second application example in this article illustrates this method for binary data with covariates. Psioda and Ibrahim (2018) discusses sampling prior elicitation in detail.

Approximation for GLMs

Because running `power.glm.fixed.a0` and `power.glm.random.a0` is potentially time-consuming, an approximation method based on asymptotic theory (Ibrahim et al., 2015) has been implemented for the model with fixed a_0. In order to attain the exact sample size needed for the desired power, the user can start with the approximation to get a rough estimate of the sample size required, using `power.glm.fixed.a0` with `approximate=TRUE`. The second application example below illustrates the use of the approximation method. For normal data, the closed form of the distribution of the MLE of β is derived and used to compute power. For other types of data, the Newton-Raphson algorithm is used. Only canonical links are allowed.

Examples

Design of a non-inferiority trial for medical devices

We first consider the non-inferiority design application of Chen et al. (2011) considering a model for binary outcomes for treatment and control groups with no covariates. The goal of that application was to design a trial to evaluate a new generation of drug-eluting stent (DES) ("test device") with the first generation of DES ("control device"). The primary endpoint is the 12-month Target Lesion Failure (TLF), defined as any of ischemia-driven revascularization of the target lesion (TLR), myocardial infarction (MI) (Q-wave and non-Q-wave) related to the target vessel, or (cardiac) death related to the target vessel. Historical information can be borrowed from two previously conducted trials involving the first generation of DES. Table 2 summarizes the historical data.

We will illustrate Bayesian SSD incorporating historical data using the power prior with fixed a_0 and the normalized power for a_0 modeled as random. Let $\mathbf{y}_t^{(n_t)} = (y_{t1}, \cdots, y_{tn_t})$ and $\mathbf{y}_c^{(n_c)} = (y_{c1}, \cdots, y_{cn_c})$ denote the responses from the current trial for the test device and the control device, respectively. The total sample size is $n = n_t + n_c$. We assume the i-th observation from the test group y_{ti} follows Bern(μ_t), and the i-th observation from the control group y_{ci} follows Bern(μ_c). Note that the notation used in our package is different from the notation used in Chen et al. (2011),

	12-Month TLF
	% TLF (# of failure/n_{0k})
Historical Trial 1	8.2% (44/535)
Historical Trial 2	10.9% (33/304)

Table 2: Summary of two historical trials involving the first generation of DES. The primary endpoint is the 12-month Target Lesion Failure (TLF). The pooled proportion for the two historical control datasets is 9.2%.

which assumes y_{ti} follows Bern(p_t) and $\mu_t = \log\left(\frac{p_t}{1-p_t}\right)$. The hypotheses for non-inferiority testing are

$$H_0 : \mu_t - \mu_c \geq \delta$$

and

$$H_1 : \mu_t - \mu_c < \delta,$$

where δ is a prespecified non-inferiority margin. We set $\delta = 4.1\%$. We choose beta($10^{-4}, 10^{-4}$) for the initial prior for μ_c, which performs similarly to the uniform improper initial prior for $\log\left(\frac{\mu_c}{1-\mu_c}\right)$ used in Chen et al. (2011) in terms of operating characteristics. We compute the Bayesian power and type I error defined in (1). Power is computed under the assumption that $\mu_t = \mu_c$ and type I error rate is computed under the assumption that $\mu_t = \mu_c + \delta$. For sampling priors, a point mass prior at $\mu_c = 9.2\%$ is used for $\pi^{(s)}(\mu_c)$ where 9.2% is the pooled proportion for the two historical control datasets, and a point mass prior at $\mu_t = \mu_c$ is used for $\pi^{(s)}(\mu_t)$. For all computations, we use $\frac{n_t}{n_c} = 3$, $N = 10,000$, and $\gamma = 0.95$, where N is the number of simulated trials and γ is a prespecified posterior probability threshold for rejecting the null hypothesis. For this example, we consider $n_t = 750$ and $a_{01} = a_{02} = 0.3$. Power can be calculated with following code in **BayesPPD**. The `historical` matrix is defined where each row represents a historical dataset, and the three columns represent the sum of responses, sample size and a_0, respectively, of the historical control data. Since point mass sampling priors are used for μ_t and μ_c, `samp.prior.mu.t` and `samp.prior.mu.c` are both scalars. For Bernoulli outcomes, beta initial priors are used for μ_t and μ_c, with hyperparameters specified by `prior.mu.t.shape1`, `prior.mu.t.shape2`, `prior.mu.c.shape1` and `prior.mu.c.shape2`.

```
historical <- matrix(0, ncol=3, nrow=2)
historical[1,] <- c(44, 535, 0.3)
historical[2,] <- c(33, 304, 0.3)

set.seed(1)
power <- power.two.grp.fixed.a0(data.type="Bernoulli",
      n.t=750, n.c=round(750/3), historical=historical,
      samp.prior.mu.t=0.092, samp.prior.mu.c=0.092,
      prior.mu.t.shape1=0.0001, prior.mu.t.shape2=0.0001,
      prior.mu.c.shape1=0.0001,prior.mu.c.shape2=0.0001,
      delta=0.041, N=10000)
power$power/type I error
[1] 0.8428
```

When a_0 is random, the normalized power prior is used and the priors for a_{01} and a_{02} are beta(1,1), as in Chen et al. (2011). We use the default settings for the upper limits, lower limits and slice widths for a_{01} and a_{02}. We run 20,000 iterations of the slice sampler. The same initial priors and sampling priors are used as in the fixed a_0 case. The code is shown below for $n_t = 750$.

```
historical <- matrix(0, ncol=2, nrow=2)
historical[1,] <- c(44, 535)
historical[2,] <- c(33, 304)

set.seed(1)
power <- power.two.grp.random.a0(data.type="Bernoulli",
      n.t=750, n.c=round(750/3),historical=historical,
      samp.prior.mu.t=0.092, samp.prior.mu.c=0.092,
      prior.mu.t.shape1=0.0001, prior.mu.t.shape2=0.0001,
      prior.mu.c.shape1=0.0001,prior.mu.c.shape2=0.0001,
```

Total sample size		1000	1080	1200	1280	1480
n_t		750	810	900	960	1110
n_c		250	270	300	320	370
		Power				
$a_0 = (0.3, 0.3)$	**BayesPPD**	0.843	0.858	0.889	0.898	0.924
	Chen et al. (2011)	0.840	0.856	0.884	0.892	0.923
Random a_0	**BayesPPD**	0.864	0.885	0.909	0.921	0.937
	Chen et al. (2011)	0.843	0.878	0.897	0.902	0.914
		Type I Error Rate				
$a_0 = (0.3, 0.3)$	**BayesPPD**	0.030	0.027	0.032	0.030	0.032
	Chen et al. (2011)	0.030	0.027	0.028	0.030	0.032
Random a_0	**BayesPPD**	0.032	0.027	0.031	0.031	0.031
	Chen et al. (2011)	0.038	0.031	0.029	0.036	0.039

Table 3: Power and type I error rate comparisons for Chen et al. (2011) and **BayesPPD** for a few different sample sizes. We use $N = 10,000$ simulated trials. Two models are considered, a power prior with a_0 fixed at 0.3 for both historical trials and the normalized power prior. We observe that the results provided by **BayesPPD** closely match the results provided in Chen et al. (2011).

```
      prior.a0.shape1=1,prior.a0.shape2=1,
      delta=0.041, gamma=0.95,
      nMC=20000, nBI=250, N=10000)
power$`power/type I error`
[1] 0.864
```

Table 3 compares power calculations from Chen et al. (2011) and **BayesPPD** for a few different sample sizes. We observe that the results provided by **BayesPPD** closely match the results provided in Chen et al. (2011).

