The R Journal

Volume 17/2, June 2025

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

Contents

Prospective authors will find detailed and up-to-date submission instructions on the Journal's homepage.

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 Rob J Hyndman

In this issue

On behalf of the editorial board, I am pleased to present Volume 17 Issue 2 of the R Journal. This issue features ten research articles, plus news from CRAN, BioConductor and the R Foundation.

The first article concerns the *CRAN Task View initiative*, which provides valuable curated lists of R packages in specific areas. The article describes the current status of the initiative and how users can contribute.

The remaining articles discuss various R packages that have been developed, covering a wide range of topics. Each article provides an overview of the package, its functionality, and examples of its use. All packages discussed are available on CRAN. Supplementary material with fully reproducible code is available for download from the Journal website.

Rob J Hyndman
Monash University

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

The CRAN Task View Initiative

by Achim Zeileis, Roger Bivand, Dirk Eddelbuettel, Kurt Hornik, Julia Piaskowski, and Nathalie Vialaneix

Abstract CRAN Task Views have been available on the Comprehensive R Archive Network since 2005. They provide guidance about which CRAN packages are relevant for tasks related to a certain topic, and can also facilitate automatic installation of all corresponding packages. Motivated by challenges from the growth of CRAN and the R community as a whole since 2005, all of the task view infrastructure and workflows were rethought and relaunched in 2021/22 in order to facilitate maintenance, and to foster deeper interactions with the R community. The redesign encompassed the establishment of a group of CRAN Task View Editors, moving all task view sources to dedicated GitHub repositories, adopting well-documented workflows with a code of conduct, and leveraging R/Markdown files (rather than XML) for the content of the task views.

1 Motivation and background

The Comprehensive R Archive Network (CRAN, https://CRAN.R-project.org/), comprised of approximately one hundred mirror pages around the world along with a content-delivery network (CDN), is the canonical site for downloading not only the base R system (R Core Team, 2025) but also more than twenty thousand extension packages (actually, 22749 at the time of writing) contributed by users from the R community. CRAN is a very useful resource within the R ecosystem, but its content can be hard to grasp and navigate. Especially users new to R can struggle with getting started and finding the relevant R packages for the tasks they want to accomplish.

To mitigate this problem, Zeileis (2005), in cooperation with the CRAN team, introduced "*CRAN Task Views*" that provide guidance about which CRAN packages are relevant for tasks related to a given topic. Task views offer a brief overview of the included packages on a dedicated CRAN page (see https://CRAN.R-project.org/web/views/ for an overview) and also enable the automatic installation of all packages from a task view using the **ctv** package (Zeileis and Hornik, 2024). The views are intended to have a sharp focus so that it is sufficiently clear which packages should be included (or excluded). They are not meant to endorse the "best" packages for a given task but they can distinguish a shorter list of most relevant/important *core* packages from the remaining *regular* packages.

While CRAN Task Views alone were certainly not able to overcome all the challenges of navigating CRAN, they proved to be useful enough to be continued over the subsequent years. However, due to the growth of CRAN (see, e.g., Hornik et al., 2024) and of the R community as a whole since the introduction of task views in 2005, the thriving of the task views was limited by several design decisions regarding their format and the corresponding workflows:

- The format for authoring a new task view was based on XML (extensible markup language). This required that task view maintainers as well as anyone who wanted to contribute write XML and HTML (hypertext markup language) directly.
- The format required that the packages included in a task view needed to be described in the information text and listed separately to be categorized into *core* and *regular* packages.
- Task views were typically proposed by individual contributors (and only in some cases by teams).
- The onboarding process for CRAN was mostly coordinated by Achim Zeileis alone.
- All task views were maintained in a single subversion (SVN) repository on the R-Forge platform (Theußl and Zeileis, 2009) to which all task view maintainers had access.
- Contributions from the R community were mostly limited to e-mails to the respective maintainers (except where the maintainers set up other channels of communication

themselves, e.g., through a separate GitHub repository).

This setup worked sufficiently well in 2005 when there were about 500 packages on CRAN and initially 4 task views, listing on average 20 packages each. (Within a year, the number of task views increased to 12.) But by now there are 48 task views, with median and mean numbers of CRAN packages covered 111 and 123, respectively. Overall, these task views cover 4937 CRAN packages, which is about 22% of all CRAN packages. This change in scope and scale necessitated a change in infrastructure and workflows that the new *CRAN Task View Initiative*, described in detail in the next section, aims to provide.

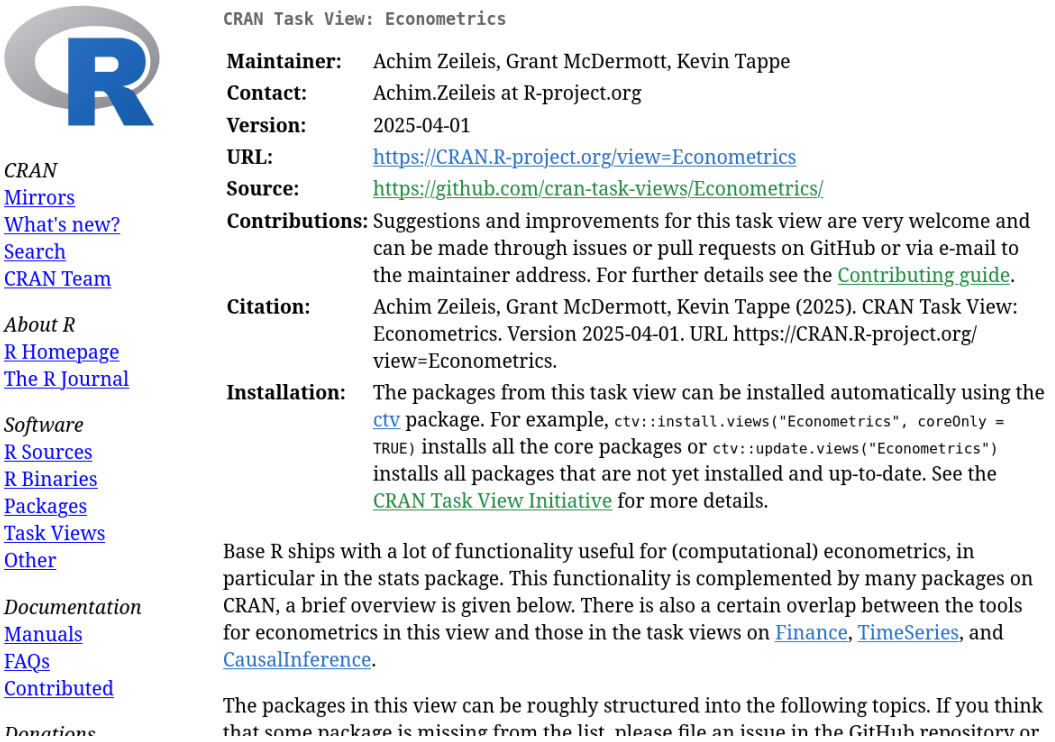
2 New design

Motivated by the challenges from the growth of CRAN and the R community as a whole, almost all aspects of the CRAN Task Views were rethought and relaunched in 2021/22 in order to facilitate maintenance and foster more interactions with the R community. The important changes are:

- The *CRAN Task View Initiative* is now overseen by a group of *CRAN Task View Editors* (rather than an individual) who review proposals of new task views, support the onboarding of the corresponding maintainers, and monitor the activity in existing task views. The corresponding official e-mail is `CRAN-task-views@R-project.org`.
- All activities are hosted on GitHub (rather than R-Forge) in a dedicated organization (`https://github.com/cran-task-views/`) which provides interfaces and workflows that many R users are familiar with. In particular, it offers a wide range of possibilities for the community to engage with the task views, most notably through issues and pull requests.
- Each task view is hosted in a separate repository within the `cran-task-views` organization (rather than all in one repository), giving the maintainers more freedom while preserving sufficient control for the editors. Also, separate projects provide better visibility for each task view and a clearer separation of responsibilities.
- For new task view proposals, the principal maintainer of a task view is expected to assemble a team of 1 to 5 co-maintainers to share the workload and reflect different perspectives. The same was also strongly encouraged for older task views that had previously been maintained by a single person. Ideally the co-maintainers should be a diverse group in terms of gender, origin, scientific field, etc.
- The file format for authoring task views is now based on R/Markdown (rather than XML and HTML directly). The files can be processed and rendered into HTML output for CRAN with dedicated functions from the **ctv** package, leveraging the popular packages **knitr** (Xie, 2015) and **rmarkdown** (Xie et al., 2018) based on the document converter pandoc (MacFarlane, 2025).
- As the task view files are now dynamic R/Markdown documents, it was easy to avoid certain redundancies from the old XML-based format, e.g., the package list with *core* and *regular* packages is compiled automatically from the information text and does not have to be specified separately.
- All contributions must now explicitly adhere to the code of conduct of the initiative, adapted from the Contributor Covenant code of conduct.

More details on the CRAN Task View Initiative are available in the GitHub repository `https://github.com/cran-task-views/ctv`. This provides an overview of the activities, detailed documentation, and the possibility to raise issues that concern the initiative as a whole (rather than individual task views).

The current CRAN Task View Editors are Roger Bivand, Dirk Eddelbuettel, Julia Piaskowski, Nathalie Vialaneix, and Achim Zeileis. Former contributors include: Henrik Bengtsson, Rocío Joo, David Meyer, and Heather Turner. The primary contributor from the CRAN team is Kurt Hornik.



CRAN Task View: Econometrics

Maintainer:	Achim Zeileis, Grant McDermott, Kevin Tappe
Contact:	Achim.Zeileis at R-project.org
Version:	2025-04-01
URL:	https://CRAN.R-project.org/view=Econometrics
Source:	https://github.com/cran-task-views/Econometrics/
Contributions:	Suggestions and improvements for this task view are very welcome and can be made through issues or pull requests on GitHub or via e-mail to the maintainer address. For further details see the Contributing guide.
Citation:	Achim Zeileis, Grant McDermott, Kevin Tappe (2025). CRAN Task View: Econometrics. Version 2025-04-01. URL https://CRAN.R-project.org/view=Econometrics.
Installation:	The packages from this task view can be installed automatically using the ctv package. For example, `ctv::install.views("Econometrics", coreOnly = TRUE)` installs all the core packages or `ctv::update.views("Econometrics")` installs all packages that are not yet installed and up-to-date. See the CRAN Task View Initiative for more details.

Base R ships with a lot of functionality useful for (computational) econometrics, in particular in the stats package. This functionality is complemented by many packages on CRAN, a brief overview is given below. There is also a certain overlap between the tools for econometrics in this view and those in the task views on Finance, TimeSeries, and CausalInference.

The packages in this view can be roughly structured into the following topics. If you think that some package is missing from the list, please file an issue in the GitHub repository or

CRAN
Mirrors
What's new?
Search
CRAN Team

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Task Views
Other

Documentation
Manuals
FAQs
Contributed

Donations

Figure 1: Screenshot of the header and introduction of the "Econometrics" task view.

3 Using task views

First and foremost, the web page of each task view can be read by interested users (see Figure 1 for an example). The clear structure and focus of the task views should help the readers to quickly gain a first overview of a topic and to search for specific relevant packages more efficiently.

Another benefit of task views that has probably been underappreciated for a long time is that they allow the easy installation of the associated packages (either all of them or just the core packages). The headers of all task view web pages now promote this possibility explicitly. To make use of this, the ctv package needs to be installed, e.g., via

```
install.packages("ctv")
```

The package provides the two functions `install.views()` and `update.views()`, where the latter only installs those packages which are not installed and up-to-date. For example, in order to install the full "Econometrics" task view either one of the following two calls can be used:

```
ctv::install.views("Econometrics")
ctv::update.views("Econometrics")
```

Moreover, two functions are provided for querying the information from task views from within R. First, information on a single task view can be obtained with the `ctv()` function.

```
ctv::ctv("Econometrics")

#> CRAN Task View
#> --------------
#> Name:       Econometrics
#> Topic:      Econometrics
#> Maintainer: Achim Zeileis, Grant McDermott, Kevin Tappe
```

```
#> Contact:    Achim.Zeileis@R-project.org
#> Version:    2025-09-01
#> Repository: https://CRAN.R-project.org
#> Source:     https://github.com/cran-task-views/Econometrics/
#> Packages:  AER*, aod, apollo, apt, bacondecomp, bayesm, betareg, bimets, BMA,
#>           BMS, boot, bootstrap, brglm, CADFtest, car*, censReg, clubSandwich,
...
```

To list all task views available from CRAN, the function `available.views()` can be used:

```
ctv::available.views()
```

```
#> CRAN Task Views
#> ---------------
#> Name: ActuarialScience
#> Topic: Actuarial Science
#> Maintainer: Christophe Dutang, Vincent Goulet
#> Repository: https://CRAN.R-project.org
#> ---------------
#> Name: Agriculture
#> Topic: Agricultural Science
#> Maintainer: Julia Piaskowski, Adam Sparks, Adrian Correndo
#> Repository: https://CRAN.R-project.org
#> ---------------
...
```

The objects returned by `ctv()` and `available.views()` include additional information which is not shown by the `print()` method. This may be useful for more specific computations based on the task views. For example, a `citation` object (Hornik et al., 2012) can be obtained from the list returned by `ctv()`.

```
ctv::ctv("Econometrics")$citation
```

```
#>
#>   Zeileis A, McDermott G, Tappe K (2025). _CRAN Task View:
#>   Econometrics_. Version 2025-09-01,
#>   <https://CRAN.R-project.org/view=Econometrics>.
#>
#> A BibTeX entry for LaTeX users is
#>
#>   @Manual{,
#>     author = {Achim Zeileis and Grant McDermott and Kevin Tappe},
#>     title = {CRAN Task View: Econometrics},
#>     year = {2025},
#>     note = {Version 2025-09-01},
#>     url = {https://CRAN.R-project.org/view=Econometrics},
#>   }
NA
```

4 Contributing

Users from the R community can contribute in two ways to the CRAN Task View Initiative: They can either provide suggestions for an *existing* task view, or they can *propose a new one* on a topic that is not yet covered. In either case, all contributions must be made under the code of conduct.

Contributions to existing task views are welcome, encouraged, and in fact crucial for keeping the task views up to date. Typical contributions would be improvements in existing content (e.g., adding details, clarifications, or corrections) or suggestions of additional content (e.g., packages or links). To facilitate such contributions, each task view includes the e-mail address of the principal maintainer as well as a link to the associated GitHub repository. Thus, contributors can choose the most convenient alternative among the following ones:

- *Send an e-mail* to the principal maintainer.
- *Raise an issue* in the GitHub repository.
- *Provide a pull request* in the GitHub repository.

To avoid having a pull request become more involved or disruptive, it is frequently suggested to first discuss proposed changes by raising an issue, and to make sure that the modified task view file still works correctly (see Section 6).

For proposing new task views, the CRAN Task View Initiative provides a standardized workflow including a review and onboarding process. All steps are detailed at `https://github.com/cran-task-views/ctv/blob/main/Proposal.md`. In the following, we just outline the essential aspects.

First, it is important that the prospective maintainers consider the time and work that is required to put together a proposal, to refine it during the review and onboarding process, and most importantly to actively maintain the task view in the future.

Second, a fundamental and very crucial step is to formulate the topic of the new task view so that it has a clear scope that is neither too narrow nor too wide. The goal is not to cover "every package" remotely related to the topic but rather the set of packages that clearly fall within the scope. The coverage should be similar to what an introductory text book on the topic would cover. Non-CRAN packages may also be included but the focus of CRAN task views should be packages on CRAN (as the name conveys). Finally, task views should *not* rate the packages or endorse certain "best" packages but rather give an overview of what is available. A bit of emphasis to the more important packages can be given in two ways: (a) The most important packages can be flagged as *core* packages. (b) In the information text the more important packages can be listed first in the respective sections.

Third, based on the information formulated as outlined above, the proposal can be made in the GitHub issues of the `ctv` repository. This initiates a first review that is carried out in the comments of the issue tracker which are open to all members of the community. In all cases, the CRAN Task View Editors are explicitly invited to comment, as may be maintainers of related task views or of relevant core packages etc. Provided that there is sufficient endorsement from the CRAN Task View Editors, typically after revisions and refinements from the prospective maintainers, a proposal is accepted, initiating an onboarding process that leads to the publication of the task view on CRAN.

Finally, the maintainers of the task view are responsible for keeping it up to date by checking CRAN regularly. Contributions from the community, as described at the beginning of this section, are eminently useful for this and hence explicitly encouraged. Moreover, some R packages like **CTVsuggest** (Dijk, 2023) can support the maintainers in discovering new relevant packages on CRAN.

5 Handling package archivals

The CRAN packages listed in task views should ideally be maintained actively, so that improved versions are released by the corresponding maintainers in case the daily CRAN checks discover any issues. However, it is not straightforward to test for active maintenance fully automatically and even actively maintained packages may be temporarily archived on CRAN. Hence, the following strategy is adopted within the CRAN Task View Initiative:

When a CRAN package from a task view is archived, it is still listed in the task view like before. It is only flagged as archived in the text and is not installed automatically anymore by `install.views()` and `update.views()`.

This strategy gives both the package maintainers and the task view maintainers some time to resolve the situation. Specifically, the task view maintainers can decide whether to

- *exclude* the package from the task view immediately, e.g., if it was archived for policy violations, at the request of the maintainer, or did not have any updates for many years and is not associated with a public repository;
- *wait some more* for an improved version, e.g., when they see that the package maintainers already started addressing the problem; or
- *reach out to the package maintainers* to check if they intend to release a corrected version or even to help with releasing an improved version.

To help discovering archived packages and initiating one of the actions above, the CRAN team regularly checks whether any task view contains packages that have been archived on CRAN for 60 days or more. If so, they create an issue in the corresponding task view repository.

After the period of grace (100 days) ends, the situation should be resolved by the task view maintainers, typically by excluding packages that are still archived from the task view. For sufficiently relevant packages, it may be sensible to replace the package listing by a link, e.g., to a GitHub repository for the package.

6 R/Markdown format

The file format for CRAN task views leverages the R/Markdown format (Xie et al., 2018) so that standard Markdown can be used for formatting and structuring the text and a handful of special R functions are provided to link to CRAN packages, other task views, GitHub projects, etc. The format is mostly self-explanatory and is illustrated below using an excerpt from the `Econometrics` task view:

```
---
name: Econometrics
topic: Econometrics
maintainer: Achim Zeileis, Grant McDermott, Kevin Tappe
email: Achim.Zeileis@R-project.org
version: 2025-04-01
source: https://github.com/cran-task-views/Econometrics/
---

Base R ships with a lot of functionality useful for (computational) econometrics,
in particular in the stats package. This functionality is complemented by many
packages on CRAN, a brief overview is given below. There is also a certain
overlap between the tools for econometrics in this view and those in the task
views on `r view("Finance")`, `r view("TimeSeries")`, and
`r view("CausalInference")`.

Further information can be formatted with standard Markdown syntax, e.g., for
_emphasizing text_ or showing something really important in **bold face**.
R/Markdown syntax with special functions can be used to link to a standard
package like `r pkg("mlogit")` or an important "core" package like
`r pkg("AER", priority = "core")`.

### Links
- [The Title of a Relevant Homepage](http://path/to/homepage/)
```

The document structure consists of three main blocks: (a) Some metainformation is given in the YAML header at the beginning (separated by lines with `---`), followed by (b) the information in the main text, and (c) a concluding special section called `### Links`. Details are explained in the official documentation: https://github.com/cran-task-views/ctv/blob/main/Documentation.md.

The information in the main text should be a short description of the packages, explaining which packages are useful for which tasks. Additionally, short R code chunks with special functions are used for linking to CRAN resources: `pkg()` for regular packages, `pkg(..., priority = "core")` for important *core* packages, and `view()` for related task views. Moreover, code projects in other repositories can be linked by dedicated functions, e.g., `bioc()` or `github()` for packages on Bioconductor or GitHub, respectively.

In order to check whether a task view file has been formatted properly it can be read into R and rendered to HTML (see also Figure 1) which can be opened and inspected in a browser. Additionally, the function `check_ctv_packages()` can be used to check whether some of the listed packages are actually not available on CRAN or not currently maintained (archived). The functions are illustrated below, assuming that the `Econometrics.md` file is in the local working directory:

```
ctv::ctv2html("Econometrics.md", cran = TRUE)
browseURL("Econometrics.html")
ctv::check_ctv_packages("Econometrics.md")
```

Note that the extension `.md` (rather than `.Rmd`) has been adopted for the files so that GitHub renders a Markdown preview on the fly. Finally, in case the maintainers of a task view want to leverage GitHub Actions, a `validate-ctv` workflow is provided at https://github.com/cran-task-views/ctv/tree/main/validate-ctv which runs the functions above and processes the results.

7 Available task views

Currently, there are 48 task views on CRAN with the following names and maintainers:

- *ActuarialScience* (Dutang, Goulet).
- *Agriculture* (Piaskowski, Sparks, Correndo).
- *Bayesian* (Park, Cameletti, Pang, Quinn).
- *CausalInference* (Mayer, Zhao, Greifer, Huntington-Klein, Josse).
- *ChemPhys* (Mullen).
- *ClinicalTrials* (Wang, Jaki, Harris, Doyle, Meyer, Igl).
- *Cluster* (Grün).
- *CompositionalData* (Hron, Palarea-Albaladejo, Templ, Menafoglio).
- *Databases* (Tang, Balamuta).
- *DifferentialEquations* (Petzoldt, Soetaert).
- *Distributions* (Dutang, Kiener, Swihart).
- *DynamicVisualizations* (Zhang, Cook, Lyttle).
- *Econometrics* (Zeileis, McDermott, Tappe).
- *Environmetrics* (Simpson).
- *Epidemiology* (Jombart, Rolland, Gruson).
- *ExperimentalDesign* (Groemping, Morgan-Wall).
- *ExtremeValue* (Dutang).
- *Finance* (Eddelbuettel).
- *FunctionalData* (Scheipl, Arnone, Hooker, Tucker, Wrobel).
- *GraphicalModels* (Hojsgaard).
- *HighPerformanceComputing* (Eddelbuettel).
- *Hydrology* (Albers, Prosdocimi).
- *MachineLearning* (Hothorn, Frick, Kook).

- *MedicalImaging* (Whitcher, Clayden, Muschelli).
- *MetaAnalysis* (Dewey, Viechtbauer).
- *MissingData* (Josse, Mayer, Tierney, Vialaneix).
- *MixedModels* (Bolker, Piaskowski, Tanaka, Alday, Viechtbauer).
- *ModelDeployment* (Tang, Balamuta).
- *NaturalLanguageProcessing* (Wild).
- *NetworkAnalysis* (Telarico, Krivitsky, Hollway).
- *NumericalMathematics* (Borchers, Hankin, Sokol).
- *OfficialStatistics* (Templ, Kowarik, Schoch).
- *Omics* (Aubert, Hocking, Vialaneix).
- *Optimization* (Schwendinger, Borchers).
- *Paleontology* (Gearty, Jones, Dillon, Godoy, Drage, Dean, Farina).
- *Pharmacokinetics* (Denney, Nayak).
- *Phylogenetics* (Gearty, O'Meara, Berv, Ballen, Ferreira, Lapp, Schmitz, Smith, Upham, Nations).
- *Psychometrics* (Mair, Rosseel, Gruber).
- *ReproducibleResearch* (Blischak, Hill, Marwick, Sjoberg, Landau).
- *Robust* (Maechler).
- *Spatial* (Bivand, Nowosad).
- *SpatioTemporal* (Pebesma, Bivand).
- *SportsAnalytics* (Baumer, Nguyen, Matthews).
- *Survival* (Allignol, Latouche).
- *TeachingStatistics* (Northrop).
- *TimeSeries* (Hyndman, Killick).
- *Tracking* (Joo, Basille).
- *WebTechnologies* (Sepulveda, Beasley).

8 Outlook

The new CRAN Task View Initiative has redesigned and relaunched the infrastructure and workflows for CRAN Task Views so that they can thrive in the years to come. In particular, many tools are used that are well-established in the R community such as **knitr**/**rmarkdown** or collaborations through GitHub projects. Moreover, various steps have been taken in order to assure that all task views are actively maintained and foster contributions from the community, either in terms of additions/improvements for existing task views, or in the form of new proposals.

Since announcing the new initiative in Spring 2022, many task views were already improved, e.g., by adding new content, extending the maintainer teams, or incorporating feedback from the community. Additionally, there were already twelve successful new task view proposals (ActuarialScience, Agriculture, CausalInference, CompositionalData, DynamicVisualizations, Epidemiology, MixedModels, NetworkAnalysis, Omics, Paleontology, Phylogenetics, SportsAnalytics). This shows that the review and onboarding process for the new task views works successfully. While the review and revision times are sometimes somewhat long (as the work on the task views is typically not a top priority in the jobs of either the editors or the task view authors), we feel that the reviews are very constructive, often involve other community members, and yield task views of higher quality.

While these steps already accomplished important improvements in the initiative, further challenges remain for the future. Apart from improving the breadth and depth of the task views, the most important aim is probably to better connect with those R users who would profit from the information provided in the task views. We feel that this was easier in the mid-2000s for two reasons:

1. CRAN (and hence its task views) were actively browsed/searched by many R users, whereas today many more users will expect that useful content is presented to them, e.g., via either search engines or large language models.

2. Twenty years ago, the R community was smaller, and there was a common understanding of "free" software as a contract between developers and users to actively share in the progress of a given project (as in the spirit of the "Debian Social Contract", see https://www.debian.org/social_contract, especially point 2). Since then the R community has grown in size and complexity, the typical career paths of new members have changed, and R is increasingly perceived as one among many free-of-charge software applications. But in contrast to R, most of the other free-of-charge software is actually paid for by surrendering data, not by sharing a responsibility for the progress of the project.

Both of these points have probably affected the expectations of users with regard to how much effort to apply to learning about the software they have chosen to use (but which they are not purchasing). This needs to be taken into account in further improvement of the effectiveness of task views. Therefore, one goal is to improve discoverability, including tags for search engine optimization, and prompts for chatbots based on large language models (LLMs). In terms of tags, DublinCore and Highwire Press metainformation tags have already been added to the HTML pages. In terms of LLM-based chatbots, it appears that prompts such as *"Where can I find more information on R packages for X?"*, where *X* is (closely related to) a task view topic, typically links to the corresponding CRAN Task Views.

Another goal is to better connect with those sub-communities for whom the task views are relevant. This could be accomplished by including more representatives from these sub-communities in broader teams of task view maintainers, in order to restore the open source social contract and sharing the responsibility for the task views. Moreover, listing the task views in more online overviews/tutorials geared towards these different sub-communities would help to spread the information.

We end with a call to the R community to support us in accomplishing these goals:

- If you find task views useful, pass on the information to your colleagues and students. For example by including links in tutorials or course materials or by spreading the word on your social media channels.
- If task views have been useful for you, e.g., for improving the set of tools for research projects, consider citing the relevant task views in the resulting manuscripts.
- If you have ideas for improving task views, please let us know as described in Section 4. Suggestions regarding the task view infrastructure in general are very welcome as are concrete ideas for the different existing task views.
- If you are interested in establishing a task view on a new topic in your field, recruit contributors and submit a proposal (as outlined in Section 4).

Any of the above are highly appreciated and would help us and the maintainers of CRAN Task Views to continue to aid R users and developers in making topical and informed choices about the R packages in their workflows.

References

D. Dijk. *CTVsuggest: CRAN Task View Package Recommendations*, 2023. URL https://dylandijk.github.io/CTVsuggest/. R package version 0.0.0.9000. [p8]

K. Hornik, D. Murdoch, and A. Zeileis. Who did what? The roles of R package authors and how to refer to them. *The R Journal*, 4(1):64–69, June 2012. doi:10.32614/RJ-2012-009. [p7]

K. Hornik, U. Ligges, and A. Zeileis. Changes on CRAN. *The R Journal*, 16(4):2073–4859, 2024. URL https://journal.R-project.org/news/RJ-2024-4-cran/. [p4]

J. MacFarlane. *Pandoc: A Universal Document Converter*, 2025. URL https://pandoc.org/. Version 3.6.4. [p5]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2025. URL https://www.R-project.org/. [p4]

S. Theußl and A. Zeileis. Collaborative software development using R-Forge. *The R Journal*, 1(1):9–14, May 2009. doi:10.32614/RJ-2009-007. [p4]

Y. Xie. *Dynamic Documents with R and knitr*. Chapman & Hall/CRC, Boca Raton, 2nd edition, 2015. doi:10.1201/9781315382487. URL https://yihui.org/knitr/. [p5]

Y. Xie, J. J. Allaire, and G. Grolemund. *R Markdown: The Definitive Guide*. Chapman & Hall/CRC, Boca Raton, 2018. doi:10.1201/9781138359444. URL https://bookdown.org/yihui/rmarkdown. [p5, 9]

A. Zeileis. CRAN task views. *R News*, 5(1):39–40, May 2005. URL https://CRAN.R-project.org/doc/Rnews/. [p4]

A. Zeileis and K. Hornik. *ctv: CRAN Task Views*, 2024. R package version 0.9-6. doi:10.32614/CRAN.package.ctv. [p4]

Achim Zeileis
Universität Innsbruck
Department of Statistics
https://www.zeileis.org/
ORCiD: 0000-0003-0918-3766
Achim.Zeileis@R-project.org

Roger Bivand
Norwegian School of Economics
Department of Economics
https://www.nhh.no/en/employees/faculty/roger-bivand/
ORCiD: 0000-0003-2392-6140
Roger.Bivand@R-project.org

Dirk Eddelbuettel
University of Illinois Urbana-Champaign
Department of Statistics
https://dirk.eddelbuettel.com
ORCiD: 000-0001-6419-907X
Dirk.Eddelbuettel@R-project.org

Kurt Hornik
WU Wirtschaftsuniversität Wien
Institute for Statistics and Mathematics
https://statmath.wu.ac.at/~hornik/
ORCiD: 0000-0003-4198-9911
Kurt.Hornik@R-project.org

Julia Piaskowski
University of Idaho
College of Agriculture and Life Sciences
https://jpiaskowski.gitlab.io/
ORCiD: 0000-0003-3700-0859
julia.piask@gmail.com

Nathalie Vialaneix
INRAE Toulouse
Unité MIA-T
https://www.nathalievialaneix.eu/
ORCiD: 0000-0003-1156-0639
Nathalie.Vialaneix@inrae.fr

boinet: An R Package for Bayesian Optimal Interval Design for Dose Finding Based on Both Efficacy and Toxicity Outcomes

by Yusuke Yamaguchi, Kentaro Takeda and Kazushi Maruo

Abstract Bayesian optimal interval based on both efficacy and toxicity outcomes (BOIN-ET) design is a model-assisted oncology phase I/II trial design, aiming to establish an optimal biological dose accounting for efficacy and toxicity in the framework of dose-finding. Some extensions of BOIN-ET design are also available to allow for time-to-event efficacy and toxicity outcomes based on cumulative and pending data (time-to-event BOIN-ET: TITE-BOIN-ET), ordinal graded efficacy and toxicity outcomes (generalized BOIN-ET: gBOIN-ET), and their combination (TITE-gBOIN-ET). **boinet** package is an R package to implement the BOIN-ET design family. The package supports the conduct of simulation studies to assess operating characteristics of BOIN-ET, TITE-BOIN-ET, gBOIN-ET, and TITE-gBOIN-ET designs, where users can choose design parameters in flexible and straightforward ways depending on their own application. We demonstrate its capability and effectiveness using several simulation studies.

1 Introduction

A paradigm shift in anti-cancer treatment with the emergence of molecular targeted agents and immunotherapies is calling for a reform of dose-finding strategy in phase I oncology clinical trials. Traditional dose-finding procedures designed to establish the maximum tolerated dose underlie an assumption of a monotonic dose-efficacy relationship; that is, the higher dose level is assumed to be more toxic and efficacious. However, the monotonic dose-efficacy relationship would be doubtful in many cases of the targeted agents and immunotherapies, where the efficacy may reach a plateau beyond a given dose level or even take an umbrella shape (Calabrese and Baldwin, 2001; LoRusso et al., 2010; Reynolds, 2010). Accordingly, a concept of optimal biological dose (OBD) has been introduced into the framework of dose-finding (Corbaux et al., 2019; Fraisse et al., 2021). The OBD, which is generally defined as a tolerable dose having adequate efficacy under an assumption of non-monotonic dose-efficacy relationships, is established with a dose-escalation rule based jointly on toxicity and efficacy. Numerous dose-finding designs to incorporate the trade-off between toxicity and efficacy have been proposed, broadly classified as algorithm-based designs (Lin and Ji, 2020), model-based designs (Thall and Cook, 2004; Yuan et al., 2016; Riviere et al., 2018), and model-assisted designs (Takeda et al., 2018; Yuan et al., 2019; Li et al., 2020; Lin et al., 2020; Yin and Yuan, 2020; Lin and Ji, 2021). Particularly, the model-assisted designs are known to yield superior performance compared with more complicated model-based designs and to be implemented in as simple a way as the algorithm-based designs (Yuan et al., 2019).

Bayesian optimal interval based on efficacy and toxicity outcomes (BOIN-ET) design proposed by Takeda et al. (2018) is one of the model-assisted designs and has been widely used to establish the OBD, where the dose-escalation decision is determined by minimizing a posterior probability of incorrect dose assignments in terms of both efficacy and toxicity. Furthermore, the BOIN-ET design has been extended in two ways to accommodate important clinical features unique to molecular targeted agents and immunotherapies. Firstly, the occurrence of efficacy responses and/or adverse events may be delayed due to the mechanism of the agents. Such late-onset efficacy and toxicity outcomes usually require a longer assessment window than the typical first one or two cycles for the traditional cytotoxic agents (Postel-Vinay et al., 2009; Weber et al., 2015). Ignoring the feature of late-onset toxicity

may cause potential dose reductions or interruptions in later phase trials. To allow for sequential enrollment even before patients have completed the required efficacy and toxicity assessment window, Takeda et al. (2020) proposed time-to-event BOIN-ET (TITE-BOIN-ET) design, which extends the BOIN-ET design to use cumulative and pending data of both efficacy and toxicity for the dose-escalation decision (Lin and Yuan, 2020). Secondly, the efficacy and toxicity outcomes are often summarized by binary outcomes, scored as an overall response (ORR) or not an ORR for efficacy, and scored as a dose-limiting toxicity (DLT) or not a DLT for toxicity; however, the molecular targeted agents and immunotherapies are more likely to be featured by multiple low or moderate grades of toxicities than DLTs (Penel et al., 2011). Besides, the efficacy response may prefer to be evaluated by the combination of ORR and long-term stable disease (SD) in solid tumors and by accounting for the difference between complete remission (CR) and partial remission (PR) in lymphoma. To allow for the ordinal graded efficacy and toxicity assessment, Takeda et al. (2022a) proposed generalized BOIN-ET (gBOIN-ET), which extends the BOIN-ET design to use ordinal graded efficacy and toxicity data for the dose-escalation decision. Takeda et al. (2023) proposed time-to-event generalized BOIN-ET (TITE-gBOIN-ET) to address simultaneously the aforementioned challenges of the late-onset ordinal graded efficacy and toxicity outcomes. **Keyboard** package (Yuan et al., 2025) and **escalation** package (Brock et al., 2024) are existing R packages that implement other types of the OBD-finding designs, while there are no R packages that provide functions to implement the BOIN-ET, TITE-BOIN-ET, gBOIN-ET, and TITE-gBOIN-ET design.

boinet package (Yamaguchi and Takeda, 2025) is an R package to implement the BOIN-ET design family. The package supports the conduct of simulation studies to assess operating characteristics of BOIN-ET, TITE-BOIN-ET, gBOIN-ET, and TITE-gBOIN-ET, where users can choose design parameters in flexible and straightforward ways depending on their own application. It should be noted that the selection of OBD in the BOIN-ET design family is carried out at the end of the trial after completing the dose-escalation procedure: (i) the toxicity and efficacy probabilities at each dose level are estimated using all data obtained from the entire cohort, and then (ii) given the estimated toxicity and efficacy probabilities, the utility to measure the toxicity-efficacy trade-off is computed and serves as a basis for the OBD selection. The **boinet** package offers several options regarding methods to estimate the efficacy probability and measures to select the OBD. The operating characteristics of the design are summarized by the percentage of times that each dose level is selected as OBD and the average number of patients who are treated at each dose level. The percentage of times that the study is terminated and the expected study duration are also provided. We demonstrate the capability and effectiveness of the **boinet** package.

This article is organized as follows. We first describe the BOIN-ET, TITE-BOIN-ET, gBOIN-ET, and TITE-gBOIN-ET designs implemented by the package. We then provide details on how to use the **boinet** package through simulation studies. The article is concluded with a brief discussion.

2 Case example

The BOIN-ET design family is a relatively new methodological development, and while they are being implemented in ongoing trials, published case results using these specific designs are still forthcoming. We here provide one case study that has actually implemented the TITE-BOIN-ET design.

A two-part phase I/II study evaluating a combination of luspatercept and erythropoiesis-stimulating agent (ESA) in patients with low-risk myelodysplastic syndromes without ring sideroblasts having failed to achieve a response after ESA without disease progression is being conducted (Ades et al., 2024). The primary objective of this dose-finding part of the trial was to determine the optimal dose in terms of both toxicity and efficacy for luspatercept + ESA, for selection in the randomized part B of the trial. Based on a TITE-BOIN-ET design, four dose levels were tested: Dose level 1 (n=3); Dose level 2 (n=3); Dose Level 3 (n=3); Dose

Level 4 (n=15). The toxicity was measured by a dose-limiting toxicity at Day 21 of cycle 1 for non-hematological toxicity, up to Day 42 for hematological toxicity. The efficacy was measured by a response rate (complete response + partial response + stable disease with hematological improvement), and the assessment window for efficacy was 21 days.

3 Methodology

Consider a dose-finding trial aiming to determine the OBD of a test drug, and suppose that J dose levels are investigated by sequentially assigning small cohorts of patients to the next dose levels until reaching a maximum number of patients enrolled. Suppose that patients in the current cohort have been treated at dose level j, and let n_j denote the cumulative number of patients treated at dose level j for $j = 1, \ldots, J$. Let τ_T and τ_E be the assessment window for toxicity and efficacy respectively. We below describe the dose-escalation procedure in the BOIN-ET, TITE-BOIN-ET, gBOIN-ET, and TITE-gBOIN-ET designs, followed by an introduction to the OBD selection. Further details are found in the original articles (Takeda et al., 2018, 2020, 2022b,a, 2023).

3.1 Dose-escalation procedure in BOIN-ET and TITE-BOIN-ET design

Let p_{Tj} and p_{Ej} be a true toxicity and efficacy probability at dose level j for $j = 1, \ldots, J$, respectively. Let ϕ^p and δ^p be a target toxicity and efficacy probability respectively, which are pre-specified by trial physicians according to the disease and the definition of toxicity and efficacy. Lower and upper toxicity boundaries are pre-specified as λ_1^p and λ_2^p respectively, which satisfy a condition of $0 \leq \lambda_1^p \leq \phi^p \leq \lambda_2^p < 1$. A lower efficacy boundary is also pre-specified as η_1^p with a condition of $0 \leq \eta_1^p < \delta^p < 1$. Let $\hat{p}_{Tj}$ and $\hat{p}_{Ej}$ be an observed toxicity and efficacy probability at dose level j respectively, which are calculated using the cumulative toxicity and efficacy data in different ways for the BOIN-ET and TITE-BOIN-ET design (see subsections below). Then, the dose level used in the next cohort is determined by the following rule:

1. If $\hat{p}_{Tj} \leq \lambda_1^p$ and $\hat{p}_{Ej} \leq \eta_1^p$, escalate to dose level $j+1$.
2. If $\hat{p}_{Tj} < \lambda_2^p$ and $\hat{p}_{Ej} > \eta_1^p$, stay at the same dose level j.
3. If $\hat{p}_{Tj} \geq \lambda_2^p$, de-escalate to dose level $j-1$.
4. If $\lambda_1^p < \hat{p}_{Tj} < \lambda_2^p$ and $\hat{p}_{Ej} \leq \eta_1^p$, consider the following additional rules:
   - 4-1 If dose level $j+1$ has never been used until the current cohort, escalate to dose level $j+1$.
   - 4-2 Otherwise, choose $\arg\max_{j' \in \{j-1, j, j+1\}} \hat{p}_{Ej'}$ as the dose level for the next cohort; if more than two dose levels have the same highest observed efficacy probability, randomly select one of them.

Optimal values of $(\lambda_1^p, \lambda_2^p, \eta_1^p)$ are determined as values that minimize a posterior probability of incorrect decisions under the following six hypotheses:

$$H_{1j} : (p_{Tj} = \phi_1^p, p_{Ej} = \delta_1^p), \quad H_{2j} : (p_{Tj} = \phi_1^p, p_{Ej} = \delta^p), \quad H_{3j} : (p_{Tj} = \phi^p, p_{Ej} = \delta_1^p),$$
$$H_{4j} : (p_{Tj} = \phi^p, p_{Ej} = \delta^p), \quad H_{5j} : (p_{Tj} = \phi_2^p, p_{Ej} = \delta_1^p), \quad H_{6j} : (p_{Tj} = \phi_2^p, p_{Ej} = \delta^p)$$

where ϕ_1^p is the highest toxicity probability that is deemed sub-therapeutic such that dose-escalation should be pursued, and ϕ_2^p is the lowest toxicity probability that is deemed overly toxic such that dose de-escalation is needed. δ_1^p is the minimum probability deemed efficacious such that the dose levels with less than δ_1^p are considered sub-therapeutic. Here, ϕ_1^p, ϕ_2^p, and δ_1^p are design parameters to be pre-specified along with the target probabilities of ϕ^p and δ^p. Given non-informative prior probabilities of the six hypotheses; i.e., $Pr(H_{1j}) =$

$Pr(H_{2j}) = Pr(H_{3j}) = Pr(H_{4j}) = Pr(H_{5j}) = Pr(H_{6j}) = 1/6$, a grid search approach can be used to find the optimal values of $(\lambda_1^p, \lambda_2^p, \eta_1^p)$ conditioned on $\phi_1^p < \lambda_1^p < \phi^p < \lambda_2^p < \phi_2^p$ and $\delta_2^p < \eta_2^p < \delta^p$. For example, considering $\phi^p = 0.3$, $\phi_1^p = 0.1\phi^p$, $\phi_2^p = 1.4\phi^p$, $\delta^p = 0.6$, and $\delta_1^p = 0.6\delta^p$, the optimal values of $(\lambda_1^p, \lambda_2^p, \eta_1^p)$ are given by $\lambda_1^p = 0.14$, $\lambda_2^p = 0.35$, and $\eta_1^p = 0.48$.

A key difference between the BOIN-ET and TITE-BOIN-ET design is the timing to determine the dose level used in the next cohort and to start the enrollment for the next cohort. The BOIN-ET design requires all the patients enrolled in the current cohort to complete the toxicity and efficacy assessment windows; in other words, the decision time is always the time when the last patient enrolled in the cohort completes the toxicity and efficacy assessment windows. In contrast, the TITE-BOIN-ET design allows for flexibility in the decision time, where the dose escalation decision is based on the cumulative data at a certain decision time. Differences in the decision time between the BOIN-ET and TITE-BOIN-ET designs lead to different calculations of the observed toxicity and efficacy probabilities, $\hat{p}_{Tj}$ and $\hat{p}_{Ej}$ for $i = 1, \ldots, n_j$, which are described below.

BOIN-ET (Takeda et al., 2018): Let y_{Tij} and y_{Eij} be a binary toxicity and efficacy outcome observed within the assessment windows of τ_T and τ_E for the i-th patient at dose level j, respectively. Given the cumulative toxicity and efficacy data for dose level j; i.e., $\{y_{Tij} : i = 1, \ldots, n_j\}$ and $\{y_{Eij} : i = 1, \ldots, n_j\}$, we have $\hat{p}_{Tj} = \sum_{i=1}^{n_j} y_{Tij}/n_j$ and $\hat{p}_{Ej} = \sum_{i=1}^{n_j} y_{Eij}/n_j$.

TITE-BOIN-ET (Takeda et al., 2020; Lin and Yuan, 2020): Let $\tilde{y}_{Tij}$ be a binary toxicity outcome observed at a certain decision time for the i-th patient at dose level j, where $\tilde{y}_{Tij} = 1$ if the patient has experienced the toxicity and $\tilde{y}_{Tij} = 0$ if the patient has not yet experienced the toxicity. Let γ_{Tij} denote that $\tilde{y}_{Tij}$ is ascertained ($\gamma_{Tij} = 1$) or is still pending ($\gamma_{Tij} = 0$) at the decision time, and u_{Tij} denote the time to the toxicity outcome if $\tilde{y}_{Tij} = 1$ and to censoring if $\tilde{y}_{Tij} = 0$. Note that $(\tilde{y}_{Tij}, \gamma_{Tij}) = (0, 1)$ indicates the censoring at $u_{Tij} = \tau_T$, and $(\tilde{y}_{Tij}, \gamma_{Tij}) = (0, 0)$ indicates the censoring at $u_{Tij} < \tau_T$. In the same way, $(\tilde{y}_{Eij}, \gamma_{Eij}, u_{Eij})$ is defined for efficacy data of the i-th patient at dose level j. Given the cumulative toxicity and efficacy data for dose level j at the decision time; i.e., $\{(\tilde{y}_{Tij}, \gamma_{Tij}, u_{Tij}) : i = 1, \ldots, n_j\}$ and $\{(\tilde{y}_{Eij}, \gamma_{Eij}, u_{Eij}) : i = 1, \ldots, n_j\}$, we have $\hat{p}_{Tj} = \sum_{i=1}^{n_j} \tilde{y}_{Tij}/\tilde{n}_{Tj}$ and $\hat{p}_{Ej} = \sum_{i=1}^{n_j} \tilde{y}_{Eij}/\tilde{n}_{Ej}$, where $\tilde{n}_{Tj} = \sum_{i=1}^{n_j} (\gamma_{Tij} + (1 - \gamma_{Tij}) u_{Tij}/\tau_T)$ and $\tilde{n}_{Ej} = \sum_{i=1}^{n_j} (\gamma_{Eij} + (1 - \gamma_{Eij}) u_{Eij}/\tau_E)$.

3.2 Dose-escalation procedure in gBOIN-ET and TITE-gBOIN-ET design

Yuan et al. (2007) proposed an equivalent toxicity score (ETS) to measure the relative severity of different toxicity grades in the dose allocation procedure and established the following severity weights related to DLT: (1) grade 0 and 1 toxicities are not of concern (no DLT); (2) two grade 2 toxicities are equivalent to a grade 3 toxicity (0.5 DLT); (3) a grade 3 toxicity is equivalent to a DLT (1 DLT); and (4) a grade 4 toxicity is equivalent to 1.5 grade 3 toxicities (1.5 DLT). A target ETS is obtained by a weighted sum of overall toxicity grades. Takeda et al. (2022a) extended this concept to an efficacy score named the equivalent efficacy score (EES) to measure the relative effectiveness of different efficacy outcomes and used the relative effectiveness of different efficacy outcomes: (1) progressive disease (PD) is not an effective outcome (no response); (2) an SD is equivalent to 0.25 PRs (0.25 responses); (3) a PR is equivalent to a response (1 response); and (4) a CR is equivalent to 3.0 PRs (3.0 responses). A target EES is obtained by a weighted sum of overall efficacy grades. In addition, the normalized ETS and EES are defined as $\text{ETS}^* = \text{ETS}/\text{ETS}_{max}$ and $\text{EES}^* = \text{EES}/\text{EES}_{max}$ respectively, where ETS_{max} is the ETS for the most severe toxicity grade and EES_{max} is the EES for the most desirable efficacy response. The normalized ETS and EES ranging from 0 to 1 are assumed to follow quasi-Bernoulli distributions.

Let μ_{Tj} and μ_{Ej} be a true quasi-Bernoulli toxicity and efficacy probability at dose level j for $j = 1, \ldots, J$, respectively. Let ϕ^μ and δ^μ be a target quasi-Bernoulli toxicity and efficacy

probability respectively. Lower and upper quasi-Bernoulli toxicity boundaries are pre-specified as λ_1^μ and λ_2^μ respectively, which satisfy a condition of $0 \leq \lambda_1^\mu \leq \phi^\mu \leq \lambda_2^\mu < 1$. A lower quasi-Bernoulli efficacy boundary is also pre-specified as η_1^μ with a condition of $0 \leq \eta_1^\mu < \delta^\mu < 1$. Let $\hat{\mu}_{Tj}$ and $\hat{\mu}_{Ej}$ be an observed quasi-Bernoulli toxicity and efficacy probability at dose level j respectively, which are calculated using the cumulative toxicity and efficacy data in different ways for the gBOIN-ET and TITE-gBOIN-ET design (see subsections below). Then, the dose level used in the next cohort is determined by the following rule:

1. If $\hat{\mu}_{Tj} \leq \lambda_1^\mu$ and $\hat{\mu}_{Ej} \leq \eta_1^\mu$, escalate to dose level $j+1$.
2. If $\hat{\mu}_{Tj} < \lambda_2^\mu$ and $\hat{\mu}_{Ej} > \eta_1^\mu$, stay at the same dose level j.
3. If $\hat{\mu}_{Tj} \geq \lambda_2^\mu$, de-escalate to dose level $j-1$.
4. If $\lambda_1^\mu < \hat{\mu}_{Tj} < \lambda_2^\mu$ and $\hat{\mu}_{Ej} \leq \eta_1^\mu$, consider the following additional rules:
   - 4-1 If dose level $j+1$ has never been used until the current cohort, escalate to dose level $j+1$.
   - 4-2 Otherwise, choose $\arg\max_{j' \in \{j-1, j, j+1\}} \hat{\mu}_{Ej'}$ as the dose level for the next cohort; if more than two dose levels have the same highest observed quasi-Bernoulli efficacy probability, randomly select one of them.

Optimal values of $(\lambda_1^\mu, \lambda_2^\mu, \eta_1^\mu)$ are determined as values that minimize a posterior probability of incorrect decisions under the following six hypotheses:

$$H_{1j}: (\mu_{Tj} = \phi_1^\mu, \mu_{Ej} = \delta_1^\mu), \quad H_{2j}: (\mu_{Tj} = \phi_1^\mu, \mu_{Ej} = \delta^\mu), \quad H_{3j}: (\mu_{Tj} = \phi^\mu, \mu_{Ej} = \delta_1^\mu),$$
$$H_{4j}: (\mu_{Tj} = \phi^\mu, \mu_{Ej} = \delta^\mu), \quad H_{5j}: (\mu_{Tj} = \phi_2^\mu, \mu_{Ej} = \delta_1^\mu), \quad H_{6j}: (\mu_{Tj} = \phi_2^\mu, \mu_{Ej} = \delta^\mu)$$

where ϕ_1^μ is the highest quasi-Bernoulli toxicity probability that is deemed sub-therapeutic such that dose-escalation should be pursued, and ϕ_2^μ is the lowest quasi-Bernoulli toxicity probability that is deemed overly toxic such that dose de-escalation is needed. δ_1^μ is the minimum quasi-Bernoulli probability deemed efficacious such that the dose levels with less than δ_1^μ are considered sub-therapeutic. Given non-informative prior probabilities of the six hypotheses; i.e., $Pr(H_{1j}) = Pr(H_{2j}) = Pr(H_{3j}) = Pr(H_{4j}) = Pr(H_{5j}) = Pr(H_{6j}) = 1/6$, a grid search approach can be used to find the optimal values of $(\lambda_1^\mu, \lambda_2^\mu, \eta_1^\mu)$ conditioned on $\phi_1^\mu < \lambda_1^\mu < \phi^\mu < \lambda_2^\mu < \phi_2^\mu$ and $\delta_2^\mu < \eta_2^\mu < \delta^\mu$. For example, considering $\phi^\mu = 0.313$ (which is the case of target ETS $= 0.47$ and $\text{ETS}_{max} = 1.5$), $\phi_1^\mu = 0.1\phi^\mu$, $\phi_2^\mu = 1.4\phi^\mu$, $\delta^\mu = 0.583$ (which is the case of target EES $= 1.75$ and $\text{EES}_{max} = 3.0$), and $\delta_1^\mu = 0.6\delta^\mu$, the optimal values of $(\lambda_1^\mu, \lambda_2^\mu, \eta_1^\mu)$ are given by $\lambda_1^\mu = 0.14$, $\lambda_2^\mu = 0.37$, and $\eta_1^\mu = 0.46$.

The relationship between the gBOIN-ET and TITE-gBOIN-ET designs is the same as that between the BOIN-ET and TITE-BOIN-ET designs. The gBOIN-ET design requires all the patients enrolled in the current cohort to complete the toxicity and efficacy assessment windows. In contrast, the TITE-gBOIN-ET design allows for flexibility in the decision time, where the dose escalation decision is based on the cumulative data at a certain decision time. The observed quasi-Bernoulli toxicity and efficacy probabilities, $\hat{\mu}_{Tj}$ and $\hat{\mu}_{Ej}$ for $i = 1, \ldots, n_j$, in the gBOIN-ET and TITE-gBOIN-ET design are calculated as follows.

gBOIN-ET (Takeda et al., 2022a): Let x_{Tij} and x_{Eij} be a quasi-Bernoulli toxicity and efficacy outcome (i.e., normalized ETS and EES) observed within the assessment windows of τ_T and τ_E for the i-th patient at dose level j, respectively. Given the cumulative toxicity and efficacy data for dose level j; i.e., $\{x_{Tij} : i = 1, \ldots, n_j\}$ and $\{x_{Eij} : i = 1, \ldots, n_j\}$, we have $\hat{\mu}_{Tj} = \sum_{i=1}^{n_j} x_{Tij}/n_j$ and $\hat{\mu}_{Ej} = \sum_{i=1}^{n_j} x_{Eij}/n_j$.

TITE-gBOIN-ET (Takeda et al., 2023): Let $\tilde{x}_{Tij}$ be a quasi-Bernoulli toxicity outcome (i.e., normalized ETS) observed at a certain decision time for the i-th patient at dose level j, where $\tilde{x}_{Tij} = 1$ if the patient has experienced the toxicity and $\tilde{x}_{Tij} = 0$ if the patient has not yet experienced the toxicity. Let κ_{Tij} denote that $\tilde{x}_{Tij}$ is ascertained ($\kappa_{Tij} = 1$) or is still pending ($\kappa_{Tij} = 0$) at the decision time, and v_{Tij} denote the time to the toxicity outcome if $\tilde{x}_{Tij} = 1$ and to censoring if $\tilde{x}_{Tij} = 0$. Note that $(\tilde{x}_{Tij}, \kappa_{Tij}) = (0, 1)$ indicates the censoring at $v_{Tij} = \tau_T$, and $(\tilde{x}_{Tij}, \kappa_{Tij}) = (0, 0)$ indicates the censoring at $v_{Tij} < \tau_T$. In the same way, $(\tilde{x}_{Eij}, \kappa_{Eij}, v_{Eij})$ is defined for efficacy data of the i-th patient at dose level j. Given the cumulative toxicity and efficacy data for dose level j at the decision time; i.e., $\{(\tilde{x}_{Tij}, \kappa_{Tij}, v_{Tij}) : i = 1, \ldots, n_j\}$ and $\{(\tilde{x}_{Eij}, \kappa_{Eij}, v_{Eij}) : i = 1, \ldots, n_j\}$, we have $\hat{\mu}_{Tj} = \sum_{i=1}^{n_j} \tilde{x}_{Tij} / \dot{n}_{Tj}$ and $\hat{\mu}_{Ej} = \sum_{i=1}^{n_j} \tilde{x}_{Eij} / \dot{n}_{Ej}$, where $\dot{n}_{Tj} = \sum_{i=1}^{n_j} (\kappa_{Tij} + (1 - \kappa_{Tij}) v_{Tij} / \tau_T)$ and $\dot{n}_{Ej} = \sum_{i=1}^{n_j} (\kappa_{Eij} + (1 - \kappa_{Eij}) v_{Eij} / \tau_E)$.

3.3 Optimal biological dose selection

In the BOIN-ET design family, the selection of OBD is carried out at the end of the trial after completing the dose-escalation procedure. Firstly, the (quasi-Bernoulli) toxicity and efficacy probabilities at each dose level are estimated using all data obtained from the entire cohort. Then, given the estimated (quasi-Bernoulli) toxicity and efficacy probabilities, the utility to measure the toxicity-efficacy trade-off is computed and serves as a basis for the OBD selection. We below describe the estimation method of the toxicity and efficacy probabilities and the OBD selection measures.

Estimation of (quasi-Bernoulli) toxicity probability To ensure the monotonically increasing dose-toxicity relationship, isotonic regression (Bril et al., 1984) is usually used to estimate the (quasi-Bernoulli) toxicity probability. Specifically, the pool-adjacent-violators algorithm (Barlow et al., 1972) is performed on the observed (quasi-Bernoulli) toxicity probability to obtain the estimated (quasi-Bernoulli) toxicity probability.

Estimation of (quasi-Bernoulli) efficacy probability The dose-efficacy relationship is typically unknown at the design stage and may show nonmonotonic patterns. There are three methods available to estimate the (quasi-Bernoulli) efficacy probability:

- Simple use of the observed (quasi-Bernoulli) efficacy probability
- Model averaging of multiple unimodal isotonic regressions (Lin and Yin, 2017)
- Fractional polynomial logistic regression (Takeda et al., 2018)

Measure to select the OBD Given the estimated (quasi-Bernoulli) toxicity and efficacy probabilities, there are four different measures to select the OBD. Three of them are to select a dose that maximizes a utility quantifying the toxicity-efficacy trade-off, and the other is to select an admissible dose that offers the highest estimated efficacy probability. Specifically, the OBD is selected as a dose level of:

- Maximizing utility defined by a weighted function (Lin and Yin, 2017; Zhou et al., 2019)
- Maximizing utility defined by truncated linear functions (Li et al., 2020; Lin and Ji, 2021)
- Maximizing utility defined by scoring (Lin et al., 2020)
- Having the highest (quasi-Bernoulli) efficacy probability among admissible doses (Takeda et al., 2018)

Yamaguchi et al. (2024) reported the comparative performance of several OBD selection approaches, in which users can find guidance on the choice of (i) the method to estimate the efficacy probability, and (ii) the measure to select the OBD. In general, the most straightforward measure, selecting an admissible dose with the highest efficacy, is recommended for selecting the OBD due to its simplicity, and the efficacy probability modeling would lead to limited gains in the OBD selection (Yamaguchi et al., 2024). Default options used in many functions of **boinet** package are (i) simple use of the observed (quasi-Bernoulli) efficacy probability, and (ii) having the highest (quasi-Bernoulli) efficacy probability among admissible doses.

3.4 Additional dose-escalation rules

Some additional dose-escalation rules implemented in the **boinet** package are described below. During the dose-escalation, dose skipping is not allowed from a safety viewpoint. In addition, a dose elimination rule is introduced to avoid patient assignment to inefficient or severely toxic dose levels. A dose level is eliminated from the investigation if a posterior probability that the toxicity probability is greater than the target toxicity probability is larger than a certain threshold, which is set to 0.95 as the default. Apart from this, a dose level is eliminated from the investigation if a posterior probability that the efficacy probability is less than the minimum efficacy probability is larger than a certain threshold, which is set to 0.99 as the default. When dose levels at which the next cohort patients would be assigned have been eliminated, the trial is terminated, and the patient enrollment is stopped. At the end of the trial, if all dose levels are eliminated due to these criteria, the OBD is determined as not applicable.

Other cases in which early trial termination is implemented include (i) when the decision is de-escalation from the lowest dose, and (ii) when the number of patients treated at the current dose level reaches a certain threshold, which is set to the maximum sample size as default. Moreover, when the decision is escalation from the highest dose, the stay decision is made alternatively. In addition, if no patients are allocated to some dose levels (for example, a case where the dose-escalation procedure does not reach the highest dose level), these dose levels are not included in the estimation of toxicity and efficacy probabilities.

4 Implementation

We below illustrate the four main functions from the **boinet** package: `boinet()`, `tite.boinet()`, `gboinet()`, and `tite.gboinet()` which conduct simulation studies with a particular scenario and provide operating characteristics of the BOIN-ET, TITE-BOIN-ET, gBOIN-ET, and TITE-gBOIN-ET designs, respectively. The **boinet** package is available from the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/package=boinet. Further to the conduct of simulation studies, the package offers a function, `obd.select()`, which can be used for selecting the OBD at the end of an actual trial. The illustration of this function is omitted here but it asks users to enter the estimated toxicity and efficacy probabilities for each dose level, the design parameters (e.g., target toxicity and efficacy probability), and the measure to select the OBD, returning an OBD.

As simulation study settings used commonly for the BOIN-ET, TITE-BOIN-ET, gBOIN-ET, and TITE-gBOIN-ET design, consider a dose-finding trial with six dose levels and suppose that three patients were enrolled per cohort. In particular, the trial treated three patients at the lowest dose level in the first cohort and then sequentially assigned three patients to the next dose level until reaching the maximum sample size of 36 patients (i.e., 12 cohorts at the maximum). The target (quasi-Bernoulli) toxicity and efficacy probability were set to 0.33 and 0.60 respectively. The toxicity and efficacy assessment windows were 30 and 45 days respectively, and the accrual rate (average number of days necessary to enroll 1 patient) was 10 days. These simulation study settings were given by:

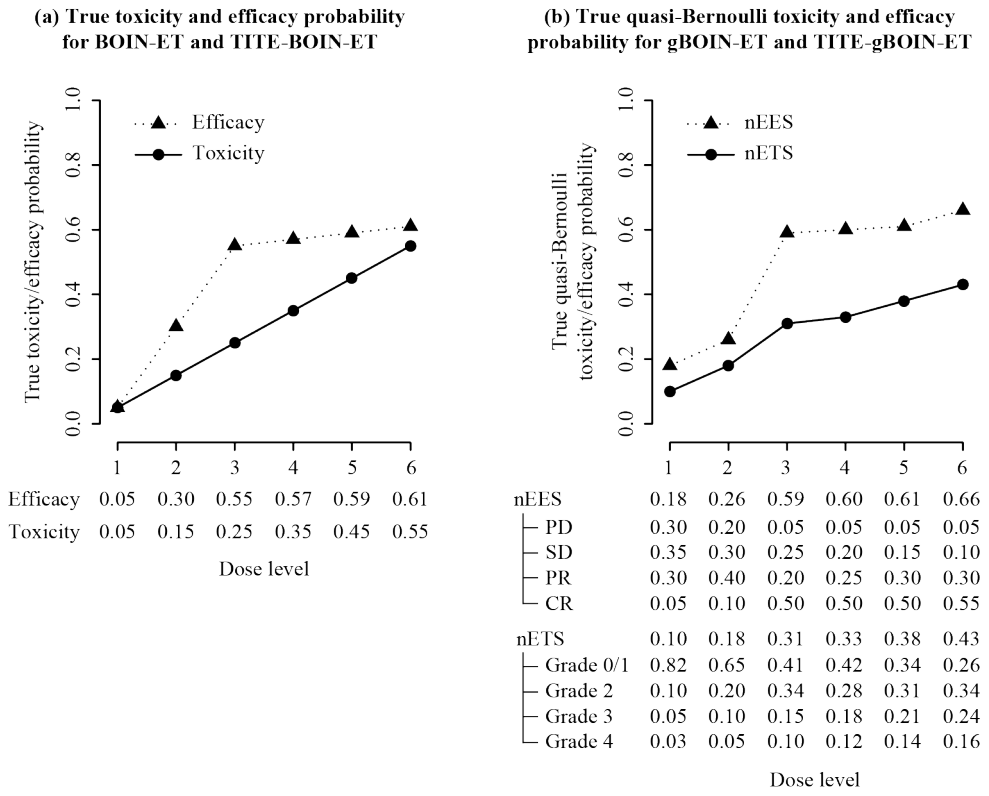


Figure 1: Dose-toxicity and dose-efficacy relationship considered in simulation study.

```
> n.dose      <- 6   # Six dose levels
> start.dose  <- 1   # Starting with the lowest dose level
> size.cohort <- 3   # Three patients assigned to each cohort
> n.cohort    <- 12  # Twelve cohorts at the maximum

> # Target (quasi-Bernoulli) toxicity and efficacy probability
> phi   <- 0.33
> delta <- 0.60
>
> tau.T   <- 30  # Thirty days of toxicity assessment windows
> tau.E   <- 45  # Forty-five days of efficacy assessment windows
> accrual <- 10  # Ten days of accrual rate
```

4.1 BOIN-ET

The `boinet()` function supports the conduct of simulation studies for the BOIN-ET design and returns a list of the summary results. The usage of the `boinet()` function is as follows:

```
boinet(
  n.dose, start.dose, size.cohort, n.cohort,
  toxprob, effprob,
  phi=0.3, phi1=phi*0.1, phi2=phi*1.4, delta=0.6, delta1=delta*0.6,
  alpha.T1=0.5, alpha.E1=0.5, tau.T, tau.E,
  te.corr=0.2, gen.event.time="weibull",
  accrual, gen.enroll.time="uniform",
  stopping.npts=size.cohort*n.cohort,
  stopping.prob.T=0.95, stopping.prob.E=0.99,
  estpt.method="obs.prob", obd.method="max.effprob",
  w1=0.33, w2=1.09,
```

```
  plow.ast=phi1, pupp.ast=phi2, qlow.ast=delta1/2, qupp.ast=delta,
  psi00=40, psi11=60,
  n.sim=1000, seed.sim=100)
```

Some of the key arguments are detailed in Table 1, 2, 3, and 4 in Appendix. Here, `toxprob` and `effprob` are vectors of true toxicity and efficacy probabilities with a length equal to the number of dose levels investigated.

We considered a scenario of the true toxicity and efficacy probabilities as given in Figure 1(a):

```
> toxprob <- c(0.05,0.15,0.25,0.35,0.45,0.55)
> effprob <- c(0.05,0.30,0.55,0.57,0.59,0.61)
```

and used (i) the observed efficacy probability for the efficacy probability estimation and (ii) the highest estimated efficacy probability as the measure to select the OBD:

```
> estpt.method <- "obs.prob"
> obd.method   <- "max.effprob"
```

Then, the `boinet()` is implemented with the previously specified arguments:

```
> boinet(
+   n.dose=n.dose, start.dose=start.dose,
+   size.cohort=size.cohort, n.cohort=n.cohort,
+   toxprob=toxprob, effprob=effprob,
+   phi=phi, delta=delta,
+   tau.T=tau.T, tau.E=tau.E, accrual=accrual,
+   estpt.method=estpt.method, obd.method=obd.method)
```

and returns the following output:

```
Simulation results:

                 Dose1  Dose2  Dose3  Dose4  Dose5  Dose6
Toxicity prob.    0.05   0.15   0.25   0.35   0.45   0.55
Efficacy prob.    0.05   0.30   0.55   0.57   0.59   0.61
No. Pts treated   3.40   7.00  15.90   7.00   2.00   0.60
Select %          1.70  12.00  54.40  25.50   5.30   0.80

No OBD %                 0.3
Trial duration (days)  778.9

Trial design settings:

Target toxicity prob.    0.330
Target efficacy prob.    0.600
Lower toxicity boundary  0.153
Upper toxicity boundary  0.390
Lower efficacy boundary  0.480

Tox. assessment window (days)  30
Eff. assessment window (days)  45
Accrual rate (days)            10

Efficacy prob. estimation: obs.prob

OBD selection: max.effprob
```

The simulation results include the number or the percentage of times that each dose level was selected as OBD (`Select %`), the average number of patients who were treated at each dose level (`No. Pts treated`), the percentage of times that the study was terminated (`Stop %`), and the expected study duration (`Trial duration (days)`). The trial design settings, such as the lower and upper toxicity/efficacy boundaries used for the dose-escalation, are also displayed. The average number of patients treated at Dose 3 was 15.90, and Dose 3 was selected as an OBD in 54.40% of the simulated trials, on average. The OBD was not selected in 0.3% of the simulated trials, on average. The trial duration was an average of 778.9 days.

4.2 TITE-BOIN-ET

The `tite.boinet()` function supports the conduct of simulation studies for the TITE-BOIN-ET design and returns a list of the summary results. The usage of the `tite.boinet()` function is as follows:

```
tite.boinet(
  n.dose, start.dose, size.cohort, n.cohort,
  toxprob, effprob,
  phi=0.3, phi1=phi*0.1, phi2=phi*1.4, delta=0.6, delta1=delta*0.6,
  alpha.T1=0.5, alpha.E1=0.5, tau.T, tau.E,
  te.corr=0.2, gen.event.time="weibull",
  accrual, gen.enroll.time="uniform",
  stopping.npts=size.cohort*n.cohort,
  stopping.prob.T=0.95, stopping.prob.E=0.99,
  estpt.method="obs.prob", obd.method="max.effprob",
  w1= 0.33, w2=1.09,
  plow.ast=phi1, pupp.ast=phi2, qlow.ast=delta1/2, qupp.ast=delta,
  psi00=40, psi11=60,
  n.sim=1000, seed.sim=100)
```

Some of the key arguments are detailed in Table 1, 2, 3, and 4 in Appendix. It should be noted that the TITE-BOIN-ET design has a suspension rule which holds off the decision on dose allocation (i.e., patient enrollment to the next cohort) until adequate information is collected, where the dose allocation is allowed only if more than 50% of patients have finished the assessment at the current dose level.

We considered a scenario of the true toxicity and efficacy probabilities as given in Figure 1(a):

```
> toxprob <- c(0.05,0.15,0.25,0.35,0.45,0.55)
> effprob <- c(0.05,0.30,0.55,0.57,0.59,0.61)
```

and used (i) the observed efficacy probability for the efficacy probability estimation and (ii) the highest estimated efficacy probability as the measure to select the OBD:

```
> estpt.method <- "obs.prob"
> obd.method   <- "max.effprob"
```

Then, the `tite.boinet()` is implemented with the previously specified arguments:

```
> tite.boinet(
+   n.dose=n.dose, start.dose=start.dose,
+   size.cohort=size.cohort, n.cohort=n.cohort,
+   toxprob=toxprob, effprob=effprob,
+   phi=phi, delta=delta,
+   tau.T=tau.T, tau.E=tau.E, accrual=accrual,
+   estpt.method=estpt.method, obd.method=obd.method)
```

and returns the following output:

```
Simulation results:

                 Dose1  Dose2  Dose3  Dose4  Dose5  Dose6
Toxicity prob.    0.05   0.15   0.25   0.35   0.45   0.55
Efficacy prob.    0.05   0.30   0.55   0.57   0.59   0.61
No. Pts treated   3.60   6.40  14.60   7.40   3.00   0.90
Select %          1.80  12.90  55.10  22.50   6.20   1.00

No OBD %                 0.5
Trial duration (days)  476.5

Trial design settings:

Target toxicity prob.    0.330
Target efficacy prob.    0.600
Lower toxicity boundary  0.153
Upper toxicity boundary  0.390
Lower efficacy boundary  0.480

Tox. assessment window (days)  30
Eff. assessment window (days)  45
Accrual rate (days)            10

Efficacy prob. estimation: obs.prob

OBD selection: max.effprob
```

The average number of patients treated at Dose 3 was 14.60, and Dose 3 was selected as an OBD in 55.10% of the simulated trials, on average. The OBD was not selected in 0.5% of the simulated trials, on average. The trial duration was an average of 476.5 days.

4.3 gBOIN-ET

The `gboinet()` function supports the conduct of simulation studies for the gBOIN-ET design and returns a list of the summary results. The usage of the `gboinet()` function is as follows:

```
gboinet(
  n.dose, start.dose, size.cohort, n.cohort,
  toxprob, effprob, sev.weight, res.weight,
  phi, phi1=phi*0.1, phi2=phi*1.4, delta, delta1=delta*0.6,
  alpha.T1=0.5, alpha.E1=0.5, tau.T, tau.E,
  te.corr=0.2, gen.event.time="weibull",
  accrual, gen.enroll.time="uniform",
  stopping.npts=size.cohort*n.cohort,
  stopping.prob.T=0.95, stopping.prob.E=0.99,
  estpt.method="obs.prob", obd.method="max.effprob",
  w1=0.33, w2=1.09,
  plow.ast=phi1, pupp.ast=phi2, qlow.ast=delta1/2, qupp.ast=delta,
  psi00=40, psi11=60,
  n.sim=1000, seed.sim=100)
```

Some of the key arguments are detailed in Table 1, 2, 3, and 4 in Appendix. Here, `toxprob` and `effprob` are matrices of true quasi-Bernoulli toxicity and efficacy probabilities with a row length equal to the number of outcome categories and a column length equal to the number of dose levels investigated. `sev.weight` and `res.weight` are weights assigned to toxicity and efficacy outcome categories respectively. For example, considering the ETS and EES, we can use `sev.weight=c(0.00,0.50,1.00,1.50)` for grade 0/1, 2, 3, and 4

toxicity and `res.weight=c(0.00,0.25,1.00,3.00)` for PD, SD, PR, and CR efficacy response, respectively

We considered a scenario of the true quasi-Bernoulli toxicity and efficacy probabilities as given in Figure 1(b):

```
> toxprob <- rbind(c(0.82,0.65,0.41,0.42,0.34,0.26),
+                  c(0.10,0.20,0.34,0.28,0.31,0.34),
+                  c(0.05,0.10,0.15,0.18,0.21,0.24),
+                  c(0.03,0.05,0.10,0.12,0.14,0.16))
>
> effprob <- rbind(c(0.30,0.20,0.05,0.05,0.05,0.05),
+                  c(0.35,0.30,0.25,0.20,0.15,0.10),
+                  c(0.30,0.40,0.20,0.25,0.30,0.30),
+                  c(0.05,0.10,0.50,0.50,0.50,0.55))
```

The weights assigned to toxicity and efficacy outcome categories were given by:

```
> sev.weight <- c(0.00,0.50,1.00,1.50)
> res.weight <- c(0.00,0.25,1.00,3.00)
```

and we used (i) the observed efficacy probability for the efficacy probability estimation and (ii) the highest estimated efficacy probability as the measure to select the OBD:

```
> estpt.method <- "obs.prob"
> obd.method   <- "max.effprob"
```

Then, the `gboinet()` is implemented with the previously specified arguments:

```
> gboinet(
+   n.dose=n.dose, start.dose=start.dose,
+   size.cohort=size.cohort, n.cohort=n.cohort,
+   toxprob=toxprob, effprob=effprob,
+   sev.weight=sev.weight, res.weight=res.weight,
+   phi=phi, delta=delta,
+   tau.T=tau.T, tau.E=tau.E, accrual=accrual,
+   estpt.method=estpt.method, obd.method=obd.method)
```

and returns the following output:

```
Simulation results:

                 Dose1  Dose2  Dose3  Dose4  Dose5  Dose6
Tox.cat1          0.82   0.65   0.41   0.42   0.34   0.26
Tox.cat2          0.10   0.20   0.34   0.28   0.31   0.34
Tox.cat3          0.05   0.10   0.15   0.18   0.21   0.24
Tox.cat4          0.03   0.05   0.10   0.12   0.14   0.16
nETS              0.10   0.18   0.31   0.33   0.38   0.43
Eff.cat1          0.30   0.20   0.05   0.05   0.05   0.05
Eff.cat2          0.35   0.30   0.25   0.20   0.15   0.10
Eff.cat3          0.30   0.40   0.20   0.25   0.30   0.30
Eff.cat4          0.05   0.10   0.50   0.50   0.50   0.55
nEES              0.18   0.26   0.59   0.60   0.61   0.66
No. Pts treated   3.80   7.20  17.60   5.40   1.70   0.30
Select %          2.90   8.20  63.10  19.50   5.50   0.70

No OBD %                 0.1
Trial duration (days)  780.1
```

```
Trial design settings:

Target toxicity prob.    0.330
Target efficacy prob.    0.600
Lower toxicity boundary  0.153
Upper toxicity boundary  0.390
Lower efficacy boundary  0.480

Tox. assessment window (days)  30
Eff. assessment window (days)  45
Accrual rate (days)            10

Efficacy prob. estimation: obs.prob

OBD selection: max.effprob
```

The average number of patients treated at Dose 3 was 17.60, and Dose 3 was selected as an OBD in 63.10% of the simulated trials, on average. The OBD was not selected in 0.1% of the simulated trials, on average. The trial duration was an average of 780.1 days.

4.4 TITE-gBOIN-ET

The `tite.gboinet()` function supports the conduct of simulation studies for the gBOIN-ET design and returns a list of the summary results. The usage of the `tite.gboinet()` function is as follows:

```
tite.gboinet(
  n.dose, start.dose, size.cohort, n.cohort,
  toxprob, effprob, sev.weight, res.weight,
  phi, phi1=phi*0.1, phi2=phi*1.4, delta, delta1=delta*0.6,
  alpha.T1=0.5, alpha.E1=0.5, tau.T, tau.E,
  te.corr=0.2, gen.event.time="weibull",
  accrual, gen.enroll.time="uniform",
  stopping.npts=size.cohort*n.cohort,
  stopping.prob.T=0.95, stopping.prob.E=0.99,
  estpt.method="obs.prob", obd.method="max.effprob",
  w1=0.33, w2=1.09,
  plow.ast=phi1, pupp.ast=phi2, qlow.ast=delta1/2, qupp.ast=delta,
  psi00=40, psi11=60,
  n.sim=1000, seed.sim=100)
```

Some of the key arguments are detailed in Table 1, 2, 3, and 4 in Appendix. It should be noted that the TITE-gBOIN-ET design has a suspension rule which holds off the decision on dose allocation (i.e., patient enrollment to the next cohort) until adequate information is collected, where the dose allocation is allowed only if more than 50% of patients have finished the assessment at the current dose level.

We considered a scenario of the true quasi-Bernoulli toxicity and efficacy probabilities as given in Figure 1(b):

```
> toxprob <- rbind(c(0.82,0.65,0.41,0.42,0.34,0.26),
+                  c(0.10,0.20,0.34,0.28,0.31,0.34),
+                  c(0.05,0.10,0.15,0.18,0.21,0.24),
+                  c(0.03,0.05,0.10,0.12,0.14,0.16))
>
> effprob <- rbind(c(0.30,0.20,0.05,0.05,0.05,0.05),
+                  c(0.35,0.30,0.25,0.20,0.15,0.10),
```

```
+                  c(0.30,0.40,0.20,0.25,0.30,0.30),
+                  c(0.05,0.10,0.50,0.50,0.50,0.55))
```

The weights assigned to toxicity and efficacy outcome categories were given by:

```
> sev.weight <- c(0.00,0.50,1.00,1.50)
> res.weight <- c(0.00,0.25,1.00,3.00)
```

and we used (i) the observed efficacy probability for the efficacy probability estimation and (ii) the highest estimated efficacy probability as the measure to select the OBD:

```
> estpt.method <- "obs.prob"
> obd.method   <- "max.effprob"
```

Then, the `tite.gboinet()` is implemented with the previously specified arguments:

```
> tite.gboinet(
+   n.dose=n.dose, start.dose=start.dose,
+   size.cohort=size.cohort, n.cohort=n.cohort,
+   toxprob=toxprob, effprob=effprob,
+   sev.weight=sev.weight, res.weight=res.weight,
+   phi=phi, delta=delta,
+   tau.T=tau.T, tau.E=tau.E, accrual=accrual,
+   estpt.method=estpt.method, obd.method=obd.method)
```

and returns the following output:

```
Simulation results:

                 Dose1  Dose2  Dose3  Dose4  Dose5  Dose6
Tox.cat1          0.82   0.65   0.41   0.42   0.34   0.26
Tox.cat2          0.10   0.20   0.34   0.28   0.31   0.34
Tox.cat3          0.05   0.10   0.15   0.18   0.21   0.24
Tox.cat4          0.03   0.05   0.10   0.12   0.14   0.16
nETS              0.10   0.18   0.31   0.33   0.38   0.43
Eff.cat1          0.30   0.20   0.05   0.05   0.05   0.05
Eff.cat2          0.35   0.30   0.25   0.20   0.15   0.10
Eff.cat3          0.30   0.40   0.20   0.25   0.30   0.30
Eff.cat4          0.05   0.10   0.50   0.50   0.50   0.55
nEES              0.18   0.26   0.59   0.60   0.61   0.66
No. Pts treated   3.90   7.20  15.70   6.20   2.40   0.60
Select %          3.00   8.10  61.00  20.70   5.90   1.20

No OBD %                0.1
Trial duration (days)  441.0

Trial design settings:

Target toxicity prob.    0.330
Target efficacy prob.    0.600
Lower toxicity boundary  0.153
Upper toxicity boundary  0.390
Lower efficacy boundary  0.480

Tox. assessment window (days)  30
Eff. assessment window (days)  45
Accrual rate (days)            10
```

```
Efficacy prob. estimation: obs.prob

OBD selection: max.effprob
```

The average number of patients treated at Dose 3 was 15.70, and Dose 3 was selected as an OBD in 61.00% of the simulated trials, on average. The OBD was not selected in 0.1% of the simulated trials, on average. The trial duration was an average of 441.0 days.

5 Summary

The concept of OBD is now widely accepted as an alternative to MTD in oncology dose-finding trials, and the BOIN-ET design family is simple and flexible enough to establish the OBD while accounting for toxicity and efficacy in the framework of dose-finding. The **boinet** package has been developed to support the conduct of simulation studies to assess operating characteristics of BOIN-ET, TITE-BOIN-ET, gBOIN-ET, and TITE-gBOIN-ET designs. Users can choose design parameters in flexible and straightforward ways depending on their own application, and several options regarding methods to estimate the efficacy probability and measures to select the OBD are applicable. The package would assist practitioners in efficiently assessing the design features of their studies.

References

L. Ades, T. Cluzeau, T. Comont, L. Aguinaga, A. Stamatoullas, M. Meunier, E. Gyan, A. Garnier, M. D'Aveni, S. Thepot, M. Sebert, M. Moldovan, A. W. Petrich, K. Lemarie, F. Chermat, M. Fontenay, S. Chevret, and P. Fenaux. Combining ESA and Luspatercept in non-RS MDS patients having failed ESA - results of the phase 1-2 part a of the GFM combola study. *blood*, 144:351–352, 2024. URL https://doi.org/10.1182/blood-2024-204791. [p16]

R. E. Barlow, D. J. Bartholomew, J. M. Bremner, and H. D. Brunk. *Statistical Inference under Order Restrictions: The Theory and Application of Isotonic Regression*. Wiley, London, 1972. [p20]

G. Bril, R. Dykstra, C. Pillers, and T. Robertson. Isotonic regression in two independent variables. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 33(3):352–357, 1984. URL https://doi.org/10.2307/2347723. [p20]

K. Brock, D. Slade, and M. Sweeting. *escalation: A Modular Approach to Dose-Finding Clinical Trials*, 2024. URL https://CRAN.R-project.org/package=escalation. R package version 0.1.10. [p16]

E. J. Calabrese and L. A. Baldwin. U-shaped dose-responses in biology, toxicology, and public health. *Annual Review of Public Health*, 22:15–33, 2001. URL https://doi.org/10.1146/annurev.publhealth.22.1.15. [p15]

P. Corbaux, M. El-Madani, M. Tod, J. Peron, D. Maillet, J. Lopez, G. Freyer, and B. You. Clinical efficacy of the optimal biological dose in earlyphase trials of anti-cancer targeted therapies. *European Journal of Cancer*, 120:40–46, 2019. URL https://doi.org/10.1016/j.ejca.2019.08.002. [p15]

J. Fraisse, D. Dinart, D. Tosi, C. Bellera, and C. Mollevi. Optimal biological dose: a systematic review in cancer phase i clinical trials. *BMC Cancer*, 21(60), 2021. URL https://doi.org/10.1186/s12885-021-07782-z. [p15]

P. Li, R. Liu, J. Lin, and Y. Ji. TEPI-2 and UBI: designs for optimal immuno-oncology and cell therapy dose finding with toxicity and efficacy. *Journal of Biopharmaceutical Statistics*, 30(6): 979–992, 2020. URL https://doi.org/10.1080/10543406.2020.1814802. [p15, 20]

R. Lin and G. Yin. STEIN: A simple toxicity and efficacy interval design for seamless phase i/ii clinical trials. *Statistics in Medicine*, 36(26):4106–4120, 2017. URL https://doi.org/10.1002/sim.7428. [p20]

R. Lin and Y. Yuan. Time-to-event model-assisted designs for dose-finding trials with delayed toxicity. *Biostatistics*, 21(4):807–824, 2020. URL https://doi.org/10.1093/biostatistics/kxz007. [p16, 18]

R. Lin, Y. Zhou, F. Yan, D. Li, and Y. Yuan. BOIN12: Bayesian optimal interval phase i/ii trial design for utility-based dose finding in immunotherapy and targeted therapies. *JCO Precision Oncology*, 4:PO.20.00257, 2020. URL https://doi.org/10.1200/PO.20.00257. [p15, 20]

X. Lin and Y. Ji. The joint i3+3 (ji3+3) design for phase i/ii adoptive cell therapy clinical trials. *Journal of Biopharmaceutical Statistics*, 30(6):993–1005, 2020. URL https://doi.org/10.1080/10543406.2020.1818250. [p15]

X. Lin and Y. Ji. Probability intervals of toxicity and efficacy design for dose-finding clinical trials in oncology. *Statistical Methods in Medical*, 30(3):843–856, 2021. URL https://doi.org/10.1177/0962280220977009. [p15, 20]

P. M. LoRusso, S. A. Boerner, and L. Seymour. An overview of the optimal planning, design, and conduct of phase i studies of new therapeutics. *Clinical Cancer Research*, 16(6):1710–1718, 2010. URL https://doi.org/10.1158/1078-0432.CCR-09-1993. [p15]

N. Penel, A. Adenis, S. Clisant, and J. Bonneterre. Nature and subjectivity of dose-limiting toxicities in contemporary phase 1 trials: comparison of cytotoxic versus non-cytotoxic drugs. *Investigational New Drugs*, 29(6):1414–1419, 2011. URL https://doi.org/10.1007/s10637-010-9490-7. [p16]

S. Postel-Vinay, H.-T. Arkenau, D. Olmos, J. E. Ang, J. Barriuso, S. Ashley, U. Banerji, J. De-Bono, I. Judson, and S. Kaye. Clinical benefit in phase-i trials of novel molecularly targeted agents: does dose matter? *British Journal of Cancer*, 100(9):1373–1378, 2009. URL https://doi.org/10.1038/sj.bjc.6605030. [p15]

A. R. Reynolds. Potential relevance of bell-shaped and u-shaped dose-responses for the therapeutic targeting of angiogenesis in cancer. *Dose Response*, 8(3):253–284, 2010. URL https://doi.org/10.2203/dose-response.09-049.Reynolds. [p15]

M.-K. Riviere, Y. Yuan, J.-H. Jourdan, F. Dubois, and S. Zohar. Phase i/ii dose-finding design for molecularly targeted agent: Plateau determination using adaptive randomization. *Statistical Methods in Medical Research*, 27(2):466–479, 2018. URL https://doi.org/10.1177/0962280216631763. [p15]

K. Takeda, M. Taguri, and S. Morita. BOIN-ET: Bayesian optimal interval design for dose finding based on both efficacy and toxicity outcomes. *Pharmaceutical Statistics*, 17(4):383–395, 2018. URL https://doi.org/10.1002/pst.1864. [p15, 17, 18, 20]

K. Takeda, S. Morita, and M. Taguri. TITE-BOIN-ET: Time-to-event Bayesian optimal interval design to accelerate dose-finding based on both efficacy and toxicity outcomes. *Pharmaceutical Statistics*, 19(3):335–349, 2020. URL https://doi.org/10.1002/pst.1995. [p16, 17, 18]

K. Takeda, S. Morita, and M. Taguri. gBOIN-ET: The generalized Bayesian optimal interval design for optimal dose-finding accounting for ordinal graded efficacy and toxicity in early clinical trials. *Biometrical Journal*, 64(7):1178–1191, 2022a. URL https://doi.org/10.1002/bimj.202100263. [p16, 17, 18, 19]

K. Takeda, Q. Xia, S. Liu, and A. Rong. TITE-gBOIN: Time-to-event Bayesian optimal interval design to accelerate dose-finding accounting for toxicity grades. *Pharmaceutical Statistics*, 21(2):496–506, 2022b. URL https://doi.org/10.1002/pst.2182. [p17]

K. Takeda, Y. Yamaguchi, M. Taguri, and S. Morita. TITE-gBOIN-ET: Time-to-event generalized Bayesian optimal interval design to accelerate dose-finding accounting for ordinal graded efficacy and toxicity outcomes. *Biometrical Journal*, 64(7):e2200265, 2023. URL https://doi.org/10.1002/bimj.202200265. [p16, 17, 20]

P. F. Thall and J. D. Cook. Dose-finding based on efficacy-toxicity trade-offs. *Biometrics*, 60 (3):684–693, 2004. URL https://doi.org/10.1111/j.0006-341X.2004.00218.x. [p15]

J. S. Weber, J. C. Yang, M. B. Atkins, and M. L. Disis. Toxicities of immunotherapy for the practitioner. *Journal of Clinical Oncology*, 33(18):2092–2099, 2015. URL https://doi.org/10.1200/JCO.2014.60.0379. [p15]

Y. Yamaguchi and K. Takeda. *boinet: Conduct Simulation Study of Bayesian Optimal Interval Design with BOIN-ET Family*, 2025. URL https://CRAN.R-project.org/package=boinet. R package version 1.3.0. [p16]

Y. Yamaguchi, K. Takeda, S. Yoshida, and K. Maruo. Optimal biological dose selection in dose-finding trials with model-assisted designs based on efficacy and toxicity: a simulation study. *Journal of Biopharmaceutical Statistics*, 34(3):379–393, 2024. URL https://doi.org/10.1080/10543406.2023.2202259. [p21]

J. Yin and Y. Yuan. Checkerboard: a Bayesian efficacy and toxicity interval design for phase i/ii dose-finding trials. *Journal of Biopharmaceutical Statistics*, 30(6):1006–1025, 2020. URL https://doi.org/10.1080/10543406.2020.1815033. [p15]

X. Yuan, C. Li, H. Sun, L. Tang, and H. Pan. *Keyboard: Bayesian Designs for Early Phase Clinical Trials*, 2025. URL https://CRAN.R-project.org/package=Keyboard. R package version 0.1.3. [p16]

Y. Yuan, R. Chappell, and H. Bailey. The continual reassessment method for multiple toxicity grades: a Bayesian quasi-likelihood approach. *Biometrics*, 63(1):173–179, 2007. URL https://doi.org/10.1111/j.1541-0420.2006.00666.x. [p18]

Y. Yuan, H. Q. Nguyen, and P. F. Thall. *Bayesian Designs for Phase I-II Clinical Trials*. Chapman and Hall/CRC, New York, 2016. [p15]

Y. Yuan, J. J. Lee, and S. G. Hilsenbeck. Model-assisted designs for early-phase clinical trials: simplicity meets superiority. *JCO Precision Oncology*, 3:PO.19.00032, 2019. URL https://ascopubs.org/doi/10.1200/PO.19.00032. [p15]

Y. Zhou, J. J. Lee, and Y. Yuan. A utility-based Bayesian optimal interval (U-BOIN) phase i/ii design to identify the optimal biological dose for targeted and immune therapies. *Statistics in Medicine*, 38(28):5299–5316, 2019. URL https://doi.org/10.1002/sim.8361. [p20]

Yusuke Yamaguchi
Quantitative Sciences and Evidence Generation, Astellas Pharma Global Development, Inc.
2375 Waterview Drive, Northbrook, Illinois 60062
United States
yusuke-yamaguchi@astellas.com

Kentaro Takeda
Quantitative Sciences and Evidence Generation, Astellas Pharma Global Development, Inc.
2375 Waterview Drive, Northbrook, Illinois 60062
United States
kentaro.takeda@astellas.com

Kazushi Maruo
Department of Biostatistics, Faculty of Medicine, University of Tsukuba

1-1-1, Tennodai, Tsukuba, Ibaraki 305-8577
Japan
kazushi.maruo@gmail.com

6 Appendix: Key arguments used in `boinet()`, `tite.boinet()`, `gboinet()`, and `tite.gboinet()` function

Argument	Description
`phi`	Numeric value between 0 and 1 specifying the target (quasi-Bernoulli) toxicity probability. Represents the maximum acceptable toxicity rate. Default is 0.3 (30%).
`phi1`	Numeric value specifying the highest (quasi-Bernoulli) toxicity probability that is deemed sub-therapeutic such that dose-escalation should be pursued. Default is phi*0.1.
`phi2`	Numeric value specifying the lowest (quasi-Bernoulli) toxicity probability that is deemed overly toxic such that dose de-escalation is needed. Default is phi*1.4.
`delta`	Numeric value between 0 and 1 specifying the target (quasi-Bernoulli) efficacy probability. Represents the desired minimum efficacy rate. Default is 0.6 (60%).
`delta1`	Numeric value specifying the minimum probability deemed efficacious such that the dose levels with efficacy < delta1 are considered sub-therapeutic. Default is delta*0.6.