Study of acquired immunodeficiency syndrome (AIDS)

Using data from two trials that study the effect of Zidovudine on AIDS, ACTG019 and ACTG036, we will demonstrate how **BayesPPD** can be used for coefficient estimation as well as power and type I error rate calculation for generalized linear models in designs that incorporate historical data.

Zidovudine (AZT) is an inhibitor of the replication of the human immunodeficiency virus (HIV). The ACTG019 study was a double-blind placebo-controlled clinical trial comparing AZT with a placebo in adults with asymptomatic HIV who had CD4 cell counts of fewer than 500 per cubic millimeter. The results were published in Volberding et al. (1990). For this example, we use only the control group data from ACTG019. The binary primary endpoint is death or development of AIDS or AIDS-related complex (ARC). We consider four of the measured covariates used, CD4 cell count (x01) (cell count per cubic millimetre of serum), age (x02), treatment (x03) and race (x04). The covariates CD4 cell count and age are continuous, while the others are binary. The ACTG036 study was also a placebo-controlled clinical trial comparing AZT with a placebo in asymptomatic patients with hereditary coagulation disorders and HIV infection. The results were published in Merigen et al (1991). The endpoint and covariates used are the same as those in the ACTG019 trial. Table 4 summarizes the endpoint and covariates for the two studies.

First, we standardize age for ease of interpretation and take the log of CD4 cell count count.

```
data(actg019)
data(actg036)
Y0 <- actg019$outcome
X0 <- actg019[,-1]
X0$age_std <- scale(X0$age)
X0$T4_log <- log(X0$T4count)
```

	ACTG019 (control group)	ACTG036
No. of patients	404	183
AZT treatment, n (%)	NA	89 (48.6)
CD4 cell count, mean (SD)	332.5 (109.3)	297.7 (130.5)
Age, y; mean (SD)	34.5 (7.7)	30.4(11.2)
White race, n (%)	377 (93.3)	166 (90.7)
Death or ARC, n (%)	36 (8.9)	11 (6.0)

Table 4: Summary of the ACTG019 trial (control group) and the ACTG036 trial data. The binary primary endpoint is death or development of ARC. The sample size of the ACTG019 trial is much larger than the ACTG036 trial. The covariate and endpoint summaries are comparable for the two datasets.

	$a_0 = 0$		$a_0 = 0.5$		$a_0 = 1$		$a_0 \sim$ beta(1,1)	
Intercept	9.14	[3.83; 16.34]	4.89	[1.24; 8.27]	3.95	[0.94; 6.98]	4.39	[1.41; 7.54]
AZT	−0.15	[−1.80; 1.42]	−0.95	[−2.16; 0.25]	−1.00	[−2.12; 0.14]	−0.96	[−2.14; 0.08]
Age (standardized)	0.32	[−0.42; 1.04]	0.36	[−0.01; 0.74]	0.38	[0.11; 0.68]	0.38	[0.06; 0.67]
Race	0.36	[−2.35; 3.23]	0.72	[−1.10; 2.75]	0.93	[−0.83; 3.05]	0.73	[−0.86; 2.44]
log(CD4)	−2.42	[−3.61; −1.35]	−1.48	[−2.04; −0.89]	−1.32	[−1.78; −0.84]	−1.37	[−1.91; −0.86]

Table 5: Posterior mean and 95% credible interval for β incorporating historical data for the four priors, a_0 fixed at 0, 0.5, and 1 and a_0 modeled as random with a beta$(1, 1)$ prior. There is evidence suggesting a negative association between AZT and death but the evidence is not substantial by common criteria (e.g., posterior probability > 0.95).

```
X0 <- as.matrix(X0[,c("age_std","race","T4_log")])

Y <- actg036$outcome
X <- actg036[,-1]
X$age_std <- scale(X$age)
X$T4_log <- log(X$T4count)
X <- as.matrix(X[,c("treat","age_std","race","T4_log")])
```

Suppose we are interested in analyzing the relationship between the outcome and the covariates after incorporating historical information. The code below demonstrates the analysis based on a power prior with a_0 fixed at 0.5 and using only the ACTG019 study data as prior information.

```
set.seed(1)
historical <- list(list(y0=Y0, x0=X0, a0=0.5))
result <- glm.fixed.a0(data.type="Bernoulli", data.link="Logistic", y=Y, x=X,
+                       historical=historical, nMC=10000, nBI=250)
colMeans(result$posterior.samples)
[1]  4.8931870 -0.9459501  0.3645510  0.7201122 -1.4784046
```

Table 5 displays the posterior mean and 95% credible interval for β for four different priors, a_0 fixed at 0, 0.5, and 1 and a_0 modeled as random with a beta$(1, 1)$ prior. There is evidence suggesting a negative association between AZT and death but the evidence is not substantial by common criteria (e.g., posterior probability > 0.95).

For this example we consider designing a new clinical trial that is similar to the historical trial, ACTG019. We hope to acquire a range of sample sizes that can achieve powers around 0.8 to test the hypotheses

$$H_0 : \beta_1 \geq 0$$

and

$$H_1 : \beta_1 < 0$$

based on the chosen sampling priors. Here, β_1 represents the treatment effect of AZT. First, we generate the input for `samp.prior.beta`, a matrix of samples for β representing a discrete

approximation of the sampling prior. For β_1, we sample from a truncated normal distribution with mean -0.5, which is our guess of the effect size of AZT. The distribution is truncated to avoid extreme, implausible values for β_1. For the other parameters, the sampling prior is fixed at the posterior mean of the parameter given the historical data, which can be easily obtained using `glm.fixed.a0` with `current=FALSE`. We then combine the sampling prior for β_1 and the other parameters into a matrix, as follows:

```
library(truncnorm)
set.seed(1)
historical.sp <- list(list(y0=Y0, x0=X0, a0=1))
result <- glm.fixed.a0(data.type="Bernoulli", data.link="Logistic",
                       historical=historical.sp,
                       nMC=10000, nBI=250, current.data = FALSE)
beta.sp <- result$posterior.samples
nSP <- 10000
mat.sp <- matrix(rep(colMeans(beta.sp), each=nSP), nrow=nSP)
beta1.sp <- rtruncnorm(nSP, a=-2, b=-0.1, mean=-0.5)
samp.prior.beta <- cbind(mat.sp[,1], beta1.sp, mat.sp[,2:4])
```

Next, we use `power.glm.fixed.a0` with `approximate=TRUE` to obtain a rough estimate of the sample size required to achieve a power of 0.8. The code below experiments with sample sizes 800, 1000 and 1200. We observe that to reach a power of 0.8, the sample size should be approximately 800 when a_0 is fixed at 0.5.

```
set.seed(1)
sample.sizes <- c(800,1000,1200)
historical <- list(list(y0=Y0, x0=X0, a0=0.5))
results <- NULL
for(i in 1:length(sample.sizes)){
    result <- power.glm.fixed.a0(data.type="Bernoulli", data.size=sample.sizes[i],
                                 historical=historical,
                                 samp.prior.beta=samp.prior.beta,
                                 delta=0, gamma=0.95, approximate=TRUE, N=10000)
    results <- c(results, result$`power/type I error`)
}
results
[1] 0.8037 0.8177 0.8391
```

Finally, we calculate the exact power using the normalized power prior with a_0 modeled as random. The `normalizing.constant` function provides the value for `a0.coefficients` of `power.glm.random.a0`. Since there is only one historical dataset, the `grid` is simply a matrix with one column. The code below demonstrates the usage when sample size is 800. We run $25,000$ iterations of the slice sampler for each of the $10,000$ simulated datasets. The corresponding power is 0.7936. Power curves for the four different priors for sample sizes ranging from 750 to 1200 are plotted in Figure 1. The underlying estimated power values are displayed in Table 6 in the Appendix.

```
grid <- matrix(seq(0.05,1,by=0.1))
historical <- list(list(y0=Y0, x0=X0))
a0_coef <- normalizing.constant(grid=grid, historical=historical,
                                data.type="Bernoulli",data.link="Logistic")
result <- power.glm.random.a0(data.type="Bernoulli",data.link="Logistic",
                              data.size=800, historical=historical,
                              samp.prior.beta=samp.prior.beta,
                              a0.coefficients = a0_coef,
                              delta=0, nMC=25000, nBI=250, N=10000)
result$`power/type I error`
[1]  0.7936
```

Discussion

BayesPPD facilitates Bayesian sample size determination by providing a robust suite of functions for power calculation and analysis using the power and normalized power priors for generalized linear models. A major contribution of this package is the ability to handle covariates for Bernoulli, normal, Poisson and exponential outcomes. Despite the use of MCMC algorithms for analysis and design

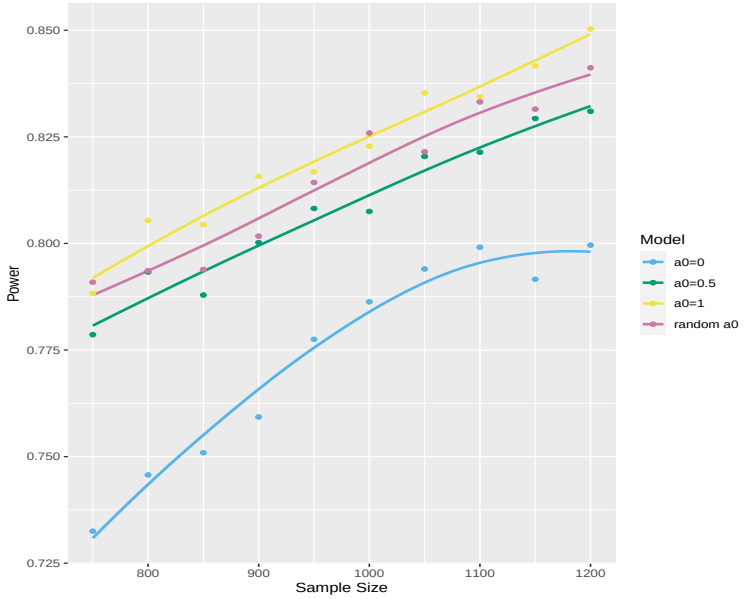


Figure 1: Power curves for a range of sample sizes for the four priors with a_0 fixed at 0 (blue line), 0.5 (green line), 1 (yellow line) and a_0 modeled as random (pink line). The dots represent the power for a particular sample size. LOESS curves have been fitted to the point estimates. We observe that for fixed a_0, the power is higher for higher values of a_0, as more historical information is borrowed. When a_0 is modeled as random, the power curve is higher than the curve with $a_0 = 0.5$ but lower than the curve with $a_0 = 1$.

simulations, **BayesPPD** is computationally efficient, with functions producing results in seconds for many application settings.

A possible extension of the package is the accommodation for longitudinal and time-to-event outcomes. Another potential feature is computing optimal hyperparameters for the beta prior on a_0 to ensure certain characteristics are met, such as the ability to adapt to prior-data conflict or prior-data agreement. The method will be based on ongoing theoretical work by the authors.

Bibliography

S. Balcome, D. Musgrove, T. Haddad, and G. L. Hickey. *bayesDP: Tools for the Bayesian Discount Prior Function*, 2021. URL https://CRAN.R-project.org/package=bayesDP. R package version 1.3.4. [p336]

L. M. Carvalho and J. G. Ibrahim. On the normalized power prior. *Statistics in Medicine*, 40(24): 5251–5275, Jul 2021. doi: 10.1002/sim.9124. [p335]

T. Chandereng, D. Musgrove, T. Haddad, G. Hickey, T. Hanson, and T. Lystig. *bayesCT: Simulation and Analysis of Adaptive Bayesian Clinical Trials*, 2020. URL https://CRAN.R-project.org/package=bayesCT. R package version 0.99.3. [p336]

M.-H. Chen and J. G. Ibrahim. Power prior distributions for regression models. *Statistical Science*, 15(1):46–60, feb 2000. doi: 10.1214/ss/1009212673. [p335, 336, 337]

M.-H. Chen, J. G. Ibrahim, P. Lam, A. Yu, and Y. Zhang. Bayesian design of noninferiority trials for medical devices using historical data. *Biometrics*, 67(3):1163–1170, Sep 2011. doi: 10.1111/j.1541-0420.2011.01561.x. [p335, 338, 343, 344, 345]

Cytel Software Corporation. *East. Software for Design Simulation and Interim Monitoring of Flexible Clinical Trials*. Cambridge, MA, 2014. URL http://www.cytel.com/software-solutions/east/. [p336]

F. De Santis. Using historical data for bayesian sample size determination. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 170(1):95–113, Jan 2007. doi: 10.1111/j.1467-985X.2006.00438.x. [p335]

Y. Duan, K. Ye, and E. P. Smith. Evaluating water quality using power priors to incorporate historical information. *Environmetrics (London, Ont.)*, 17(1):95–106, feb 2006. doi: 10.1002/env.752. [p335, 337]

D. Eddelbuettel and R. Francois. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. [p336]

B. Eggleston, D. Wilson, B. McNeil, J. Ibrahim, and D. Catellier. *BayesCTDesign: Two Arm Bayesian Clinical Trial Design with and Without Historical Control Data*, 2019. URL https://CRAN.R-project.org/package=BayesCTDesign. R package version 0.6.0. [p336]

F. Gerber and T. Gsponer. gsbDesign: An R package for evaluating the operating characteristics of a group sequential Bayesian design. *Journal of Statistical Software*, 69(11):1–23, 2016. doi: 10.18637/jss.v069.i11. [p336]

Z. Han, T. Bai, and K. Ye. *NPP: Normalized Power Prior Bayesian Analysis*, 2021. URL https://CRAN.R-project.org/package=NPP. R package version 0.4.0. [p336]