Table 1: Arguments regarding target (quasi-Bernoulli) toxicity and efficacy probability.

Argument	Description
`alpha.T1`	Numeric value specifying the probability that a (quasi-Bernoulli) toxicity outcome occurs in the late half of the toxicity assessment window. Used for event time generation. Default is 0.5.
`alpha.E1`	Numeric value specifying the probability that a (quasi-Bernoulli) efficacy outcome occurs in the late half of the efficacy assessment window. Used for event time generation. Default is 0.5.
`te.corr`	Numeric value between -1 and 1 specifying the correlation between toxicity and efficacy, specified as Gaussian copula parameter. Default is 0.2 (weak positive correlation).
`gen.event.time`	Character string specifying the distribution for generating event times. Options are "weibull" (default) or "uniform". A bivariate Gaussian copula model is used to jointly generate the time to first toxicity and efficacy outcome, where the marginal distributions are set to Weibull distribution when `gen.event.time="weibull"`, and uniform distribution when `gen.event.time="uniform"`.
`gen.enroll.time`	Character string specifying the distribution for enrollment times. Options are "uniform" (default) or "exponential". Uniform distribution is used when `gen.enroll.time="uniform"`, and exponential distribution is used when `gen.enroll.time="exponential"`.

Table 2: Arguments regarding data generation.

Argument	Description
`stopping.npts`	Integer specifying the maximum number of patients per dose for early study termination. If the number of patients at the current dose reaches this criterion, the study stops the enrollment and is terminated. Default is size.cohort*n.cohort.
`stopping.prob.T`	Numeric value between 0 and 1 specifying the early study termination threshold for toxicity. If P(toxicity > phi) > stopping.prob.T, the dose levels are eliminated from the investigation. Default is 0.95.
`stopping.prob.E`	Numeric value between 0 and 1 specifying the early study termination threshold for efficacy. If P(efficacy < delta1) > stopping.prob.E, the dose levels are eliminated from the investigation. Default is 0.99.

Table 3: Arguments regarding early study termination criteria.

Argument	Description
`estpt.method`	Character string specifying the method for estimating efficacy probabilities. Options: "obs.prob" (observed efficacy probabilities/rates), "fp.logistic" (fractional polynomial), or "multi.iso" (model averaging of multiple unimodal isotonic regression). Default is "obs.prob".
`obd.method`	Character string specifying the method for OBD selection. Options: "utility.weighted", "utility.truncated.linear", "utility.scoring", or "max.effprob" (default).
`w1`	Numeric value specifying the weight for toxicity-efficacy trade-off in "utility.weighted" method. Default is 0.33.
`w2`	Numeric value specifying the penalty weight for toxic doses in "utility.weighted" method. Default is 1.09.
`plow.ast`	Numeric value specifying the lower toxicity threshold for "utility.truncated.linear" method. Default is phi1.
`pupp.ast`	Numeric value specifying the upper toxicity threshold for "utility.truncated.linear" method. Default is phi2.
`qlow.ast`	Numeric value specifying the lower efficacy threshold for "utility.truncated.linear" method. Default is delta1/2.
`qupp.ast`	Numeric value specifying the upper efficacy threshold for "utility.truncated.linear" method. Default is delta.
`psi00`	Numeric value specifying the utility score for (toxicity=no, efficacy=no) in "utility.scoring" method. Default is 40.
`psi11`	Numeric value specifying the utility score for (toxicity=yes, efficacy=yes) in "utility.scoring" method. Default is 60.

Table 4: Arguments regarding optimal biological dose (OBD) selection.

Partitioned Local Depth (PaLD) Community Analyses in R

by Lucy D'Agostino McGowan, Katherine Moore, and Kenneth S. Berenhaut

Abstract Partitioned Local Depth (PaLD) is a framework for holistic consideration of community structure for distance-based data. This paper describes an R package, **pald**, for calculating Partitioned Local Depth (PaLD) probabilities, implementing community analyses, determining community clusters, and creating data visualizations to display community structure. We present essentials of the PaLD approach, describe how to use the **pald** package, walk through several examples, and discuss the method in relation to commonly used techniques.

0.1 Introduction

Partitioned Local Depth (PaLD) is a framework for holistic consideration of community structure for distance-based data. Leveraging a socially inspired perspective, the approach provides network-based community information which is founded on new measures of local depth and pairwise cohesion (partitioned local depth). The method does not require distributional assumptions, optimization criteria, nor extraneous inputs. A complete description of the perspective, together with a discussion of the underlying social motivation, theoretical results, and applications to additional data sets is provided in Berenhaut et al. (2022). A brief technical description is included below, directly following the introduction.

As suggested in Berenhaut et al. (2022), a main goal of PaLD is to "transform input dissimilarity comparisons into output pairwise relationship strengths (or cohesion) and resulting weighted networks". This is intended to provide within- and between-community structural information which goes beyond simple cluster labeling. Building on existing approaches to (global) depth, local depth expresses features of centrality via an interpretable probability which is free of parameters and robust to outliers. Partitioning the probabilities which define local depth, we then obtain a measure of cohesion between pairs of individuals. Both local depth and cohesion reflect aspects of relative position (rather than absolute distance) and provide a straightforward way to account for varying density across the space. As shown in Berenhaut et al. (2022), provided that two sets are separated (in the sense that the minimum between-set distance is greater than the maximum within-set distance), cohesion is invariant under the contraction and dilation of the distances within each set. This property may be particularly valuable when one has reason to believe that there is heterogeneity in density across the space.

As cohesion captures a sense of the relationship strength between points, we can then analyze and visualize the resulting community structure via a network whose edges are weighted by (mutual) cohesion. The underlying social framework motivates a straightforward yet elegant threshold for distinguishing between strongly and weakly cohesive pairs.

Networks obtained from cohesion can be displayed using a force-directed graph drawing algorithm; here we will graphically emphasize the strong ties (colored by connected component). We refer to the connected components of the network of strong ties as community "clusters". Note that to qualify as a cluster in this definition, one may not have any strong ties with those outside the cluster, and thus the existence of disjoint groups is a strong signal for separation. Here, clusters are identified without additional user inputs nor optimization criteria. If one wishes to further break the community graph into groups, one may consider using community detection methods (such as spectral clustering or the Louvain algorithm), as available, say, in the **igraph** package. The collection of strong ties may be used in place of (weighted) k-nearest neighbors in settings such as classification and smoothing. Overall, the structural information obtained from local depth, cohesion and community graphs can provide a holistic perspective to the data which does not require the use of distributional

assumptions, optimization criteria nor additional user inputs.

It is important to emphasize at the outset that community analyses go beyond simple cluster labeling to address intra- and inter-cluster structure, and can supplement results from other methods for clustering, embedding, data depth, nearest neighbors, etc. In addition, the method does not require distributional assumptions, optimization criteria, nor extraneous inputs. Theoretical considerations (see Berenhaut et al. (2022)) and examples point to distinctive properties that assist in considerations of complex data, in particular with respect to varying density and high dimensions. See the section Cultural and Psychological distance analysis, below, for some discussion in the context of density-based methods.

After a brief introduction to the partitioned local depth approach (including a concrete and instructive example), below, we present a new package, **pald**, for calculating partitioned local depths, implementing community analyses, and creating data visualizations to display community structure. This paper describes how to use the package, walks through several examples, and contrasts the method results with commonly used techniques. Together, these demonstrate both the novelty of the method and utility of the implementation in the package described.

0.2 The partitioned local depth approach

The PaLD methodology as introduced in Berenhaut et al. (2022) offers a parameter-free approach to analyzing community structure in distance-based data. First, consider a ground set $\mathcal{S}$ equipped with a meaningful measure of pairwise distance (or dissimilarity), $d : \mathcal{S} \times S \to \mathbb{R} \cup \infty$. We require only that d satisfies the requirements that for all $x, y \in S$ with $y \neq x$,

$$d(x,x) \leq d(y,x) \quad \text{and} \quad d(x,x) < d(x,y). \tag{1}$$

Note that d need not necessarily be symmetric nor positive (see (i) below).

We begin with a concept formalizing locality to a pair. For any pair $(x, y) \in \mathcal{S} \times \mathcal{S}$, the *local focus*, $U_{x,y}$ (illustrated in Figure 1 for two-dimensional Euclidean data) is defined via

$$U_{x,y} \stackrel{\text{def}}{=} \{z \in S \mid d(z,x) \leq d(y,x) \text{ or } d(z,y) \leq d(x,y)\}. \tag{2}$$

The set $U_{x,y}$ is comprised of elements that are "locally" relevant to the pair (x, y). Eq. (1) guarantees that both x and y are elements of $U_{x,y}$. For discussion from a social perspective, wherein the data are embedded in a latent social space, see Social Framework in Berenhaut et al. (2022). Recall that in (2), the distance d need not be symmetric; leveraging the social perspective, we are interested in the direct closeness of z *to* x and y, not the reverse (see also (iv), below).

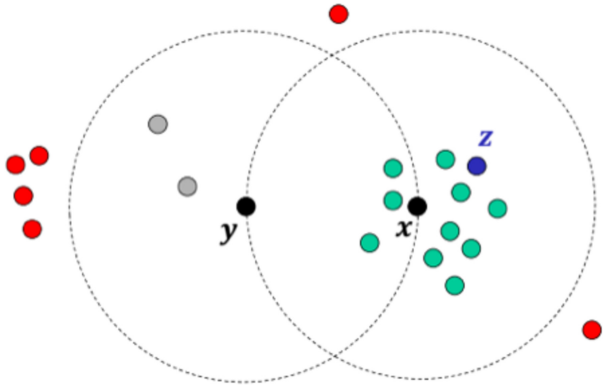


Figure 1: The local focus for two points, x and y, where $\mathcal{S}$ is a subset of $\mathbb{R}^2$, under Euclidean distance (reproduced with permission from Berenhaut et al. (2024)). The points in red are outside the focus, $U_{x,y}$. Those in green (and z in blue) are in the focus and closer to x, while those in gray are closer to y.

Now, suppose that x is fixed, and select Y and Z uniformly at random from $\mathcal{S} \setminus \{x\}$ and $U_{x,Y}$, respectively. The *local depth* of x, $\ell_{\mathcal{S}}(x)$, is defined as the probability that Z is closer to x than it is to Y (with a coin flip in the case Z is equidistant), i.e.

$$\ell_{\mathcal{S}}(x) \stackrel{\text{def}}{=} P(d(Z,x) < d(Z,Y)). \tag{3}$$

The local depth of x can be thought of as a measure of the extent to which x is (relatively) surrounded by other elements of $\mathcal{S}$ (under d).

Finally, the *cohesion*, $C_{x,w}$, of w to x, for a given $w \in \mathcal{S}$, is obtained through a partitioning of the probability defining local depth in (3). In particular,

$$C_{x,w} \stackrel{\text{def}}{=} P(Z = w \text{ and } d(Z,x) < d(Z,Y)). \tag{4}$$

The cohesion of w to x can be viewed as the contribution of w to the local depth of x. Eq. (1) guarantees that when $Z = x$, $d(Z,x) < d(Z,Y)$. For some theoretical properties of cohesion, see Berenhaut et al. (2022). Note that the sum of all cohesions is conserved at $n/2$, where $n \stackrel{\text{def}}{=} |\mathcal{S}|$; we thus have a constant mean local depth of 0.5 (independent of n).

A universal threshold for strong cohesion (see Berenhaut et al. (2022)) is defined via

$$T \stackrel{\text{def}}{=} P(Z = W \text{ and } d(Z,X) < d(Z,Y)), \tag{5}$$

where X, Y, Z, and W are selected uniformly at random from $\mathcal{S}$, $\mathcal{S} \setminus X$, $U_{X,Y}$, and $U_{X,Y}$, respectively. The threshold, T, can be conveniently calculated as one half the average of the diagonal of the matrix of pairwise cohesion values (see Berenhaut et al. (2022) for details), i.e.

$$T = \frac{1}{2}\left(\frac{1}{n}\sum_{x \in \mathcal{S}} C_{x,x}\right). \tag{6}$$

A strong relationship connection between x and w is then established when:

$$\min\{C_{x,w}, C_{w,x}\} \geq T. \tag{7}$$

For further details on PaLD, including theoretical results, discussion of the underlying social perspective and applications, see Berenhaut et al. (2022) (see also Berenhaut and Moore (2022)).

Before moving on to a simple concrete example, we provide a few remarks, for the

interested reader, on the concepts introduced above.

(i) (Distances and dissimilarities) For the purposes of the package, we require only the inequalities in (1). This implies the convenient form of the threshold in (7), see Berenhaut et al. (2022), and is crucial to the underlying social perspective of conflict. For a more general mathematical PaLD framework, wherein the concepts of locality and support are further refined see Berenhaut et al. (2024). As the definitions in (2), (3) and (4) only depend on triplet distance comparisons, local depth and cohesion are invariant under monotone transformations of distance, and negative distances are fine. If meaningful in context, distances need not be symmetric and self-distances, $d(x, x)$, may vary over x (subject to the constraint in (1)). The triangle inequality may be violated to some extent, mirroring ideas of strong triadic closure (Granovetter (1973)).

(ii) (Local foci) The concept of local foci in (2) is crucial to all that follows. It allows for consideration of the data at varying scales without the need for localizing parameters. The selection of Z in (3) and (4) reflects the larger presence of individuals in smaller foci (see Feld (1981) for discussion is social settings). The results in Berenhaut et al. (2022) regarding limiting irrelevance of density and separation under increasing concentration are driven by local comparisons. Self-cohesions (i.e., diagonal elements of C) are not necessarily equal. This is driven by the fact that by (4),

$$C_{x,x} = \frac{1}{n-1} \sum_{y \neq x} \frac{1}{|U_{x,y}|}, \tag{8}$$

and $C_{x,x}$ is dependent on local density around x.

(iii) (Non-monotonicity of the size of local foci) As distance away from an element, x, increases, monotonicity in volume of local foci is not necessary. In particular, consider the one-dimensional set $\mathcal{S} = \{B, x, A, C, D\}$, with $B = 5$, $x = 11$, $A = 16$, $C = 18$, $D = 19$. Under Euclidean distance, $d(x, A) = 5 < 6 = d(x, B)$ but $|U_{x,A}| = |\{x, A, C, D\}| = 4$ and $|U_{x,B}| = |\{x, B, A\}| = 3$. This further emphasizes the capturing of local density, an important feature of PaLD.

(iv) (Assymmetry) Note that the definitions in (2), (3) and (4) are stated to allow for asymmetric distances, as may occur for instance when considering distances on graphs. Note that even symmetric distances can easily lead to asymmetric cohesion. This is the case even for the simple one-dimensional data considered in Example 1. In considering strong relationships based on cohesion, though, symmetrization is employed in (7).

(v) (The use of weighted networks) In what follows, we will at times have occasion to consider a network with node set $\mathcal{S}$ and pairwise edges weighted via the respective values of cohesion. When viewed as a weighted (connectivity) network in this sense,

   (a) the weighted nodal degrees are the values of local depth, addressing considerations of (local) data depth,

   (b) the network may be embedded in low-dimensional Euclidean space via standard network embedding techniques, providing visualization of the data that adapts to relative density,

   (c) the isolated nodes of either the original network or that restricted to strong ties can provide information regarding outliers,

   (d) the connected components of the network restricted to strong ties (i.e. those with weights above the threshold in (6)) can be taken as “clusters" in the classical sense,

   (e) weighted or unweighted network neighbors may be used in place of k-nearest neighbors in settings such as classification and smoothing, and

   (f) some network analyses can be informative regarding data structure.

Ideas of distance on networks can be complex, though, and this is particularly so for general weighted networks (e.g. correlation or social networks). As is the case here, edge weights may reflect intensity of connection as opposed to simple pairwise distance, and it is important to exercise caution when applying naïve functions intended for unweighted networks (or simple distance-based weighted networks) in **igraph** and other software.

(vi) (Matrix methods) For convenience, we use the **igraph** package to analyze weighted networks, where appropriate. Matrix-based methods are also possible. For instance, we employ `igraph::components` to obtain community clusters from networks, but alternatively one could use blocks extraction directly from the cohesion matrix using packages such as **lintools**.

(vii) (Density-based methods and localized approaches) Complimentary density-based methods such as DBSCAN (density-based spatial clustering of applications with noise) and its hierarchical variant HDBSCAN (see for instance Campello et al. (2020)), seek to identify high-density regions as clusters. We discuss these in the context of a concrete example below (see Cultural and Psychological distance analysis). Determination of required parameter values can be challenging. There are also extent methods for considering data depth which probe local structure (see Paindaveine and Van Bever (2013) and Agostinelli and Romanazzi (2011)). These often also require localizing parameters. In Berenhaut et al. (2022) (see Theorem 2: Limiting Irrelevance of Density), it is shown that local depth and cohesion account for varying density in the sense that, provided subsets are sufficiently separated, the cohesion is maintained, as within-subset distances are contracted or dilated, without the need to search over a parameter space.

(viii) (Social latent spaces) The definitions in (2), (3), (4), and (6), are developed from a social perspective. The interested reader can refer to the section *Social Framework* in Berenhaut et al. (2022). For further discussion see Berenhaut et al. (2024) and Berenhaut and Moore (2022).

Example 1. As a concrete example demonstrating the PaLD framework, consider the one-dimensional set $\mathcal{S} \overset{\text{def}}{=} \{1, 3, 7, 8, 9, 13, 17\}$, with pairwise (Euclidean) distances given in the array below.

```
    1  3  7 8 9 13 17
1   0  2  6 7 8 12 16
3   2  0  4 5 6 10 14
7   6  4  0 1 2  6 10
8   7  5  1 0 1  5  9
9   8  6  2 1 0  4  8
13 12 10  6 5 4  0  4
17 16 14 10 9 8  4  0
```

The local depths (rounded to three decimal places) are

```
    1     3     7     8     9    13    17
0.409 0.463 0.588 0.675 0.600 0.471 0.294
```

while the pairwise cohesions (in matrix form) are given by

```
       1     3     7     8     9    13    17
1  0.210 0.127 0.036 0.024 0.012 0.000 0.000
3  0.137 0.220 0.048 0.036 0.024 0.000 0.000
7  0.048 0.048 0.220 0.137 0.109 0.028 0.000
8  0.024 0.052 0.163 0.218 0.163 0.056 0.000
```

```
9  0.024 0.024 0.109 0.137 0.220 0.063 0.024
13 0.000 0.000 0.012 0.036 0.103 0.188 0.133
17 0.000 0.000 0.000 0.000 0.012 0.107 0.175
```

where (non-diagonal) cohesion values above the threshold of

$$0.1036 = (1/14)(0.210 + 0.220 + 0.220 + 0.218 + 0.220 + 0.188 + 0.175) \tag{9}$$

are indicated in blue; see (7)). The resulting weighted network of pairwise cohesion values is displayed in Figure 2; here strong ties (with cohesion values above the threshold) are colored according to community clusters (the connected components of the network of strong ties), i.e. $\{1,3\}$, $\{7,8,9\}$ and $\{13,17\}$.

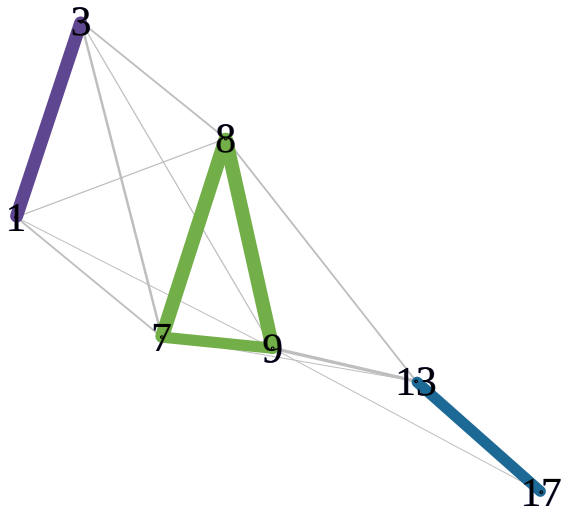


Figure 2: The community cluster network for the simple one-dimensional data considered in Example 1. Strong ties (with cohesion values above the threshold) are colored according to community clusters (the connected components of the network of strong ties), while weak ties are indicated in grey.

Note that the cohesion of the element 17 to the element 1 is zero. This reflects the fact that when an element, Y, other than 1 is selected uniformly at random from $\mathcal{S}$, and an element Z is selected uniformly at random from $U_{x,Y}$, 17 is never, at once, equal to Z and closer to 1 than it is to the opposing element Y. In fact, 17 is not an element of $U_{1,y}$, for $y \in \{3,7,8\}$. For $y \in \{9,13,17\}$, $17 \in U_{1,y}$, but in each case, $d(17,y) < d(17,1)$. This zero cohesion is reflected in the absence of any edge (grey or colored) from the node for 17 to the node for 1, in Figure 2. □

We now turn to discussion of the **pald** package.

0.3 pald

The main functions in the **pald** package can be split into 3 categories:

1. A function for computing the matrix of cohesion values, as defined in (4).
2. Functions for extracting useful information from the cohesion matrix, such as local depths, neighbors, community clusters, and graph objects.
3. Plotting functions for community graphs.

In addition, the package provides a number of pertinent example data sets which may be used to demonstrate community analysis, including a synthetic data set of two-dimensional

points created by Gionis et al. to demonstrate aggregation, clustering data generated from the scikit-learn Python package (Pedregosa et al., 2011), data describing cognate relationships between words across 87 Indo-European languages (Dyen et al., 1992), data compiled by Love and Irizarry (2015) of tissue gene expressions, data employing information from the World Values Survey (Inglehart et al., 2014) on cultural values regarding family, religion, education, and institutions for several regions (Muthukrishna et al., 2020), and three example data sets generated for the Berenhaut et al. (2022) paper.

While it is not a necessity, the **pald** package is designed to function well with the pipe operator, `|>`, particularly for those who are familiar with tidyverse-based approaches. This functionality will be demonstrated briefly below.

Creating the cohesion matrix

For the purposes of the **pald** package, the sole input for the Partitioned Local Depth (PaLD) computations is a distance matrix or `dist` object. Recall that the collection of input distances (or dissimilarities) is assumed to satisfy the requirements in (1). More generally, the method only requires triplet distance comparisons, as opposed to exact numeric distances (see Berenhaut et al. (2022)).

For demonstration purposes, we first show how one can compute a distance matrix from an input data frame with, say, two variables `x1` and `x2`. The input data may be of any dimension; in fact the PaLD framework provides advantages when considering high-dimensional data (see the **Examples** section as well as Berenhaut et al. (2022)).

```
library(pald)
df <- data.frame(
  x1 = c(6, 8, 8, 16, 4, 14),
  x2 = c(5, 4, 10, 8, 4, 10)
)
rownames(df) <- c("A", "B", "C", "D", "E", "F")
```

The `dist` function returns a (default Euclidean) pairwise distance matrix for an input data frame, as demonstrated below. If the data are already provided as a distance matrix (or `dist` object), the user can skip to the next step. Note that the distance matrix needed for the subsequent functions does not need to be a `dist` object and *need not* be symmetric.

```
d <- dist(df)
```

The function above creates a `dist` object. If converted to a matrix, this will be an $n \times n$ distance matrix, where n corresponds to the number of observations in the original data frame (in this example $n = 6$).

This `dist` object, or a distance matrix, can then be passed to the `cohesion_matrix` function in order to calculate pairwise cohesion values.

```
cohesion_matrix(d)
```

```
#>            A          B          C         D          E         F
#> A 0.25000000 0.18333333 0.06666667 0.0000000 0.18333333 0.0000000
#> B 0.14000000 0.24000000 0.05000000 0.0000000 0.10666667 0.0000000
#> C 0.07333333 0.07333333 0.20333333 0.0000000 0.03333333 0.0800000
#> D 0.00000000 0.00000000 0.00000000 0.2333333 0.00000000 0.1333333
#> E 0.14000000 0.10666667 0.03333333 0.0000000 0.24000000 0.0000000
#> F 0.00000000 0.00000000 0.05000000 0.1400000 0.00000000 0.2400000
#> attr(,"class")
#> [1] "cohesion_matrix" "matrix"          "array"
```

Equivalently, the user can use the native pipe `|>` as follows.

```
df |>
  dist() |>
  cohesion_matrix()
```

```
#>            A          B          C         D          E         F
#> A 0.25000000 0.18333333 0.06666667 0.0000000 0.18333333 0.0000000
#> B 0.14000000 0.24000000 0.05000000 0.0000000 0.10666667 0.0000000
#> C 0.07333333 0.07333333 0.20333333 0.0000000 0.03333333 0.0800000
#> D 0.00000000 0.00000000 0.00000000 0.2333333 0.00000000 0.1333333
#> E 0.14000000 0.10666667 0.03333333 0.0000000 0.24000000 0.0000000
#> F 0.00000000 0.00000000 0.05000000 0.1400000 0.00000000 0.2400000
#> attr(,"class")
#> [1] "cohesion_matrix" "matrix"          "array"
```

The *cohesion matrix* output by the `cohesion_matrix` function is the main input for the majority of the remaining functions.

Functions for extracting information from the cohesion matrix

From the *cohesion matrix*, a variety of useful quantities can be computed. Below, we create a cohesion matrix using the functions described in the previous section.

```
df |>
  dist() |>
  cohesion_matrix() -> cohesion
```

The `local_depths` function calculates the *depth* of each point, as defined in (3), outputting a vector of local depth probabilities.

```
local_depths(cohesion)
```

```
#>         A         B         C         D         E         F
#> 0.6833333 0.5366667 0.4633333 0.3666667 0.5200000 0.4300000
```

In this case, the (locally) deepest point is `A`.

The `strong_threshold` function will calculate the cohesion threshold for strong ties given in (5), which reflects typical cohesion for local points (see Berenhaut et al. (2022)). Computationally, this is equal to half the average of the diagonal of the cohesion matrix, and is a threshold that may be used to distinguish between strong and weak ties.

```
strong_threshold(cohesion)
```

```
#> [1] 0.1172222
```

Here, the threshold is a little above `0.117`.

The function `cohesion_strong` will update the cohesion matrix to set all weak ties to zero (via the `strong_threshold` function). Reflecting (7), by default, the matrix will also be symmetrized, using the entry-wise (parallel) minimum of the cohesion matrix and its transpose.

```
cohesion_strong(cohesion)
```

```
#>      A    B         C         D    E         F
#> A 0.25 0.14 0.0000000 0.0000000 0.14 0.0000000
#> B 0.14 0.24 0.0000000 0.0000000 0.00 0.0000000
#> C 0.00 0.00 0.2033333 0.0000000 0.00 0.0000000
```

```
#> D 0.00 0.00 0.0000000 0.2333333 0.00 0.1333333
#> E 0.14 0.00 0.0000000 0.0000000 0.24 0.0000000
#> F 0.00 0.00 0.0000000 0.1333333 0.00 0.2400000
#> attr(,"class")
#> [1] "cohesion_matrix" "matrix"          "array"
```

The `community_graphs` function takes the cohesion matrix and creates **igraph** objects, graphs that describe the (symmetrized) relationship structure between points. This function will output a list of three objects:

- `G`: the weighted (community) graph whose edge weights are mutual cohesion
- `G_strong`: the weighted (community) graph consisting of edges for which mutual (symmetrized) cohesion (i.e. the minimum of the two directed cohesion values for any given pair) is greater than the threshold for strong ties
- `layout`: the graph layout. By default this is provided by the Fruchterman Reingold (FR) force-directed graph drawing algorithm for the graph `G`, as implemented in the **igraph** package.

```
graphs <- community_graphs(cohesion)
graphs[["G_strong"]]
```

```
#> IGRAPH c4d81a7 UNW- 6 3 --
#> + attr: name (v/c), weight (e/n)
#> + edges from c4d81a7 (vertex names):
#> [1] A--B A--E D--F
```

Here we see that there are three connected components, ties `A-B` and `A-E` form the first community cluster, and the tie `D-F` which forms another.

The `any_isolated()` function will check whether there are any isolated points (according to cohesion).

```
any_isolated(cohesion)
```

Here, there are no isolated points, i.e. points having zero cohesion with all other points in the data (an extreme form of outlier).

The "community clusters" identified by PaLD are the connected components of the graph of strong ties, `G_strong`. To directly calculate these, we can use the `community_clusters` function. This will output a data frame with two columns; the first corresponds to the individual (`point`), as identified by the row name of the original input data frame, `df`, the second identifies the `community` that the individual belongs to.

```
community_clusters(cohesion)
```

```
#>   point community
#> A     A         1
#> B     B         1
#> C     C         2
#> D     D         3
#> E     E         1
#> F     F         3
```

In this example, three communities are identified with these six points. Points `A`, `B`, and `E` fall into Community 1. Point `C` is in Community 2 (a community of size 1) and points `D` and `F` fall into Community 3.

0.4 Plotting functions

The final category of function is that for data visualization. We can begin by visualizing the points in the data frame `df` (Figure 3). When visualizing these points, it is important to have the aspect ratio of the x and y axes equal to 1 so as to not distort distances. When using the **ggplot2** package for this visualization, one can use the command `coord_fixed(ratio = 1)`. If using the `plot` function included in the base library, one can use the `asp = 1` argument.

```
library(ggplot2)
ggplot(df, aes(x1, x2)) +
  geom_text(label = rownames(df)) +
  coord_fixed(ratio = 1) +
  xlim(c(4, 16)) +
  ylim(c(4, 16))
```

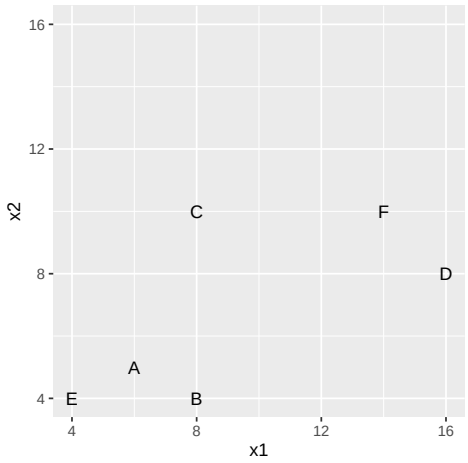


Figure 3: Visualization of the points from the data frame `df`

We can pass the cohesion matrix to the `plot_community_graphs` function to view the relationship between points (Figure 4). The function will also permit parameters that can be passed to `plot.igraph` via the `...` argument.

```
plot_community_graphs(cohesion,
                      vertex.label.cex = 2,
                      vertex.label.dist = 0.9)
```

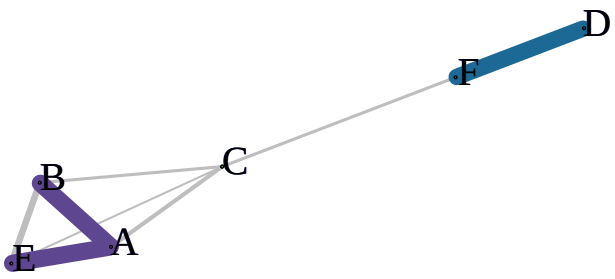


Figure 4: PaLD graph displaying the relationship between the points in data frame `df`

Notice in this plot the force-directed layout does not match that of the original data frame as seen in Figure 3. Since our original data is two-dimensional, it may be reasonable

to use the latter as the layout for plotting. Figure 5 includes this update as well as others addressing some of the aesthetics, such as employing more readable labels. The `layout` argument allows the user to pass a matrix to dictate the 2-dimensional layout of the graph. For example, if we wanted the graph to match the visualization displayed in Figure 3, we could pass `as.matrix(df)` (a matrix of the data frame `df`) to the `layout` argument (Figure 5). Here, we increase the vertex size and change the vertex label color, through the argument specifications `vertex.size = 100` and `vertex.label.color = "white"`. Additionally, to allow axes, we use `axes = TRUE`, and to put these back on the original scale we set `rescale = FALSE`, resetting the axis limits using `xlim` and `ylim`. The `par(pty = "s")` function forces the subsequent plot to be square.

```
par(pty = "s")

plot_community_graphs(cohesion,
                      layout = as.matrix(df),
                      vertex.size = 100,
                      vertex.label.color = "white",
                      axes = TRUE,
                      rescale = FALSE,
                      asp = 1,
                      xlim = c(4, 16),
                      ylim = c(4, 16))
```

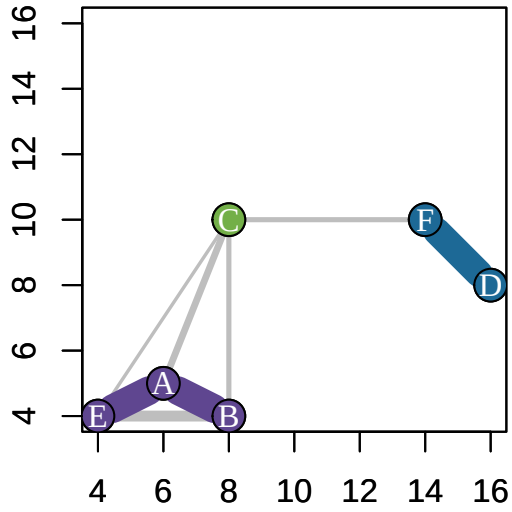


Figure 5: PaLD graph displaying the relationship between the points in data frame `df`, matching the original layout in Figure 3

0.5 Examples

We will demonstrate the utility of the **pald** package through several illustrative examples.

Community analysis for tissue gene expression data

The first example is from a subset of data from Zilliox and Irizarry (2007), McCall et al. (2011), and McCall et al. (2014), obtained from the **tissuesGeneExpression** bioconductor package (Love and Irizarry, 2015) consisting of 22,215-dimensional gene expression data from 189 tissue samples. A (Euclidean) `dist` object was created using this data set and is included in the **pald** package in an object called `tissue_dist`.

The `tissue_dist` object is a `dist` object resulting in a distance matrix with 189 rows and 189 columns.

We can create the cohesion matrix using the `cohesion_matrix` function.

```
tissue_cohesion <- cohesion_matrix(tissue_dist)
```

We can display relationships between tissue samples, both locally and globally through the `plot_community_graphs` function (Figure 6). For clarity of the display, we show how to remove the labels using `show_labels = FALSE`. We will instead color according to the labels by passing these to the `vertex.color` argument for the `plot.igraph` function (via the `...` argument). Similarly, we can add a legend using the `legend()` function, as you would for an **igraph** visualization. Additionally, we use the `edge_width_factor` and `emph_strong` arguments to adjust the width of the lines between and within PaLD communities.

```
labels <- rownames(tissue_cohesion)
plot_community_graphs(tissue_cohesion,
                      show_labels = FALSE,
                      vertex.size = 4,
                      vertex.color = as.factor(labels),
                      edge_width_factor = 35,
                      emph_strong = 5)
legend("topleft",
       legend = unique(as.factor(labels)),
       pt.bg = unique(as.factor(labels)),
       col = "black",
       pch = 21)
```

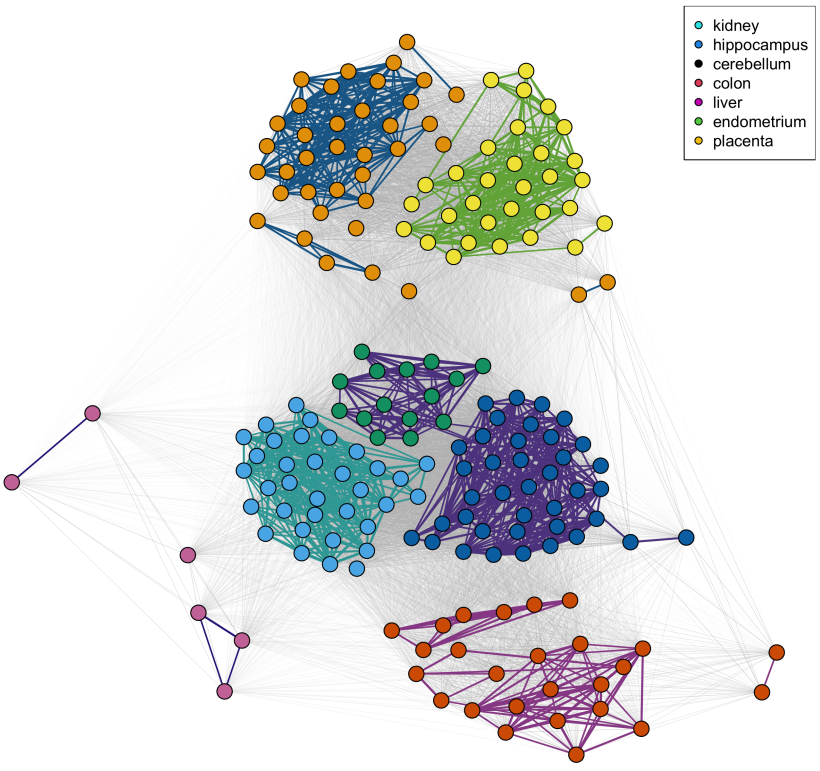


Figure 6: Community cluster network for the tissue data. The line colors indicate the PaLD communities; the point colors indicate the tissue classification.

A summary of community strong ties can be produced via the following code. Note that `tissue_graph_strong` is an **igraph** object corresponding to the graph displayed in Figure 6, restricted only to strong ties.

```
tissue_graphs <- community_graphs(tissue_cohesion)
tissue_graph_strong <- tissue_graphs[["G_strong"]]
E <- igraph::get.edgelist(tissue_graph_strong)
tissue_strong_ties <- data.frame(
  strong_ties = apply(E, 1, paste, collapse = ",")
)
tissue_strong_ties |>
  table() |>
  sort(decreasing = TRUE)
```

```
#> strong_ties
#>           kidney,kidney
#>                     273
#>             colon,colon
#>                     241
#> hippocampus,hippocampus
#>                     191
#>   cerebellum,cerebellum
#>                     186
#>             liver,liver
#>                      85
#> endometrium,endometrium
#>                      81
#>       placenta,placenta
#>                       4
#>      kidney,endometrium
#>                       3
```

Note that there are only three strong ties between different tissue types (kidney and endometrium) in the community cluster network of 1,064 strong ties total.

0.6 Cognate-based Language Families

This example explores a data set from Dyen et al. (1992) that summarizes relationships between 87 Indo-European languages from the perspective of cognates, coded using 2,655-dimensional binary vectors. A `dist` object was created from this data set and is included in the **pald** package in an object called `cognate_dist`.

Here we will demonstrate how one can further apply functions in the **igraph** package to objects output from the **pald** package. We can first use the `cohesion_matrix` function to calculate the cohesion matrix and the `community_graphs` function to create a list with the weighted community graph, the weighted community graph with only strong ties included, and the layout. From this, we can extract the graph with only the strong ties, here called `cognate_graph_strong`.

```
cognate_cohesion <- cohesion_matrix(cognate_dist)
cognate_graphs <- community_graphs(cognate_cohesion)

cognate_graph_strong <- cognate_graphs[["G_strong"]]
```

We can then use the `neighbors` function from the **igraph** package to extract the strong neighbors in this graph. For example, we can extract all neighbors for the language "French", via the following code.

```
french_neighbors <- igraph::neighbors(cognate_graph_strong, "French")
french_neighbors
```

```
#> + 8/87 vertices, named, from 5c6bb3a:
#> [1] Italian
#> [2] Ladin
#> [3] Provencal
#> [4] Walloon
#> [5] French_Creole_C
#> [6] French_Creole_D
#> [7] Spanish
#> [8] Catalan
```

Similarly, we can sort and print the associated neighborhood weights by subsetting the cohesion matrix.

```
cognate_cohesion["French", french_neighbors] |>
  sort(decreasing = TRUE)
```

```
#>         Walloon
#>      0.03258771
#>       Provencal
#>      0.02871174
#> French_Creole_C
#>      0.02406057
#> French_Creole_D
#>      0.02406057
#>           Ladin
#>      0.02094596
#>         Italian
#>      0.01997696
#>         Catalan
#>      0.01859688
#>         Spanish
#>      0.01679733
```

We can again use the `plot_community_graphs` function to visualize the community clusters (Figure 7). One may note the commonly identifiable language clusters and that, under a slight rotation, some of the underlying geography is mirrored in the plot.

```
plot_community_graphs(
  cognate_cohesion,
  edge_width_factor = 30,
  emph_strong = 3,
  vertex.size = 3,
  vertex.label.cex = 0.7,
  vertex.label.dist = 1
)
```

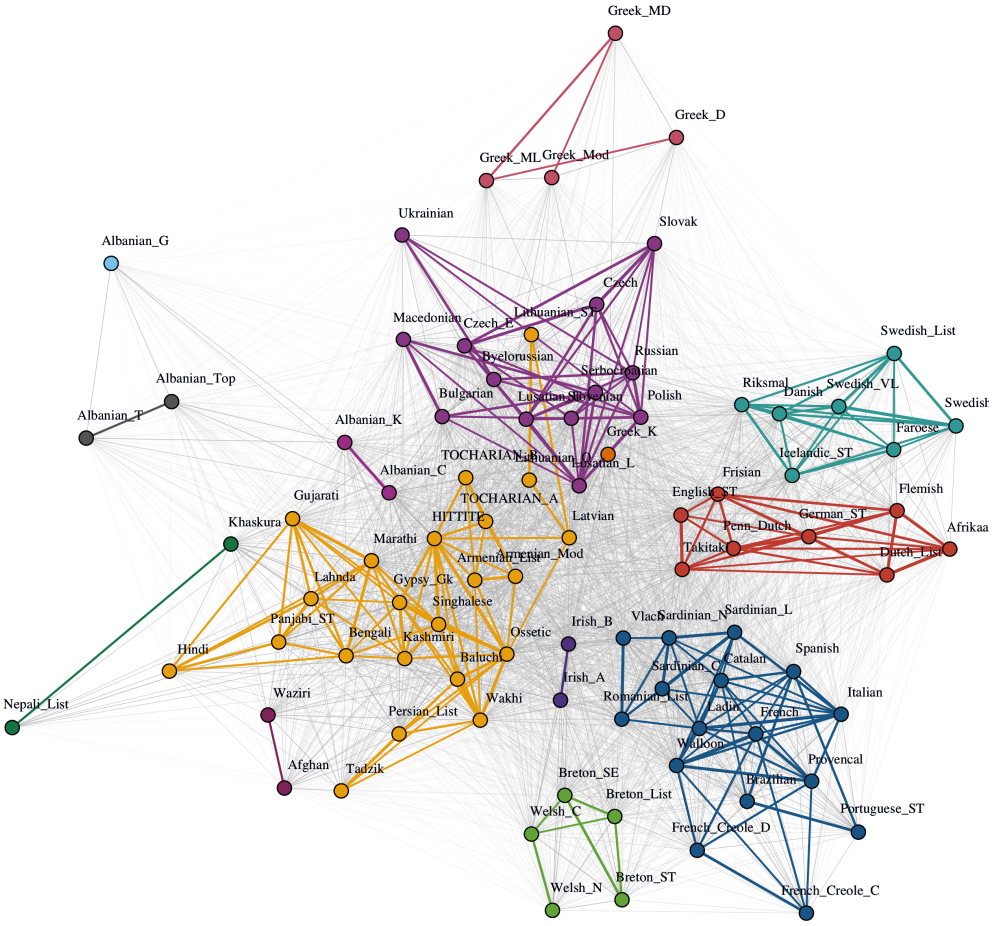


Figure 7: Community structure for 87 Indo-European languages, which employs cognate information that was coded via 2,665-dimensional binary vectors. Commonly identifiable language clusters arise along with informative inter- and intra-cluster structure. Several ancient languages are centrally located.