B. P. Hobbs, B. P. Carlin, S. J. Mandrekar, and D. J. Sargent. Hierarchical commensurate and power prior models for adaptive incorporation of historical information in clinical trials. *Biometrics*, 67 (3):1047–1056, Sep 2011. doi: 10.1111/j.1541-0420.2011.01564.x. [p335]

J. G. Ibrahim, M.-H. Chen, and D. Sinha. On optimality properties of the power prior. *Journal of the American Statistical Association*, 98(461):204–213, mar 2003. doi: 10.1198/016214503388619229. [p335]

J. G. Ibrahim, M.-H. Chen, Y. Gwon, and F. Chen. The power prior: theory and applications. *Statistics in Medicine*, 34(28):3724–3749, dec 2015. doi: 10.1002/sim.6728. [p335, 336, 343]

L. Joseph, C. E. M'Lan, and D. B. Wolfson. Bayesian sample size determination for binomial proportions. *Bayesian Analysis*, 3(2), Jun 2008. doi: 10.1214/08-BA310. [p335]

A. Kopp-Schneider, M. Wiesenfarth, W. Ruth, D. Edelmann, O. Witt, and U. Abel. Monitoring futility and efficacy in phase ii trials with bayesian posterior distributions - a calibration approach. *Biometrical Journal*, to appear, 2018. [p336]

LLC Consultants. *FACTS: Fixed and Adaptive Clinical Trial Simulator*. Austin, TC, 2014. URL http://www.berryconsultants.com/software/. [p336]

C. E. M'Lan, L. Joseph, and D. B. Wolfson. Bayesian sample size determination for case-control studies. *Journal of the American Statistical Association*, 101(474):760–772, Jun 2006. doi: 10.1198/016214505000001023. [p335]

K. Nagashima. *ph2bayes: Bayesian Single-Arm Phase II Designs*, 2018. URL https://CRAN.R-project.org/package=ph2bayes. R package version 0.0.2. [p336]

R. M. Neal. Slice sampling. *Annals of Statistics*, 31(3):705–767, 2003. [p338, 341]

B. Neuenschwander, M. Branson, and D. J. Spiegelhalter. A note on the power prior. *Statistics in Medicine*, 28:3562–3566, 2009. [p337]

B. Neuenschwander, G. Capkun-Niggli, M. Branson, and D. J. Spiegelhalter. Summarizing historical information on controls in clinical trials. *Clinical Trials*, 7(1):5–18, Feb 2010. doi: 10.1177/1740774509356002. [p335]

S. J. Pocock. The combination of randomized and historical controls in clinical trials. *Journal of chronic diseases*, 29(3):175–188, Mar 1976. doi: 10.1016/0021-9681(76)90044-8. [p335]

M. A. Psioda and J. G. Ibrahim. Bayesian design of a survival trial with a cured fraction using historical data. *Statistics in Medicine*, 37(26):3814–3831, Nov 2018. doi: 10.1002/sim.7846. [p336, 339, 343]

M. A. Psioda and J. G. Ibrahim. Bayesian clinical trial design using historical data that inform the treatment effect. *Biostatistics*, 20(3):400–415, Jul 2019. doi: 10.1093/biostatistics/kxy009. [p335]

Y. Qiu, S. Balan, M. Beall, M. Sauder, N. Okazaki, and T. Hahn. *RcppNumerical: 'Rcpp' Integration for Numerical Computing Libraries*, 2019. URL https://CRAN.R-project.org/package=RcppNumerical. R package version 0.4-0. [p341]

E. Rahme and L. Joseph. Exact sample size determination for binomial experiments. *Journal of Statistical Planning and Inference*, 66:83–93, 1998. [p335]

H. Schmidli, S. Gsteiger, S. Roychoudhury, A. O'Hagan, D. Spiegelhalter, and B. Neuenschwander. Robust meta-analytic-predictive priors in clinical trials with historical control information. *Biometrics*, 70(4):1023–1032, Dec 2014. doi: 10.1111/biom.12242. [p335]

Y. Shen, M. A. Psioda, and J. G. Ibrahim. *BayesPPD: Bayesian Power Prior Design*, 2022. URL https://CRAN.R-project.org/package=BayesPPD. R package version 1.1.0. [p335]

R. Simon. Bayesian design and analysis of active control clinical trials. *Biometrics*, 55(2):484–487, Jun 1999. doi: 10.1111/j.0006-341x.1999.00484.x. [p335]

K. Viele, S. Berry, B. Neuenschwander, B. Amzal, F. Chen, N. Enas, B. Hobbs, J. G. Ibrahim, N. Kinnersley, S. Lindborg, and et al. Use of historical control data for assessing treatment effects in clinical trials. *Pharmaceutical Statistics*, 13(1):41–54, Feb 2014. doi: 10.1002/pst.1589. [p335]

F. Wang and A. E. Gelfand. A simulation-based approach to bayesian sample size determination for performance under a given model and for separating models. *Statistical Science*, 17(2):193–208, May 2002. doi: 10.1214/ss/1030550861. [p335]

Y.-B. Wang, M.-H. Chen, L. Kuo, and P. O. Lewis. A new monte carlo method for estimating marginal likelihoods. *Bayesian Analysis*, 13(2):311–333, 2018. [p335, 338, 341]

G. Wassmer and R. Eisebitt. *ADDPLAN: Adaptive Designs – Plans and Analyses*. Reston, VA, 2005. URL http://www.addplan.com/. [p336]

S. Weber. *RBesT: R Bayesian Evidence Synthesis Tools*, 2021. URL https://CRAN.R-project.org/package=RBesT. R package version 1.6-2. [p336]

1 Additional tables

Sample size	$a_0 = 0$	$a_0 = 0.5$	$a_0 = 1$	Random a_0
750	0.732	0.779	0.788	0.791
800	0.746	0.793	0.805	0.794
850	0.751	0.788	0.804	0.794
900	0.759	0.800	0.816	0.802
950	0.778	0.808	0.817	0.814
1000	0.786	0.807	0.823	0.826
1050	0.794	0.820	0.835	0.822
1100	0.799	0.821	0.834	0.833
1150	0.792	0.829	0.842	0.832
1200	0.800	0.831	0.850	0.841

Table 6: Power for the four priors of the AIDS study.

Yueqi Shen
University of North Carolina at Chapel Hill
Department of Biostatistics
United States
ys137@live.unc.edu

Matthew A. Psioda
University of North Carolina at Chapel Hill
Department of Biostatistics
United States
matt_psioda@unc.edu

Joseph G. Ibrahim
University of North Carolina at Chapel Hill
Department of Biostatistics
United States
ibrahim@bios.unc.edu

Bioconductor Notes

by Maria Doyle and Bioconductor Core Developer Team

Introduction

Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data. The project has entered its twentieth year, with funding for core development and infrastructure maintenance secured through 2025 (NIH NHGRI 2U24HG004059). Additional support is provided by NIH NCI, Chan-Zuckerberg Initiative, National Science Foundation, Microsoft, and Amazon. In this news report, we give some updates on core team and project activities.