Community analysis for generated data

The **pald** package includes three randomly generated data frames corresponding to plots from Berenhaut et al. (2022):

- `exdata1` is a data set consisting of 8 points used to create Figure 1 in Berenhaut et al. (2022)
- `exdata2` is a data set consisting of 16 points used to create Figure 2 in Berenhaut et al. (2022)
- `exdata3` is a data set consisting of 240 points used to create Figure 4D in Berenhaut et al. (2022)

Here, we will demonstrate how to use `exdata3`. These points were generated from bivariate normal distributions with varying means and variances. There are eight "true" communities.

We can contrast resulting community clusters obtained via PaLD (i.e. connected components of the network of strong ties) with clusters as obtained through common cluster analysis techniques. Here we will consider two common clustering methods. The code below calculates the cohesion matrix (`exdata_cohesion`) as well as the community clusters obtained via PaLD (`exdata_pald`), along with $k = 8$ clusters obtained via k-means

(exdata_kmeans) and hierarchical clustering using complete linkage (exdata_hclust). Recall that there is no need to determine values for extraneous inputs in the case of PaLD (e.g. the number of clusters, as is necessary to specify for k-means and hierarchical clustering).

```
exdata_cohesion <- exdata3 |>
  dist() |>
  cohesion_matrix()

exdata_pald <- community_clusters(exdata_cohesion)$community

exdata_kmeans <- kmeans(exdata3, 8)$cluster

exdata_hclust <- exdata3 |>
  dist() |>
  hclust() |>
  cutree(k = 8)
```

The information is displayed in Figure 8.

```
par(mfrow = c(1, 3), pty = "s")
plot(
  exdata3,
  pch = 16,
  col = pald_colors[exdata_pald],
  xlab = "",
  ylab = "",
  main = "PaLD Communities",
  asp = 1
)
plot(
  exdata3,
  pch = 16,
  col = pald_colors[exdata_kmeans],
  xlab = "",
  ylab = "",
  main = "K-Means Clusters (k = 8)",
  asp = 1
)
plot(
  exdata3,
  pch = 16,
  col = pald_colors[exdata_hclust],
  xlab = "",
  ylab = "",
  main = "Hiearchical Clusters (k = 8)",
  asp = 1
)
```

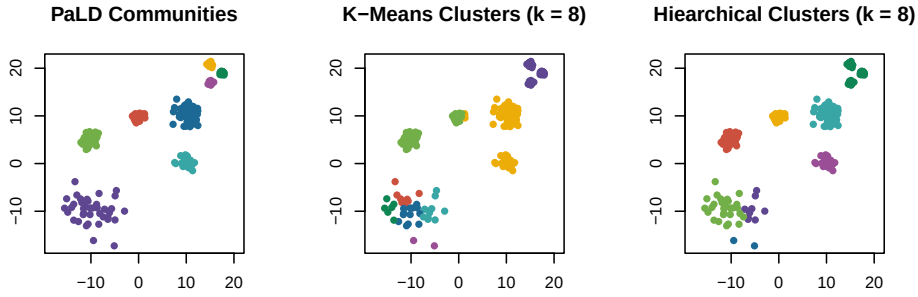


Figure 8: PaLD, k-means, and hierarchical 8-clustering of randomly generated example data (from Berenhaut et al. (2022); Figure 4D).

Cohesion is particularly useful when considering data with varying local density; see Berenhaut et al. (2022) for further examples, discussion, and theoretical results. Note that the PaLD algorithm is able to detect the eight natural groups within the data (along with inter- and intra-community structure not displayed here) without the use of any additional inputs (e.g., number of clusters) nor optimization criteria. Despite the user input of the "correct" number of clusters (i.e., $k = 8$) both k-means and hierarchical clustering do not provide the desired result.

Cultural and Psychological distance analysis

In this example we perform a PaLD analysis for cultural distances obtained in Muthukrishna et al. (2020) from two recent waves of the World Values Survey (2005 to 2009 and 2010 to 2014; see Inglehart et al. (2014)). Distances are computed using the cultural fixation index (CFST), which is a measure built on the framework of fixation indices from population biology (Bell et al. (2009); Cavalli-Sforza et al. (1994)). Recall that the foundation of PaLD in within-triplet comparisons allows for the employment of application-dependent and non-Euclidean measures of dissimilarity. The `dist` object is included in the **pald** package (`cultures`). We will first create the cohesion matrix using the `dist` object `cultures`, and proceed to plot the community graph.

```
cultures_cohesion <- cohesion_matrix(cultures)

plot_community_graphs(
  cultures_cohesion,
  edge_width_factor = 30,
  emph_strong = 3,
  vertex.label.cex = 0.7,
  vertex.size = 3,
  vertex.label.dist = 1
)
```

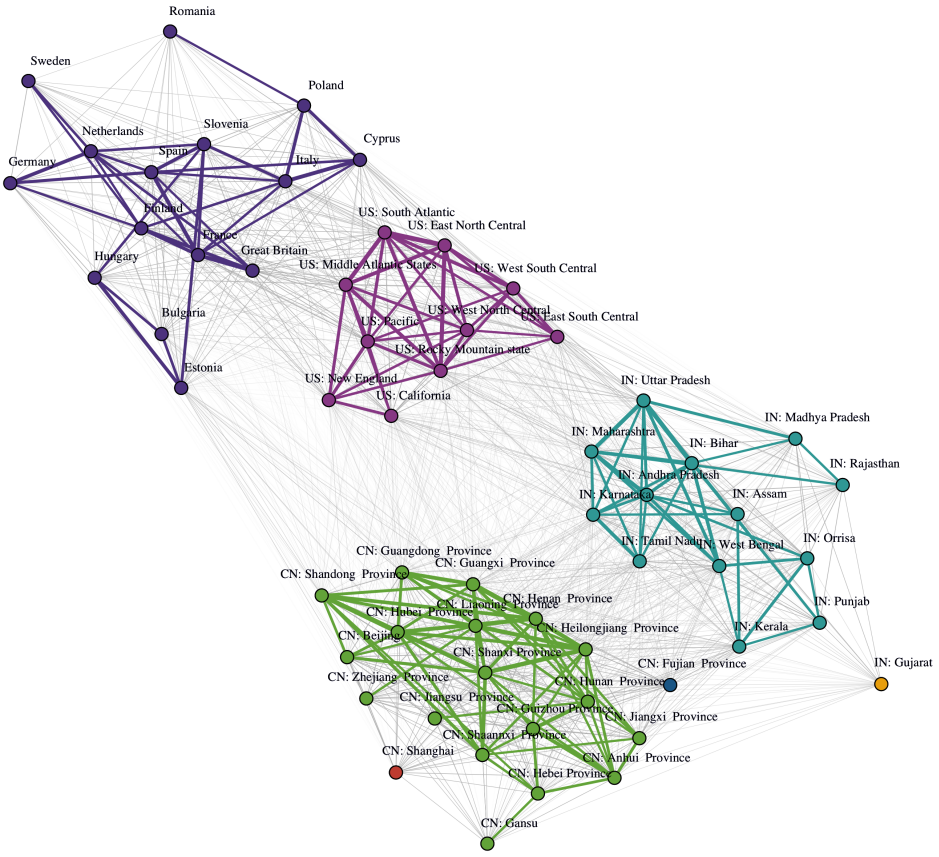


Figure 9: Community structure for cultural distance data.

In addition to viewing the local and global community structure as seen in Figure 9, the **pald** package allows for a two-dimensional display of cohesion against distance for the data, via the `dist_cohesion_plot` function, as seen below (Figure 10).

```
dist_cohesion_plot(cultures, mutual = TRUE)
```

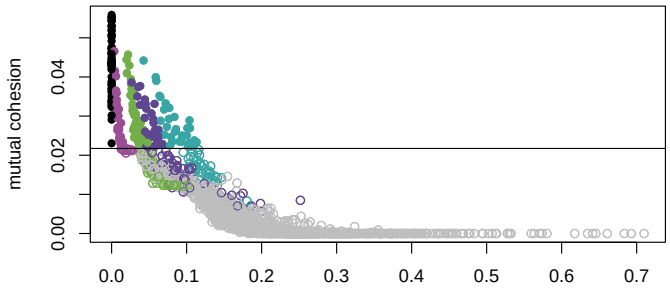


Figure 10: A plot of cohesion versus distance for the data. The identified communities are colored as in Figure 9.

Notice here that the magnitude of the distances within each of the identified communities varies substantially between regions; in fact, the most disparate two regions in the United States (at distance ≈ 0.027) are far closer than the two most similar in India (at distance ≈ 0.043). Despite this, India remains a cohesive whole, and locally disparate regions in

the United States such as East South Central and California are not strongly cohesive. For discussion of subtleties in local density (see Berenhaut et al. (2022). Additionally, currently available techniques require specification of parameters, as seen in the previous section. For further discussion of the cultural distance data in relation to community analysis see Berenhaut et al. (2022).

Given the inherent variation in density over regions, it is of interest to consider complementary results from density-based methods such as DBSCAN and its variant HDBSCAN. Employing the **dbscan** package (see Hahsler et al. (2019)), for DBSCAN with parameters `eps=0.34` and `minPts=3`, we have a result with five clusters and six regions being classified as noise. Therein, (excluding noise) the Indian region is split into two communities. Similarly, for HDBSCAN, with `minPts=3`, we have a result with five clusters and eight noise points. Here again (excluding the noise points), India has been split into two communities. The parameter choices used for both methods were determined by maximizing normalized mutual information (NMI; see Ana and Jain (2003)), when comparing with the underlying partition of the regions into Europe, United States, India and China, by regional information), giving values of 0.848 and 0.819, respectively. For comparison, the corresponding NMI value for the result in Figure 9 is 0.933. Further discussion is available in Berenhaut et al. (2022). It is important to note the required parameter choices for both DBSCAN and HDBSCAN and that results can be somewhat sensitive to changes in these. For instance, for DBSCAN, an increase in the value of `minPts` from 3 to 5, and a decrease in `eps` value from 0.34 to 0.30 leads to a large number of noise points (25), and a decrease in NMI value from 0.848 to 0.685. For HDBSCAN an increase in the value of `minPts` from 3 to 5 leads to 15 noise points, and a decrease in NMI value from 0.819 to 0.760, while a decrease in the value of `minPts` from 3 to 2 leads to 10 clusters and a decrease in NMI to 0.762.

0.7 Computational considerations

Computation of the cohesion matrix as implemented in the package is of order $O(n^3)$. The method is highly parallelizable, though, and recent work has resulted in extensive speed-up (see Devarakonda and Ballard (2024)). Approximations are also available (see Baron et al. (2021)). Note that the method is entirely deterministic, and one does not need to search a parameter space nor select initial values.

0.8 Summary

This paper introduces the **pald** package, demonstrating its utility for providing novel parameter-free community analysis which can easily be implemented for a variety of data sets, supplementing results from other methods for clustering, embedding, data depth, nearest neighbors, etc. Example code is provided along with discussion in the context of other commonly used R-based approaches.

References

C. Agostinelli and M. Romanazzi. Local depth. *Journal of Statistical Planning and Inference*, 141(2):817–830, 2011. [p39]

L. F. Ana and A. K. Jain. Robust data clustering. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 2, pages II–II. IEEE, 2003. [p53]

J. D. Baron, R. Darling, J. L. Davis, and R. Pettit. Partitioned k-nearest neighbor local depth for scalable comparison-based learning. *arXiv preprint arXiv:2108.08864*, 2021. [p53]

A. V. Bell, P. J. Richerson, and R. McElreath. Culture rather than genes provides greater scope for the evolution of large-scale human prosociality. *Proceedings of the National Academy of Sciences*, 106(42):17671–17674, 2009. [p51]

K. S. Berenhaut and K. E. Moore. Communities in data. *SIAM News Blog*, 2022. [p37, 39]

K. S. Berenhaut, K. E. Moore, and R. L. Melvin. A social perspective on perceived distances reveals deep community structure. *Proceedings of the National Academy of Sciences*, 119(4), 2022. [p35, 36, 37, 38, 39, 41, 42, 49, 51, 53]

K. S. Berenhaut, J. D. Foley, and L. Lyu. Generalized partitioned local depth. *Journal of Statistical Theory and Practice*, 18(1):10, 2024. [p37, 38, 39]

R. J. Campello, P. Kröger, J. Sander, and A. Zimek. Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(2):e1343, 2020. [p39]

L. L. Cavalli-Sforza, L. Cavalli-Sforza, P. Menozzi, and A. Piazza. *The history and geography of human genes*. Princeton university press, 1994. [p51]

A. Devarakonda and G. Ballard. Sequential and shared-memory parallel algorithms for partitioned local depths. In *Proceedings of the 2024 SIAM Conference on Parallel Processing for Scientific Computing*, pages 53–64, 2024. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611977967.5. [p53]

I. Dyen, J. B. Kruskal, and P. Black. An Indoeuropean classification: A lexicostatistical experiment. *Transactions of the American Philosophical Society*, 82(5):iii, 1992. doi: 10.2307/1006517. [p41, 47]

S. L. Feld. The focused organization of social ties. *American Journal of Sociology*, 86(5): 1015–1035, 1981. [p38]

A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data*, 1:1–30. [p41]

M. S. Granovetter. The strength of weak ties. *American Journal of Sociology*, 78(6):1360–1380, 1973. [p38]

M. Hahsler, M. Piekenbrock, and D. Doran. dbscan: Fast density-based clustering with R. *Journal of Statistical Software*, 91:1–30, 2019. [p53]

R. Inglehart, C. Haerpfer, A. Moreno, C. Welzel, K. Kizilova, J. Díez-Medrano, M. Lagos, P. Norris, E. Ponarin, B. Puranen, et al. World values survey: All rounds–country-pooled datafile 1981-2014. 2014. [p41, 51]

M. Love and R. Irizarry. *tissuesGeneExpression. Subset of gene expression data*, 2015. URL https://github.com/genomicsclass/tissuesGeneExpression. [p41, 45]

M. N. McCall, K. Uppal, H. A. Jaffee, M. J. Zilliox, and R. A. Irizarry. The gene expression barcode: leveraging public data repositories to begin cataloging the human and murine transcriptomes. *Nucleic Acids Research*, 39(suppl_1):D1011–D1015, 2011. [p45]

M. N. McCall, H. A. Jaffee, S. J. Zelisko, N. Sinha, G. Hooiveld, R. A. Irizarry, and M. J. Zilliox. The gene expression barcode 3.0: improved data processing and mining tools. *Nucleic Acids Research*, 42(D1):D938–D943, 2014. [p45]

M. Muthukrishna, A. V. Bell, J. Henrich, C. M. Curtin, A. Gedranovich, J. McInerney, and B. Thue. Beyond Western, Educated, Industrial, Rich, and Democratic (WEIRD) psychology: Measuring and mapping scales of cultural and psychological distance. *Psychological Science*, 31(6):678–701, 2020. [p41, 51]

D. Paindaveine and G. Van Bever. From depth to local depth: a focus on centrality. *Journal of the American Statistical Association*, 108(503):1105–1119, 2013. [p39]

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011. [p41]

M. J. Zilliox and R. A. Irizarry. A gene expression bar code for microarray data. *Nature Methods*, 4(11):911–913, 2007. [p45]

Lucy D'Agostino McGowan
Wake Forest University
Winston-Salem, NC
27106
mcgowald@wfu.edu

Katherine Moore
Amherst College
Amherst, MA
1002
kmoore@amherst.edu

Kenneth S. Berenhaut
Wake Forest University
Winston-Salem, NC
27106
berenhks@wfu.edu

Student t-Lévy regression model in yuima

by Hiroki Masuda, Lorenzo Mercuri, and Yuma Uehara

Abstract This paper presents an estimation and simulation method in the R package **yuima** for a linear regression model driven by a Student-t Lévy process with constant scale and arbitrary degrees of freedom. This process finds applications in several fields, for example finance, physics, biology, etc. The first challenge involves simulating sample paths at high-frequency levels, as only unit-time increments are Student-t distributed. In **yuima**, we solve this problem by means of the inverse Fourier transform for simulating the increments of a Student-t Lévy defined on an interval with any length. The second challenge is the joint estimation of trend, scale, and degrees of freedom, a problem not previously explored in the literature. In **yuima**, we develop a two-step estimation procedure that efficiently deals with this issue. Numerical examples are given in order to explain methods and classes used in the **yuima** package.

1 Introduction

The **yuima** package in R provides several simulation and estimation methods for stochastic processes (YUIMA Project Team, 2024; Brouste et al., 2014; Iacus and Yoshida, 2018). This paper introduces new classes and methods in **yuima** for simulating and estimating a t-Lévy regression model based on high-frequency observations. This model can be seen as a generalization of a t-Lévy process (Heyde and Leonenko, 2005; Cufaro Petroni, 2007) by adding covariates, which can be either deterministic or stochastic processes. Adding covariates to a continuous-time stochastic process is a widely used approach for constructing new processes in various fields. For example, in finance, periodic deterministic covariates can be employed to capture the seasonality observed in commodity markets (Sørensen, 2002). Alternatively, in insurance and medicine, covariates are often incorporated into mortality rate dynamics to account for age and cohort effects (see Haberman and Renshaw, 2009; Castro-Porras et al., 2021, and references therein).

In such fields, data often exhibit heavy-tailed behavior in the margins and thus, the inclusion of a t-Lévy process as a driving noise would be useful. However, despite the simple mathematical definition of a t-Lévy regression model, it poses significant challenges both in deriving its mathematical properties and in implementing numerical methods. A key difficulty arises from the fact that the t-Lévy driving noise is not closed under convolution. Consequently, the distribution of its increments follows a t-distribution only over a unit-time interval.

Motivated by this fact, in **yuima**, we propose three simulation methods for a t-Lévy regression model. The key element is the construction of a random number generator for the noise increments, which is essential for several discretized simulation methods and involves the numerical inversion of the characteristic function. More specifically, our method is based on the approximation of the cumulative distribution function, and our method can relieve numerical instability compared with directly using the density function especially for simulating small time increments. Additionally, **yuima** provides a two-stage estimation procedure and the corresponding estimator has consistency and asymptotic normality as shown in Masuda et al. (2024).

The rest of this paper is organized as follows. We briefly summarize the t-Lévy regression model in Section 2. We introduce the new **yuima** classes and methods in Section 3 and we show some examples with simulated and real data that highlight their usage in Section 4. Finally, Section 5 concludes the paper.

2 Student t-Lévy regression model

In this section, we review the main characteristics of the t-Lévy regression model and the t-Lévy process used as its driving noise. We highlight the main issues that arise in the simulation and estimation procedures for these models. For a complete discussion on all mathematical aspects and the properties of these procedures in both cases, we refer to Masuda et al. (2024).

The proposed t-Lévy regression model is a continuous-time stochastic process of the form:

$$Y_t = X_t \cdot \mu + \sigma J_t, \qquad t \in [0, T_n], \tag{1}$$

where the q-dimensional vector process $X = (X_t)$ with càdlàg paths contains the covariates; the dot denotes the inner product in $\mathbb{R}^q$ and $J = (J_t)$ is a scaled t-Lévy process such that its unit-time distribution $\mathcal{L}(J_1)$ is given by:

$$\mathcal{L}(J_1) = t_\nu := t_\nu(0, 1), \tag{2}$$

where $t_\nu(\mu_1, \sigma_1)$ denotes the scaled Student-t distribution[1] with density: The parameters $\nu > 0$, and $\sigma > 0$ represent the degree of freedom, location, and scale parameters, respectively (see Heyde and Leonenko, 2005; Cufaro Petroni, 2007, for more details on a t-Lévy process). For this model, we consider the situation where we estimate these three unknown parameters based on a discrete-time sample $\{(X_{t_j}, Y_{t_j})\}_{j=0}^{[nT_n]}$ with $t_j = t_j^n := \frac{j}{n}$ and $T_n \to \infty$ as $n \to \infty$.

To develop the **yuima** simulation and estimation algorithms for the model in (1), we need to address two problems: first, the simulation of the sample path of $J = (J_t)$ on a small time grid with $\Delta t \neq 1$ which is essential for handling the increments:

$$\Delta_j Y = \Delta_j X \cdot \mu + \sigma \Delta_j J, \quad j = 0, 1, \ldots, [nT_n]$$

where $\Delta_j Z$ denotes $Z_{t_j} - Z_{t_{j-1}}$ for any stochastic process $Z = (Z_t)$. Second, the identification of an efficient procedure for estimating the model parameters (μ, σ) in (1) and the degree of freedom ν in (2). We will introduce both the simulation and estimation methods below.

Due to the stationary behavior of the L'evy increments, i.e. $\mathcal{L}(\Delta_i J) = \mathcal{L}(J_h)$ and $h := t_i - t_{i-1} = \frac{1}{n}$ for $i = 1, \ldots, [nT_n]$, $\mathcal{L}(J_h)$ admits the Lebesgue density:

$$\begin{aligned} x \mapsto & \frac{1}{\pi} \int_0^\infty \cos(ux) \{\varphi_{J_1,\nu}(u)\}^h du \\ & = \left(\frac{2^{1-\nu/2}}{\Gamma(\nu/2)} \right)^h \frac{1}{\pi} \int_0^\infty \cos(ux) u^{\nu h/2} \left(K_{\nu/2}(u)\right)^h du, \end{aligned} \tag{3}$$

where $\varphi_{J_1,\nu}(u)$, the characteristic function of $\mathcal{L}(J_1) = t_\nu$ is given by

$$\varphi_{J_1,\nu}(u) := \frac{2^{1-\nu/2}}{\Gamma(\nu/2)} |u|^{\nu/2} K_{\nu/2}(|u|), \qquad u \in \mathbb{R} \tag{4}$$

and $K_\nu(t)$ denotes the modified Bessel function of the second kind ($\nu \in \mathbb{R}$, $t > 0$):

$$K_\nu(t) = \frac{1}{2} \int_0^\infty s^{\nu-1} \exp\left\{ -\frac{t}{2} \left(s + \frac{1}{s} \right) \right\} ds.$$

We here remark that although the explicit expression of (3) cannot be obtained for general h, the tail index of $\mathcal{L}(J_h)$ is the same as that of $\mathcal{L}(J_1)$ (Berg and Vignat, 2008, Theorem 2),

[1] Let J_1 be a scaled Student-t random variable with parameters $\nu > 0$ $\mu_1 = 0$ and $\sigma_1 = 1$, its transformation $Z_1 := \sqrt{\nu} J_1$ is a Student-t r.v. see @johnson1995continuous Ch. 28 with the following density function:

$$f_{Z_1}(z; \nu) := \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\pi\nu}\Gamma(\frac{\nu}{2})} \left\{ 1 + \frac{z^2}{\nu} \right\}^{-(\nu+1)/2}.$$

and hence, modeling tail index via student t-Lévy process makes sense. A classical method for simulating t-Lévy increments based on this Fourier inversion representation is through the rejection method, as discussed in Hubalek (2005) and Devroye (1981). However, this method can become time-consuming and unstable when dealing with very small values of h, primarily due to the oscillatory behavior encountered during the numerical evaluation of the density. Such a problem often arises since to express the high-frequency observed situation from a continuous process, we should take a finer mesh of simulating the underlying process than that of simulating the discrete observations. For this issue, in **yuima**, the Random Number Generator for the increments is developed using the inverse quantile method. The first step involves integrating the density in (3) to obtain the cumulative distribution function (CDF). This approach leads to a more stable behavior in the tails of the CDF compared to the density tails, owing to a mitigation effect observed during the numerical evaluation of the following double integral:

$$F(y) = \int_{-\infty}^{y} \frac{1}{\pi} \int_{0}^{\infty} \cos(ux)\{\varphi_{J_1,\nu}(u)\}^h du dx.$$

Further details on our approach are provided in the next section. Another way to simulate t-Lévy process is using series representation. Numerically, for each time t, we need ad-hoc truncation of the infinite sum. The Gaussian approximation of a small-jump part is also valid (Asmussen and Rosiński, 2001), but the associated error may not be easy to control in a practical manner. For more details, see Massing (2018).

Following Masuda et al. (2024), **yuima** provides a two-stage estimation algorithm for $(\mu, \sigma, \nu) \in \Theta_\mu \times \Theta_\sigma \times \Theta_\nu$. Denote by (μ_0, σ_0, ν_0) the true parameter values of the model in (1); the two-step algorithm can be summarized as follows:

1. An estimator $\hat{a} := (\hat{\mu}_n, \hat{\sigma}_n)$ of $a_0 = (\mu_0, \sigma_0)$ is obtained by maximizing the *Cauchy quasi-likelihood*:
$$\hat{a}_n := (\hat{\mu}_n, \hat{\sigma}_n) \in \underset{a \in \overline{\Theta_\mu \times \Theta_\sigma}}{\operatorname{argmax}} \mathbb{H}_{1,n}(a). \tag{5}$$
The Cauchy quasi-(log-)likelihood $\mathbb{H}_{1,n}(a)$ conditional on X has the following form:
$$\begin{aligned}\mathbb{H}_{1,n}(a) &:= \sum_{j=1}^{N_n} \log\left\{\frac{1}{h\sigma}\phi_1\left(\frac{\Delta_j Y - \mu \cdot \Delta_j X}{h\sigma}\right)\right\} \\ &= C_n - \sum_{j=1}^{N_n}\left\{\log\sigma + \log\left(1 + \epsilon_j(a)^2\right)\right\},\end{aligned}$$
where ϕ_1 denotes the density function of the standard Cauchy distribution and the term C_n does not depend on $a = (\mu, \sigma)$. $N_n := [nB_n]$ is the number of observations in the part $[0, B_n]$ of the entire period $[0, T_n]$ where (B_n) is a positive sequence satisfying $B_n \le T_n$ and $n^{\epsilon''} \lesssim B_n \lesssim n^{1-\epsilon'}$ for some $\epsilon', \epsilon'' \in (0,1)$.

2. Then, we construct an estimator $\hat{\nu}_n$ of ν_0 using the *Student-t quasi-likelihood* on the "unit-time'' residual sequence $\hat{\epsilon}_i$:
$$\hat{\epsilon}_i := \hat{\sigma}_n^{-1}\left(Y_i - Y_{i-1} - \hat{\mu}_n \cdot (X_i - X_{i-1})\right),$$
for $i = 1, \ldots, [T_n]$, which is expected to be approximately i.i.d. t_ν-distributed. Therefore $\hat{\nu}_n$ solves:
$$\hat{\nu}_n \in \underset{\nu \in \overline{\Theta}_\nu}{\operatorname{argmax}} \mathbb{H}_{2,n}(\nu),$$
where
$$\mathbb{H}_{2,n}(\nu) := \sum_{i=1}^{[T_n]}\left(-\frac{1}{2}\log\pi + \log\Gamma\left(\frac{\nu+1}{2}\right) - \log\Gamma\left(\frac{\nu}{2}\right) - \frac{\nu+1}{2}\log\left(1 + \hat{\epsilon}_i^2\right)\right).$$

We note that the first estimation scheme is based on the locally Cauchy property of J: $h^{-1}J_h \xrightarrow{\mathcal{L}} t_1$ (standard Cauchy) as $h \to 0$. Let $\hat{u}_{a,n} := \sqrt{N_n}(\hat{a}_n - a_0)$ and $\hat{u}_{\nu,n} := \sqrt{T_n}(\hat{\nu}_n - \nu_0)$. Under some regularity conditions on the covariate process $X = (X_t)$ (see Masuda et al., 2024, Assumption 2.1 in for all requirements on X), the estimators have the following joint asymptotic normality:

$$(\hat{\Gamma}_{a,n}^{1/2}\hat{u}_{a,n}, \hat{\Gamma}_{\nu,n}^{1/2}\hat{u}_{\nu,n}) \xrightarrow{\mathcal{L}} N_{q+2}(0, I_{q+2}),$$

where ψ_1 denotes the trigamma function and

$$\hat{\Gamma}_{a,n} = \mathrm{diag}\left(\frac{1}{2\hat{\sigma}_n^2 N_n}\sum_{j=1}^{N_n}\left(\frac{1}{h}\Delta_j X\right)^{\otimes 2}, \frac{1}{2\hat{\sigma}_n^2}\right),$$

$$\hat{\Gamma}_{\nu,n} = \frac{1}{4}\left(\psi_1\left(\frac{\hat{\nu}_n}{2}\right) - \psi_1\left(\frac{\hat{\nu}_n + 1}{2}\right)\right),$$

Since $\hat{\Gamma}_{a,n}^{1/2}$ and $\hat{\Gamma}_{\nu,n}$ can be constructed only by the observations, we can easily obtain the confidence intervals of each parameter.
We conclude this section by noting that the function $\mathbb{H}_{2,n}(\nu)$ is concave with respect to ν, as demonstrated in Masuda et al. (2024). This result follows directly from the fact that the driving noise in the t-Lévy regression model is a scaled t-Lévy process with a unit-time distribution as defined in (2).

3 Classes and methods for t-lévy regression models

This section provides an overview of the new classes and methods introduced in **yuima** for the mathematical definition, trajectory simulation, and estimation of a Student Lévy Regression model. To handle this model, the first step involves constructing an object of the `yuima.LevyRM-class`. As an extension of the `yuima-class` refer to Brouste et al. (2014) for more details, `yuima.LevyRM-class` inherits slots such as `@data`, `@model`, `@sampling`, and `@functional` from its parent class. The remaining slots store specific information related to the Student-t-Lévy regression model.

Notably, the slot `@unit_Levy` contains an object of the `yuima.th` class, which represents the mathematical description of the Student-t Lévy process J_t (see the subsequent section for detailed explanations). The labels of the regressors are saved in the slot `@regressors`, while slots `@LevyRM` and `@paramRM` respectively cache the names of the output process Y_t and a `string` vector reporting the regressors' coefficients, the scale parameter, and the degree of freedom. The `yuima.th-class` is obtained by the new `setLaw_th` constructor. This function requires the arguments used for the numerical inversion of the characteristic function, and its usage is discussed in the next section.

Once the `yuima.th-object` is created, we define the system of stochastic differential equations (SDEs) that describes the behavior of the regressors, with their mathematical definitions stored in an object of the `yuima.model-class`. Both `yuima.th` and `yuima.model` objects are used as inputs for the `setLRM` constructor, which returns an object of the `yuima.LevyRM-class`. The following chunk code reports the input for this new function.

```
setLRM(unit_Levy, yuima_regressors, LevyRM = "Y", coeff = c("mu", "sigma0"), data = NULL,
  sampling = NULL, characteristic = NULL, functional = NULL)
```

As is customary for any class extending the `yuima-class`, the `simulate` method enables the generation of sample paths for the Student Lévy Regression model. To simulate trajectories, an object of the `yuima.sampling-class` is constructed to represent an equally spaced grid-time used in trajectory simulation. The regressors' paths are obtained using the Euler scheme, while the increments of the Student-t Lévy process are simulated using the random number generator available in the slot `@rng` of the `yuima.th-object`.

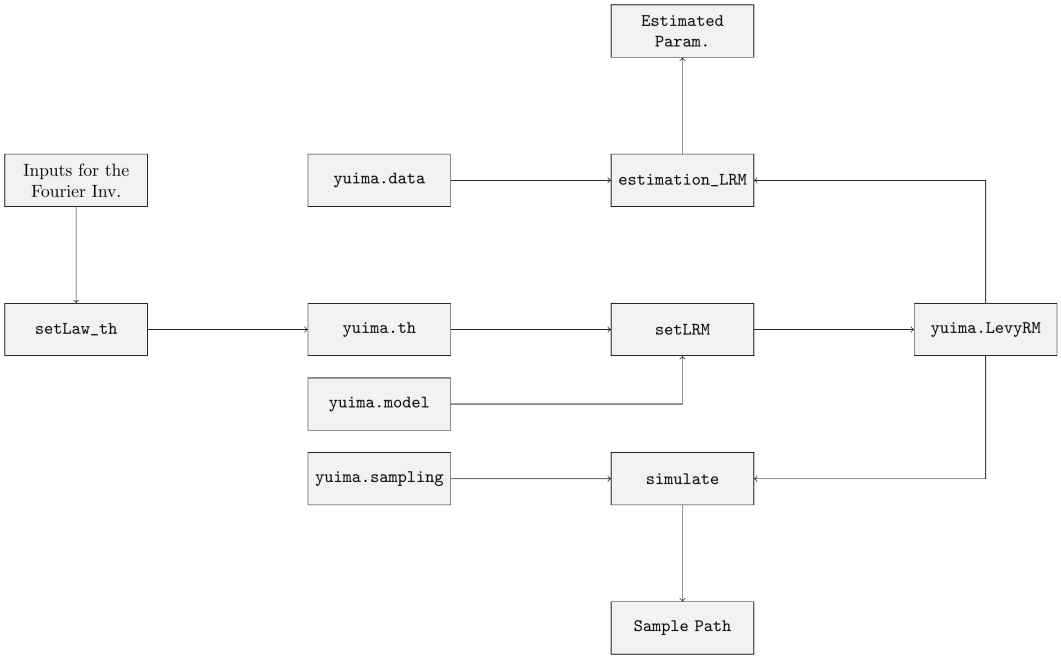


Figure 1: Scheme of classes and methods in yuima for the t-Lévy Regression model.

The last method available in **yuima** is `estimation_LRM`. This method allows the users to estimate the model using either real or simulated data. The estimation follows a two-step procedure, introduced in Masuda et al. (2024).

```
estimation_LRM(start, model, data, upper, lower)
```

For this function, the minimal inputs are `@start`, `@model`, `@data`, `@upper` and `@lower`. The arguments `@start` are the initial points for the optimization routine while `@upper` and `@lower` correspond to the box constraints. The `@yuima.LevyRM-object` is passed to the function through the input `@model` while the input `@data` is used to pass the dataset to the internal optimization routine. Figure 1 describes these new classes and methods, along with their respective usage.

3.1 yuima.th: A new class for mathematical description of a Student-t Lévy process

In this section, the steps for the construction of an `yuima.th-object` are presented. As remarked in Section 3, this object contains all information on the scaled Student-t Lévy process. Moreover, detailed information is provided regarding the numerical algorithms utilized for evaluating the density function (3). The construction of this object is accomplished using the `setLaw_th` constructor, and the subsequent code snippet displays its corresponding arguments.

```
setLaw_th(h = 1, method = "LAG", up = 7, low = -7, N = 180, N_grid = 1000,
  regular_par = NULL)
```

The input `h` is the length of the step-size of each time interval for the Student-t Lévy increment $\Delta J_h = J_h - J_0$. Its default value, `h=1`, indicates that the `yuima.th-object` describes completely the process J_t at time 1. The argument `method` refers to the type of quadrature used for the computation of the integral in (3) while the remaining arguments govern the precision of the integration routine.

The `yuima.th-class` inherits the slots `@rng`, `@density`, `@cdf` and `@quantile` from its parent class `yuima-law` Masuda et al. (2022). These slots store respectively the random number generator, the density function, the cumulative distribution function and the quantile function of J_h.

As mentioned in Section 1, the density function with $h \neq 1$ does not have a closed-form formula and, therefore, the inversion of the characteristic function is necessary. **yuima** provides three methods for this purpose: the Laguerre quadrature, the COS method and the Fast Fourier Transform. The Gauss-Laguerre quadrature is a numerical integration method employed for evaluating integrals in the following form:

$$\mathcal{I} = \int_0^{+\infty} f(x)\, e^{-x} \mathrm{d}x.$$

This procedure has been recently used for the computation of the density of the variance gamma and the transition density of a CARMA model driven by a time-changed Brownian motion (Loregian et al., 2012; Mercuri et al., 2021), this motivates its application in this paper.

Let $f(x)$ be a continuous function defined on $[0, +\infty)$ such that:

$$\mathcal{I} = \int_0^{+\infty} f(x)\, e^{-x} \mathrm{d}x < +\infty;$$

the integral $\mathcal{I}$ can be approximated as follows:

$$\mathcal{I} \approx \sum_{j=1}^{N} \omega(k_j)\, f(k_j), \tag{6}$$

where k_j is the jth-root of the N-order Laguerre polynomial[2] $L_N(x)$ and the weights $\omega(k_j), j = 1, \ldots, N$ are defined as:

$$\omega(k_j) = \frac{k_j}{(N+1)^2 L_{N+1}^2(k_j)}. \tag{7}$$

To apply the approximation in (6), we rewrite the inversion formula in (3) as follows:

$$\begin{aligned} f(x) &= \frac{1}{\pi}\left(\frac{2^{1-\frac{\nu}{2}}}{\Gamma(\nu/2)}\right)^h \int_0^{+\infty} \cos(ux)\, u^{\nu h/2} \left(K_{\nu/2}(u)\right)^h \mathrm{d}u \\ &= \frac{1}{\pi}\left(\frac{2^{1-\frac{\nu}{2}}}{\Gamma(\nu/2)}\right)^h \int_0^{+\infty} \cos(ux)\, u^{\nu h/2} \left(K_{\nu/2}(u)\right)^h e^u e^{-u} \mathrm{d}u. \end{aligned} \tag{8}$$

Applying the result in (6), the density function of J_h can be approximated with the formula reported below:

$$\hat{f}_N(x_j) = \frac{1}{\pi}\left(\frac{2^{1-\frac{\nu}{2}}}{\Gamma(\nu/2)}\right)^h \sum_{j=1}^{N} \cos(k_j x)\, k_j^{\nu h/2} \left(K_{\nu/2}(k_j)\right)^h e^{k_j} \omega(k_j). \tag{9}$$

Notably, the approximation formula's precision in equation (9) can be enhanced through the argument `N` in the `setLaw_th` constructor. The roots k_j and the weights $\omega(k_j)$ are internally computed using the `gauss.quad` function from the R package **statmod** (Smyth et al., 2023; Giner and Smyth, 2016), with a maximum allowed order of 180 for the Laguerre polynomial.

The COS method is based on the Fourier Cosine expansion employed for an even function with the compact domain $[-\pi, \pi]$. This method has been widely applied in the finance literature for the computation of the exercise probability of an option and its no-arbitrage price, we refer to Fang and Oosterlee (2009), Fang and Oosterlee (2011) and references therein for details. Let $g(\theta) : [-\pi, \pi] \to \mathbb{R}$ be an even function, its Fourier Cosine expansion reads:

$$g(\theta) = \frac{1}{2}a_0 + \sum_{k=1}^{\infty} a_k \cos(k\theta) \tag{10}$$

[2]The Laguerre polynomial can be defined recursively as follows: $L_0(x), L_1(x) = 1 - x, \ldots, L_N(x) = \frac{(2N-x)L_{N-1}(x) - (N-1)L_{N-2}(x)}{N}$

with

$$a_k = \frac{2}{\pi}\int_0^{\pi} g(\theta)\cos(k\theta)\,\mathrm{d}\theta.$$

Denoting with $f(x)$ the density function of J_h, the new function $\bar{g}(\theta)$ is defined as:

$$\bar{g}(\theta) := f\left(\frac{L}{\pi}\theta\right)\frac{L}{\pi}\mathbb{1}_{\{-\pi\leq\theta\leq\pi\}}, \tag{11}$$

can be applied. The coefficient a_k is determined as follows:

$$\begin{aligned} a_k &= \frac{2}{\pi}\int_0^{\pi}\bar{g}(\theta)\cos(k\theta)\,\mathrm{d}\theta \\ &= \frac{2}{\pi}\int_0^{\pi} f\left(\frac{L}{\pi}\theta\right)\cos(k\theta)\,\frac{L}{\pi}\mathrm{d}\theta. \end{aligned}$$

Setting $\theta = \frac{\pi}{L}x$, we have:

$$a_k = \frac{2}{\pi}\int_0^{L} f(x)\cos\left(k\frac{\pi}{L}x\right)\mathrm{d}x. \tag{12}$$

The coefficient a_k can be rewritten using the characteristic function of J_h at $k\frac{\pi}{L}$, i.e.:

$$a_k = \frac{2}{\pi}\left[\varphi\left(k\frac{\pi}{L}\right) - \int_L^{+\infty} f(x)\cos\left(k\frac{\pi}{L}x\right)\mathrm{d}x\right]. \tag{13}$$

For a sufficiently large L, the coefficient a_k can be approximated as follows:

$$a_k \approx \frac{2}{\pi}\varphi\left(k\frac{\pi}{L}\right). \tag{14}$$

The following series expansion is achieved:

$$f(x) = \frac{1}{2}\bar{a}_0 + \sum_{k=1}^{+\infty}\bar{a}_k\cos\left(k\frac{\pi}{L}x\right) \tag{15}$$

with

$$\bar{a}_k = a_k\frac{\pi}{L} \approx \frac{2}{L}\varphi\left(k\frac{\pi}{L}\right).$$

Finally, (15) can be approximated by truncating the series as follows:

$$\hat{f}_N(x) = \frac{1}{2}\hat{a}_{0,L} + \sum_{k=1}^{N}\hat{a}_{k,L}\cos\left(k\frac{\pi}{L}x\right) \tag{16}$$

with

$$\hat{a}_{k,L} = \frac{2}{L}\varphi\left(k\frac{\pi}{L}\right).$$

The precision of $\hat{f}_{N,L}(x)$ in (16) depends on N and L. Users can select these two quantities using `N` and the couple `(up, low)` respectively. L is computed internally in the `setLaw_th` constructor as `L = max(|low|,up)`. The last method available in **yuima** for the computation of the density of t-Lévy increments over a time interval with length h is based on the inversion of the characteristic function by means of the Fast Fourier Transform `FFT` Singleton (1969), Cooley and Tukey (1965) [3]. Denoting the density function as $f(x)$ and the characteristic function as $\varphi(u) := \mathbb{E}\left[e^{iuJ_h}\right]$ for the scaled t-Lévy J_h, the density $f(x)$ can be obtained as follows:

[3] Since **yuima** manages SDEs driven by a Lévy process. For a user-defined noise, the random number generator in the simulation algorithm and the density function for the estimation procedure are constructed using the **yuima** function `FromCF2yuima_law` see the documentation YUIMA Project Team (2024) for more details. This function internally implements the same `FFT` algorthim described in this paper.

$$\begin{aligned} f(x) &= \frac{1}{2\pi}\int_{-\infty}^{+\infty} e^{-iux}\mathbb{E}\left[e^{iuJ_h}\right]\mathrm{d}u \\ &= \frac{1}{2\pi}\int_{-\infty}^{+\infty} e^{-iux}\varphi(u)\,\mathrm{d}u. \end{aligned} \tag{17}$$

As first step, we apply to the integral in (17) the change variable $u = 2\pi\omega$ and we have:

$$f(x) = \int_{-\infty}^{+\infty} e^{-i2\pi\omega x}\varphi(2\pi\omega)\,\mathrm{d}\omega. \tag{18}$$

We consider discrete support for x and ω. The x grid has the following structure:

$$x_0 = a, \ldots, x_j = a + j\Delta x, \ldots, x_N = b \tag{19}$$

with $\Delta x = \frac{b-a}{N}$ where $N+1$ is the number of the points in the grid in (19). Similarly, we define a ω grid:

$$\omega_0 = -\frac{N}{2}\Delta\omega, \ldots, \omega_n = -\frac{N}{2} + n\Delta\omega \ldots, \omega_N = \frac{N}{2}\Delta\omega \tag{20}$$

where $\Delta\omega = \frac{1}{N\Delta x} = \frac{1}{b-a}$. Both grids have the same dimension $N+1$, Δx shrinks as $N \to +\infty$ while $\Delta\omega$ reduces for large values of $b-a$. For any x_j in the grid we can approximate the integral in (17) using the left Riemann summation, therefore, we have the approximation $\hat{f}_N(x_j)$:

$$\begin{aligned} \hat{f}_N(x_j) &= \Delta\omega \sum_{n=1}^{N} e^{2i\pi\left(-\frac{N}{2}\Delta\omega + (n-1)\Delta u\right)x_j}\varphi\left(-\pi N\Delta u + 2\pi(n-1)\right) \\ &= \Delta\omega e^{i\pi N\Delta\omega}\sum_{n=1}^{N} e^{-2i\pi(n-1)\Delta u(a+(j-1)\Delta x)}\varphi\left(-\pi N\Delta\omega + 2\pi(n-1)\right) \\ &= \Delta\omega e^{i\pi N\Delta\omega}\sum_{n=1}^{N} e^{\frac{-2i\pi(n-1)(j-1)}{N}}\varphi\left(-\pi N\Delta\omega + 2\pi(n-1)\right)e^{\frac{-2i\pi(n-1)a}{N}}. \end{aligned} \tag{21}$$

The last equality in (21) is due to the identity $\Delta\omega\Delta x = \frac{1}{N}$. To evaluate the summation in (21), we use the `FFT` algorithm and

$$\hat{f}_N(x_j) = \Delta\omega e^{i\pi N\Delta\omega}\texttt{FFT}\left[\varphi\left(-\pi N\Delta\omega + 2\pi(n-1)\right)e^{\frac{-2i\pi(n-1)a}{N}}\right]. \tag{22}$$

In this case, we have two sources of approximation errors that we can control using the arguments `up`, `low` and `N` in the `setLaw_th` constructor. `N` denotes the number of intervals in the grid used for the Left-Riemann summation while `up`, `low` are used to compute the step size of this grid.

Once the density function has been obtained using one of the three methods described above, it is possible to approximate the cumulative distribution function using the Left-Riemann summation computed on the grid in (19), therefore the cumulative distribution function $F(\cdot)$ for each x_j on the grid is determined as follows:

$$\hat{F}(x_j) = \sum_{x_k < x_j}\hat{f}_N(x_k)\,\Delta x. \tag{23}$$

In this way, we can construct a table that we can use internally in the `cdf` and `quantile` functions. Moreover, to evaluate the cumulative distribution function at any $x \in (x_{j-1}, x_j)$, we interpolate linearly its value using the couples $(x_{j-1}, \hat{F}(x_{j-1}))$ and $(x_j, \hat{F}(x_j))$. The random numbers can be obtained using the inversion sampling method.