Software

The current Bioconductor release is 3.16. It is compatible with R 4.2 and consists of 2183 software packages, 416 experiment data packages, 909 up-to-date annotation packages, 28 workflows, and 3 books. Books are built regularly from source and therefore fully reproducible; an example is the community-developed Orchestrating Single-Cell Analysis with Bioconductor.

Bioconductor in Python!

We're excited to introduce BiocPy, a project aimed at enabling Bioconductor workflows in Python. Analysts today use a variety of languages in their workflows, including R/Bioconductor for statistical analysis and Python for imaging or machine learning tasks. BiocPy aims to facilitate interoperability between R and Python by providing standardized data structures based on existing Bioconductor data structures. These include genomic ranges for interval based operations, summarized experiments and other derivatives for analyzing genomic experiments. To learn more, visit the BiocPy GitHub organization.

Core team updates

Default Branch Renaming

The default branch (that corresponds to Bioconductor devel version) will be renamed to devel on git.bioconductor.org for all packages to move forward with diversity and inclusiveness. The core has been doing extensive testing and is ready to move forward within the next 2-3 weeks.

Maintainers will temporarily still be able to push to either devel or master to allow time for adaptation, with the goal of eventually making it an error in maybe 2-3 release cycles (1-1.5 years) We will be making announcements on bioc-devel mailing list, Twitter, Mastodon, Slack, etc with an upcoming/heads up and an announcement when it is live.

Partnering with Outreachy

Bioconductor participated as a mentoring community in Outreachy's December 2022 - March 2023 internship round. Outreachy provides open source and open science internships to individuals impacted by systemic bias and underrepresentation in tech. Bioconductor received funding for two interns through Outreachy's general fund for two projects. Kirabo Atuhurira, a software engineering student from Kampala, Uganda, was chosen to work on the BSgenomeForge, which attempts to simplify making Bsgenome data packages. Beryl Kanali, a masters student from Nairobi, Kenya, was chosen to work on Sweave2Rmd, which attempts to modernize Sweave vignettes by converting them into R Markdown. You can read more about their experience at Our experience as Outreachy interns with Bioconductor.

Bioconductor will also participate in Outreachy's May 2023 - August 2023 internship round to offer internships with Bioconductor community projects.

Conferences

The BioC2023 and EuroBioC2023 conferences have both been announced, and abstract submissions are open. BioC Asia 2023 will be announced later this year.

BioC2023 will be held in Boston, USA from Aug 2-4. Abstracts can be submitted until March 19.

EuroBioC2023 will be held in Ghent, Belgium from Sep 20-22. Abstracts can be submitted until April 14.

Blog

The Bioconductor blog is used for articles of interest by and for the community. Recent articles include:

- Bioconductor Carpentries instructors Year 1
  An update on the Bioconductor Carpentries global training program - applicants selected for Year 1
- Introducing CuratedAtlasQueryR package
  CuratedAtlasQueryR is a new package that enables easy programmatic exploration of CELLxGENE single-cell human cell atlas data.
- Our experience as Outreachy interns with Bioconductor
  Bioconductor participated in the Outreachy Internship program for the December 2022 cohort. Our interns share their experience working on their various projects and Bioconductor in general.
- Bioconductor Hacktoberfest 2022 review and looking ahead to 2023
  Last October, Bioconductor joined Hacktoberfest and got great community contributions there. Here, we describe its success and notify the community we plan to participate in 2023.

Boards updates

Community Advisory Board

- The Community Advisory Board (CAB) are voting in new members. Outgoing members are Susan Holmes, Leonardo Collado-Torres, Yagoub A. I. Adam, Matt Ritchie, Benilton Carvalho, we gratefully thank them for their service.
- There is a Call for Bioconductor Community-Led events supported by the CAB. Apply in this form

Committees and Working groups updates

If you are interested in becoming involved with one of these groups please contact the group leader(s).

- Cloud Methods held their first meeting Feb 2023. The working group will meet monthly to develop standards and tools for the Bioconductor community related to the cloud.
- Social media working group. A new group, started in Feb 2023, have set up Bioconductor Mastodon acccount.

- Teaching Committee
  - Are planning a hackathon for the Bioconductor Carpentries RNA-seq lesson.
  - Will be participating in Galaxy Training Smorgasbord 2023, a free online global event, May 22-26.
- Website working group are planning the redesign of `bioconductor.org`.

Other

- Summer school *"Statistical Data Analysis for Genome-Scale Biology"* CSAMA 2023 in Bressanone-Brixen, Italy, June 11-16 2023. Application deadline March 15.
- Course *"Statistical Analysis of Genome Scale Data"* at CSHL in New York, USA, June 30 - July 13 2023. Application deadline March 15.
- Workshop *"Opportunities for Bioconductor and ELIXIR communities to co-develop training infrastructure"* accepted for the ELIXIR All Hands meeting in Dublin, Ireland, June 5-8 2023.
- Workshop *"Orchestrating Large-Scale Single-Cell Analysis with Bioconductor"* accepted for ISMB/ECCB 2023 in Lyon, France, July 23 2023.

Using Bioconductor

Start using Bioconductor by installing the most recent version of R and evaluating the commands

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install()
```

Install additional packages and dependencies, e.g., SingleCellExperiment, with

```
BiocManager::install("SingleCellExperiment")
```

Docker images provides a very effective on-ramp for power users to rapidly obtain access to standardized and scalable computing environments. Key resources include:

- bioconductor.org to install, learn, use, and develop Bioconductor packages.
- A list of available software linking to pages describing each package.
- A question-and-answer style user support site and developer-oriented mailing list.
- A community slack workspace (sign up) for extended technical discussion.
- The F1000Research Bioconductor gateway for peer-reviewed Bioconductor workflows as well as conference contributions.
- The Bioconductor YouTube channel includes recordings of keynote and talks from recent conferences including BioC2022, EuroBioC2022, and BiocAsia2021, in addition to video recordings of training courses.
- Our package submission repository for open technical review of new packages.

Upcoming and recently completed events are browsable at our events page.

The Technical and and Community Advisory Boards provide guidance to ensure that the project addresses leading-edge biological problems with advanced technical approaches, and adopts practices (such as a project-wide Code of Conduct that encourages all to participate. We look forward to welcoming you!

1 References

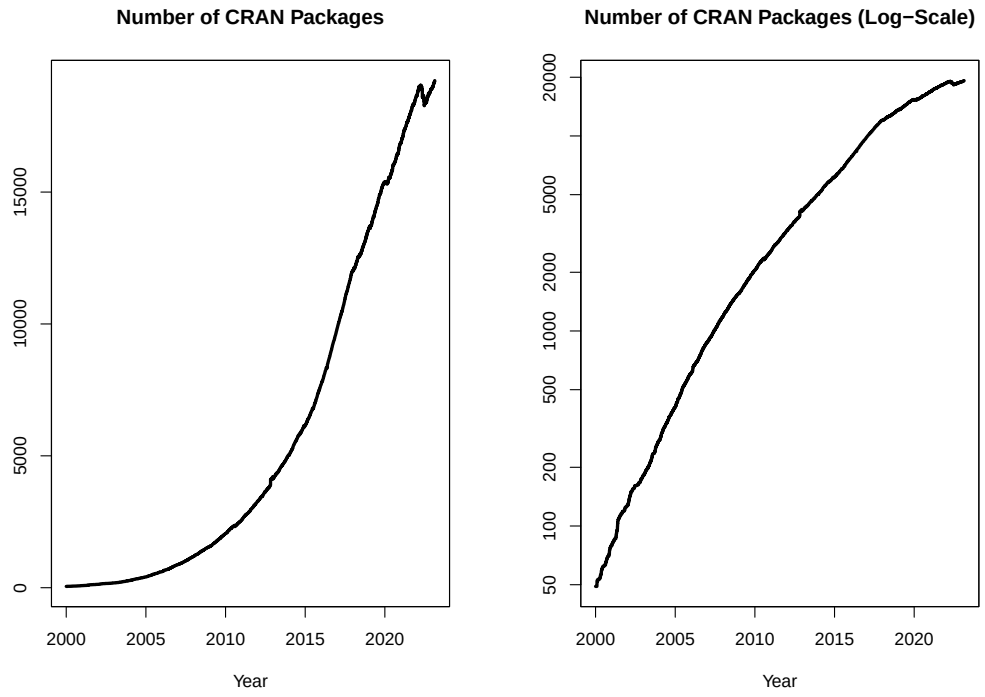
Maria Doyle
University of Limerick,Bioconductor Community Manager

Bioconductor Core Developer Team
Dana-Farber Cancer Institute, Roswell Park Comprehensive Cancer Center, City University of New York, Fred Hutchinson Cancer Research Center, Mass General Brigham

Changes on CRAN

by Kurt Hornik, Uwe Ligges, and Achim Zeileis

In the past 3 months, 462 new packages were added to the CRAN package repository. 219 packages were unarchived, 330 were archived and 1 had to be removed. The following shows the growth of the number of active packages in the CRAN package repository:



On 2023-01-31, the number of active packages was around 19135.

CRAN package submissions

From November 2022 to January 2023 CRAN received 6904 package submissions. For these, 12365 actions took place of which 8114 (66%) were auto processed actions and 4251 (34%) 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	1786	1594	1140	0	0	2164	724	706
manual	1523	81	365	209	54	1422	515	82

These include the final decisions for the submissions which were

	archive	publish
auto	1665 (24.7%)	1733 (25.7%)
manual	1506 (22.3%)	1844 (27.3%)

where we only count those as *auto* processed whose publication or rejection happened automatically in all steps.

CRAN mirror security

Currently, there are 96 official CRAN mirrors, 80 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).

CRAN Task View Initiative

Currently there are 42 task views (see https://cran.r-project.org/web/views/), with median and mean numbers of CRAN packages covered 103 and 116, respectively. Overall, these task views cover 4050 CRAN packages, which is about 21% 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

News from the Forwards Taskforce

by Heather Turner

Forwards is an R Foundation taskforce working to widen the participation of under-represented groups in the R project and in related activities, such as the *useR!* conference. This report rounds up activities of the taskforce during the second half of 2022.

Accessibility

Liz Hare gave a workshop on Writing Meaningful Alt-Texts for Data Visualizations in R for R-Ladies New York and contributed a chapter on the same topic to the Urban Institute's Do No Harm Guide: Centering Accessibility in Data Visualization. Liz also participated in a Book Dash for the Turing Way, a handbook for reproducible, ethical and collaborative data science, discussing technical accessibility, e.g., bandwidth and hardware requirements, as well as disability-related issues.

s gwynn sturdevant contributed to an article for the Justice, Equity, Diversity, and Inclusion (JEDI) Corner of Amstat News, on the topic From Visualization to 'Sensification', considering how senses other than sight might be used for statistical communication.

Yanina Bellini Saibene and Andrea Sánchez-Tapia contributed to a *CarpentryCon 2022* panel on Translation at The Carpentries, sharing their experience of translating technical and educational materials into Spanish. The panel was part of a wider effort by The Carpentries to make their materials available in languages other than English, which will help more people to teach or learn data science skills, including R programming.

Community engagement

RainbowR continued to build on its relaunch, led by Ella Kaye and Zane Dax. Inspired by a post on the RainbowR Twitter, community member Laura Bakala developed the **gglgbtq** package, which provides **ggplot2** palettes and themes based on various pride flags. Zane led the development of the tidyrainbow GitHub repository that collates several datasets pertaining to the LGBTQ+ community, where LGBTQ+ folk are explicitly represented and where it is not assumed that gender is binary. These data sets provide great opportunities for data analysis and visualisation in R and other languages.

RainbowR are meeting bimonthly online, giving people an opportunity to present, in an informal and safe space, anything R related they had been working on. Ella recently set up a Mastodon account @rainbowr@tech.lgbt, which has helped to grow the community, with several new folk joining the RainbowR Slack as a result.

Conferences

Yanina Bellini Saibene was co-chair of the LatinR 2022 conference. Now in its 5th year, 900 people attended the online conference, with keynotes, regular talks and tutorials in each of the conference languages: English, Spanish and Portuguese. A charge was introduced for tutorials reducing no-shows and enabling tutors to be paid, but scholarships were offered thanks to donations. Yanina was a keynote speaker at *CarpentryCon 2022*, where she spoke about Achieving the change we want one conference at a time (in Spanish). In the talk she shared actions that can be taken to make events more diverse and inclusive, based on 20 years of experience organizing face-to-face, virtual and hybrid events. Many of these actions are summarized in the article "Ten simple rules to host an inclusive conference" (Joo et al. 2022), which distils learnings from organizing *useR! 2021*.

Teaching

Paola Corrales and Yanina Bellini Saibene gave a tutorial From spreadsheets to R at the Software Sustainability Institute's Research Software Camp: Next Steps in Coding. The tutorial was presented in Spanish - one of two events during the research camp in a language other than English.

For anyone wanting to teach package development, we would like to highlight the Forwards materials which are organized into four 1-hour hands-on modules: Packages in a Nutshell, Setting up your system, Your first package, and Package documentation. The material is released under a CC BY-NC-SA license.

R Contribution

Saranjeet Kaur Bhogal continued work on the R Development Guide, as part of the R project's Google Summer of Docs project, alongside the second technical writer, Lluís Revilla. The case study summarises the progress made during the official project period. There is some funding remaining from the grant, so there are plans to continue work in 2023.