We conclude this section with a numerical comparison among the three methods im-

plemented in **yuima**. To assess the precision of our code we use as a target the cumulative distribution function of a Student-t with $\nu = 3$ computed through the R function `pt` (R Core Team, 2024). To conduct this comparison, we construct three `yuima.th-objects` as displayed in the code snippet reported below[4].

```
# To instal YUIMA from Github repository
R> library(pak)
R> pak::pkg_install("yuimaproject/yuima")

##########
# Inputs #
##########
R> library(yuima)
R> nu <- 3
R> h <- 1 # step size for the interval time
R> up <- 10
R> low <- -10

# Support definition for variable x
R> x <- seq(low,up,length.out=100001)

# Definition of yuima.th-object
R> law_LAG <- setLaw_th(h = h, method = "LAG", up = up, low = low, N = 180) # Laguerre
R> law_COS <- setLaw_th(h = h, method = "COS", up = up, low = low, N = 180) # COS
R> law_FFT <- setLaw_th(h = h, method = "FFT", up = up, low = low, N = 180) # FFT

# Cumulative Distribution Function: we apply the cdf method to the yuima.th-object
R> Lag_time <- system.time(ycdf_LAG <- cdf(law_LAG, x, list(nu=nu))) # Laguerre
R> COS_time <- system.time(ycdf_COS<-cdf(law_COS, x, list(nu=nu))) # COS
R> FFT_time <- system.time(ycdf_FFT<-cdf(law_FFT, x, list(nu=nu))) # FFT
```

User can specify the numerical methods of the inversion of the characteristic function using the input `method`. This argument assumes three values: `"LAG"` for the Gauss-Laguerre quadrature, `"COS"` for the Cosine Series Expansion and `"FFT"` for the Fast Fourier Transform. Once the `yuima.th-object` has been constructed, its cumulative distribution function is computed by applying the **yuima** method[5] `cdf` that returns the cumulative distribution function for the `numeric` vector `x`. To enable a direct comparison with the outputs of the R functions `dt`, `pt`, `rt`, and `qt`, the **yuima** methods `dens`, `cdf`, `quantile`, and `rand` refer to the increments of a t-Lévy process Z_t, rather than the corresponding scaled t-Lévy process J_t. Specifically, when $t = 1$, the functions `pt` and `cdf` compute the same cumulative distribution function, as illustrated in Figure 2.

For all numerical approaches available for `cdf`, we have a good level of precision that is also confirmed by Table 1. As expected the fastest method is the `FFT` which seems to be also the most precise.

To further investigate this fact, we study the behaviour of the cumulative distribution function when h varies. For $h = 0.01$, we observe an oscillatory behaviour on tails for the `FFT` and `COS` while the Laguerre method seems to be more stable as shown in Figure 3. To have a fair comparison, we set `N = 180`, however we notice that the precision of `FFT` can be drastically improved by tuning the inputs `N`, `up` and `low`.

[4]We recommend to download the latest **yuima** version using the function `pk_install` available in **pak** Csárdi and Hester (2024).

[5]An object of `yuima.th-class` inherits the `dens` method for the density computation, `cdf` for the evaluation of the cumulative distribution function, `quantile` for the quantile function and `rand` for the generation of the random sample. We refer to Masuda et al. (2022) for the usage of these methods.

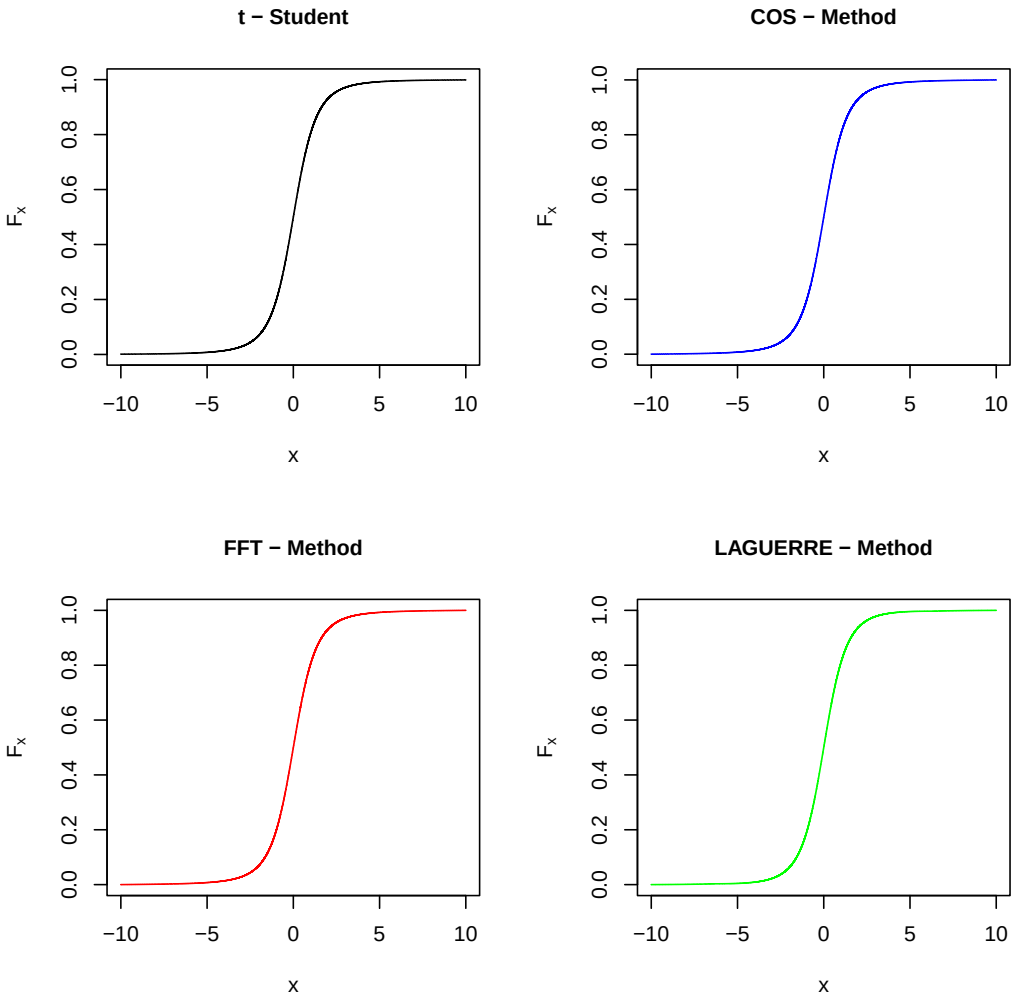


Figure 2: Cumulative distribution function comparison for a Student-t random variable with $\nu = 3$.

Table 1: Summary comparison of the distance between each method available in yuima and the cumulative distribution function obtained using the R function 'pt'.

Metric	COS	FFT	LAG
RMSE	2.10e-02	2.10e-02	0.064000
Max	3.20e-02	3.20e-02	0.085000
Min	7.88e-05	2.18e-05	0.000497
sec.	1.47e+00	4.00e-02	1.200000

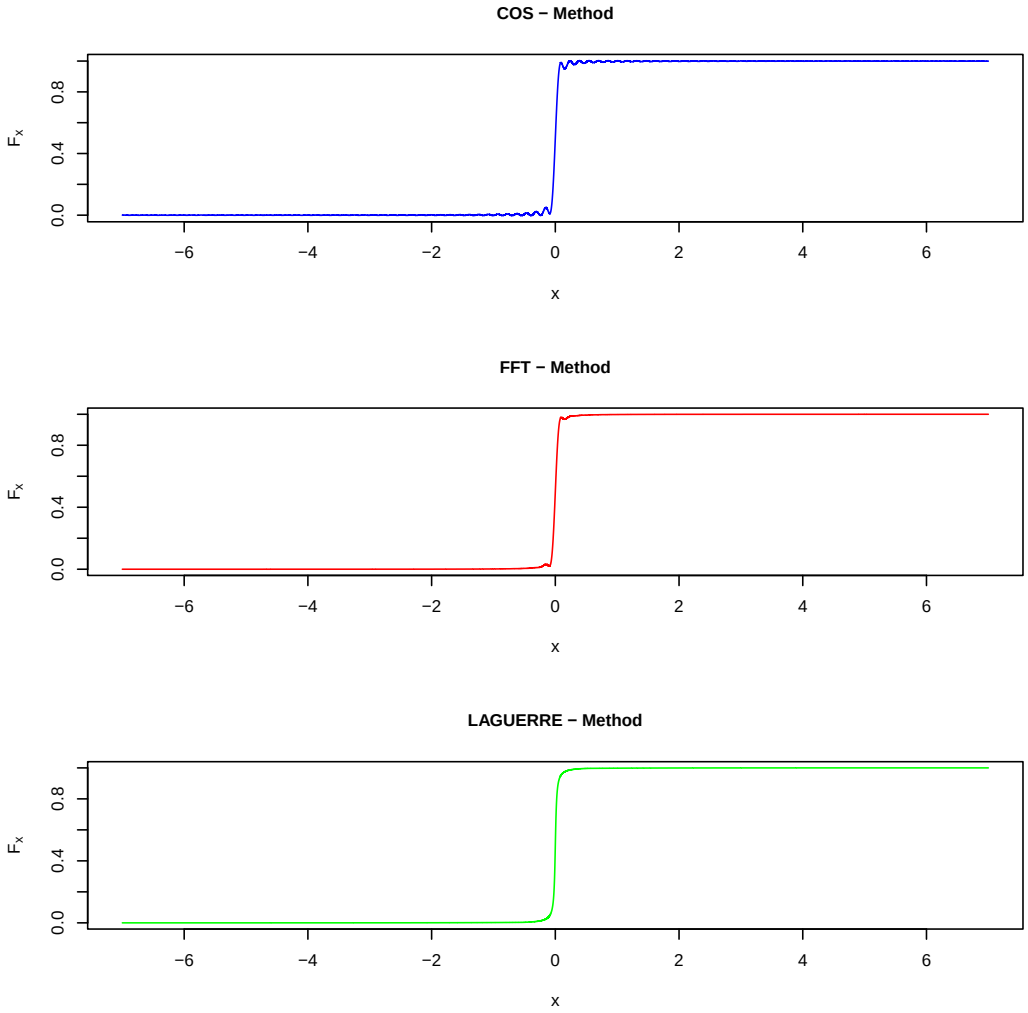


Figure 3: Cumulative distribution function comparison for a Student-t Lévy increment with $\nu = 3$ and $h = 0.01$.

4 Numerical examples

This section presents a series of numerical examples demonstrating the practical application of newly developed classes and methods within the Student-t Regression model. Specifically, we showcase the simulation and estimation of models where the regressors are determined by deterministic functions of time in the first example. The second example introduces integrated stochastic regressors. Lastly, we perform an analysis using real data in the final example.

4.1 Model with Deterministic Regressors

In this example, we consider two deterministic regressors and the dynamics of the model have the following form:

$$Y_t = \mu_1 \cos(5t) + \mu_2 \sin(t) + \sigma J_t \tag{24}$$

with the true values $(\mu_1, \mu_2, \sigma_0, \nu_0) = (5, -1, 3, 3)$. The estimation of the model (24) has been investigated in Masuda et al. (2024) where the empirical distribution of the studentized estimates has been discussed. To use the simulation method in **yuima** we have to write the dynamics of the regressors $X_{1,t} := \cos(5t)$ and $X_{2,t} =:$, that is:

$$\mathrm{d}\begin{bmatrix} X_{1,t} \\ X_{2,t} \end{bmatrix} = \begin{bmatrix} -5\sin(5t) \\ \cos(t) \end{bmatrix} \mathrm{d}t \tag{25}$$

with the initial condition:

$$X_{1,0} = 1,\ X_{2,0} = 0.$$

In the following, we show how to implement this model in **yuima**. In this example, we set $h_n = 1/50$, and thus the number of the observations n over unit time is 50. We also set the terminal time of the whole observations $T_n = 50$ and that of the partial observations $B_n = 15$.

```
R> library(yuima)
##########
# Inputs #
##########
# Inputs for Fourier Inversion
R> method_Fourier <- "FFT"; up <- 6; low <- -6; N <- 2^17; N_grid <- 60000

# Inputs for the sample grid time
R> Factor <- 1
R> Factor1 <- 1
R> initial <- 0; Final_Time <- 50 * Factor; h <- 0.02/Factor1
```

We notice that the variable `Factor1` controls the step size of the time grid. Indeed for different values of this quantity, we have a different value for h for example `Factor1 = 1, 2, 4, 7.2` corresponds `h = 0.02, 0.01, 0.005, 1/365`.

The first step is the definition of an object of `yuima.th-class` using the constructor `setLaw_th`. This object contains the random number generator, the density function, the cumulative distribution function and the quantile function for constructing the increments of a Student-t Lévy process.

```
#####################################
# Example 1: Deterministic Regressors #
#####################################

R> mu1 <- 5; mu2 <- -1; scale <- 3; nu <- 3 # Model Parameters
```

```
# Model Definition
R> law1 <- setLaw_th(method = method_Fourier, up = up, low = low,
  N = N, N_grid = N_grid) # yuima.th

R> class(yuima_law)
[1] "yuima.th"
attr(,"package")
[1] "yuima"
```

The next step is to define the dynamics of the regressors described in (25). This set of differential equations is defined in **yuima** using the standard constructor `setModel`. Once an object containing the mathematical description of the regressors has been defined, we use `setLRM` to obtain the `yuima.LevyRM`. In the following, we report the command lines for the definition of the Student Lévy Regression Model in (24).

```
R> regr1 <- setModel(drift = c("-5*sin(5*t)", "cos(t)"), diffusion = matrix("0",2,1),
     solve.variable = c("X1","X2"), xinit = c(1,0)) # Regressors definition

R> Mod1 <- setLRM(unit_Levy = law1, yuima_regressors = regr1) # t-Regression Model

R> class(Mod1)
[1] "yuima.LevyRM"
attr(,"package")
[1] "yuima"
```

Using the object `Mod1` we simulate a trajectory of the model in (24) using the **yuima** method `simulate`.

```
# Simulation
R> samp <- setSampling(initial, Final_Time, n = Final_Time/h)
R> true.par <- unlist(list(mu1 = mu1, mu2 = mu2, sigma0 = scale, nu = nu))

R> set.seed(1)

R> sim1 <- simulate(Mod1, true.parameter = true.par, sampling = samp)
```

Figure 4 reports the simulated sample paths for the regressors $X_{1,t}$, $X_{2,t}$ and the Student Lévy Regression model Y_t.

The next step is to study the behaviour of the two-step estimation procedure described in Section 2. To run this procedure, we use the method `estimation_LRM` and then we initialize randomly the optimization routine as shown in the following command lines.

```
# Estimation
R> lower <- list(mu1 = -10, mu2 = -10,  sigma0 = 0.1)
R> upper <- list(mu1 = 10, mu2 = 10, sigma0 = 10.01)

R> start <- list(mu1 = runif(1, -10, 10), mu2 = runif(1, -10, 10),
+    sigma0 = runif(1, 0.01, 4))

R> Bn <- 15*Factor

R> est1 <- estimation_LRM(start = start, model = sim1, data = sim1@data,
+    upper = upper, lower = lower, PT = Bn)

R> class(est1)
```

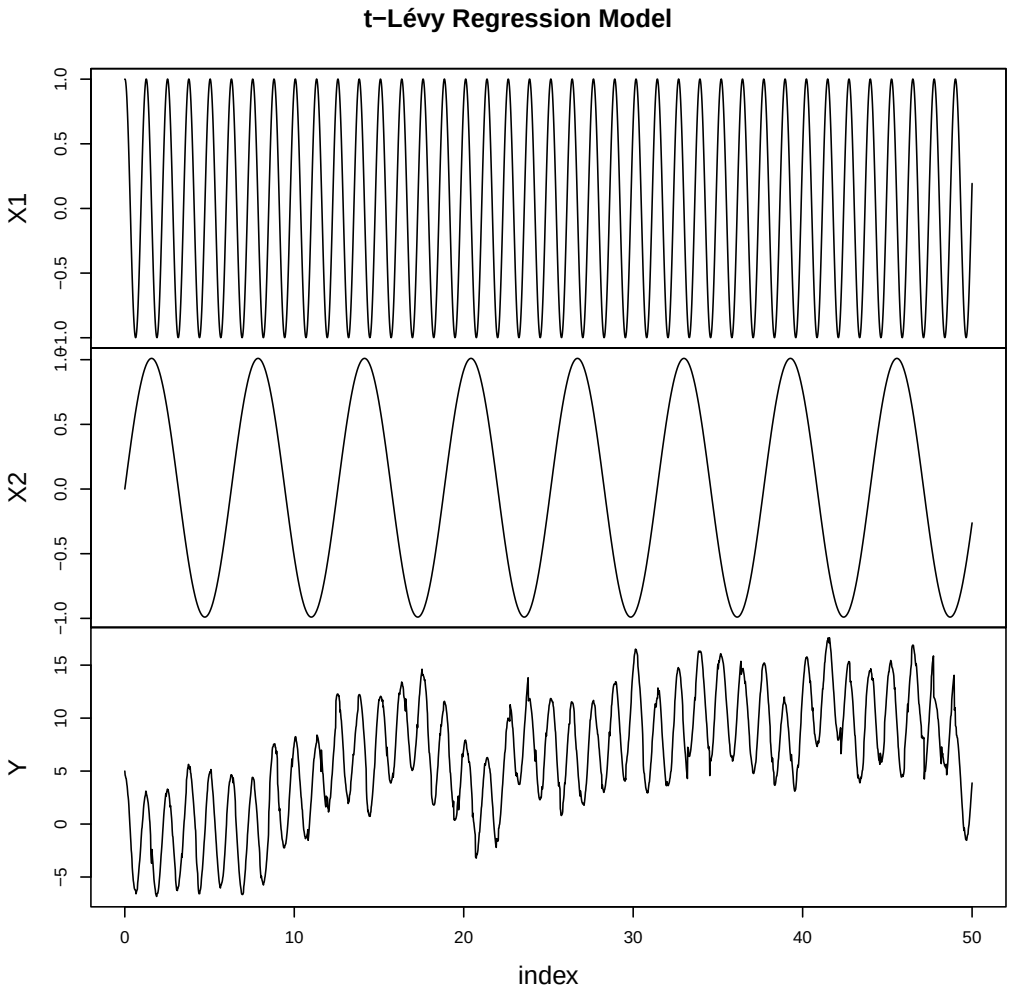


Figure 4: Simulated Trajectory of the Student Lévy Regression model defined in eqrefeq:Regr1.

```
[1] "yuima.qmle"
attr(,"package")
[1] "yuima"

R> summary(est1)
Quasi-Maximum likelihood estimation

Call:
estimation_LRM(start = start, model = sim1, data = sim1@data,
    upper = upper, lower = lower, PT = Bn)

Coefficients:
        Estimate Std. Error
mu1     5.029555 0.03538355
mu2    -1.128905 0.18026088
sigma0  2.425147 0.12523404
nu      2.735344 0.47457435

-2 log L: 3236.535 73.3833
```

It is also possible to construct the dataset without the method `simulate`. This result can be achieved by constructing an object of `yuima.th-class` that represents the increment of the Student-t Lévy process on the time interval with length h.

```
# Dataset construction without YUIMA
R> time_t <- index(get.zoo.data(sim1@data)[[1]])
R> X1 <- zoo(cos(5*time_t), order.by = time_t)
R> X2 <- zoo(sin(time_t), order.by = time_t)
R> law1_h <- setLaw_th(h = h, method = method_Fourier, up = up, low = low,
     N = N, N_grid = N_grid)

R> print(c(law1_h@h,law1@h))
[1] 0.02 1.00
```

The object `law1_h` refers to the Student-t Lévy increment over an interval with length 0.02 as it can be seen looking at the slot `...@h`.

```
R> set.seed(1)
R> names(nu)<- "nu"
# Simulation a t-Levy process Z_t.
R> Z_t <- zoo(cumsum(c(0,rand(law1_h, Final_Time/h,nu))), order.by = time_t)
# Sample path a scaled t-Levy process J_t
R> J_t <- Z_t/sqrt(nu)
# Sample path of a t-Levy regression model
R> Y <- mu1 * X1 + mu2 * X2 + scale * J_t
R> data1_a <- merge(X1, X2, Y)
```

We observe that, as highlighted in Section 3.1, the **yuima** method `rand` returns the increments of a t-Lévy process Z_t. To obtain the trajectory of the scaled t-Lévy process J_t in (24), the process Z_t needs to be rescaled by the factor $\sqrt{\nu}$. The estimation is performed by means of the method `estimation_LRM` as in the previous example.

```
R> est1_a <- estimation_LRM(start = start, model = Mod1, data = setData(data1_a),
+    upper = upper, lower = lower, PT = Bn)
R> summary(est1_a)
Quasi-Maximum likelihood estimation
```

```
Call:
estimation_LRM(start = start, model = Mod1, data = setData(data1_a),
    upper = upper, lower = lower, PT = Bn)

Coefficients:
        Estimate Std. Error
mu1     4.987778 0.03642823
mu2    -1.164138 0.18558422
sigma0  2.495930 0.12888927
nu      2.407683 0.41221590

-2 log L: 3232.784 85.32438
```

Using the result stored in the `summary` we can construct a confidence interval using the command lines reported below.

```
R> info_sum <- summary(est1_a)@coef
R> alpha <- 0.025
R> Confidence_Int_95 <- rbind(info_sum[,1]+info_sum[,2]*qnorm(alpha),
+     info_sum[,1]+info_sum[,2]*qnorm(1-alpha),unlist(true.par),
+     info_sum[,1])
R> rownames(Confidence_Int_95) <- c("LB","UB","True_par","Est_par")
R> print(Confidence_Int_95, digit = 4)
           mu1      mu2  sigma0     nu
LB       4.916  -1.5279   2.243  1.600
UB       5.059  -0.8004   2.749  3.216
True_par 5.000  -1.0000   3.000  3.000
Est_par  4.988  -1.1641   2.496  2.408
```

Based on the aforementioned examples, the estimated value for the σ parameter deviates from its true value. To further investigate this observation, we conducted a comparative analysis using the three integration methods discussed in Section 3.1. This exercise was repeated for three different values of $h = 0.01, 0.005$ and $\frac{1}{365}$. To ensure a fair comparison among the Laguerre, Cosine Series expansion, and Fast Fourier Transform methods, we set the argument `N = 180`. The obtained results are presented in Table 2.

Table 2: Estimated parameters for $h = 0.01, 0.005, 1/365$ and different integration methods. The number of points for the inversion of the characteristic function is $N = 180$. The parenthesis shows the asymptotic standard error. The last row reports the seconds necessary for the simulation of a sample path.

	Laguerre Method			COS Method			FFT Method		
Parameter	Lag_0.01	Lag_0.005	Lag_1_365	COS_0.01	COS_0.005	COS_1_365	FFT_0.01	FFT_0.005	FFT_1_365
$\hat{\mu_1}$	4.968	5.019	4.976	4.939	5.065	4.875	4.934	5.069	4.876
	(0.029)	(0.020)	(0.014)	(0.041)	(0.049)	(0.054)	(0.042)	(0.049)	(0.054)
$\hat{\mu_2}$	-1.065	-1.044	-0.914	-1.192	-1.195	-0.555	-1.179	-1.178	-0.528
	(0.146)	(0.104)	(0.074)	(0.210)	(0.247)	(0.277)	(0.213)	(0.248)	(0.277)
$\hat{\sigma_0}$	2.770	2.794	2.657	3.993	6.654	10.010	4.055	6.674	10.010
	(0.101)	(0.072)	(0.051)	(0.145)	(0.172)	(0.192)	(0.148)	(0.172)	(0.193)
$\hat{\nu}$	3.462	3.281	3.067	2.827	2.223	2.227	5.749	8.310	15.175
	(0.615)	(0.580)	(0.538)	(0.492)	(0.377)	(0.378)	(1.063)	(1.571)	(2.940)
sec.	2.07	2.02	2.07	1.10	1.24	1.34	0.15	0.24	0.39

In the Laguerre method, we observe that reducing the step size h leads to improved estimates of ν, however looking at the asymptotic standard error, the estimates for σ_0 seem to maintain the bias. As for the remaining methods, the estimates exhibit unsatisfactory performance due to the imposed restriction of `N = 180` (that is the maximum value allowed for the Laguerre quadrature in the evaluation of the density in (8)). This outcome is not unexpected, given the Laguerre method's ability to yield a valid distribution even with a

relatively small value of N, as demonstrated in Figure 3. Nevertheless for the COS and the FFT, it is possible to enhance the results by increasing the value of the argument N that yields a more accurate result in the simulation of the sample path for the model described in (24).

Table 3: Estimated parameters for $N = 1000, 5000, 10^4$ in the COS and the FFT methods. The step size h is $1/365$. The parenthesis shows the asymptotic standard error. The last row reports the seconds necessary for the simulation of a sample path.

	COS Method			FFT Method		
Parameter	COS_1000	COS_5000	COS_10000	FFT_1000	FFT_5000	FFT_10000
μ_1	4.968	4.975	4.975	4.968	4.975	4.975
	(0.034)	(0.016)	(0.016)	(0.035)	(0.306)	(0.016)
μ_2	-0.887	-0.909	-0.909	-0.886	-0.908	-0.908
	(0.174)	(0.082)	(0.082)	(0.177)	(0.022)	(0.082)
σ_0	3.300	2.964	2.964	3.363	2.963	2.964
	(0.064)	(0.057)	(0.057)	(0.065)	(0.057)	(0.057)
ν	2.713	2.765	2.765	3.279	2.760	2.702
	(0.470)	(0.480)	(0.480)	(0.579)	(0.479)	(0.468)
sec.	3.49	14.83	33.70	0.38	0.44	0.45

Table 3 shows the estimated parameters for the varying value of N and $h = 1/365$. As expected increasing the precision in the quadrature improved estimates for both methods. Notably for $N \geq 5000$, all estimates fall within the asymptotic confidence interval at the 95% level.

This example suggests a possible strategy for selecting the optimal N for the grid defined in (20), particularly when $h = \frac{1}{365}$ (commonly used for daily data). The strategy consists of the following steps:

- Estimate the model using Laguerre quadrature with the maximum allowable polynomial order (180).
- Using the estimates from the previous step as an initial guess, re-estimate the model using the FFT method with $N = 500$. If the resulting estimates are sufficiently close to those obtained in the previous step, stop.
- Otherwise, using the Laguerre estimates as the initial guess, double the number of intervals in the grid in (20) and re-estimate the model using the FFT. Repeat this process until two successive estimates are sufficiently close. The Euclidean norm can be used to determine whether the estimates are sufficiently close.

4.2 Model with integrated stochastic regressors

In this section, we consider two examples whose regressors are stochastic. Moreover, to satisfy the regularity conditions, the regressors are supposed to be an integrated version of stochastic processes.

Example 1 In this example, we consider the following continuous time regression model with a single regressor:

$$Y_t = \mu \int_0^t X_u du + \sigma J_t, \quad J_1 \sim t_\nu, \tag{26}$$

with the true values $(\mu_0, \sigma_0, \nu_0) = (-3, 3, 2.5)$, and the process X is supposed to be the Lévy driven Ornstein-Uhlenbeck process defined as:

$$dX_t = -X_t dt + 2dZ_t,$$

where the driving noise Z is the Lévy process with $Z_1 \sim \text{NIG}(1, 0, 1, 0)$. The normal inverse Gaussian (NIG) random variable is defined as the normal-mean variance mixture

of the inverse Gaussian random variable, and the probability density function of $Z_t \sim \mathrm{NIG}(\alpha, \beta, \delta t, \mu t)$ is given by

$$x \mapsto \frac{\alpha\delta t \exp\{\delta t\sqrt{\alpha^2-\beta^2} + \beta(x-\mu t)\} K_1(\alpha\psi(x;\delta t,\mu t))}{\pi\psi(x;\delta t,\mu t)}$$

where $\alpha^2 := \gamma^2 + \beta^2$ and $\psi(x;\delta t,\mu t) := \sqrt{(\delta t)^2 + (x-\mu t)^2}$. More detailed theoretical properties of the NIG-Lévy process are given for example in **?**.

For the simulation by **yuima**, we formally introduce the following system:

$$d\begin{bmatrix} X_{1,t} \\ X_{2,t} \end{bmatrix} = \begin{bmatrix} X_{2,t} \\ -X_{2,t} \end{bmatrix} dt + \begin{bmatrix} 0 \\ 2 \end{bmatrix} dZ_t, \tag{27}$$

with the initial condition:

$$X_{1,0} = 0,\ X_{2,0} = 0.$$

The process $X_{1,t}$ corresponds to the regressor $\int_0^t X_u du$ in the regression model (26). Due to the specification of the new **yuima** function, we will additionally construct the "full" model:

$$Y_t = \mu_1 X_{1,t} + \mu_2 X_{2,t} + \sigma J_t, \quad J_1 \sim t_\nu, \tag{28}$$

and simulate the trajectory of Y with $\mu_2 = 0$. After that, we will estimate the parameters (μ_1, σ, ν) by the original model (26). From now on, we show the implementation of this model in **yuima**. The sampling setting is unchanged from the previous example code: $h_n = 1/50, n = 1/h_n = 50, T_n = 50, B_n = 15$.

```
R> library(yuima)
##########
# Inputs #
##########
# Inputs for Fourier Inversion
R> method_Fourier = "FFT"; up = 6; low = -6; N = 2^17; N_grid = 60000

# Inputs for the sample grid time
R> Factor <- 1
R> Factor1 <- 1
R> initial <- 0; Final_Time <- 50 * Factor; h <- 0.02/Factor1
```

The role of the variable `Factor1` is the same as in the previous section. By using the constructor `setLaw_th`, we define `yuima.th-class` in a similar manner. After that, we construct the "full" model (28) by the constructors `setModel` and `setLRM`. A trajectory of Y and its regressors can be simulated by the YUIMA method `simulate`.

```
#####################################################
# Regressor: Integrated NIG-Levy driven OU process #
#####################################################
R> mu1 <- -3; mu2 <- 0;  scale <- 3; nu <- 2.5 # Model Parameters

# Model Definition
R> lawILOU <- setLaw_th(method = method_Fourier, up = up, low = low, N = N,
+    N_grid = N_grid) # yuima.th

R> regrILOU <- setModel(drift = c("X2", "-X2"), jump.coeff = c("0", "2"),
+    solve.variable = c("X1","X2"), xinit = c(0,0), measure.type = "code",
+    measure = list(df = "rNIG(z, 1, 0, 1, 0)")) # Regressors definition

# t-Regression model
```

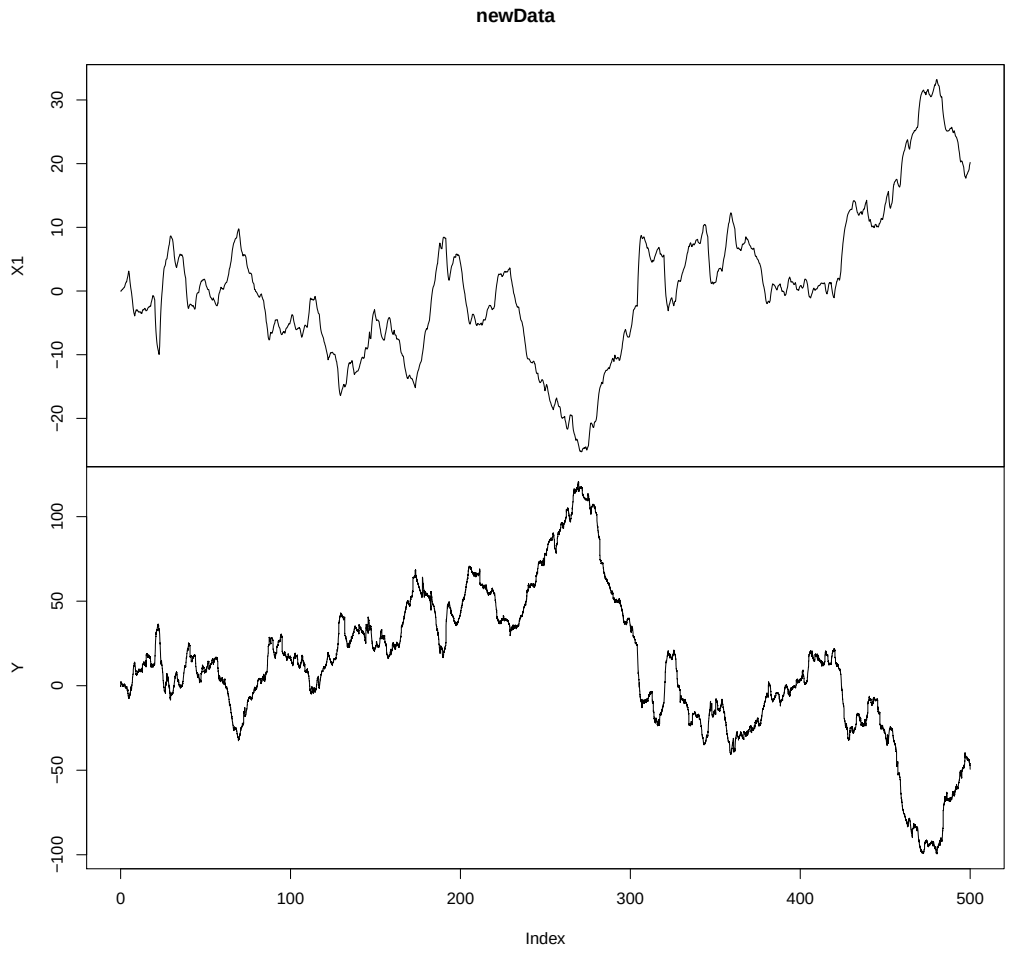


Figure 5: Simulated Trajectory of the Student Lévy Regression model defined in eqrefeq:yuex1

```
R> ModILOU <- setLRM(unit_Levy = lawILOU, yuima_regressors = regrILOU)

# Simulation
R> sampILOU <- setSampling(initial, Final_Time, n = Final_Time/h)
R> true.parILOU <- unlist(list(mu1 = mu1, mu2 = mu2, sigma0 = scale, nu = nu))
R> set.seed(1)
R> simILOU <- simulate(ModILOU, true.parameter = true.parILOU, sampling = sampILOU)
```

Next we extract the trajectories of Y and X_1 from the `yuima.LevyRM-class`: `simILOU` and construct the new model for the estimation of the parameters. Figure 5 shows the simulated sample paths for the regressor and the response process Y.

```
# Data extraction
R> Dataset <- get.zoo.data(simILOU)
R> newData <- Dataset[-2]
R> newData <- merge(newData$X1, newData$Y)
R> colnames(newData) <- c("X1","Y")
R> plot(newData)

# Define the model for estimation
R> regrILOU1 <- setModel(drift = c("0"), diffusion = matrix(c("0"),1,1),
+    solve.variable = c("X1"), xinit = c(0)) # Regressors definition
```

```
# t-Regression model
R> ModILOU1 <- setLRM(unit_Levy = lawILOU, yuima_regressors = regrILOU1)

# Estimation
R> lower1 <- list(mu1 = -10,  sigma0 = 0.1)
R> upper1 <- list(mu1 = 10, sigma0 = 10.01)
R> startILOU1 <- list(mu1 = runif(1, -10, 10), sigma0 = runif(1, 0.01, 4))

R> Bn <- 15*Factor
R> Data1 <- setData(newData)
R> estILOU1 <- estimation_LRM(start = startILOU1, model = ModILOU1, data = Data1,
+    upper = upper1, lower = lower1, PT = Bn)
R> summary(estILOU1)
Quasi-Maximum likelihood estimation

Call:
estimation_LRM(start = startILOU1, model = ModILOU1, data = Data1,
    upper = upper1, lower = lower1, PT = Bn)

Coefficients:
        Estimate Std. Error
mu1    -2.959621 0.02112734
sigma0  2.778659 0.03208519
nu      2.404843 0.13018410

-2 log L: 69711.32 854.3838
```

Example 2 In this example, we consider the following regression model:

$$Y_t = \mu_1 \int_0^t V_{1,u} du + \mu_2 \int_0^t V_{2,u} du + \sigma J_t, \quad J_1 \sim t_\nu \tag{29}$$

where the process $X = (V_1, V_2)$ satisfy

$$dV_{1,t} = \frac{1}{\epsilon}(V_{1,t} - V_{1,t}^3 - V_{2,t} + s)dt, \tag{30}$$

$$dV_{2,t} = (\gamma V_{1,t} - V_{2,t} + \alpha)dt + \sigma' dw_t, \tag{31}$$

with

$$(\epsilon, s, \gamma, \alpha, \sigma') = (1/3, 0, 3/2, 1/2, 2),$$

and standard Wiener process w. We set the true values

$$(\mu_{1,0}, \mu_{2,0}, \sigma_0, \nu_0) = (8, -4, 8, 3).$$

The process V is the so-called stochastic FitzHugh-Nagumo process which is a classical model for describing a neuron; V_1 expresses the membrane potential of the neuron and V_2 represents a recovery variable. Its theoretical properties such as hypoellipticity, and Feller and mixing properties are well summarized in León and Samson (2018). The paper also provides a nonparametric estimator of the invariant density and spike rate. Similarly, other integrated degenerate diffusion can be considered as regressors. Its ergodicity for the regularity conditions is studied for example in Wu (2001). For the statistical inference for degenerate diffusion processes, we refer to Ditlevsen and Samson (2019) and Gloter and Yoshida (2021).

For the implementation of the regression model (29) on YUIMA, we formally consider

the following dynamics:

$$d\begin{bmatrix} V_{1,t} \\ V_{2,t} \\ V_{3,t} \\ V_{4,t} \end{bmatrix} = \begin{bmatrix} V_{3,t} \\ V_{4,t} \\ 3(V_{3,t} - V_{3,t}^3 - V_{4,t}) \\ \frac{3}{2}V_{3,t} - V_{4,t} + \frac{1}{2} \end{bmatrix} dt + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 2 \end{bmatrix} dw_t, \tag{32}$$

with the initial condition:

$$V_{1,0} = 0,\ V_{2,0} = 0,\ V_{3,0} = 0,\ V_{4,0} = 0.$$

The first and second elements correspond to the regressor in (29). As in the previous example, we first simulate data by the "full" model defined as:

$$Y_t = \mu_1 V_{1,t} + \mu_2 V_{2,t} + \mu_3 V_{3,t} + \mu_4 V_{4,t} + \sigma J_t, \quad J_1 \sim t_\nu, \tag{33}$$

with $\mu_3 = \mu_4 = 0$, and after that, we extract the simulated data and estimate the parameters based on the original regression model (29). Below we show how to implement on **yuima**. In this example, we set $h_n = 1/200, n = 1/h_n = 200, T_n = 1000, B_n = 300$. The values of them are controlled by the variables `Factor` and `Factor1` in the example code.

```
R> library(yuima)
##########
# Inputs #
##########
# Inputs for Fourier Inversion
R> method_Fourier = "FFT"; up = 6; low = -6; N = 2^17; N_grid = 60000

# Inputs for the sample grid time
R> Factor <- 20
R> Factor1 <- 4
R> initial <- 0; Final_Time <- 50 * Factor; h <- 0.02/Factor1

#############################################################
# Regressor: Integrated stochastic FitzHugh-Nagumo process #
#############################################################

R> mu1 <- 8; mu2 <- -4; mu3 <- 0; mu4 <- 0;  scale <- 8; nu <- 3 # Model Parameters
# Model Definition
R> lawFN <- setLaw_th(method = method_Fourier, up = up, low = low,
+    N = N, N_grid = N_grid) # yuima.th

R> regrFN <- setModel(drift = c("V3", "V4", "3*(V3-V3^3-V4)", "1.5*V3-V4+0.5"),
+    diffusion = matrix(c("0", "0", "0", "2"),4,1),
+    solve.variable = c("V1","V2", "V3", "V4"),
+    xinit = c(0,0,0,0)) # Regressors definition

R> ModFN <- setLRM(unit_Levy = lawFN, yuima_regressors = regrFN) #  t-regression model

# Simulation
R> sampFN <- setSampling(initial, Final_Time, n = Final_Time/h)
R> true.parFN <- unlist(list(mu1 = mu1, mu2 = mu2, mu3 = mu3, mu4 = mu4,
+    sigma0 = scale, nu = nu))
R> set.seed(12)
R> simFN <- simulate(ModFN, true.parameter = true.parFN, sampling = sampFN)
```

In this example, both `Factor` and `Factor1` are larger than the previous example, and hence it takes a higher computational load than the previous one. Figure 6 shows the

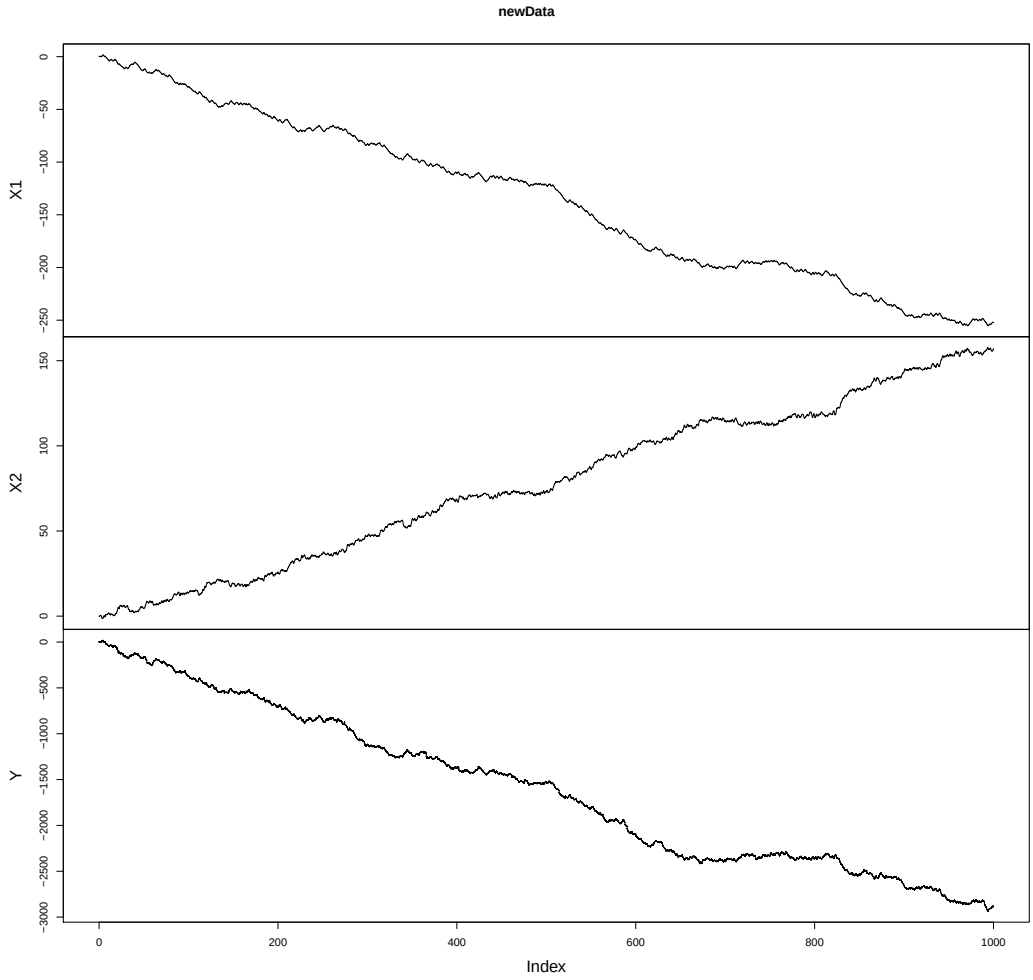


Figure 6: Simulated Trajectory of the Student Lévy Regression model defined in eqrefeq:yuex2.

simulated sample paths for the regressor and the Student Lévy Regression model Y. Now we move on to the estimation phase.

```
R> Dataset <- get.zoo.data(simFN)
R> newData <- Dataset[-c(3,4)]
R> newData <- merge(newData$X1,newData$X2, newData$Y)
R> colnames(newData) <- c("X1","X2", "Y")
R> plot(newData)

# Define the model for estimation
R> regrFN1 <- setModel(drift = c("0", "0"), diffusion = matrix(c("0", "0"),2,1),
+    solve.variable = c("X1", "X2"), xinit = c(0,0)) # Regressors definition
R> ModFN1 <- setLRM(unit_Levy = lawFN, yuima_regressors = regrFN1) # t-Regression model.

# Estimation
R> lower1 <- list(mu1 = -10, mu2 = -10,  sigma0 = 0.1)
R> upper1 <- list(mu1 = 10, mu2 =10, sigma0 = 10.01)
R> startFN1 <- list(mu1 = runif(1, -10, 10), mu2 = runif(1, -10, 10),
+    sigma0 = runif(1, 0.01, 4))

R> Bn <- 15*Factor
R> Data1 <- setData(newData)
```

```
R> estFN1 <- estimation_LRM(start = startFN1, model = ModFN1, data = Data1,
+    upper = upper1, lower = lower1, PT = Bn)
R> summary(estFN1)
Quasi-Maximum likelihood estimation

Call:
estimation_LRM(start = startFN1, model = ModFN1, data = Data1,
    upper = upper1, lower = lower1, PT = Bn)

Coefficients:
        Estimate Std. Error
mu1     8.041838 0.04895791
mu2    -4.041518 0.04495255
sigma0  7.712157 0.04452616
nu      3.020972 0.11837331

-2 log L: 403960.9 1291.516
```

4.3 Real data regressors

In this section, we show how to use the **yuima** package for the estimation of a Student Lévy Regression model in a real dataset. We consider a model where the daily price of the Standard and Poor 500 Index is explained by the VIX index and two currency rates: the YEN-USD and the Euro Usd rates. The dataset was provided by `Yahoo.finance` and ranges from December 14th 2014 to May 12th 2023. We downloaded the data using the function `getSymbol` available in `quantmod` library. To consider a small value for the step size h we estimate our model every month $h = 1/30$. We report below the code for storing the market data in an object of `yuima.data-class`

```
R> library(yuima)
R> library(quantmod)
R> getSymbols("^SPX", from = "2014-12-04", to = "2023-05-12")
# Regressors
R> getSymbols("EURUSD=X", from = "2014-12-04", to = "2023-05-12")
R> getSymbols("VIX", from = "2014-12-04", to = "2023-05-12")
R> getSymbols("JPYUSD=X", from = "2014-12-04", to = "2023-05-12")

R> SP <- zoo(x = SPX$SPX.Close, order.by = index(SPX$SPX.Close))
R> Vix <- zoo(x = VIX$VIX.Close/1000, order.by = index(VIX$VIX.Close))
R> EURUSD <- zoo(x = `EURUSD=X`[,4], order.by = index(`EURUSD=X`))
R> JPYUSD <- zoo(x = `JPYUSD=X`[,4], order.by = index(`JPYUSD=X`))
R> Data <- na.omit(na.approx(merge(Vix, EURUSD, JPYUSD, SP)))
R> colnames(Data) <- c("VIX", "EURUSD", "JPYUSD", "SP")
R> days <- as.numeric(index(Data))-as.numeric(index(Data))[1]

# equally spaced grid time data
R> Data <- zoo(log(Data), order.by = days)
R> Data_eq <- na.approx(Data, xout = days[1] : tail(days,1L))
R> yData <- setData(zoo(Data_eq, order.by = index(Data_eq)/30)) # Data on monthly basis
```

We decided to work with log-price (see `Data` variable in the above code) to have quantities defined on the same support of a Student-t Lévy process, i.e.: the real line. Since the estimation method in **yuima** requires that the data are observed on an equally spaced grid time, we interpolate linearly to estimate possible missing data to get a log price for each day. Figure 7 shows the trajectory for each financial series.