Michael Chirico and others supported R Contribution Working Group (RCWG) initiatives to encourage more people to contribute translations to base R. RCWG member Gergely Daróczi set up a Weblate server to provide a user-friendly web interface for adding/editing translations. This was used during the R translatón/Hackaton de tradução do R that was organized by RCWG/LatinR as a satellite to *LatinR 2022*, led by Beatriz Milz, Ángela Sanzo, Macarena Quiroga, and Caio Lente. Use of the Weblate server has gradually increased and Michael Lawrence made a recent commit to R-devel, which bundled over 3000 translations from the platform across 8 languages and 17 translators.

Heather Turner and Ella Kaye joined RCWG members Elin Waring and Gabriel Becker in running monthly Office Hours for R contributors. These sessions provide an opportunity to discuss how to get started contributing to R and to look at open bugs or work on translations together. This has led to some bugs being closed such as Bug 16158: Error in predict.lm for rank-deficient cases and Bug 17853: stats::factanal print method drops labels when sort=TRUE. The Office Hours are held at two times on the same day to cater for different time zones, see the R Contributors Events page or Meetup for upcoming events.

We also started planning for the R Project Developer Sprint 2023, to be hosted at the University of Warwick, Coventry, UK, from August 30 to September 1. This sprint will bring together novice and experienced contributors, to work alongside members of the R Core Team on contributions to base R. Thanks to the event sponsors, full board on-campus accommodation will be provided during the sprint and funding is available for travel if required. Anyone interested to attend is encouraged to self-nominate via the application form by **Friday 10 March**. Additional sponsors are also welcome.

Social Media

RCWG and Forwards have also set up Mastodon accounts, follow them at @R_Contributors@hachyderm.io and @R_Forwards@hachyderm.io, respectively.

Changes in Membership

New members

We welcome the following members to the taskforce:

- On-ramps team: Allison Vuong
- Surveys team: Daniela Cialfi

References

Joo, Rocío, Andrea Sánchez-Tapia, Sara Mortara, Yanina Bellini Saibene, Heather Turner, Dorothea Hug Peter, Natalia Soledad Morandeira, et al. 2022. "Ten Simple Rules to Host an Inclusive Conference." *PLOS Computational Biology* 18 (7): 1–13. https://doi.org/10.1371/journal.pcbi.1010164.

Heather Turner
University of Warwick
UK
Heather.Turner@R-Project.org

Changes in R

by Tomas Kalibera, Sebastian Meyer, and Kurt Hornik

Abstract We present important changes in the development version of R (referred to as R-devel, to become R 4.3). Some statistics on bug tracking activities in 2022 are also provided.

1 R-devel selected changes

R 4.3.0 is due to be released around April 2023. The following gives a selection of the most important changes in R-devel, which are likely to appear in the new release.

Dates and times

There are a number of (robustness) improvements in the handling of dates and times. These include warnings about extrapolation for datetimes before 1902/1900, finer control for padding when printing years, improved detection of offset with `strftime()` (`%z`), inclusion of system time zone and information on timezone support implementation in `sessionInfo()`, more robust handling of hand-crafted `POSIXlt` objects, optional support for using system timezone support on recent macOS, improved detection of the system time zone on Windows and improved default tick locations and default formats in `axis.Date()` and `axis.POSIXct()`.

Encoding support

- Performance of regular expression operations in R has been improved by reducing the costs of encoding conversions. With `perl=FALSE`, all inputs have to be converted to UTF-16 or UTF-32, and the conversion is now faster. With `perl=TRUE`, performance has been improved by opting out from duplicate checks for UTF-8 validity in PCRE2. With `fixed=TRUE`, performance has been improved by taking advantage of the properties of UTF-8. One of the motivations for the speedups was to reduce the incentive for using `useBytes=TRUE` with regular expression operations, which often leads to incorrect results or errors due to producing invalid strings.

  See Speedups in operations with regular expressions for more information.

- The support for encoding-agnostic string operations in R using the "bytes" encoding has been improved. It is now possible to read a text file directly as bytes. Regexp operations, when creating new strings by splitting or substituting, now also flag them as "bytes" when any of the input has been flagged as such. This simplifies encoding-agnostic parsing of files such as `DESCRIPTION`. `iconv(,from="")` now respects the encoding flag of the input string, making it easier to recover from type-instability in return values of regular expression operations. Improving the support for encoding-agnostic operations using the "bytes" encoding comes together with stricter checking of validity of real strings in a character encoding, e.g. "unknown/native", which has been helpful in revealing user errors. In the long term, it should also help to simplify encoding support in R.

  See Improvements in handling bytes encoding for more information. The blog includes a detailed introduction to string and encoding support in R.

  See Why to avoid \x in regular expressions for related information on the danger of using `\x` escapes in regular expressions, which leads to errors, that are now more likely to be detected by R. This is closely related as `\x` is a common way to create invalid strings.

Graphics

- The grDevices and grid packages have new functions for rendering typeset glyphs, primarily: `grDevices::glyphInfo()` and `grid::grid.glyph()`.
- The behaviour of compositing operators in `grid::grid.group()` has been tweaked to allow consistency across graphics devices.
- The `grDevices::quartz()` device will support gradient fills, pattern fills, clipping paths, masks, compositing operators, affine transformations, stroked/filled paths, and glyphs. To be soon merged to R-devel.

Accessibility on Windows

- `Rgui` console on Windows now works better with the open-source NVDA screen reader when the "full" blinking cursor is selected. This is due to improved implementation of the console cursor (when it is displayed and hidden with respect to application startup and window focus) on which makes it easier for the screen reader to detect where the cursor is. Previously, NVDA was not able to read out the character under the cursor moved by the arrow keys.
- The drop-field GraphApp control, which is used in the `Rgui` configuration editor, has been extended so that it can be left by pressing the TAB key, so without using the mouse.
- GraphApp has been extended to allow reverse-order navigation through the controls using Shift+TAB key, which can now be done also in the `Rgui` configuration editor.

Other selected changes

- Using vectors of more than one element with the logical operators `&&` and `||` will give an error in R 4.3.0 (a warning in R 4.2.x, a check error since R 3.6.0).
- Support for working with concordances has been extended from Sweave to help files. A concordance is a mapping between lines in an intermediate file (e.g., `.tex` or `.html`) and lines in the corresponding input file (e.g., `.Rnw` or `.Rd`), which, for example, allows relating problems in the intermediate file to the source file from which it was generated.