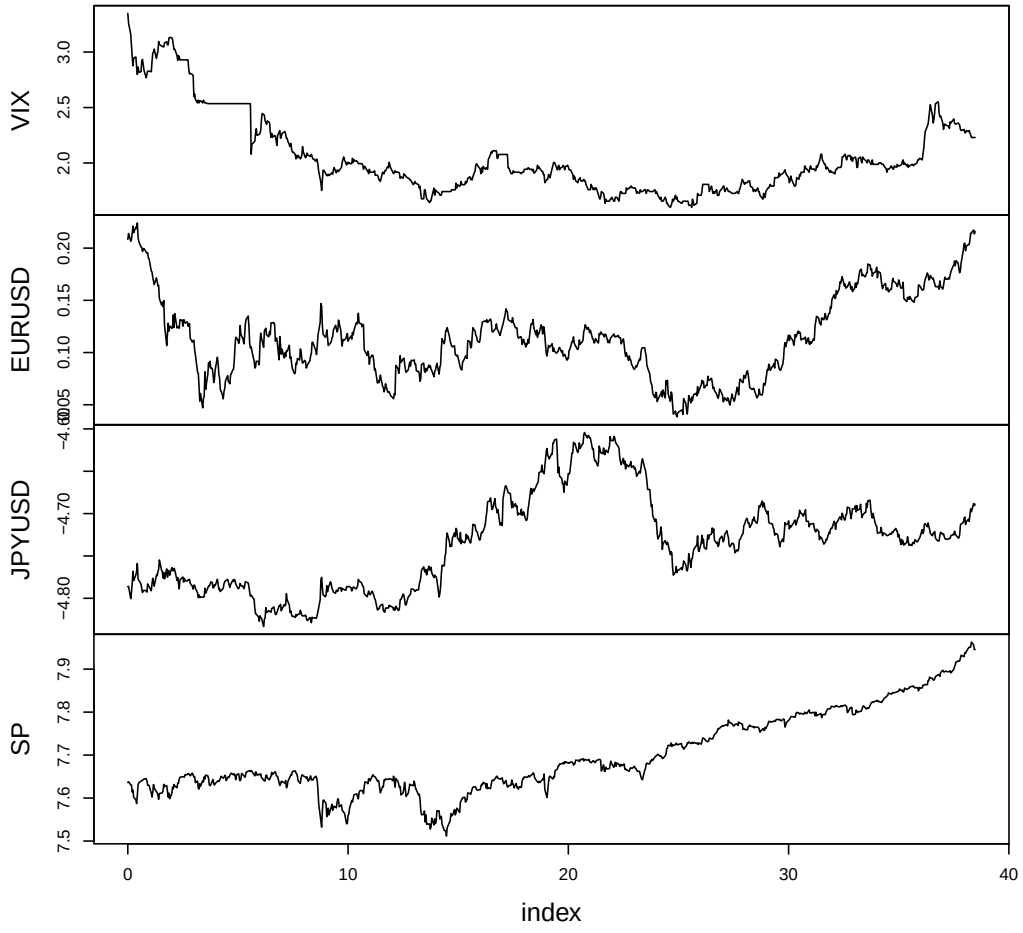


Figure 7: Financial data observed on a monthly equally grid time ranging from December 14th 2014 to May 12th 2023.

To estimate the model we perform the same steps discussed in the previous examples and we report below the code for reproducing our result.

```
# Inputs for integration in the inversion formula
R> method_Fourier <- "FFT"; up <- 6; low <- -6; N <- 10000; N_grid <- 60000

# Model Definition
R> law <- setLaw_th(method = method_Fourier, up = up, low = low, N = N,
+    N_grid = N_grid)
R> regr <- setModel(drift = c("0", "0", "0"), diffusion = matrix("0",3,1),
+    solve.variable = c("VIX", "EURUSD", "JPYUSD"), xinit = c(0,0,0))
Mod <- setLRM(unit_Levy = law, yuima_regressors = regr, LevyRM = "SP")

# Estimation
R> lower <- list(mu1 = -100, mu2 = -200, mu3 = -100,  sigma0 = 0.01)
R> upper <- list(mu1 = 100, mu2 = 100, mu3 = 100, sigma0 = 200.01)
R> start <- list(mu1 = runif(1, -100, 100), mu2 = runif(1, -100, 100),
+    mu3 = runif(1, -100, 100), sigma0 = runif(1, 0.01, 100))
R> est <- estimation_LRM(start = start, model = Mod, data = yData, upper = upper,
+    lower = lower, PT = floor(tail(index(Data_eq)/30,1L)/2))
R> summary(est)

Quasi-Maximum likelihood estimation

Call:
estimation_LRM(start = start, model = Mod, data = yData, upper = upper,
    lower = lower, PT = floor(tail(index(Data_eq)/30, 1L)/2))

Coefficients:
           Estimate  Std. Error
mu1     0.000335328 0.004901559
mu2    -0.062235188 0.033027232
mu3     0.016291220 0.039382559
sigma0  0.082031691 0.004859138
nu      3.062728822 0.616466913

-2 log L: -1506.067 48.17592
```

5 Conclusion

In this paper, we have presented classes and methods for a t-Lévy regression model. In particular, the simulation and the estimation algorithm have been introduced from a computational point of view. Moreover three different algorithms have been developed for computing the density, the cumulative distribution and the quantile functions and the random number generator of the t-Lévy increments defined over a non-unitary interval. These latter methods can be also used in any stochastic model available in **yuima** for the definition of the underlying noise.

Based on our simulated data, for the estimation of the degrees of freedom the Laguerre method seems to be more accurate. However, due to the restriction on the number of roots used in the evaluation of the integral, we notice a bias on the estimation of the scale parameter. Such bias seems to be observable also in the other implemented methods when the same number of points is used in the evaluation of the integral for the inversion of the characteristic function. However for the COS and the FFT methods, the numerical precision can be improved. In particular, it is possible to ensure that the estimates fall in the asymptotic confidence interval at 95% level even if the estimates of the degrees of freedom

in the simplest model (first example) seem to underestimate the true value. As concerns computational time, the FFT methods with a sufficiently large computational precision ($N \geq 5000$) and h sufficiently small seems to be an acceptable compromise.

Acknowledgment

This work is supported by JST CREST Grant Number JPMJCR2115, Japan and by the MUR-PRIN2022 Grant Number 20225PC98R, Italy CUP codes: H53D23002200006 and G53D23001960006.

References

S. Asmussen and J. Rosiński. Approximations of small jumps of lévy processes with a view towards simulation. *Journal of Applied Probability*, 38(2):482–493, 2001. URL http://www.jstor.org/stable/3215901. [p58]

C. Berg and C. Vignat. On some results of Cufaro Petroni about Student t-processes. *Journal of Physics A: Mathematical and Theoretical*, 41(26):265004, 2008. doi: 10.1088/1751-8113/41/26/265004. [p57]

A. Brouste, M. Fukasawa, H. Hino, S. Iacus, K. Kamatani, Y. Koike, H. Masuda, R. Nomura, T. Ogihara, Y. Shimuzu, M. Uchida, and N. Yoshida. The yuima project: A computational framework for simulation and inference of stochastic differential equations. *Journal of Statistical Software*, 57(4):1–51, 2014. doi: 10.18637/jss.v057.i04. URL https://www.jstatsoft.org/index.php/jss/article/view/v057i04. [p56, 59]

L. V. Castro-Porras, R. Rojas-Martínez, C. A. Aguilar-Salinas, O. Y. Bello-Chavolla, C. Becerril-Gutierrez, and C. Escamilla-Nuñez. Trends and age-period-cohort effects on hypertension mortality rates from 1998 to 2018 in Mexico. *Scientific Reports*, 11(1):17553, 2021. URL https://pubmed.ncbi.nlm.nih.gov/34475436/. [p56]

J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, 19(90):297–301, 1965. doi: https://doi.org/10.2307/2003354. [p62]

G. Csárdi and J. Hester. *pak: Another Approach to Package Installation*, 2024. URL https://CRAN.R-project.org/package=pak. R package version 0.8.0. [p64]

N. Cufaro Petroni. Mixtures in nonstable Lévy processes. *J. Phys. A*, 40(10):2227–2250, 2007. ISSN 1751-8113. doi: 10.1088/1751-8113/40/10/001. URL http://dx.doi.org/10.1088/1751-8113/40/10/001. [p56, 57]

L. Devroye. On the computer generation of random variables with a given characteristic function. *Comput. Math. Appl.*, 7(6):547–552, 1981. ISSN 0097-4943. doi: 10.1016/0898-1221(81)90038-9. URL http://dx.doi.org/10.1016/0898-1221(81)90038-9. [p58]

S. Ditlevsen and A. Samson. Hypoelliptic diffusions: filtering and inference from complete and partial observations. *J. R. Stat. Soc. Ser. B. Stat. Methodol.*, 81(2):361–384, 2019. ISSN 1369-7412. doi: https://doi.org/10.1111/rssb.12307. [p75]

F. Fang and C. W. Oosterlee. A novel pricing method for European options based on Fourier-cosine series expansions. *SIAM Journal on Scientific Computing*, 31(2):826–848, 2009. doi: https://doi.org/10.1137/080718061. [p61]

F. Fang and C. W. Oosterlee. A Fourier-based valuation method for Bermudan and barrier options under Heston's model. *SIAM Journal on Financial Mathematics*, 2(1):439–463, 2011. doi: https://doi.org/10.1137/100794158. [p61]

G. Giner and G. K. Smyth. statmod: probability calculations for the inverse Gaussian distribution. *R Journal*, 8(1):339–351, 2016. doi: 10.32614/RJ-2016-024. URL https://doi.org/10.32614/RJ-2016-024. [p61]

A. Gloter and N. Yoshida. Adaptive estimation for degenerate diffusion processes. *Electron. J. Stat.*, 15(1):1424–1472, 2021. doi: 10.1214/20-ejs1777. URL https://doi.org/10.1214/20-ejs1777. [p75]

S. Haberman and A. Renshaw. On age-period-cohort parametric mortality rate projections. *Insurance: Mathematics and Economics*, 45(2):255–270, 2009. ISSN 0167-6687. doi: https://doi.org/10.1016/j.insmatheco.2009.07.006. URL https://www.sciencedirect.com/science/article/pii/S016766870900081X. [p56]

C. C. Heyde and N. N. Leonenko. Student processes. *Adv. in Appl. Probab.*, 37(2):342–365, 2005. ISSN 0001-8678. doi: 10.1239/aap/1118858629. URL http://dx.doi.org/10.1239/aap/1118858629. [p56, 57]

F. Hubalek. On simulation from the marginal distribution of a Student t and generalized hyperbolic lévy process. 2005. URL https://api.semanticscholar.org/CorpusID:19037661. [p58]

S. M. Iacus and N. Yoshida. Simulation and inference for stochastic processes with yuima. *A comprehensive R framework for SDEs and other stochastic processes. Use R*, 2018. URL https://link.springer.com/book/10.1007/978-3-319-55569-0. [p56]

J. R. León and A. Samson. Hypoelliptic stochastic FitzHugh-Nagumo neuronal model: mixing, up-crossing and estimation of the spike rate. *Ann. Appl. Probab.*, 28(4):2243–2274, 2018. ISSN 1050-5164. doi: 10.1214/17-AAP1355. URL https://doi.org/10.1214/17-AAP1355. [p75]

A. Loregian, L. Mercuri, and E. Rroji. Approximation of the variance gamma model with a finite mixture of normals. *Statistics & Probability Letters*, 82(2):217–224, 2012. doi: https://doi.org/10.1016/j.spl.2011.10.004. URL https://www.sciencedirect.com/science/article/pii/S0167715211003257. [p61]

T. Massing. Simulation of Student–lévy processes using series representations. *Computational Statistics*, 33(4):1649–1685, 2018. doi: 10.1007/s00180-018-0814-y. [p58]

H. Masuda, L. Mercuri, and Y. Uehara. Noise inference for ergodic Lévy driven SDE. *Electronic Journal of Statistics*, 16(1):2432–2474, 2022. doi: 10.1214/22-EJS2006. URL https://doi.org/10.1214/22-EJS2006. [p60, 64]

H. Masuda, L. Mercuri, and Y. Uehara. Quasi-Likelihood Analysis for Student-Lévy Regression. *Statistical Inference for Stochastic Processes*, pages 1–34, 2024. doi: https://doi.org/10.1007/s11203-024-09317-2. [p56, 57, 58, 59, 60, 67]

L. Mercuri, A. Perchiazzo, and E. Rroji. Finite mixture approximation of CARMA (p, q) models. *SIAM Journal on Financial Mathematics*, 12(4):1416–1458, 2021. doi: https://doi.org/10.1137/20M1363248. [p61]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2024. URL https://www.R-project.org/. [p64]

R. Singleton. An algorithm for computing the mixed radix fast Fourier transform. *IEEE Transactions on audio and electroacoustics*, 17(2):93–103, 1969. URL https://ieeexplore.ieee.org/document/1162042. [p62]

G. Smyth, L. Chen, Y. Hu, P. Dunn, B. Phipson, and Y. Chen. *statmod: Statistical Modeling*, 2023. URL https://cran.r-project.org/web/packages/statmod/index.html. R package version 1.5.0. [p61]

C. Sørensen. Modeling seasonality in agricultural commodity futures. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 22(5):393–426, 2002. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/fut.10017. [p56]

L. Wu. Large and moderate deviations and exponential convergence for stochastic damping Hamiltonian systems. *Stochastic Process. Appl.*, 91(2):205–238, 2001. ISSN 0304-4149. doi: 10.1016/S0304-4149(00)00061-2. URL https://doi.org/10.1016/S0304-4149(00)00061-2. [p75]

YUIMA Project Team. *yuima: The YUIMA Project Package for SDEs*, 2024. URL https://cran.r-project.org/web/packages/yuima/index.html. R package version 1.15.27. [p56, 62]

Hiroki Masuda
The University of Tokyo
Graduate School of Mathematical Sciences,
3-8-1 Komaba Meguro-ku Tokyo 153-8914, Japan
https://www.ms.u-tokyo.ac.jp/teacher_e/masuda_e.html
ORCiD: 0000-0002-5553-9201
hmasuda@ms.u-tokyo.ac.jp

Lorenzo Mercuri
University of Milan
Department of Economics, Management and Quantitative Methods
Via Del Conservatorio Milano 7, Italy.
https://www.unimi.it/en/ugov/person/lorenzo-mercuri
ORCiD: 0000-0003-0490-2952
lorenzo.mercuri@unimi.it

Yuma Uehara
Kansai University
Department of Mathematics, Faculty of Engineering Science, 3-3-35 Yamate-cho Suita-shi Osaka 564-8680, Japan
https://kugakujo.kansai-u.ac.jp/html/100000728_en.html
ORCiD: 0000-0002-7084-0457
y-uehara@kansai-u.ac.j

Sfislands: An R Package for Accommodating Islands and Disjoint Zones in Areal Spatial Modelling

by Kevin Horan, Katarina Domijan, and Chris Brunsdon

Abstract Fitting areal models which use a spatial weights matrix to represent relationships between geographical units can be a cumbersome task, particularly when these units are not well-behaved. The two chief aims of **sfislands** are to simplify the process of creating an appropriate neighbourhood matrix, and to quickly visualise the predictions of subsequent models. The package uses visual aids in the form of easily-generated maps to help this process. This paper demonstrates how **sfislands** could be useful to researchers. It begins by describing the package's functions in the context of a proposed workflow. It then presents two worked examples showing a selection of potential use-cases. These range from earthquakes in Indonesia, to river crossings in London. We aim to show how the **sfislands** package streamlines much of the human workflow involved in creating and examining such models.

1 Introduction

A key feature which differentiates spatial statistics is the non-independence of observations and the expectation that neighbouring units will be more similar than non-neighbouring ones (Tobler, 1970). If this is not accounted for, the assumptions of many types of models will be violated. The relationships between all spatial units in a study can be represented numerically in a spatial weights matrix. In order to build this, we must first decide on what constitutes being a neighbour. We might see this as a continuous relationship where degree of neighbourliness is a function of connectivity, which could be represented as some measure of distance. Alternatively it could be a binary situation where each pair of units either are (1) or are not (0) neighbours. This can be based on a condition such as contiguity of some sort, or a distance constraint. It is the job of the modeller to formulate a hypothesis which justifies their choice of neighbourhood structure.

For R users, the **spdep** package (Bivand, 2022) has long been popular for the creation of these matrices. More recently, in reference to the increasing use of **sf** structures (Pebesma, 2018), the **sfdep** package (Parry and Locke, 2024) has presented generally similar functionality by wrapping **spdep** functions with functions that follow the **sf** naming convention (function names starting with `st_`), as well as a "use a data.frame for everything" attitude.

The most appropriate form of neighbourhood structure will depend on the specific context. Briz-Redón et al. (2021) compared different structures in the context of COVID-19 data. They note that Earnest et al. (2007) found that distance-based matrices were more appropriate when examining birth defects in Australia, whereas Duncan et al. (2017) found that a first-order contiguity structure produced a better fit than others in the context of lip cancer incidence in Scotland.

The most commonly used neighbourhood structure is one based on first-order queen contiguity, where units are considered neighbours if they share at least a vertex or boundary. However, as the name suggests, this will lead to problems when non-contiguous units such as islands or exclaves are present. Less obviously, depending on how the geographic units are described, areas on either side of rivers may be inappropriately classified as neighbours or not neighbours. Furthermore, the presence of infrastructure such as tunnels, bridges or ferry services might be satisfactory to meet our hypothesis of the required degree of connectivity to be considered neighbours. Again, such information may not be apparent from a basic set of polygons. In order to create what a researcher considers to be an appropriate neighbourhood structure, incorporating all of the domain knowledge that

they might have about the system, it should be simple and intuitive to add and remove connections between spatial units. This might mean adding links to account for man-made infrastructure, or cutting links to incorporate natural barriers such as rivers or mountains.

The aim of **sfislands** (Horan et al., 2024) is to deal with the situations described above in a convenient and open manner. It allows us to set up a structure, quickly map it, and then examine whether or not we are happy with how it represents our hypothesis of relationships between units. The structure can then be edited and the process re-iterated until we have described a spatial relationship structure with which we are satisfied.

It should be noted that while this package offers convenient tools for the examination, visualisation, addition and removal of neighbourhood linkages between units, such an approach to dealing with disconnected units is not always appropriate and other methodologies are available. These issues are discussed in more depth by Bivand and Portnov (2004) and Freni-Sterrantino et al. (2018).

The above can be considered as the *pre-functions* of the package. A second category of features, which we refer to as *post-functions*, are for use after the creation of a model. Having fit a model with **mgcv** (Wood, 2011) in particular, the process of extracting estimates for certain types of effects can be somewhat awkward. These *post-functions* augment the original dataframe with these estimates and their standard errors in tidy format. They also allow for quick visualisation of the output in map form.

1.1 Typical use-cases

In this paper, we will look at two examples to show different use-cases for **sfislands**. The first example focuses on earthquakes in Indonesia. It shows a scenario where all of the functions are used, from setting up contiguities, to modelling and examining the predictions of the model. The second example looks at London and how, despite an absence of islands, the presence of a river means that some of the pre-functions of **sfislands** can be useful.

2 Why use sfislands?

Below, we outline some of the benefits of the package in the context of a proposed workflow for fitting areal spatial models.

Step 1: *Pre-functions* for setting up neighbourhood structure

1. It addresses an issue commonly seen in online help forums where an inexperienced user wishes to get started with a model but fails at the first hurdle because their neighbourhood structure contains empty records. **sfislands** will include a contiguity for all units.

2. It gives tools to immediately visualise this structure as a map.

3. These maps are created using **ggplot2** (Wickham, 2016), which allows users to apply additional styling and themes using **ggplot2** syntax.

4. As the nodes can be labelled by index, it makes it very easy to add and remove connections as appropriate with confidence and without reference to the names of these areas.

5. Connections which have been induced by a function from the package but which are not based on geographical contiguity can be accessed to ensure openness in the process.

Table 1: Pre-functions: setting up a neighbourhood structure.

function	purpose
st_bridges()	create a neighbourhood contiguity structure, with a k-nearest neighbours condition for islands
st_quickmap_nb()	check structure visually on map
st_check_islands()	check the contiguities which have been assigned to islands
st_force_join_nb()	enforce changes by adding connections
st_force_cut_nb()	enforce changes by removing connections

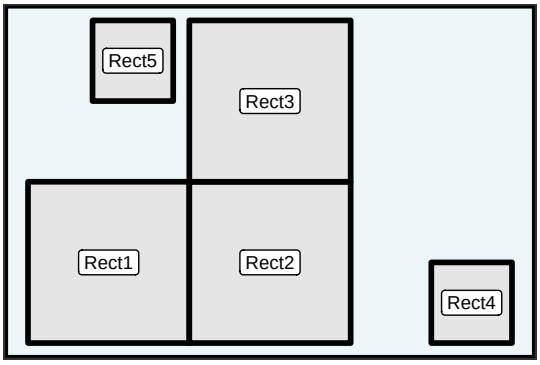


Figure 1: Simplified scenario with five rectangles.

Step 2: Modelling

These neighbourhood structures can be used in modelling packages such as **mgcv**, **brms** (Bürkner, 2017), **r-inla** (Bakka et al., 2018) and more.

Step 3: *Post-functions* for models

1. It simplifies the process of extracting estimates from models, such as those with random effects and Markov random field structures created using **mgcv**. Compatibility with more packages can be added at a future date.

2. These effects can be quickly visualised as **ggplot2** maps.

3 Pre-functions

The first group of functions, shown in Table 1, deals with the creation of a neighbourhood structure in the presence of discontiguities. The resultant structure can be quickly mapped to check if it is satisfactory. Connections can be forcibly added or removed by name or index number. By an iterative process of changes and examination of a quickly-generated guide map, a satisfactory structure can be decided upon.

We will now go through each function in more detail using the set of rectangles shown in Figure 1 for demonstration purposes. Rectangles 1, 2 and 3 are contiguous while 4 and 5 can be viewed as "islands".

3.1 st_bridges()

This function requires at least two arguments: an **sf** dataframe and, from that, the name of one column of unique row identifiers, ideally names, of each spatial unit. It creates a neighbourhood structure where non-island units are joined by first-order queen contiguity, while island units are joined to their k-nearest neighbours. The output is a *named* neighbourhood structure in either list or matrix form as desired, which can be either a standalone object

or included as an additional column in the original **sf** dataframe. While we have chosen to append the neighbourhood structure to the original data frame in this way by default, the user should be warned that any subsequent row sub-setting (filter) operation on this object will invalidate the list column involved. While it is not necessary in all modelling packages for the neighbourhood list or matrix to be *named*, it is good practice to do so and is mandatory when using, for example, **mgcv**.

One solution when confronted with islands in a dataset is to simply exclude them from the analysis. In the first two examples of using `st_bridges()`, we have chosen to ignore islands with the argument `remove_islands = TRUE` and to return a list and matrix structure respectively by specifying this in the `nb_structure` argument and choosing `add_to_dataframe = FALSE`:

```
# output a named list

st_bridges(rectangles,
           "name",
           remove_islands = TRUE,
           nb_structure = "list",
           add_to_dataframe = FALSE) |>
  head()

#> $Rect1
#> [1] 2 3
#>
#> $Rect2
#> [1] 1 3
#>
#> $Rect3
#> [1] 1 2

# output a named matrix

st_bridges(rectangles,
           "name",
           remove_islands = TRUE,
           nb_structure = "matrix",
           add_to_dataframe = FALSE) |>
  head()

#>       [,1] [,2] [,3]
#> Rect1    0    1    1
#> Rect2    1    0    1
#> Rect3    1    1    0
```

Alternatively, in the following examples, we choose to join islands to their 1 nearest neighbour, which is the default setting, and to return the output as a column called "nb" in the original **sf** dataframe (`add_to_dataframe = "TRUE"` is the default setting):

```
# output a named list as a column "nb" in original dataframe

st_bridges(rectangles,
           "name",
           link_islands_k = 1,
           nb_structure = "list") |>
  head()
```

```
#> Simple feature collection with 5 features and 2 fields
#> Geometry type: POLYGON
#> Dimension:     XY
#> Bounding box:  xmin: 0 ymin: 0 xmax: 6 ymax: 4
#> CRS:           NA
#>    name      nb                       geometry
#> 1 Rect1    2, 3 POLYGON ((0 0, 0 2, 2 2, 2 ...
#> 2 Rect2 1, 3, 4 POLYGON ((2 0, 2 2, 4 2, 4 ...
#> 3 Rect3 1, 2, 5 POLYGON ((2 2, 2 4, 4 4, 4 ...
#> 4 Rect4       2 POLYGON ((5 0, 5 1, 6 1, 6 ...
#> 5 Rect5       3 POLYGON ((0.8 3, 0.8 4, 1.8...

# output a named matrix as a column "nb" in original dataframe

st_bridges(rectangles,
           "name",
           link_islands_k = 1,
           nb_structure = "matrix") |>
  head()

#> Simple feature collection with 5 features and 2 fields
#> Geometry type: POLYGON
#> Dimension:     XY
#> Bounding box:  xmin: 0 ymin: 0 xmax: 6 ymax: 4
#> CRS:           NA
#>    name nb.1 nb.2 nb.3 nb.4 nb.5                       geometry
#> 1 Rect1    0    1    1    0    0 POLYGON ((0 0, 0 2, 2 2, 2 ...
#> 2 Rect2    1    0    1    1    0 POLYGON ((2 0, 2 2, 4 2, 4 ...
#> 3 Rect3    1    1    0    0    1 POLYGON ((2 2, 2 4, 4 4, 4 ...
#> 4 Rect4    0    1    0    0    0 POLYGON ((5 0, 5 1, 6 1, 6 ...
#> 5 Rect5    0    0    1    0    0 POLYGON ((0.8 3, 0.8 4, 1.8...
```

These structures can serve as the input to models in **brms**, **r-inla**, **rstan** (Stan Development Team, 2020) or **mgcv**. **brms** requires a matrix structure while **mgcv** models use a list. Rather than having a separate neighbours object, it is included in the original **sf** dataframe as a named list or matrix, in the spirit of the **sfdep** package.

3.2 st_quickmap_nb()

It is much more intuitive to examine these structures visually than in matrix or list format. This can be done with the `st_quickmap_nb()` function as shown in Figure 2.

```
# default is 'nodes = "point"'

st_bridges(rectangles,
           "name",
           link_islands_k = 1) |>
  st_quickmap_nb()
```

If we wish to make edits, it might be more useful to represent the nodes numerically rather than as points (Figure 3).

```
# with 'nodes = "numeric"'

st_bridges(rectangles,
           "name",
```

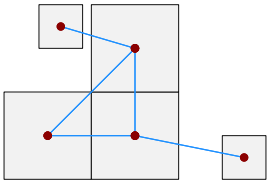


Figure 2: Queen contiguity and islands connected to nearest neighbour.

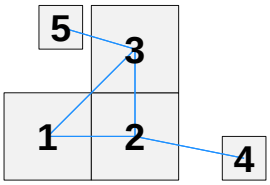


Figure 3: Queen contiguity and islands connected to nearest neighbour. Nodes are shown as numeric indices.

```
          link_islands_k = 1) |>
  st_quickmap_nb(nodes = "numeric")
```

3.3 st_check_islands()

This function will show us transparently what connections have been made which are not based on contiguity. It gives both the name and index number of each pair of added connections. In this example, two pairs have been added.

```
# show summary of non-contiguous connections in a dataframe

st_bridges(rectangles,
           "name",
           link_islands_k = 1) |>
  st_check_islands()

#>   island_names island_num nb_num nb_names
#> 1        Rect4          4      2    Rect2
#> 2        Rect5          5      3    Rect3
```

3.4 st_force_join_nb()

If we feel that 4 should also be connected to 3, this can be done by forcing a join (Figure 4).

```
# add an extra connection using numeric index

st_bridges(rectangles, "name",
           link_islands_k = 1) |>
  st_force_join_nb(3,4) |>
  st_quickmap_nb(nodes = "numeric")
```

3.5 st_force_cut_nb()

And perhaps there is a wide river between rectangles 1 and 2 which justifies removing the connection. We will edit it this time using names (Figure 5).

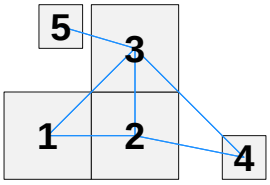


Figure 4: With an additional connection between 3 and 4.

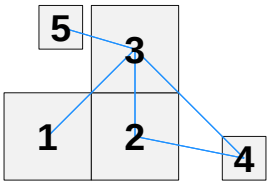


Figure 5: With the removal of connection between 1 and 2.

```
# remove an existing connection using unit name, not index

st_bridges(rectangles, "name",
           link_islands_k = 1) |>
  st_force_join_nb(3,4) |>
  st_force_cut_nb("Rect1","Rect2") |>
  st_quickmap_nb(nodes = "numeric")
```

Having decided upon an appropriate neighbourhood structure, the next step is to use this in the context of a model. The use of such structures is particularly associated with CAR (conditional autoregressive) or ICAR-type (intrinsic conditional autoregressive) models (Besag, 1974). These are often implemented in a Bayesian framework using **brms**, **r-inla** or **rstan**. For example, the **brms** ICAR structure requires the neighbourhood relationships to be in matrix form. The pre-functions will output the neighbourhood structure in the desired format for use in any of these frameworks. A convenient frequentist alternative is to use the **mgcv** package which requires a named list of neighbours. It has the functionality to create such models using `bs="mrf"`. It also has the ability to combine these with a hierarchical structure using `bs="re"`. While the outputs from the Bayesian structures mentioned above can be extracted in the same way as any other component of the model, it can be somewhat awkward to get the estimates from **mgcv** models. **sfislands** has two post-functions to conveniently extract and visualise these.

4 Post-functions

Table 2 shows the second set of functions in the package and their purpose.

4.1 st_augment()

This function augments the original dataframe with the estimated means and standard errors of the spatially varying predictions from a fitted **mgcv** model in a similar manner to how the **broom** package (Robinson et al., 2023) operates. The `geometry` column, as per convention, remains as the last column of the augmented dataframe, while the predictions are positioned

Table 2: Post-functions: tidy estimates from mgcv.

function	purpose
st_augment()	augment the original dataframe with model predictions
st_quickmap_preds()	generate quick maps of these predictions

Table 3: The naming procedure for augmented columns from different mgcv structures.

mgcv syntax	column name
s(region, bs = 're')	random.effect.region
s(region, covariate, bs = 're')	random.effect.covariate | region
s(sub-region, bs = 'mrf', xt = list(nb = data$nb))	mrf.smooth.sub-region
s(sub-region, by = covariate, bs = 'mrf', xt = list(nb = data$nb))	mrf.smooth.covariate | sub-region

immediately before it. [1] The spatially varying predictions which `st_augment()` extracts from an **mgcv** model are

- random effects (which are called in **mgcv** with `bs='re'`), and
- ICAR components (`bs='mrf'`).

Consider the model structure described in the code below using **mgcv** syntax. In this model y is the dependent variable which is being estimated with a fixed intercept, a fixed slope for some covariate, a set of random intercepts and slopes for the covariate at a *region* level, and a set of ICAR varying intercepts and slopes at a lower *sub-region* level.

```
# creating an mgcv model

mgcv::gam(
  y ~ covariate +                  # fixed intercept and effect for covariate
    s(region, bs = "re") +                 # random intercept at level region
    s(region, covariate, bs = "re") +         # random slopes at level region
    s(sub-region,
      bs = 'mrf',
      xt = list(nb = data$nb),
      k = k) +                    # ICAR varying intercept at level sub-region
    s(sub-region, by = covariate,
      bs = 'mrf',
      xt = list(nb = data$nb),
      k = k),           # ICAR varying slope for covariate at level sub-region
  data = data,
  method = "REML")
```

When labelling the new prediction columns which are augmented to the original dataframe from such a model, `st_augment()` follows the formula syntax of the **lme4** package (Bates et al., 2015), where the pipe symbol (`|`) indicates "*grouped by*". Table 3 shows how the augmented columns in this scenario would be named. Each column name begins with either `random.effect.` or `mrf.smooth.` as appropriate. An additional column is also added for the standard error of each prediction, as calculated by **mgcv**. These columns are named as above but with `se.` prepended (e.g. `se.random.effect.region`).

4.2 st_quickmap_preds()

These estimates can then be quickly mapped. As it is possible to include more than 1 spatially varying component, the output of this function is a list of plots. They can be viewed individually by indexing, or all at once using, for example, the `plotlist` argument from the `ggarrange()` function which is part of the **ggpubr** (Kassambara, 2023) package. We will see this function in practice in the following example. The maps which it generates are automatically titled and subtitled according to the type of effect. For example, the map showing predictions for `random.effect.region` will have "*region*" as its title and "*random.effect*" as its subtitle.

[1] In a similar way, `st_augment()` can also be used to append the random effects from **lme4** (Bates et al., 2015) and **nlme** (Pinheiro et al., 2023) models to an **sf** dataframe, which can then be easily mapped using `st_quickmap_preds()`. Compatibility with models created using different packages can be introduced in the future.

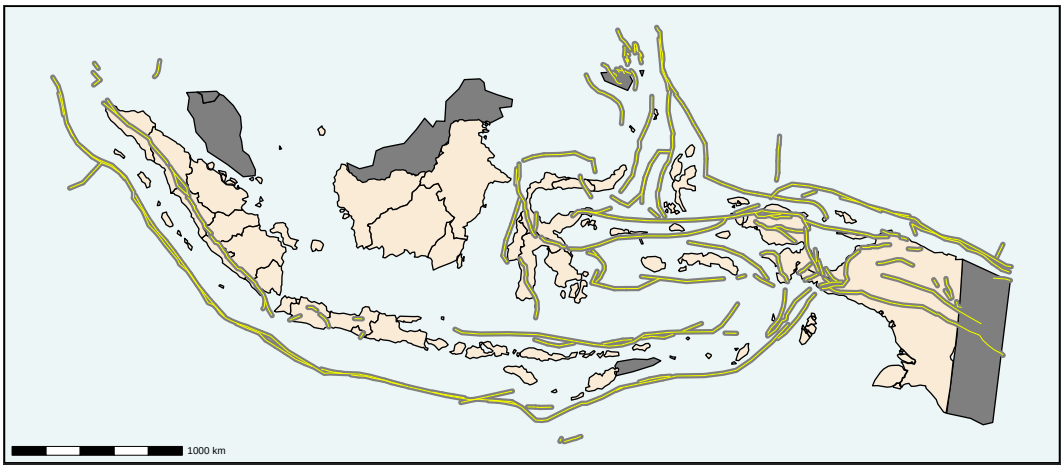


Figure 6: Indonesia faults. Surrounded by a 10 kilometre buffer.

5 Indonesia (example 1)

Modelling earthquakes in Indonesia serves as a good example to demonstrate this package. Firstly, Indonesia is composed of many islands. Secondly, earthquake activity is known to be associated with the presence of faults which exist below sea level and thus do not respect land boundaries. Therefore it is reasonable to expect similar behaviour in nearby provinces regardless of whether or not they are contiguous. We aim to model the incidence, or count per unit area, of earthquake activity by province across Indonesia, controlling for proximity to faults.

5.1 Data

The data for this section have been downloaded from the National Earthquake Information Center, USGS earthquake catalogue. The datasets with accompanying explanations are available at `https://github.com/horankev/quake_data`. They capture all recorded earthquakes in and close to Indonesia from the beginning of January 1985 to the end of December 2023. Figure 6 shows a map of Indonesia, divided into 33 provinces, with other neighbouring or bordering countries filled in grey. The many local faults which lie within 300km of the shore are shown in yellow with green outlines.

To get an interpretable measure of the concentration of faults in any area, these faults are transformed from linestrings to polygons by setting a buffer of 10km around them, which explains their green outline. Now both our faults and the sizes of provinces are in units of kilometres squared. This means we can generate a unitless metric of what proportion of any administrative unit is covered by these buffered faults. This measure across provinces is shown in Figure 7.

Earthquake incidence per province has been calculated as the total number of earthquakes with an epicentre within that province per unit area. We have restricted counts to earthquakes >5.5 on the moment magnitude scale, which is the point at which they are often labelled as potentially damaging.

The occurrences of these earthquakes are shown in Figure 8, their total per province in Figure 9, and finally, their incidence or count per square kilometre can be seen in Figure 10.

5.2 Model

As this is count data, we will model it as a Poisson distribution with λ as the mean count per province. For $i = 1, ..., n$ provinces, the dependent variable in this model is

$$y_i = \text{earthquake count}_i \tag{1}$$

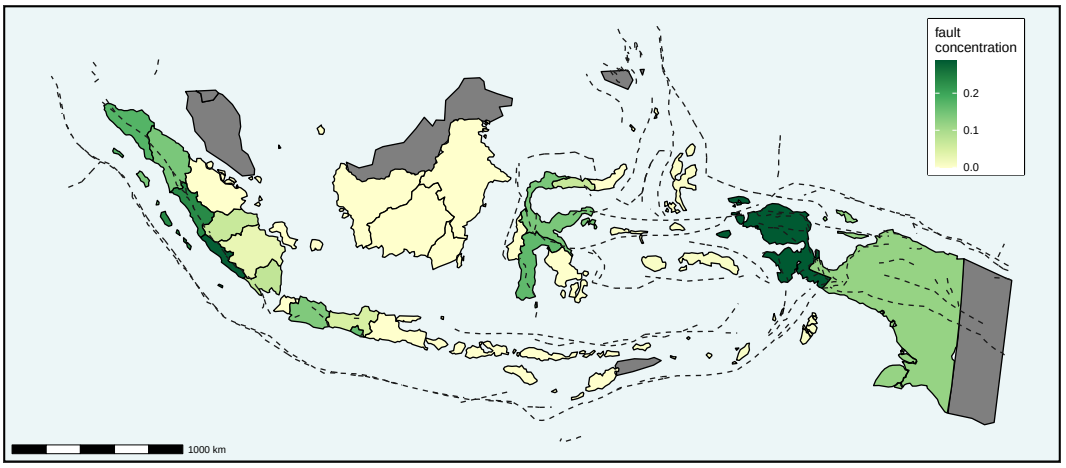


Figure 7: Indonesia fault concentration. Square kilometre of buffered fault per square kilometre of province area.

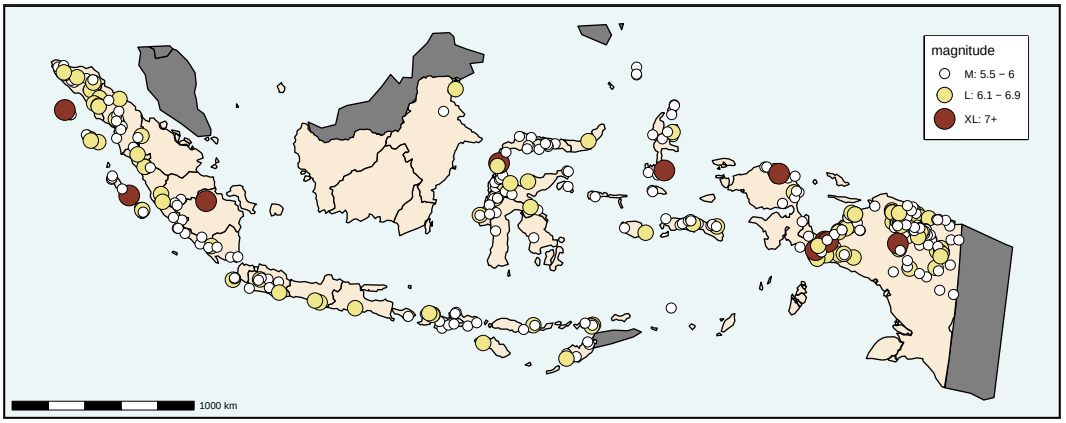


Figure 8: Earthquakes in Indonesia of magnitude > 5.5, 1985-2023. Categorised by magnitude as medium, large or extra-large.

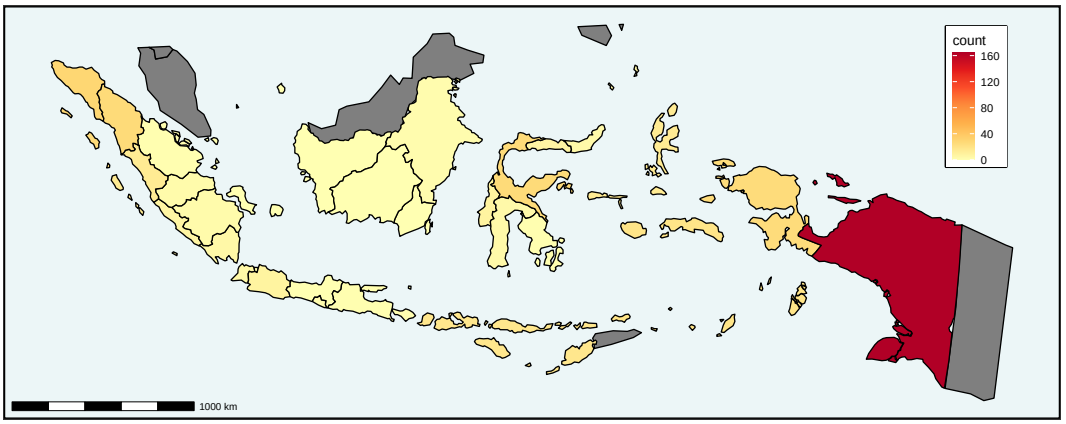


Figure 9: Earthquake count in Indonesia, 1985-2023, mag > 5.5: count by province.

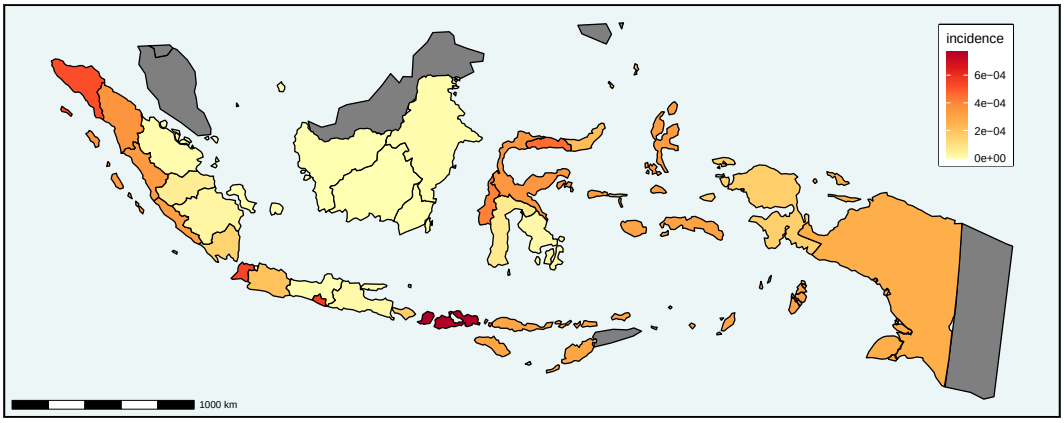


Figure 10: Earthquake incidence in Indonesia, 1985-2023, mag > 5.5: count per square kilometre by province.

while the explanatory variable is

$$x_i = \text{fault concentration}_i = \frac{\text{area of buffered faults in province}_i}{\text{province area}_i}. \tag{2}$$

Firstly, when excluding the incidence and just modelling counts, where y_i = earthquake count in province$_i$, the Poisson model is of the following form:

$$y_i|\lambda_i \sim \text{Pois}(\lambda_i) \tag{3}$$

with

$$E(y_i|\lambda_i) = \lambda_i. \tag{4}$$

We model

$$log(\lambda_i) = \beta_0 + \beta_1 x_i + \gamma_i. \tag{5}$$

where γ_i is a term with a correlation structure reflecting a province's location relative to other provinces.

We can describe these relationships by setting up a neighbourhood structure based on queen contiguity where a pair of provinces are considered neighbours if they share at least one point of boundary. This can be modelled as a Markov random field to generate an ICAR model with a spatially varying term. Each of these terms will be correlated with the others according to the neighbourhood structure we have defined.

The Markov random field here follows a multivariate Gaussian distribution. γ_i is a vector of province effects having a distribution with mean **0** and precision **P** where

$[\mathbf{p}]_{ij} = v_i$ if $i = j$ and v_i is the number of adjacent provinces to province i,

$[\mathbf{p}]_{ij} = -1$ if provinces i and j are adjacent, and

$[\mathbf{p}]_{ij} = 0$ otherwise.

A further constraint that $\Sigma_j \gamma_j = 0$ is applied so that the distribution is identifiable.

We now include an offset term (here, area) because we are more interested in modelling the incidence than in the actual count, such that

$$log(\frac{\lambda_i}{\text{area}_i}) = \beta_0 + \beta_1 x_i + \gamma_i \tag{6}$$

which is equivalent to

$$log(\lambda_i) = \beta_0 + \beta_1 x_i + \gamma_i + log(\text{area}_i). \tag{7}$$

We are still modelling $log(\lambda)$ rather than the incidence, but we are adding an offset to adjust for differing areas. Modelling $log(\lambda)$ and adding an offset is equivalent to modelling

incidence, and coefficients can be interpreted that way.

When interpreting the estimated coefficients of the model, it can be useful to look at it in the following form:

$$\lambda_i = e^{\beta_0+\beta_1 x_i+\gamma_i}\text{area}_i. \tag{8}$$

Having described the type of model we wish to implement, we now show how **sfislands** can be used to streamline the process.

5.3 Pre-functions

Such models, however, cannot incorporate locations which have no neighbours. In the case of Indonesia, this is quite problematic. It is composed of many islands. The estimated count of islands according to Andréfouët et al. (2022) is 13,558. While it is not unusual for a country to have a number of often small offshore islands, Indonesia is entirely composed of (at least portions of) an archipelago of islands, so many of these islands or groups of islands are individual provinces in their own right. We might like to hypothesise that just because a province is a disconnected island, this should not mean that it is independent of other nearby provinces in terms of earthquake incidence. A standard first-order queen contiguity structure would mean the exclusion of disconnected units entirely from the model. An alternative strategy of assigning neighbour status based on a distance metric would overcome this, but the threshold size of distance necessary for such a structure might be inappropriately large for the non-islands provinces. Many extra unwanted contiguities could be added when only those related to disconnected units were desired. We would like to use a compromise between these two strategies.

In this case, we use `st_bridges()` for setting up the queen contiguity structure as usual, but with the additional stipulation that unconnected units (provinces which are islands or collections of islands) are considered neighbours to their k nearest provinces. For this example, we have set the value of k to 2. The resulting neighbourhood structure is shown in Figure 11. Note how it can be styled with a combination of internal arguments (size, colour, fill etc.) and additional **ggplot2** layers.

```
# join islands to k=2 nearest neighbours
# various arguments exist for altering colours and sizes
# additional ggplot themes and layers can be added

st_bridges(provinces_df, "province", link_islands_k = 2) |>
  st_quickmap_nb(fillcol = "antiquewhite1",
                 bordercol = "black", bordersize = 0.5,
                 linkcol = "darkblue", linksize = 0.8,
                 pointcol = "red", pointsize = 2) +
  theme(panel.background = element_rect(fill = "#ECF6F7", colour = "black",
                                        linewidth=1.5),
        axis.text = element_blank()) +
  geom_sf(data=nearby_countries_df,
          fill="gray50", linewidth=0.5, colour="black")
```

This neighbourhood structure now has no unconnected provinces so it is suitable for use in an ICAR model. However, if we are not entirely happy with this structure because of some domain knowledge about the inter-relationships between certain island provinces, we might wish to

- add some additional contiguities using `st_force_join_nb()`
- and remove one using `st_force_cut_nb()`.

To cater for the possibility that a modeller might not be familiar with the names of the various geographic units but still wishes to enforce alterations to their relationships, we

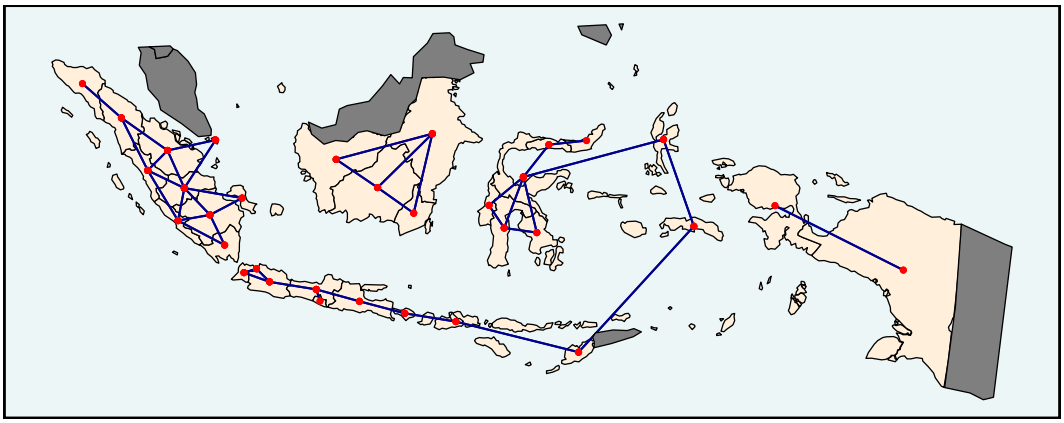


Figure 11: Neighbourhood structure for Indonesian provinces created by `st_bridges()` with `k=2`.

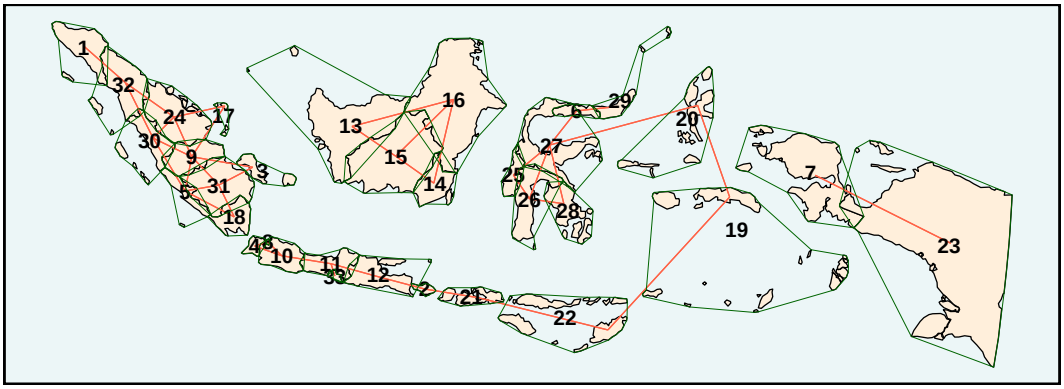


Figure 12: Neighbourhood structure for Indonesian provinces. Viewed with `st_quickmap_nb()`, using the arguments `nodes = 'numeric'` and `concavehull = TRUE`.

can look at a map (Figure 12) where the nodes are shown by index number instead of as points (using the argument `nodes='numeric'`). This makes it easy to cut and join neighbour connectivities as desired. Furthermore, there is an option to show concave hulls drawn around each unit (using `concavehull = TRUE`). This is also shown in Figure 12. These shapes are not used in the assignment of contiguities but it can be useful to see them in a situation such as Indonesia where many individual provinces are actually multipolygons of more than one island. Without them, it is not clear whether an island is a province in its own right, or which group of islands together form one province.

```
# with 'concavehull = TRUE' and 'nodes = "numeric"'

st_bridges(provinces_df, "province", link_islands_k = 2) |>
  st_quickmap_nb(fillcol = "antiquewhite1",
                 bordercol = "black", bordersize = 0.5,
                 linkcol = "tomato", linksize = 0.5,
                 nodes = "numeric",
                 numericcol = "black", numericsize = 6,
                 concavehull = TRUE,
                 hullcol = "darkgreen", hullsize = 0.2) +
  theme(panel.background = element_rect(fill = "#ECF6F7", colour = "black",
                                        linewidth=1.5),
        axis.text = element_blank())
```

Having enforced some adjustments to the neighbourhood structure, outlined in the code below, the new structure can be seen in Figure 13. Edge effects have also been mitigated by imposing additional connections on the two extreme provinces (1 and 23), which would

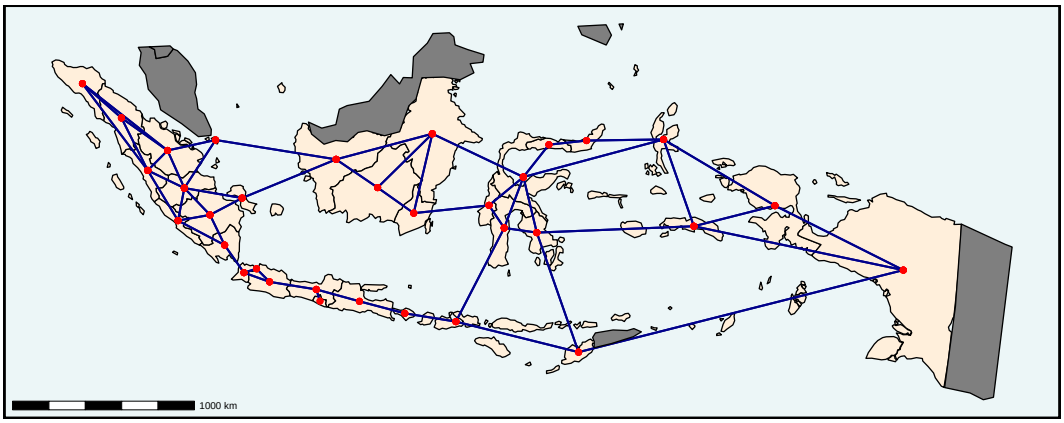


Figure 13: Neighbourhood structure for Indonesian provinces, after alterations using `st_force_join()` and `st_force_cut()`. As many connections are being enforced at once, we can feed these to the function as a data frame rather than using consecutive function calls as before.

otherwise have only one neighbour, so that they now also include their two next closest neighbours.

```
# a series of forced joins and cuts by index number

joins_df <- tribble(
  ~x, ~y,
  1, 24,
  1, 30,
  3, 13,
  13, 17,
  14, 25,
  20, 29,
  19, 23,
  16, 27,
  22, 23,
  7, 19,
  7, 20,
  19, 28,
  4, 18,
  21, 26,
  22, 28
)

st_bridges(provinces_df, "province", link_islands_k = 2) |>
  st_force_join_nb(xy_df = joins_df) |>
  st_force_cut_nb(19,22) |>
  st_quickmap_nb(fillcol = "antiquewhite1",
                 bordercol = "black", bordersize = 0.5,
                 linkcol = "darkblue", linksize = 0.8,
                 pointcol = "red", pointsize = 2) +
  theme(panel.background = element_rect(fill = "#ECF6F7", colour = "black",
                                        linewidth=1.5),
        axis.text = element_blank()) +
  geom_sf(data=nearby_countries_df,
          fill="gray50", linewidth=0.5, colour="black") +
  annotation_scale()
```

5.4 mgcv model

We now create the ICAR model using, in this case, the **mgcv** package. We will be able to use the output of `st_bridges`, which we have named `prep_data`, as both the data source for the model and the neighbourhood structure (by specifying the column `nb` which contains the neighbourhood list).

```
mod_pois_mrf <- gam(damaging_quakes_total ~
                      fault_concentration +
                      s(province, bs='mrf', xt=list(nb=prep_data$nb), k=24) +
                      offset(log(area_province)),
                    data=prep_data, method="REML",family = "poisson")
```

We can see from the summary below that the adjusted R-squared is **0.983** and deviance explained is **93.3%**. The coefficient for `fault_concentration` confirms an expected positive mean global association between earthquake and fault incidence.

```
#>
#> Family: poisson
#> Link function: log
#>
#> Formula:
#> damaging_quakes_total ~ fault_concentration + s(province, bs = "mrf",
#>     xt = list(nb = prep_data$nb), k = 24) + offset(log(area_province))
#>
#> Parametric coefficients:
#>                     Estimate Std. Error z value Pr(>|z|)
#> (Intercept)          -9.5648     0.1744 -54.845  < 2e-16 ***
#> fault_concentration   5.9971     1.9245   3.116  0.00183 **
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Approximate significance of smooth terms:
#>               edf Ref.df Chi.sq p-value
#> s(province) 19.19     23  166.6  <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> R-sq.(adj) =  0.983   Deviance explained = 93.3%
#> -REML = 104.81  Scale est. = 1         n = 33
```

Returning to the initial question, what is the additional risk level of earthquakes in a province, having controlled for the concentration of faults? This can be seen as a measure of the activity level of faults locally and it is spatially smoothed by the autoregressive process. It is represented in the model summary by the component `s(province)`. However, the extraction of individual predictions for this component for each province from the `mgcv` model requires a number of steps. We now demonstrate how these are streamlined into a single function by the **sfislands** package.

5.5 Post-functions

The function `st_augment()` allows us to add the spatially varying predictions from the model as new columns to the original dataframe in a process similar to that of the **broom** package. For instance, we see from the output of the following code chunk that the original dataframe is now augmented with columns called `mrf.smooth.province` and `se.mrf.smooth.province` which show the predictions for the γ_i component and their standard errors. Note that this is how we would expect them to be named, based on the previous

discussion surrounding Table 3. They are positioned immediately before the final `geometry` column of the `sf` dataframe, and after the neighbours list column, `nb`.

```
# column names of augmented dataframe

mod_pois_mrf |>
  st_augment(prep_data) |>
  names() |>
  dput()

#> c("province", "province_id", "S", "M", "L", "XL", "quake_total",
#> "quake_density", "damaging_quakes_total", "damaging_quakes_density",
#> "area_fault_within", "area_province", "fault_concentration",
#> "nb", "mrf.smooth.province", "se.mrf.smooth.province", "geometry"
#> )
```

This output can now be piped into the `st_quickmap_preds()` function to get a quick visualisation of these estimates for γ_i on a map, as shown in Figure 14. Again, note that the title and subtitle of the image are as previously discussed.

```
# st_quickmap_preds() outputs a list of ggplots

plot_mrf <- mod_pois_mrf |>
  st_augment(prep_data) |>
  st_quickmap_preds(scale_low = "darkgreen",
                    scale_mid = "ivory",
                    scale_high = "darkred",
                    scale_midpoint = 0)

# in this case, there is only one plot in the list
# so we call it by index
# it is then supplemented with additional ggplot functions

plot_mrf[[1]] +
  coord_sf(datum=NA, default = TRUE) +
  theme(panel.background = element_rect(fill = "#ECF6F7", colour = "black",
                                        linewidth=1.5),
        axis.text = element_blank()) +
  geom_sf(data=provinces_df, fill=NA, colour="black", linewidth=0.5) +
  geom_sf(data=nearby_countries_df, fill="gray50", colour="black",
          linewidth=0.5) +
  labs(fill="relative\nincidence") +
  annotation_scale() +
  coord_sf(datum=NA, default = TRUE) +
  theme(legend.position = "inside",
        legend.position.inside = c(0.92,0.77),
        legend.box.background = element_rect(colour = "black", linewidth = 1),
        legend.title = element_text())
```

If we wish to apply the inverse link function (the exponential function in the case of this Poisson model) to map these values to a more interpretable scale, this will not be generated by the function `st_quickmap_preds()`. Instead, we must use the augmented dataframe which is produced by `st_augment()` and create the appropriate extra column with the usual **tidyverse** `mutate()` function. This allows us to produce the map in Figure 15. As these coefficients are multiplicatively related to the earthquake incidence, values below 1 imply an earthquake incidence which is lower than expected.

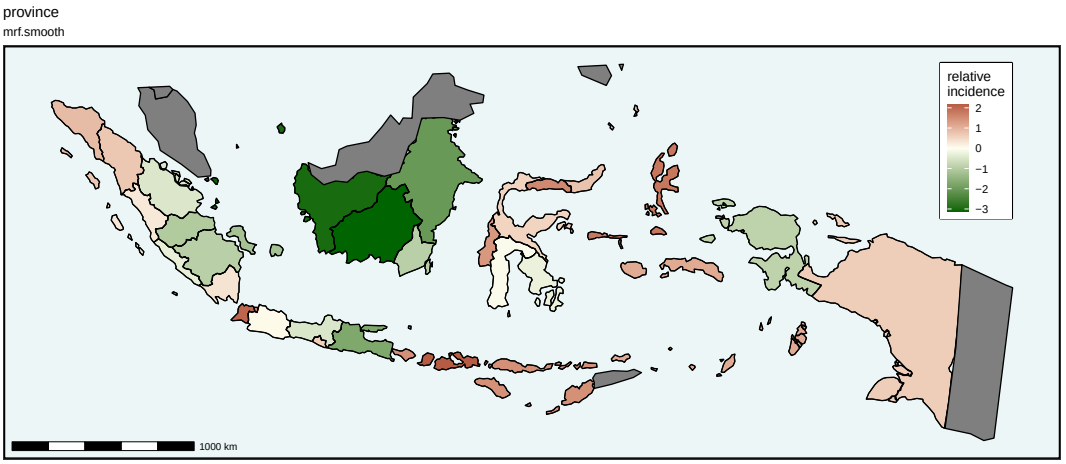


Figure 14: Estimates of γ_i shown as a map using `st_quickmap_preds()`.

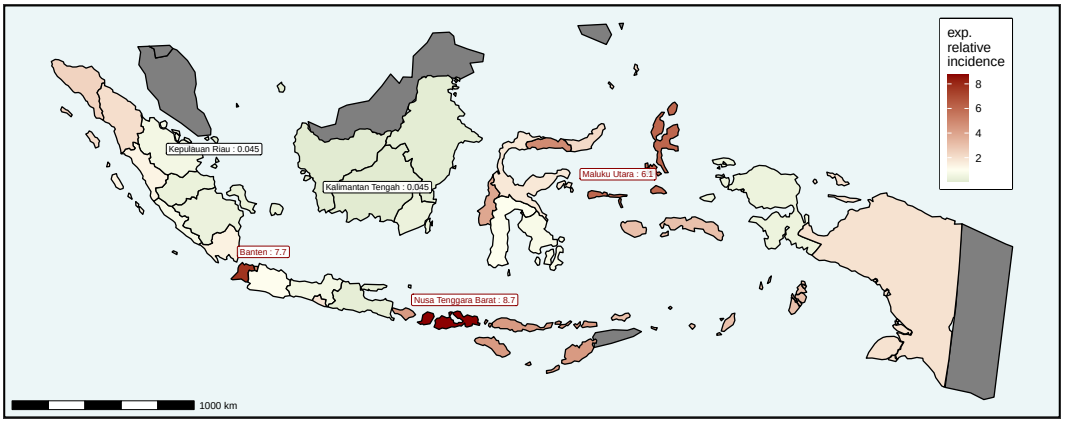


Figure 15: Map showing estimates of $\exp(\gamma_i)$. This is produced by adding an additional column to the dataframe produced by `st_augment()`.

The provinces with the 3 most elevated incidences are labelled in red. We can see that, controlling for the effects of proximity to faults, the province of Nusa Tenggara Barat has 8.7 times the expected incidence, or number of major earthquakes per square kilometre. The two lowest-scoring provinces, labelled in green, have essentially no incidence of earthquake epicentres within their boundaries, controlling for what their proximity to faults alone would suggest.

5.6 Workflow summary

In this example, we have gone through a number of stages carefully, making changes to contiguities that we deemed appropriate as we went. However, in practice, at least in a first iteration, it might not be necessary to go through all of these steps. A rough and ready model, complete with spatially varying coefficients and visual output, can be generated with **sfislands** using nothing more than three or four lines of code, such as the following:

```
# workflow:

# 1. set up neighbourhood structure

prep_data <- st_bridges(provinces_df, "province")

# 2. define model
```

```
mod <- gam(quake_mlxl_total ~
                  fault_concentration +
                  s(province, bs='mrf', xt=list(nb=prep_data$nb), k=22) +
                  offset(log(area_province)),
        data=prep_data, method="REML",family = "poisson")

# 3. augment tidy estimates

tidy_ests <- st_augment(mod, prep_data)

# 4. visualise them

st_quickmap_preds(tidy_ests)
```

6 London (example 2)

The next example looks only at using the *pre-functions* of **sfislands**, but in a situation where the presence of actual *islands* is not the problem we seek to deal with. Consider the wards of London (sourced from the Greater London Authority's London Datastore) and available at https://github.com/horankev/london_liverpool_data. In Figure 16 the `st_bridges()` function is applied to them to construct a queen contiguity neighbourhood structure. As can be seen from the `st_check_islands()` function, this collection of London wards contains no isolated units.

```
st_bridges(london, "GSS_CODE") |>
  st_check_islands()

#> No disconnected units were found in original data

#> [1] 0
```

The `st_quickmap_nb()` function gives an immediate visual representation of the structure. [2] Because this map is created using **ggplot2**, it can be easily supplemented by adding a layer showing the course of the river Thames which is also visible in Figure 16.

```
# same as sfdep:st_contiguity() as there are no islands
# an extra layer for the river Thames

st_bridges(london, "GSS_CODE") |>
  st_quickmap_nb() +
  geom_sf(data=thames, colour="blue", linewidth=1.5) +
  theme(panel.background = element_rect(fill = "#F6F3E9", colour = "black",
                                        linewidth=1.5))
```

When a study area has a river running through it, problems can arise with constructing appropriate neighbourhood structures. Depending on how the geometries are defined, the presence of a river can cause problems in two ways. In one situation, the river could be expressed as a polygon in its own right meaning that, using the condition of queen contiguity, it severs any potential contiguity between units on either side of its banks. In this situation, no spatial units will be neighbours with the units directly across the river from them. At the other extreme, if the river is not included as a geometry (as is the case here) all units on opposing banks are automatically considered neighbours.

Depending on the presence of river crossings, two areas which are physically quite close but on opposing banks might be very distinct. If there is no means of crossing the river

[2] `st_quickmap_nb()` can also be used to visualise any contiguity structure created by `spdep` or `sfdep` as long as that structure is included in an `sf` dataframe as a column named `nb`.

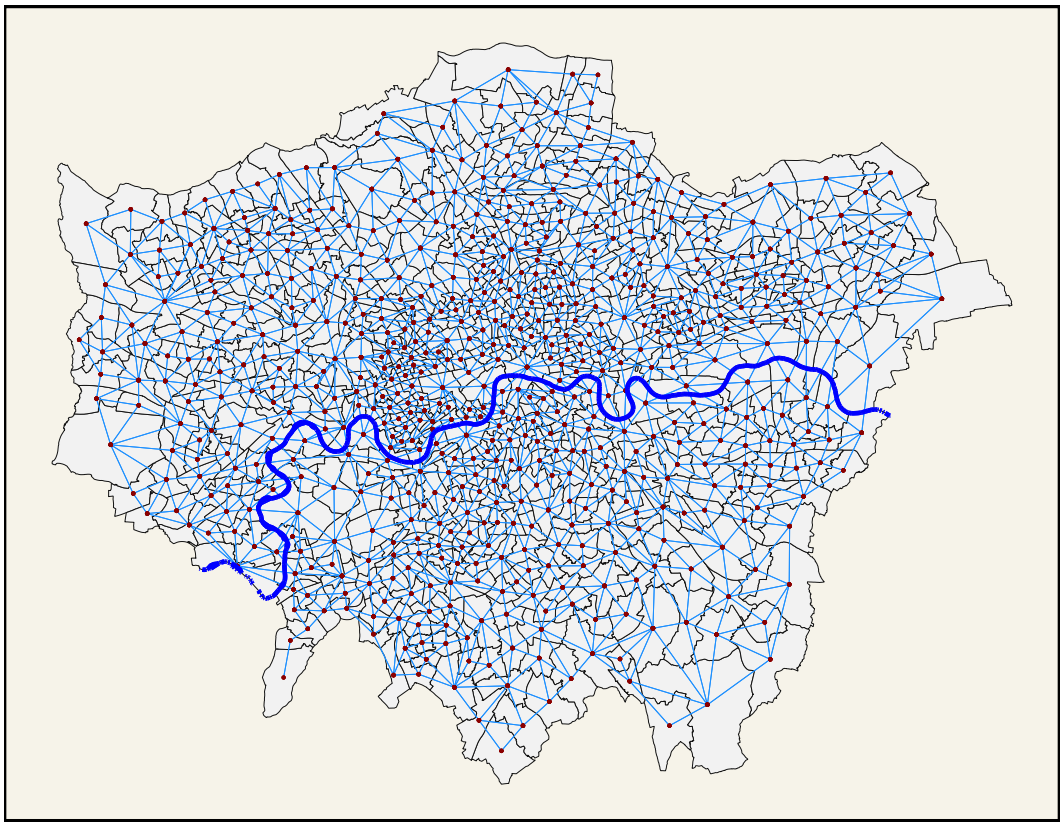


Figure 16: Wards of Greater London. Queen contiguity.

within a reasonable distance, somebody living on the banks of a river might be more likely to go about their life primarily on their side of the river, despite the short distance as the crow flies of facilities on the other side. This could be relevant in terms of, say, modelling of house prices where we might want to incorporate issues such as local amenities into a neighbourhood structure.

sfislands provides convenient functions for this sort of situation. Let us start by restricting our wards of interest to just those which are on either side of the river Thames. Figure 17 shows the resultant contiguities when the river is ignored.

```
# which wards are alongside the river

riverside <- thames |> st_intersects(london) |> unlist() |> unique()

# only map these wards

st_bridges(london[riverside,],"NAME") |>
  st_quickmap_nb(linksize = 0.5) +
  geom_sf(data=thames, colour="blue", linewidth=1.5) +
  annotation_scale(location="br") +
  coord_sf(datum=NA) +
  theme(panel.background = element_rect(fill = "#F6F3E9", colour = "black",
                                        linewidth=1.5))
```

In order to take account of actual connectivity, we can add a layer showing the road and pedestrian bridges or tunnels. Details of these were sourced from the Wikipedia (2024) article titled *"List of crossings of the River Thames"*. In Figure 18, we have also drawn a 1 kilometre buffer around each crossing. This was chosen as an arbitrary measure of what might be considered a "reasonable" distance within which to consider opposing banks as being connected. The vast majority of units on opposing banks have access to a river

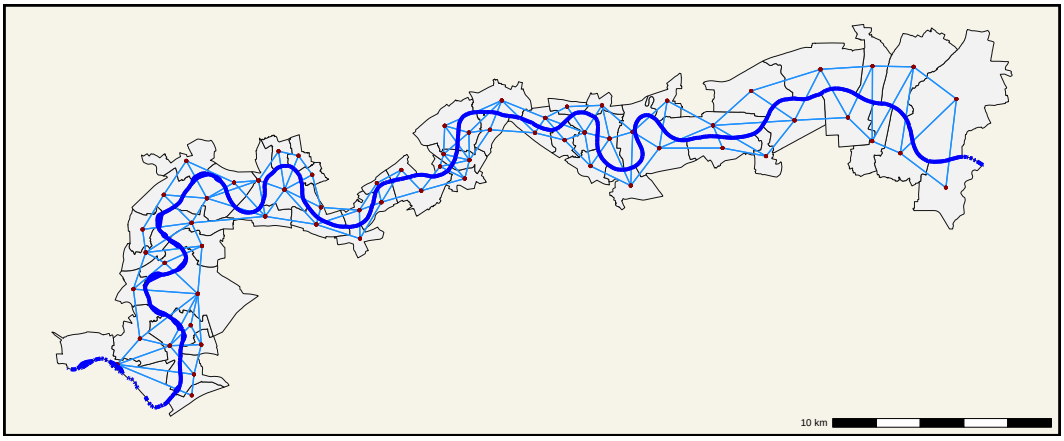


Figure 17: Riverside wards of Greater London. Queen contiguity disregarding river.

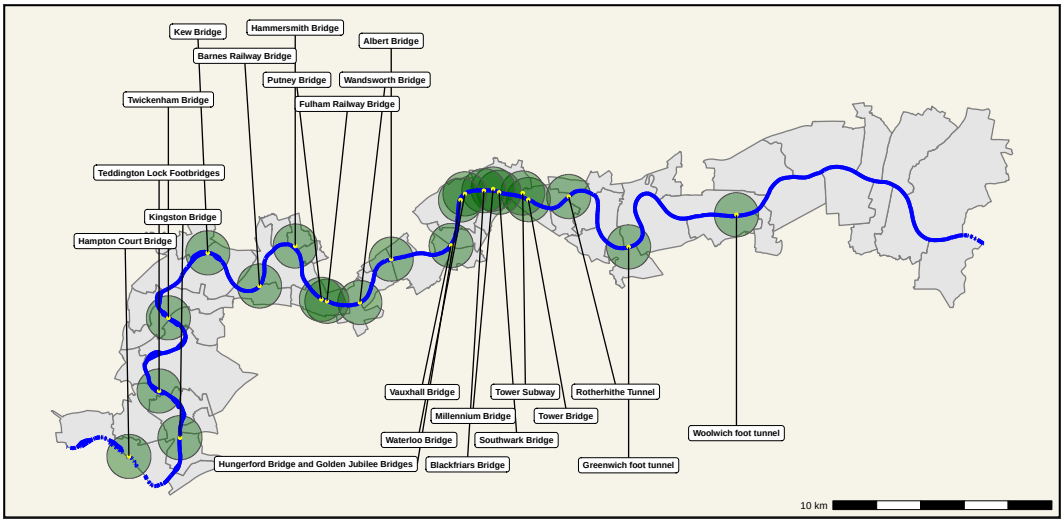


Figure 18: Riverside wards of Greater London. Road and pedestrian crossing and tunnels surrounded by 1 kilometre buffer shaded green.

crossing within this threshold and thus should be considered as neighbours. Only the extreme eastern units and one to the south west should not have a connection across the river according to this criterion.

In order to identify the changes we wish to make, we use the `nodes = "numeric"` argument in `st_quickmap_nb()`. Now we can identify each unit by its position in the contiguity structure. Here we have shaded in pink the units which are not within 1 kilometre of a river crossing (see Figure 19).

```
# with 'nodes = "numeric"'

st_bridges(london[riverside,],"NAME") |>
  st_quickmap_nb(nodes = "numeric", numericsize = 4, linksize = 0.5) +
  geom_sf(data=no_touch_buffer, fill="pink", alpha=0.3) +
  geom_sf(data=crossings_roadped |> st_buffer(1000),
          fill="darkgreen", alpha=0.3) +
  geom_sf(data=thames, colour="blue", linewidth=1.5) +
  geom_sf(data=crossings_roadped, size=1, colour="yellow") +
  annotation_scale(location="br") +
  coord_sf(datum=NA) +
  theme(panel.background = element_rect(fill = "#F6F3E9", colour = "black",
                                        linewidth=1.5))
```

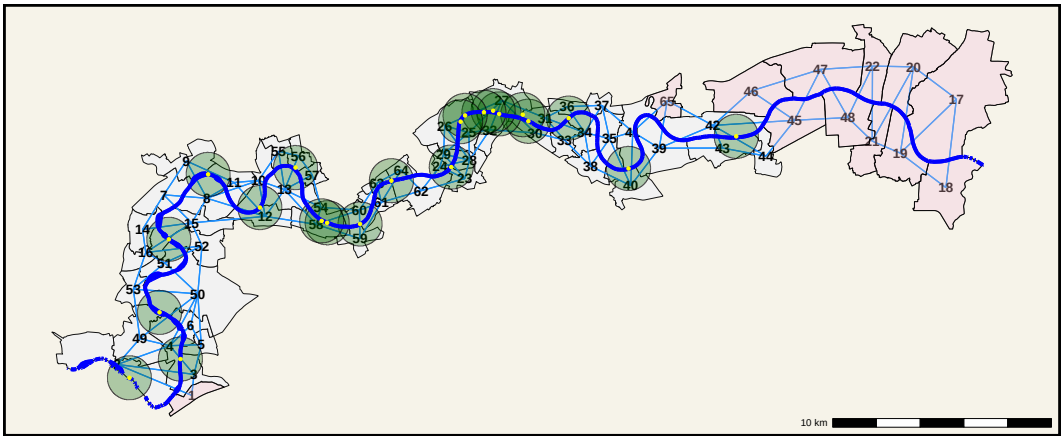


Figure 19: Riverside wards of Greater London. Index number for each ward shown at centroid. Wards which are not within 1 kilometre of a crossing are shaded pink.

This allows us to easily cut the ties across the river for these units by using the function `st_force_cut_nb()`. [3] Having made these adjustments, `st_quickmap_nb()` now shows a connectivity structure (Figure 20) which reflects our hypothesis of how influence should extend across the river in the presence or absence of crossings.

This example shows that the pre-functions of **sfislands** have uses for situations which do not involve islands. They can be used to apply domain knowledge to easily design the most appropriate neighbourhood structure.

```
# enforce cuts for the links where there is no crossing

cut_df <- tribble(
  ~x, ~y,
  18, 17,
  19, 17,
  19, 20,
  20, 21,
  21, 22,
  47, 48,
  45, 46,
  45, 47,
  39, 65,
  1, 2
)
st_bridges(london[riverside,], "NAME") |>
  st_force_cut_nb(xy_df = cut_df) |>
  st_quickmap_nb(bordercol = "black", bordersize = 0.5, linksize = 0.5) +
  geom_sf(data=no_touch_buffer, fill = "pink", alpha = 0.3) +
  geom_sf(data=crossings_roadped |> st_buffer(1000),
          fill= "darkgreen", alpha = 0.3) +
  geom_sf(data=thames, colour = "blue", linewidth = 1.5) +
  annotation_scale(location = "br") +
  coord_sf(datum=NA) +
  theme(panel.background = element_rect(fill = "#F6F3E9", colour = "black",
                                        linewidth = 1.5))
```

[3] While we are using the index of the units in this example, the function also accepts names as arguments which may be more convenient in some circumstances.

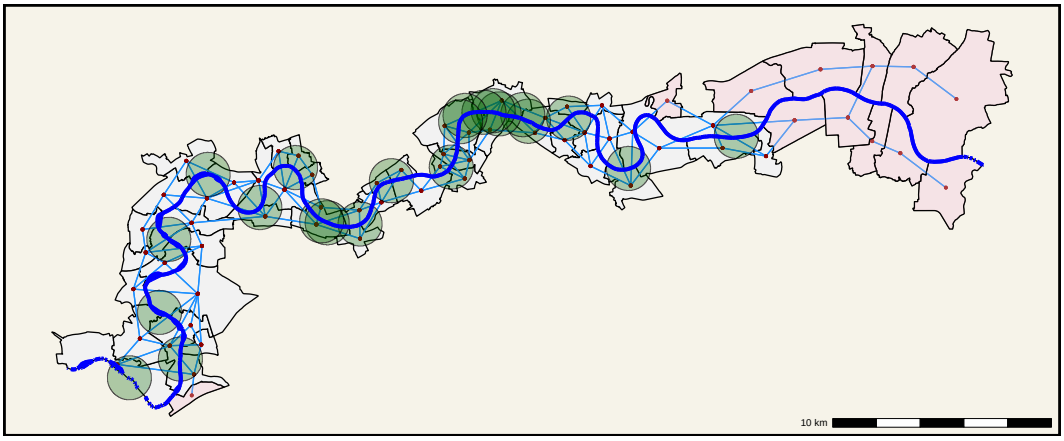


Figure 20: Riverside wards of Greater London. Contiguities across the river have been cut for the pink wards.

7 Summary

These examples have shown the varying scenarios in which **sfislands** can be useful. It aims to contribute to spatial modelling by making an awkward area less awkward. Rather than having a default attitude of ignoring islands when building neighbourhood structures based on contiguity, it offers a convenient system for enforcing linkages if that is deemed to be the most appropriate course of action. Even when no islands are present, it provides a simple procedure for tailoring a neighbourhood structure with bespoke contiguities to match a given hypothesis. It also provides helper functions to use these structures in spatial regression models, notably those built with **mgcv**, which streamline the human effort necessary to examine the estimates. In future, compatibility with other modelling packages can be added to broaden the package's capabilities.

8 Acknowledgements

This publication has emanated from research conducted with the financial support of Taighde Éireann – Research Ireland under Grant number 18/CRT/6049. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

References

S. Andréfouët, M. Paul, and A. R. Farhan. Indonesia's 13558 islands: A new census from space and a first step towards a One Map for Small Islands Policy. *Marine Policy*, 135: 104848, 2022. URL http://dx.doi.org/10.1016/j.marpol.2021.104848. [p95]

H. Bakka, H. Rue, G.-A. Fuglstad, A. Riebler, D. Bolin, E. Krainski, D. Simpson, and F. Lindgren. Spatial modelling with R-INLA: A review, 2018. URL https://arxiv.org/abs/1802.06350. [p86]

D. Bates, M. Mächler, B. Bolker, and S. Walker. Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, 67(1):1–48, 2015. URL https://doi.org/10.18637/jss.v067.i01. [p91]

J. Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(2):192–225, 1974. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1974.tb00999.x. [p90]

R. S. Bivand. R packages for analyzing spatial data: A comparative case study with areal data. *Geographical Analysis*, 54(3):488–518, 2022. URL https://doi.org/10.1111/gean.12319. [p84]

R. S. Bivand and B. A. Portnov. Exploring spatial data analysis techniques using R: The case of observations with no neighbors. In L. Anselin, R. J. G. M. Florax, and S. J. Rey, editors, *Advances in Spatial Econometrics*, Advances in Spatial Science, chapter 6, pages 121–142. Springer, 2004. URL https://ideas.repec.org/h/spr/adspcp/978-3-662-05617-2_6.html. [p85]

Á. Briz-Redón, A. Iftimi, J. F. Correcher, J. De Andrés, M. Lozano, and C. Romero-García. A comparison of multiple neighborhood matrix specifications for spatio-temporal model fitting: A case study on COVID-19 data. *Stochastic Environmental Research and Risk Assessment*, 36(1):271–282, 2021. URL http://dx.doi.org/10.1007/s00477-021-02077-y. [p84]

P.-C. Bürkner. brms: An R package for Bayesian multilevel models using Stan. *Journal of Statistical Software*, 80(1):1–28, 2017. URL https://doi.org/10.18637/jss.v080.i01. [p86]

E. W. Duncan, N. M. White, and K. Mengersen. Spatial smoothing in Bayesian models: A comparison of weights matrix specifications and their impact on inference. *International Journal of Health Geographics*, 16(1), 2017. URL http://dx.doi.org/10.1186/s12942-017-0120-x. [p84]

A. Earnest, G. Morgan, K. Mengersen, L. Ryan, R. Summerhayes, and J. Beard. Evaluating the effect of neighbourhood weight matrices on smoothing properties of conditional autoregressive (CAR) models. *International Journal of Health Geographics*, 6(1):54, 2007. URL http://dx.doi.org/10.1186/1476-072X-6-54. [p84]

A. Freni-Sterrantino, M. Ventrucci, and H. Rue. A note on intrinsic conditional autoregressive models for disconnected graphs. *Spatial and Spatio-temporal Epidemiology*, 26:25–34, 2018. URL http://dx.doi.org/10.1016/j.sste.2018.04.002. [p85]

K. Horan, K. Domijan, and C. Brunsdon. *sfislands: Streamlines the process of fitting areal spatial models*, 2024. URL https://horankev.github.io/sfislands/. R package version 1.1.2. [p85]

A. Kassambara. *ggpubr: 'ggplot2'-based publication ready plots*, 2023. URL https://CRAN.R-project.org/package=ggpubr. R package version 0.6.0. [p91]

J. Parry and D. H. Locke. *sfdep: Spatial dependence for simple features*, 2024. URL https://sfdep.josiahparry.com. R package version 0.2.4. [p84]

E. Pebesma. Simple features for R: Standardized support for spatial vector data. *The R Journal*, 10(1):439–446, 2018. URL https://doi.org/10.32614/RJ-2018-009. [p84]

J. Pinheiro, D. Bates, and R Core Team. *nlme: Linear and nonlinear mixed effects models*, 2023. URL https://CRAN.R-project.org/package=nlme. R package version 3.1-164. [p91]

D. Robinson, A. Hayes, and S. Couch. broom: Convert statistical objects into tidy tibbles. 2023. URL https://CRAN.R-project.org/package=broom. [p90]

Stan Development Team. RStan: the R interface to Stan, 2020. URL http://mc-stan.org/. [p88]

W. R. Tobler. A computer movie simulating urban growth in the Detroit region. *Economic Geography*, 46(sup1):234–240, 1970. URL https://www.tandfonline.com/doi/abs/10.2307/143141. Publisher: Routledge. [p84]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. URL https://ggplot2.tidyverse.org. [p85]

Wikipedia. List of crossings of the River Thames — Wikipedia, the free encyclopedia, 2024. URL http://en.wikipedia.org/w/index.php?title=List%20of%20crossings%20of%20the%20River%20Thames&oldid=1184426738. Accessed 2024-03-15. [p102]

S. N. Wood. Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. 73:3–36, 2011. URL https://CRAN.R-project.org/web/packages/mgcv/index.html. [p85]

Kevin Horan
Hamilton Institute, Maynooth University
Maynooth
Co. Kildare, Ireland
https://github.com/horankev
ORCiD: 0009-0003-9378-0084
kevin.horan.2021@mumail.ie

Katarina Domijan
Department of Mathematics and Statistics, Maynooth University
Maynooth
Co. Kildare, Ireland
ORCiD: 0000-0002-4268-2236
katarina.domijan@mu.ie

Chris Brunsdon
National Centre for Geocomputation, Maynooth University
Maynooth
Co. Kildare, Ireland
ORCiD: 0000-0003-4254-1780
Christopher.Brunsdon@mu.ie

rPDBapi: A Comprehensive R Package Interface for Accessing the Protein Data Bank

by Selcuk Korkmaz and Bilge Eren Yamasan

Abstract The RCSB Protein Data Bank (PDB) is a foundational resource for bioinformatics and structural biology, providing essential 3D structural data for large biological molecules. This data underpins advancements in drug discovery and genomics. The rPDBapi package aims to bridge the existing gap in accessing PDB data through the R programming language, offering a user-friendly, powerful interface that enhances the accessibility of PDB data for the R community. Leveraging the PDB's XML-based API, rPDBapi simplifies the creation of custom queries, making data retrieval more efficient. It also introduces advanced functionalities within R, such as customized search capabilities and direct data manipulation, demonstrating its potential to significantly impact research workflows in bioinformatics and structural biology. Through detailed examples, the paper illustrates how rPDBapi enables precise data retrieval and analysis, facilitating a deeper understanding of molecular functions and interactions. This contribution makes structural biology data more accessible to researchers using R, simplifying access to PDB resources.

1 Introduction

The RCSB Protein Data Bank (PDB) is an essential resource in bioinformatics and structural biology, offering a vast repository of 3D structural data of large biological molecules (Burley et al., 2022). This data is crucial for researchers aiming to unravel molecular functions and interactions, playing a pivotal role in advancing drug discovery and genomic studies (Korkmaz et al., 2018). Traditionally, access to the PDB has been facilitated through various web interfaces and a suite of programmatic tools, with Python-based solutions like PyPDB being particularly prominent (Gilpin, 2016). However, the breadth and depth of data housed within the PDB necessitate diverse access methods to cater to the specific needs and technical preferences of the global research community.

The complexity and sheer volume of the data underscore the importance of developing more intuitive and flexible tools that can streamline the process of data retrieval and analysis, thereby enhancing the efficiency of scientific research and innovation. In this context, the accessibility of the PDB through these platforms not only democratizes the availability of critical scientific data but also encourages the continuous evolution of computational tools designed to meet the growing demands of multidisciplinary research fields.

To support diverse queries, ranging from atomic interactions to protein-protein interactions within large assemblies, the PDB data hierarchy is structured to provide detailed information at various levels of biological organization. At the core are individual atoms, which are grouped into residues. Residues are organized into chains, typically representing a single protein or nucleic acid molecule. Chains are then assembled into biological assemblies, which may represent complete macromolecular complexes as they exist in biological systems. This hierarchical structure facilitates the comprehensive representation of complex biological entities.

To elaborate, the PDB data organization can be detailed as follows:

Entry:

- Each structure in the PDB is designated by a unique alphanumeric ID (PDB ID, e.g., 1Q2W) and includes annotations such as the title of the structure, the list of depositors, deposition date, release date, and experimental details.
- The entry also includes information about the methodology used for structure determination (e.g., X-ray crystallography, NMR spectroscopy, cryo-electron microscopy)

and the resolution of the structure if applicable.
- Additional metadata may include the biological significance of the structure, references to related literature, and the conditions under which the structure was determined.