  See Concordances for more information.
- The implementation of the sampling profiler, `Rprof()`, has been improved. On macOS, the profiler is now more robust against high load on the system by using low-level Mach API to avoid a race condition between initialization of pthread data and arrival of a profiler signal. This race condition could lead to a live-lock when the system has been overloaded due to a too short profiling interval. As an additional measure, `Rprof()` now refuses to use a too short profiling interval, which in the first place would lead to incorrect profiling results. To prevent a deadlock seen on Windows, the profiler has been rewritten to avoid using C runtime functions while the main thread is suspended.
- Package installation now uses C++17 as the default C++ standard (and there is initial support for C++23). Also, there now is support for a package to indicate the version of the C standard which should be used to compile it, and for the installing user to specify this. In most cases, C17 (a "bug-fix" of C11) is used by default.
- Producing PDF manuals (`R CMD Rd2pdf`) now loads standard AMS-LaTeX packages for greater coverage of math commands in Rd equations (e.g., `\lVert` and `\text`), and for consistency with the enhanced HTML math rendering introduced in R 4.2.0. This change has been backported to the R 4.2 release branch.
- The `"repos"` option is now initialized from the `repositories` file, see `?R_REPOSITORIES`, allowing the default CRAN mirror to be set therein.

2 Bug statistics for 2022

Summaries of bug-related activities over the past year were derived from the database underlying R's Bugzilla system. Overall, 180 new bugs or requests for enhancements were reported, 171 reports were closed, and 869 comments (on any report) were added by a total of 123 contributors. This amounts to one report/closure every other day, and 2–3 comments per day. The numbers of reports, closures and comments are about 20% lower than in 2021, whereas the number of contributors stayed the same. High bug activity in 2021 had largely been driven by dedicated efforts of several contributors in reviewing old reports.

Figures 1 and 2 show statistics for the numbers of new reports, closures and comments by calendar month and weekday, respectively, in 2022. The frequency of new reports was relatively stable over the year with minor peaks in January and June. There tended to be more new reports than closures, except for July and especially March with a revived effort to deal with old reports, including 9 related to the **nlme** package, which is also maintained by the R Core Team.

The top 5 components reporters have chosen for their reports were "Misc", "Language", "Low-level", "Documentation", and "Wishlist", which is the same set as in 2021. Many reports are suggestions for enhancements and placed either in the "Wishlist" or in a specific component but with severity level set to "enhancement". Bug discussions led to an average of 72 comments per month, with a minimum of 42 in August and a maximum of 111 in January. From the numbers in Figure 2 we see that the R community is also active during weekends, though at a lower frequency.

Figure 1: Bug tracking activity by month in 2022

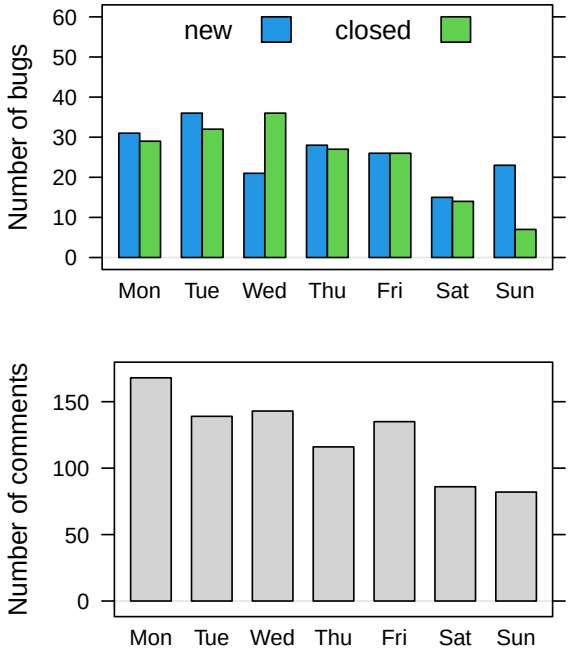


Figure 2: Bug tracking activity by weekday in 2022

Acknowledgements

Tomas Kalibera's work on the article and R development has received funding from the National Science Foundation award 1925644.

Tomas Kalibera
R Core
Czechia
Tomas.Kalibera@R-project.org

Sebastian Meyer
Friedrich-Alexander-Universität Erlangen-Nürnberg
Germany
ORCiD: 0000-0002-1791-9449
Sebastian.Meyer@R-project.org

Kurt Hornik
WU Wirtschaftsuniversität Wien
Austria
ORCiD: 0000-0003-4198-9911
Kurt.Hornik@R-project.org

R Foundation News

by Torsten Hothorn

1 Donations and members

Membership fees and donations received between 2022-09-05 and 2023-02-19.

Donations

Bharat R Adhikary (Nepal) b-data GmbH (Switzerland) RV Detailing Pros of San Diego (United States) Shalese Fitzgerald (United States) Roger Koenker (United Kingdom) Rudolph Martin (United States) Kem Phillips (United States) Panagiotis Togias (Greece) Jason Wyse (Ireland) Statistik Aargau, Aarau (Switzerland)

Supporting institutions

Ef-prime, Inc., Chuo-ku (Japan) Institute of Botany of the Czech Academy of Sciences, Pruhonice (Czechia) oikostat GmbH, Ettiswil (Switzerland) Putnam Data Sciences, LLC, Cambridge (United States)

Supporting members

Douglas Adamoski (Brazil) Tim Appelhans (Germany) Luis Biedma (Luxembourg) Michael Blanks (United States) Gordon Blunt (United Kingdom) Tamara Bozovic (New Zealand) Susan M Carlson (United States) Cédric Chambru (Switzerland) John Chandler (United States) Michael Chirico (United States) Tom Clarke (United Kingdom) Gerard Conaghan (United Kingdom) Terry Cox (United States) Robin Crockett (United Kingdom) Brandon Dahl (United States) Robert Daly (Australia) Gergely Daroczi (Hungary) Dereck de Mezquita (United States) Anna Doizy (Réunion) Fraser Edwards (United Kingdom) Anthony Alan Egerton (Malaysia) MAEL ELEGOET (France) Faisel Eskander (United Kingdom) Dane Evans (United States) Isaac Florence (United Kingdom) Neil Frazer (United States) David Freedman (United States) Keita Fukasawa (Japan) Sven Garbade (Germany) Jan Marvin Garbuszus (Germany) Gabriel Gersztein (Brazil) Anne Catherine Gieshoff (Switzerland) Pavel Goriacko (United States) Brian Gramberg (Netherlands) Spencer Graves (United States) Krushi Gurudu (United States) Hlynur Hallgrímsson (Iceland) Joe Harwood (United Kingdom) Bela Hausmann (Austria) Kieran Healy (United States) Philippe Heymans Smith (Costa Rica) Adam Hill (United States) Alexander Huelle (Germany) Lorenzo Isella (Belgium) Sebastian Jeworutzki (Germany) Nigokhos Kanaryan (Bulgaria) Curtis Kephart (United States) JUNE KEE KIM (Korea, Republic of) Miha Kosmac (United Kingdom) Jan Herman Kuiper (United Kingdom) Flavio L (Switzerland) Bernardo Lares (Venezuela) Thierry Lecerf (Switzerland) Mauro Lepore (United States) Thomas Levine (United States) Chin Soon Lim (Singapore) Joseph Luchman (United States) Mehrad Mahmoudian (Finland) Gilles Marodon (France) Daniel McNichol (United States) Amanuel Medhanie (United States) Orlando Monsalve (Switzerland) Keon-Woong Moon (Korea, Republic of) Guido Möser (Germany) yoshinobu nakahashi (Japan) Dan Orsholits (Switzerland) George Ostrouchov (United States) Antonio Paez (Canada) Elgin Perry (United States) VASILEIOS PLESSAS (United Kingdom) PierGianLuca Porta Mana (Norway) Bianca Prandi (Austria) Fergus Reig Gracia (Spain) Peter Ruckdeschel (Germany) Ingo Ruczinski (United States) Choonghyun Ryu (Korea, Republic of) Nico Schäfer (Germany) Raoul Schorer (Switzerland) Dejan Schuster (Germany) Jagat Sheth (United States) David Sides (United States) Pedro Silva (Brazil) Rachel Smith-Hunter (United States) Murray Sondergard (Canada) Matteo Starri (Italy) Tobias Strapatsas (Germany) Kai Streicher (Switzerland) ROBERT Szabo (Sweden) Fred Viole (United States) Marcus Vollmer (Germany) Dr. Alfred Wagner (Germany) Petr Waldauf (Czechia) Jaap

Walhout (Netherlands) Sandra Ware (Australia) Arne Jonas Warnke (Germany) Vaidotas Zemlys-Balevičius (Lithuania) Lim Zhong Hao (Singapore) 广宇曾(China)

Torsten Hothorn
Universität Zürich, Switzerland Torsten.Hothorn@R-project.org