Entity:

This level describes the distinct molecules within entries, categorized into three types:

- **polymer_entity:**
  - Represents macromolecules such as proteins, DNA, and RNA, defined by their amino acid or nucleotide sequences.
  - Annotations at this level include details about the biological source, such as the organism from which the sequence was derived, expression system, and any modifications to the sequence (e.g., post-translational modifications in proteins).
  - Polymer entities also include secondary structure annotations, domains, and functional sites (e.g., active sites in enzymes).
- **branched_entity:**
  - Encompasses carbohydrates composed of saccharide units linked by glycosidic bonds, which can be linear or branched.
  - These entities often play crucial roles in biological recognition processes, such as cell-cell communication, and are annotated with information about their specific linkages and glycosidic bond types.
  - The branched entity level may also include annotations about the biological function of the carbohydrate, its involvement in glycoproteins, and interactions with other biomolecules.
- **nonpolymer_entity:**
  - Covers small molecules like enzyme cofactors, ligands, inhibitors, ions, and solvent molecules.
  - Annotations at this level include details about the chemical structure (e.g., SMILES, InChI), molecular weight, charge, and other relevant physicochemical properties.
  - Nonpolymer entities are often involved in critical biological processes, such as enzyme catalysis, signaling, or structural stability, and are annotated with their roles within the macromolecular complex.

Entity Instance:

- Refers to the specific copies of entities within entries, often called "chains."
- This level provides information that can vary between instances, such as structural connectivity, secondary structure, and domain organization.
- It also includes details about the conformation of the chain in the structure, including information on any alternative conformations (e.g., alternate locations of side chains or backbone conformers).
- The entity instance level allows the representation of multiple conformations of the same entity within a single structure, which can be critical for understanding dynamic aspects of biomolecular function.

Assembly:

- Describes the biological assemblies, which represent the quaternary structure of the macromolecular complex as it exists in vivo.
- Annotations include details about the number and types of chains involved in the assembly, the symmetry and transformations required to generate the assembly, and evidence supporting the biological relevance of the assembly (e.g., experimental validation, computational prediction).

- The assembly level may also include information about the stoichiometry of the complex, interactions between different chains (e.g., protein-protein or protein-nucleic acid interactions), and the overall topology of the assembly.
- For multimeric complexes, this level is crucial for understanding the functional organization and cooperative interactions within the assembly.

Chemical Component:

- Includes all residues (e.g., amino acids in proteins, nucleotides in DNA/RNA) and small molecules found in entries.
- Annotations at this level encompass chemical descriptors such as SMILES (Simplified Molecular Input Line Entry System) and InChI (International Chemical Identifier), chemical formula, systematic names, and alternative names (e.g., common names or trade names for small molecules).
- The chemical component level also includes information about covalent modifications, such as post-translational modifications in proteins (e.g., phosphorylation, glycosylation) or synthetic modifications in nucleic acids.
- For ligands and cofactors, this level may also include details about binding affinities, interaction partners, and the specific role of the chemical component in the biological function of the macromolecular complex.

This hierarchical data organization in the PDB ensures that the complex nature of biological macromolecules and their interactions are accurately represented, enabling researchers to extract detailed insights from structural data. Whether the focus is on atomic-level interactions, the role of specific residues, or the overall architecture of a large protein complex, the PDB structure allows for comprehensive and flexible data exploration.

A notable gap remains in providing streamlined, R-based access to the PDB—a gap that the **rPDBapi** package seeks to close (Korkmaz and Yamasan, 2024). By offering a user-friendly, powerful interface to the PDB designed for the R programming community, **rPDBapi** significantly enhances the accessibility of critical PDB data. This leap in accessibility is particularly impactful given R's widespread adoption across statistical computing and bioinformatics, disciplines that rely heavily on comprehensive and intricate data analysis.

rPDBapi leverages the PDB's existing XML-based API to streamline the process of crafting custom queries, thereby simplifying what was once a complex task. This facility to generate customized queries with ease represents a fundamental shift towards more versatile and efficient data retrieval practices. It ensures that researchers can quickly adapt their data-gathering strategies to suit the unique needs of their projects, from exploratory analyses in genomic studies to the identification of potential drug targets.

Moreover, **rPDBapi** introduces advanced functionalities within R, such as specialized search capabilities, sequence retrieval, and direct data manipulation. The package's ability to construct customized JSON queries for specific data retrieval needs, coupled with its support for extracting FASTA sequences and performing nuanced searches across the PDB, exemplifies the breadth of its utility. Whether facilitating the download of structure files for molecular modeling, enabling batch queries for sequence analysis, or providing direct access to paper titles and chemical descriptions within the PDB, **rPDBapi** stands as a testament to the evolving landscape of bioinformatics tools. It not only simplifies access to a vast array of structural biology data but also fosters an environment where scientific discovery is accelerated through efficient data analysis and workflow integration. The workflow for querying and retrieving data from the RCSB PDB using the rPDBapi package is illustrated in Figure 1, highlighting the interaction between core functions, API requests, and downstream data processing.

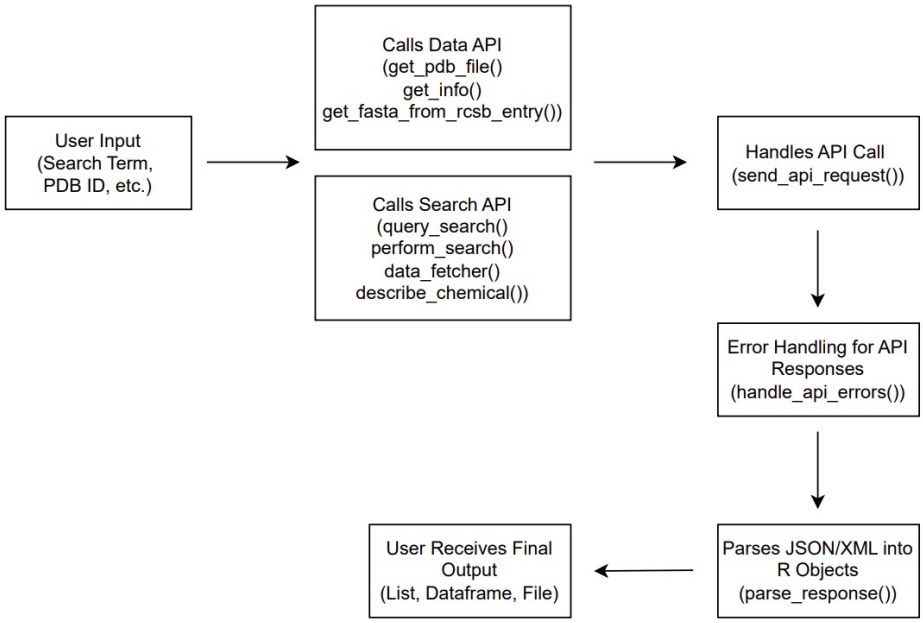


Figure 1: Data Retrieval and Query Workflow for RCSB PDB API Using rPDBapi Package

In essence, **rPDBapi** aims to significantly improve the way researchers interact with the RCSB PDB through programmatic means. By offering a more accessible pathway for R users to engage with PDB data, the package seeks to enhance the usability and flexibility of data exploration and analysis within the fields of bioinformatics and structural biology.

2 Using the Package

The accessibility and retrieval of specific data from the RCSB PDB are facilitated through various search functions, each designed to meet the nuanced needs of the scientific community. These functions are built on a foundation of low-level core functions—`send_api_request`, `handle_api_errors`, and `parse_response`—which ensure efficient interaction with the RCSB PDB's APIs. `send_api_request` handles the actual communication with the PDB server, while `handle_api_errors` manages potential issues during the request, and parse_response processes and extracts the relevant data from the server's responses. This section explores the details of these search functions, illuminating their utility in fetching lists of PDB IDs based on diverse search criteria.

2.1 Installation and Setup

rPDBapi package can be installed from CRAN (The Comprehensive R Archive Network) and loaded as follows:

```
# install.packages("rPDBapi", repos = "http://cran.us.r-project.org")
library("rPDBapi")
```

The package is currently in a state of active development. The latest version in development can be accessed via GitHub (`https://github.com/selcukorkmaz/rPDBapi`). This paper was composed utilizing **rPDBapi** version 2.1.

2.2 Retrieving PDB IDs

Utilizing `query_search` function to query PDB database enables researchers to uncover a wealth of structural information pertinent to specific biological entities or processes. These

queries can range from searches for structures associated with particular terms, such as specific molecules or complexes like hemoglobin, to more refined strategies targeting data linked to unique PubMed IDs, source organisms identified by NCBI Taxonomy IDs, or structures determined by specific experimental methodologies. Further, the PDB's functionality to compare protein structure similarities and conduct advanced searches based on authorship, organism, UniProt IDs, or PFAM numbers exemplifies the database's versatility and its role in facilitating detailed comparative analyses and structural elucidation.

A primary method of querying the PDB involves searching for structures related to a specific term, such as a biological molecule or complex. For instance, executing a query for *hemoglobin* yields a list of PDB IDs corresponding to structures associated with hemoglobin assemblies. This search modality is instrumental in gathering a broad spectrum of structural data pertinent to a particular molecule or complex, facilitating comparative analyses and structural elucidation.

The following code snippet exemplifies a search that returns the first 6 PDB IDs related to hemoglobin, showcasing the breadth of data available for this fundamental biological machinery.

```
pdbs = query_search(search_term = "hemoglobin")
head(pdbs)

#> [1] "2PGH" "3PEL" "3GOU" "6IHX" "1G08" "1G09"
```

Another refined search strategy involves querying the PDB using PubMed ID numbers. This approach allows researchers to find structures that are directly linked to specific scientific publications, ensuring a bridge between the structural data and the corresponding bibliographic sources.

```
pdbs = query_search(search_term = 32453425, query_type = "PubmedIdQuery")
pdbs

#> [1] "6XTX" "6XTY"
```

Here, structures associated with a given PubMed ID are retrieved, offering a targeted exploration of structural data underpinned by the scientific literature.

Advanced search functions further refine data retrieval capabilities. Queries can be constructed based on authorship, enabling the exploration of contributions from specific researchers or research groups. Similarly, searches can target structures derived from particular organisms or those associated with specific UniProt IDs or PFAM numbers, enhancing the precision and relevance of search results. Building on the advanced search capabilities, researchers can explore deeper into the structural biology landscape by querying the PDB for entries associated with specific Pfam IDs. Pfam IDs are used to identify protein domains, which are conserved parts of a protein's sequence and structure. The following example focuses on the Pfam ID "PF00069", which corresponds to the protein kinase domain, a common and functionally important domain in many signaling proteins.

```
# Search by PFAM number (protein kinase domain)
pdbs = query_search(search_term = "PF00069", query_type="pfam")
head(pdbs)

#> [1] "1A06" "1A9U" "1APM" "1AQ1" "1ATP" "1B38"
```

By querying the PDB for this specific Pfam ID, we can retrieve all related PDB entries and analyze the distribution of protein lengths within this domain family. This analysis provides insights into the structural variability and conservation of domain length among different proteins.

We then focus on the first 100 PDB entries for analysis, although this number can be adjusted according to research needs:

```
# Load necessary libraries
library(rPDBapi)
library(Biostrings)
library(ggplot2)

# Get first 100 entries
pdbs <- pdbs[1:100]
```

The next step involves calculating the length of each protein sequence associated with the retrieved PDB IDs. The get_protein_length function is used to fetch the FASTA sequence for each PDB ID, convert it into an AAString using the **Biostrings** package, and then determine the number of amino acids (protein length) in the sequence:

```
get_protein_length <- function(pdb_id) {
  fasta <- get_fasta_from_rcsb_entry(pdb_id, chain_id = "A")
  sequence <- AAString(fasta)
  return(nchar(sequence))
}

# Apply the function to all retrieved PDB IDs
protein_lengths <- sapply(pdbs, get_protein_length)

# Create a data frame for plotting
length_data <- data.frame(PDB_ID = names(protein_lengths), Length = protein_lengths)

head(length_data)

#>      PDB_ID Length
#> 1A06   1A06    332
#> 1A9U   1A9U    379
#> 1APM   1APM    350
#> 1AQ1   1AQ1    298
#> 1ATP   1ATP    350
#> 1B38   1B38    299
```

Finally, we visualize the distribution of protein lengths using a histogram (Figure 2). This plot provides a clear representation of how protein lengths vary within the protein kinase domain family, offering valuable insights into the structural characteristics of this important protein domain.

```
# Plot the distribution of protein lengths
ggplot(length_data, aes(x = Length)) +
  geom_histogram(binwidth = 10, fill = "blue", color = "black", alpha = 0.7) +
  labs(x = "Protein Length (Amino Acids)",
       y = "Frequency") +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 10),
    axis.title.x = element_text(size = 8),
    axis.title.y = element_text(size = 8),
    axis.text = element_text(size = 7)
  )
```

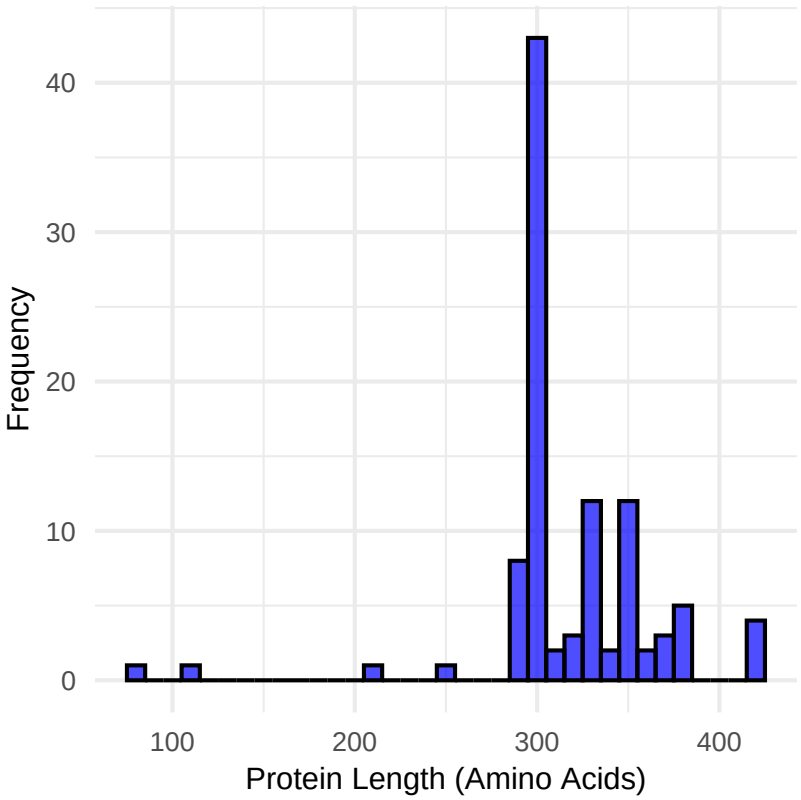


Figure 2: Histogram of protein lengths in the kinase domain family.

These examples illustrate the depth and versatility of search functions available for querying the PDB, each designed to support specific research objectives and inquiries. Through these search mechanisms, the PDB continues to serve as an invaluable resource for the scientific community, enabling the advancement of knowledge in the structural biology domain. For more examples and additional functionalities, please refer to the package documentation.

2.3 Data Search

To enhance the utility of the RCSB PDB database for researchers, the rPDBapi package provides functions such as `perform_search`, `data_fetcher`, and `describe_chemical`. While these functions facilitate advanced search capabilities and streamlined data retrieval, it is important to acknowledge that they primarily serve as wrappers for building queries and parsing responses from the powerful RCSB PDB Data API. The real strength of these functions lies in their ability to simplify and customize API interactions, enabling users to execute refined searches and efficiently fetch data based on specific criteria without needing to directly interact with the API itself.

`perform_search(...)` function

The perform_search function is designed to enhance the way researchers interact with the RCSB PDB database by allowing for highly customizable searches. Here is a general overview of the function:

```
perform_search(
  search_operator,
  return_type = "ENTRY",
  request_options = NULL,
  return_with_scores = FALSE,
```

```
  return_raw_json_dict = FALSE,
  verbosity = TRUE
)
```

This function connects with the RCSB PDB's RESTful API and lets users search the database using specific criteria provided through the search_operator parameter. This capability is particularly useful for complex searches where precise results are needed.

To refine searches, the `perform_search` function can be combined with various custom search operators. These operators enable the construction of complex search queries customized to specific needs. Detailed documentation for each search operator is available in the RCSB PDB Search Operators (https://search.rcsb.org/#search-operators). Moreover, Searchable attributes include annotations from the mmCIF dictionary, external resources, and additional annotations provided by RCSB PDB. Internal additions to the mmCIF dictionary and external resource annotations are prefixed with 'rcsb_'. For a comprehensive list of available attributes for text searches, refer to the Structure Attributes Search (https://search.rcsb.org/structure-search-attributes.html) and Chemical Attributes Search (https://search.rcsb.org/chemical-search-attributes.html) pages.

Default Search Operator

The DefaultOperator function constructs a basic search operator using a single value. This operator is useful for general searches where a straightforward query is required.

```
DefaultOperator <- function(value) {
  res <- list(value = value)
  structure(res, class = c("DefaultOperator", class(res)))
}
```

Exact Match Search Operator

The ExactMatchOperator function allows for precise search operations by matching an exact attribute value. This is particularly useful when the user needs to retrieve entries that exactly match a given criterion.

```
ExactMatchOperator <- function(attribute, value) {
  res <- list(attribute = attribute, value = value, operator = "exact_match")
  structure(res, class = c("ExactMatchOperator", class(res)))
}
```

Inclusion Search Operator

The InOperator function is designed for scenarios where the search needs to include values from a specified set. This operator is ideal for filtering results based on a list of possible values.

```
InOperator <- function(attribute, value) {
  res <- list(attribute = attribute, operator = "in", value = value)
  structure(res, class = c("InOperator", class(res)))
}
```

Contains Words and Contains Phrase Operators

For text-based searches, the ContainsWordsOperator and ContainsPhraseOperator functions are invaluable. These operators search for attributes containing certain words or specific phrases, respectively, making them perfect for querying descriptive fields.

```
ContainsWordsOperator <- function(attribute, value) {
  res <- list(attribute = attribute, operator = "contains_words", value = value)
  structure(res, class = c("ContainsWordsOperator", class(res)))
}
```

```
ContainsPhraseOperator <- function(attribute, value) {
  res <- list(attribute = attribute, operator = "contains_phrase", value = value)
  structure(res, class = c("ContainsPhraseOperator", class(res)))
}
```

Comparison and Range Search Operators

The ComparisonOperator and RangeOperator functions enable users to perform comparison checks or specify a range for attribute values. These operators are essential when the search requires evaluating attributes against numerical or date ranges.

```
ComparisonOperator <- function(attribute, value, comparison_type) {
  if (ComparisonType[[comparison_type]] == "not_equal") {
    param_dict <- list(operator = "equals", negation = TRUE)
  } else {
    param_dict <- list(operator = ComparisonType[[comparison_type]])
  }
  param_dict$attribute <- attribute
  param_dict$value <- value
  structure(param_dict, class = c("ComparisonOperator", class(param_dict)))
}

RangeOperator <- function(attribute, from_value, to_value, include_lower = TRUE,
                          include_upper = TRUE, negation = FALSE) {
  res <- list(
    operator = "range",
    attribute = attribute,
    negation = negation,
    value = list(from = from_value, to = to_value)
  )
  structure(res, class = c("RangeOperator", class(res)))
}
```

Comparison and Range Search Operators

Lastly, the ExistsOperator function checks the existence of a specific attribute within the database, ensuring that the search results only include entries where the attribute is present.

```
ExistsOperator <- function(attribute) {
  res <- list(attribute = attribute, operator = "exists")
  structure(res, class = c("ExistsOperator", class(res)))
}
```

One of the key features of `perform_search` is its ability to return different types of data depending on the user's needs. By default, it returns general information about PDB entries, but users can adjust this to retrieve specific types of data, such as structural or assembly details.

This function allows users to specify the format of the returned data by setting the return_type parameter. This parameter determines the level of detail and specificity of the search results. The available return types include:

- **entry:** Returns a list of PDB IDs, providing a general overview of relevant entries.
- **assembly:** Returns a list of PDB IDs appended with assembly IDs, formatted as [pdb_id]-[assembly_id]. This is particularly useful for researchers interested in biological assemblies.

- **polymer_entity:** Returns a list of PDB IDs appended with entity IDs in the format [pdb_id]_[entity_id], corresponding to polymeric molecular entities. This is ideal for focusing on specific protein or nucleic acid chains within the structure.
- **non_polymer_entity:** Returns a list of PDB IDs appended with entity IDs in the format [pdb_id]_[entity_id], corresponding to non-polymeric entities, such as ligands. This allows for the identification of small molecules or other non-polymeric components within a structure.
- **polymer_instance:** Returns a list of PDB IDs appended with asym IDs in the format [pdb_id].[asym_id], corresponding to instances of certain polymeric molecular entities, also known as chains. The asym_id corresponds to the _label_asym_id from the mmCIF schema, which may differ from the _auth_asym_id selected by the author at the time of deposition.
- **mol_definition:** Returns a list of molecular definition identifiers, which include chemical component entries identified by alphanumeric codes (e.g., ATP, ZN) and BIRD entries identified by BIRD IDs (e.g., PRD_000154). This return type is useful for chemical and molecular component searches.

Additionally, the perform_search function allows for further customization through options like request_options, which can be used to fine-tune the search, such as filtering results by certain criteria or setting constraints like date ranges or experimental methods.

For users who want to prioritize the most relevant results, the function can include relevance scores with the search results by setting the return_with_scores option to TRUE. This helps in quickly identifying the most pertinent entries in a large set of results.

In some cases, researchers may want to access the raw data returned by the PDB API. The return_raw_json_dict option allows users to retrieve this data in its original JSON format for further processing or analysis.

Finally, the verbosity option controls how much information is displayed during the search process. When enabled, it provides detailed feedback on the search, which can be helpful for understanding how the function is working or troubleshooting any issues.

To further enhance the search experience, the RCSB PDB Search API consolidates requests to a variety of specialized search services, allowing for highly targeted data retrieval. These services include:

- **text:** Performs attribute searches against textual annotations associated with PDB structures. This service is useful for searching within structure-related information.
- **text_chem:** Similar to the text service but focuses on molecular definitions. It's ideal for searching through chemical attributes associated with PDB entries.
- **full_text:** Executes unstructured searches across multiple text attributes related to both PDB structures and molecular definitions, making it a powerful tool for broad searches that need to cover a wide range of text fields.
- **sequence:** Employs the MMseqs2 software to perform fast sequence matching searches, similar to BLAST, based on a user-provided FASTA sequence. It supports protein, DNA, and RNA sequence searches with configurable cutoffs like E-value or percentage identity.
- **seqmotif:** Allows for short motif searches against nucleotide or protein sequences using simple patterns, PROSITE-like syntax, or regular expressions.
- **structure:** Conducts fast 3D shape matching searches using a BioZernike descriptor strategy, which can be performed in strict or relaxed modes depending on the desired accuracy.

- **strucmotif:** Facilitates structure motif searches across all available PDB structures, providing a focused approach to identifying similar structural motifs.
- **chemical:** Offers comprehensive chemical searches based on molecular formula, chemical structure, and a variety of matching criteria. It includes powerful tools for searching chemical fingerprints and performing graph-based matching for exact, strict, or relaxed criteria.

Each of these services allows researchers to customize their searches precisely to their needs, whether they are looking for specific chemical components, structural motifs, or sequence patterns.

Consider a scenario where a researcher wishes to find all non-polymer entities related to "Mus musculus" and "Homo sapiens" within the PDB. The `perform_search` function can be configured as follows:

```
search_operator = InOperator(value=c("Mus musculus", "Homo sapiens"),
                   attribute="rcsb_entity_source_organism.taxonomy_lineage.name")
return_type = "NONPOLYMER_ENTITY"
results = perform_search(search_operator, return_type)
head(results)

#> [1] "12GS_3" "13GS_4" "1A1F_4" "1A1J_4" "1A2C_5" "1A3B_4"
```

Faceted queries (or facets) add another layer of functionality to the perform_search function by allowing users to group and perform calculations on PDB data through a simple search query. Facets organize search results into categories, or "buckets," based on specified field values.

When the facets property is included in the request_options of a search request, the results are returned along with numerical counts that indicate how many matching IDs were found for each facet term. This allows researchers to quickly gauge the distribution of search results across different categories. If the query context is omitted in the search request but facets are specified, the response will contain only the facet counts, offering a summary view of the data distribution.

By combining faceted queries with the other flexible search operators and return types, the perform_search function becomes an incredibly powerful tool for bioinformatics research, enabling users to conduct comprehensive and precise searches within the RCSB PDB database.

Below is an example that demonstrates how to use the ComparisonOperator to filter results based on a specific date (2019-08-20), along with faceted queries to categorize the search results by experimental methods.

```
# Create the ComparisonOperator for the date
operator_date <- ComparisonOperator(
    attribute = "rcsb_accession_info.initial_release_date",
    value = "2019-08-20",
    comparison_type = "GREATER"
)

# Define the facets for request options
request_options <- list(
    facets = list(
        list(
            name = "Methods",
            aggregation_type = "terms",
            attribute = "exptl.method"
        )
```

```
    )
)

# Specify the return type
return_type <- "ENTRY"

# Perform the search with the specified return type and request options
results <- perform_search(
    search_operator = operator_date,
    return_type = return_type,
    request_options = request_options
)

# Display the results
results

#>  [1] "3IX2" "3QOB" "4FPQ" "5BK8" "5BKA" "5BKF" "5BKG" "5BKH" "5BKI" "5BKJ"
```

Kinases are a large family of enzymes that play crucial roles in signal transduction and cellular regulation. To analyze the sequence identity among different kinase-related proteins, we first perform a search for PDB entries using the keyword "protein kinase."

```
# Load necessary libraries
library(rPDBapi)
library(Biostrings)
library(msa)
library(r3dmol)
library(gplots)

set.seed(123)
# Define the search operator for a full-text search using the keyword "protein kinase"
search_operator <- list(
  type = "terminal",
  service = "full_text",
  parameters = list(value = "protein kinase")
)

# Perform the search for PDB entries
protein_entities <- perform_search(
  search_operator = search_operator,
  return_type = "ENTRY"
)

# Display the first 10 PDB IDs retrieved
protein_entities <- protein_entities[1:10]
protein_entities

#>  [1] "1QK1" "1U0R" "1RQF" "4UY9" "7L26" "1V0P" "4G3D" "7ZP0" "5CEN" "6CD6"
```

The search retrieves PDB entries associated with protein kinases. We then select the first 10 entries for further analysis. The next step involves retrieving the amino acid sequences for these proteins and performing multiple sequence alignment using the **msa** package.

```
# Function to retrieve and process protein sequences
get_protein_sequence <- function(pdb_id) {
  fasta <- get_fasta_from_rcsb_entry(pdb_id, chain_id = "A")
```

```
  AAString(fasta)
}

# Retrieve protein sequences for the selected PDB IDs
sequences <- lapply(protein_entities, get_protein_sequence)
names(sequences) <- protein_entities

# Perform multiple sequence alignment
alignment <- msa(AAStringSet(sequences), method = "ClustalW")

#> use default substitution matrix

# Convert alignment to a matrix for pairwise sequence identity calculation
alignment_matrix <- as.matrix(alignment)
```

We calculate the pairwise sequence identity between the aligned sequences, allowing us to quantify the similarity between the kinase-related proteins, and create a heatmap using the **gplots** package (Figure 3).

```
# Function to calculate pairwise sequence identity
pairwise_identity <- function(seq1, seq2) {
  sum(seq1 == seq2) / length(seq1) * 100
}

# Calculate pairwise sequence identity
n <- length(sequences)
identity_matrix <- matrix(NA, n, n)
rownames(identity_matrix) <- names(sequences)
colnames(identity_matrix) <- names(sequences)

for (i in 1:n) {
  for (j in i:n) {
    identity_matrix[i, j] <- pairwise_identity(alignment_matrix[i, ],
                                               alignment_matrix[j, ])
    identity_matrix[j, i] <- identity_matrix[i, j]
  }
}

par(cex.main=1)

# Visualize the pairwise sequence identity as a heatmap
heatmap.2(identity_matrix, xlab = "PDB ID", ylab = "PDB ID",
          col = colorRampPalette(c("blue", "white", "red"))(100),
        labRow = rownames(identity_matrix), labCol = colnames(identity_matrix),
         trace = "none", key = TRUE, key.title = "Identity", key.xlab = "Scale",
          density.info = "none", scale = "none")
```

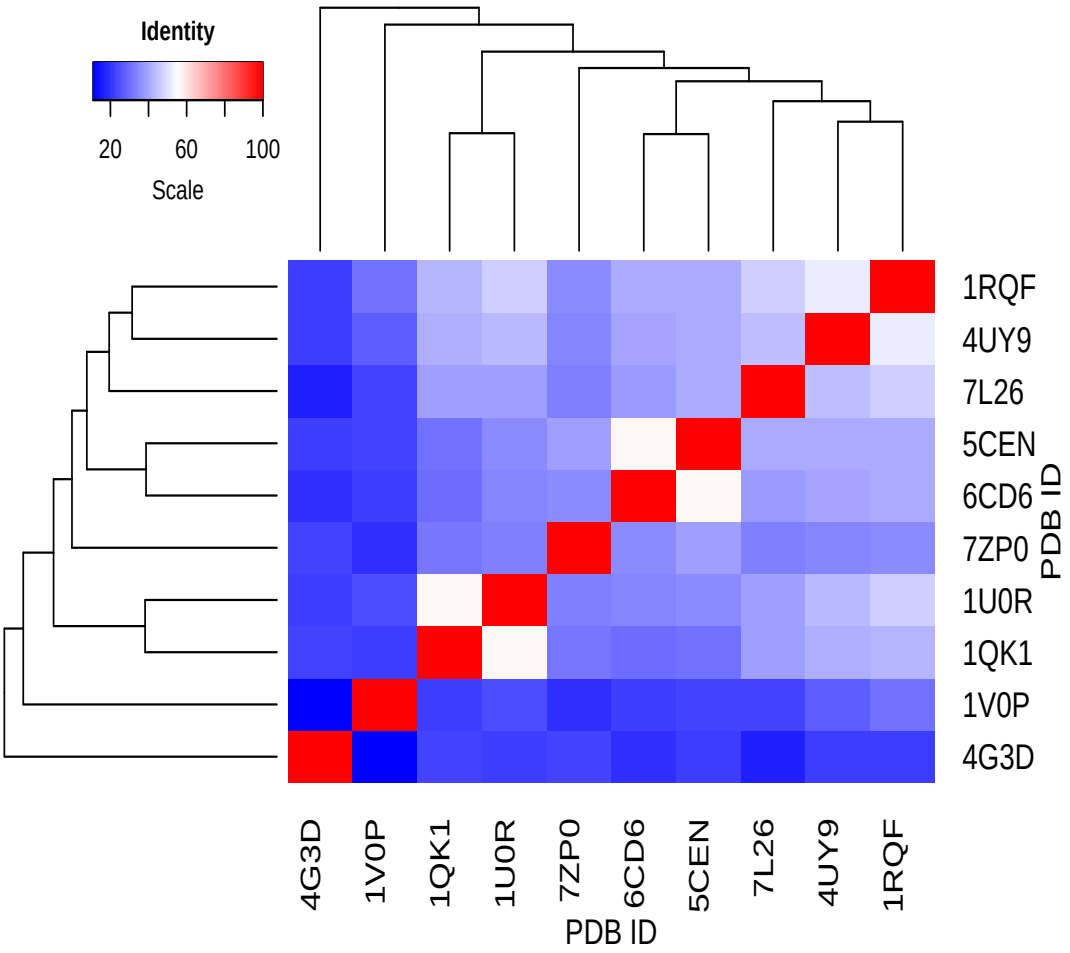


Figure 3: Heatmap of sequence identity among kinase proteins.

The resulting heatmap provides a visual representation of the sequence identity among the selected kinase proteins, with higher identity values indicated by warmer colors.

In addition to sequence analysis, visualizing the 3D structures of proteins can provide valuable insights into their functional mechanisms. We use the **r3dmol** package to create an interactive visualization of the protein structures (Figure 4-5).

```
# Function to visualize the structures using r3dmol with color
visualize_structure <- function(pdb_id) {
  pdb_path <- get_pdb_file(pdb_id, filetype = "pdb", save = TRUE)$path

  viewer <- r3dmol() %>%
    m_add_model(pdb_path, format = "pdb") %>%
    m_set_style(style = m_style_cartoon(
      color = "spectrum"
    )) %>%
    m_zoom_to()

  return(viewer)
}

# Visualize the first protein structure
visualize_structure(protein_entities[1])
```

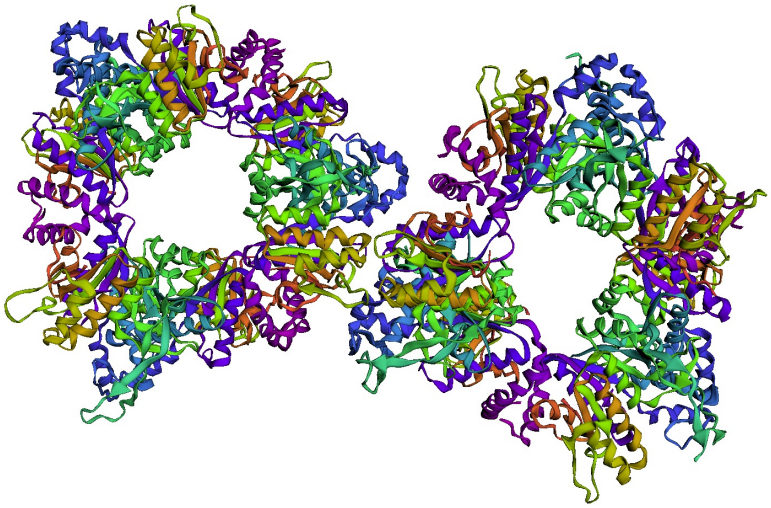


Figure 4: 3D structure of the first kinase protein.

```
# Visualize the second protein structure
visualize_structure(protein_entities[2])
```

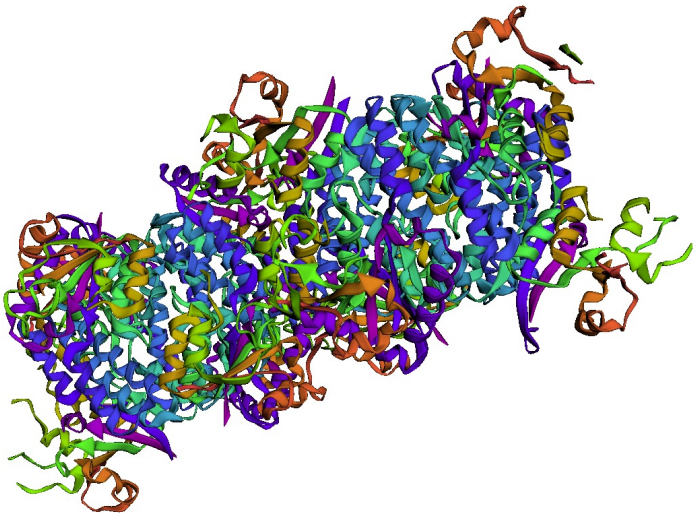


Figure 5: 3D structure of the second kinase protein.

Here, we retrieve and visualize the first two protein structures from the list of kinase-related PDB entries. The visualization uses a "spectrum" color scheme to highlight different regions of the protein structure, providing an intuitive understanding of the protein's 3D conformation.

`data_fetcher(...)` function

In the rapidly evolving field of structural biology, efficient access to specific datasets from the RCSB PDB is crucial for researchers aiming to conduct detailed analyses of biological structures. To cater to this need, the `data_fetcher` function has been developed, streamlining the process of fetching data based on user-defined IDs, data types, and properties. This function exemplifies a customized approach to data retrieval, offering flexibility in how data is fetched and presented, thereby enhancing research workflows.

Consider a scenario where a researcher is interested in retrieving and analyzing CRISPR-associated entries from the RCSB PDB database to elucidate trends over time in the deposition of CRISPR-related structures. The query_search function is employed to identify entries related to CRISPR, leveraging specified properties including molecular weight metric (molecular_weight), experimental method (method), and accession information (deposit_date).

```
properties <- list(rcsb_entry_info = c("molecular_weight"),
                   exptl = "method",
                   rcsb_accession_info = "deposit_date")

ids =  query_search("CRISPR")

head(ids)

#> [1] "2Y9H" "4ILM" "4XTK" "5FCL" "7EI1" "6TUG"
```

The data fetched by `data_fetcher` is structured as a dataframe for analytical convenience, with entries uniquely identified and their deposit dates converted to a standard date format to facilitate temporal analysis.

```
df = data_fetcher(
   id = ids,
   data_type = "ENTRY",
   properties = properties,
   return_as_dataframe = TRUE
 )

head(df)

#> # A tibble: 6 x 4
#>   ID    molecular_weight method            deposit_date
#>   <chr> <chr>            <chr>             <chr>
#> 1 2Y9H  240.96           X-RAY DIFFRACTION 2011-02-14T00:00:00Z
#> 2 4ILM  302.01           X-RAY DIFFRACTION 2012-12-31T00:00:00Z
#> 3 4XTK  226.22           X-RAY DIFFRACTION 2015-12-15T00:00:00Z
#> 4 5FCL  680.57           X-RAY DIFFRACTION 2021-03-30T00:00:00Z
#> 5 7EI1  416.5            X-RAY DIFFRACTION 2020-01-07T00:00:00Z
#> 6 6TUG  147.1            X-RAY DIFFRACTION 2009-03-18T00:00:00Z
```

Subsequent processing extracts the deposit year for each entry, enabling the aggregation of data on an annual basis.

```
# Convert character vector to Date
df$deposit_date <- as.Date(df$deposit_date, "%Y-%m-%dT%H:%M:%SZ")

# Adding year column
df$year <- as.numeric(format(df$deposit_date, "%Y"))

# Counting entries per year
df_summary <- aggregate(ID ~ year, data = df, FUN = length)

head(df_summary)

#>   year ID
#> 1 2004  7
#> 2 2005  2
```

```
#> 3 2006  6
#> 4 2007  2
#> 5 2008  4
#> 6 2009  4
```

The use of the ggplot2 library facilitates the visualization of this aggregated data, with a line plot generated to display the trend in the number of CRISPR-related database deposits over time (Figure 6).

```
ggplot(df_summary, aes(x = year, y = ID)) +
   geom_line(color = "red") +
   xlim(c(min(df_summary$year), max(df_summary$year)+1)) +
   geom_point() +
   labs( x = 'Year', y = 'Number of Deposits')
```

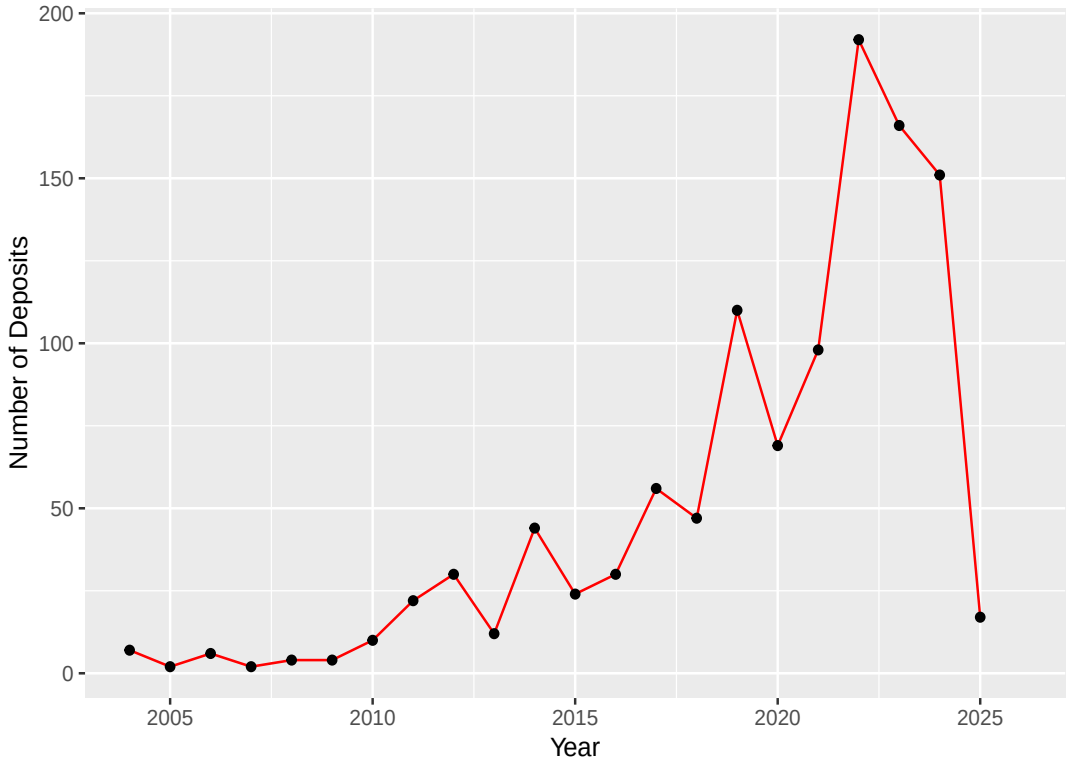


Figure 6: Number of CRISPR Deposits by Year

The data_type argument can be one of the following: ENTRY, POLYMER_ENTITY, BRANCHED_ENTITY, NONPOLYMER_ENTITY, POLYMER_ENTITY_INSTANCE, BRANCHED_ENTITY_INSTANCE, NONPOLYMER_ENTITY_INSTANCE, ASSEMBLY, and CHEMICAL_COMPONENT. The properties for each of these data types are available in JSON format on the RCSB website (https://data.rcsb.org/#data-schema). Additionally, we have provided Supplementary files that list the properties for each data type. These files help users identify the appropriate keywords for their queries.

In the following example, we retrieve information on polymer entities, such as taxonomy and cluster membership, for specific IDs.

```
properties <- list(rcsb_entity_source_organism =  c("ncbi_taxonomy_id",
                                                     "ncbi_scientific_name"),
                   rcsb_cluster_membership =  c("cluster_id", "identity"))

ids =  c("4HHB_1", "12CA_1", "3PQR_1")
```

```
df = data_fetcher(
  id = ids,
  data_type = "POLYMER_ENTITY",
  properties = properties,
  return_as_dataframe = TRUE
)

df

#> # A tibble: 3 x 5
#>   ID     ncbi_taxonomy_id ncbi_scientific_name cluster_id identity
#>   <chr>  <chr>            <chr>                <chr>      <chr>
#> 1 4HHB_1 9606             Homo sapiens         120        100
#> 2 12CA_1 9606             Homo sapiens         5          100
#> 3 3PQR_1 9913             Bos taurus           1253       100
```

For more examples and details on additional functionalities, please consult the package documentation.

`describe_chemical(...)` function

Beyond macromolecular structures, the PDB also houses extensive data on chemical compounds that play crucial roles in biological processes and interactions. To facilitate access to this valuable information, the `describe_chemical` function allows researchers to retrieve comprehensive descriptions of chemical compounds based on their unique RCSB PDB identifiers.

The `describe_chemical` function is designed to query the RCSB PDB for detailed descriptions of chemical compounds, utilizing their 3-character chemical ID as the query parameter. This function provides a detailed examination of chemical compounds, including their chemical composition, molecular properties such as formula weight and charge, and key identification markers like unique IDs and names. It furnishes standardized chemical descriptors, such as SMILES and InChI strings, facilitating computational analysis and database searches. Additionally, the function connects compounds to external databases (e.g., DrugBank, PubChem) for expanded research opportunities, and lists synonyms and alternative names to aid in cross-referencing across scientific literature. For compounds with biological significance, it may include information on molecular interactions and potential drug targets, alongside a historical record of the compound's database entries, highlighting the evolution of its scientific understanding. This comprehensive suite of data supports a wide range of scientific inquiries, from basic chemical analysis to complex pharmacological research.

A typical application of the `describe_chemical` function involves querying the database for information on adenosine triphosphate, a central molecule in cellular energy transfer, using its chemical ID 'ATP':

```
chem_desc <- describe_chemical('ATP')
chem_desc$rcsb_chem_comp_descriptor$smiles

#> [1] "c1nc(c2c(n1)n(cn2)C3C(C(C(O3)COP(=O)(O)OP(=O)(O)OP(=O)(O)O)O)O)N"
```

This example demonstrates how to retrieve the SMILES notation for ATP from the RCSB PDB using the describe_chemical function. ATP is fundamental to biochemistry and molecular biology, making its detailed chemical description highly valuable for a broad spectrum of research applications.

2.4 Data Retrieval

Accessing the RCSB PDB data is crucial for researchers aiming to model, visualize, and understand the molecular mechanisms underlying biological processes and diseases. The **rPDBapi** package offers functions such as `get_pdb_file`, `get_info`, and `get_fasta_from_rcsb_entry` to facilitate efficient data retrieval. However, it is important to acknowledge that these functions largely act as wrappers around the robust RCSB PDB Data API, which handles the majority of the search and retrieval tasks. Moreover, `get_pdb_file` relies on the **bio3d** package's `read.pdb` function to process PDB files. While these functions simplify interaction with the API and external tools by providing options for various file formats, optional compression, and FASTA sequence extraction, their primary contribution is streamlining access to existing resources rather than creating entirely new functionality.

`get_pdb_file(...)` function

Access to structural data for proteins, nucleic acids, and complex assemblies is vital for researchers aiming to model, visualize, and understand the molecular mechanisms underlying biological processes and diseases. Recognizing the importance of efficient data retrieval, the `get_pdb_file` function has been developed to facilitate the downloading of PDB files from the RCSB PDB database. This function supports various file formats and offers options for optional compression, addressing the need for streamlined access to structural data.

The `get_pdb_file` function is designed to enhance user access to the RCSB PDB by allowing the downloading of files in different formats, including the traditional PDB format, the newer CIF format, XML for those who prefer it, and structure factor files for entries where these are available. By default, the function opts for CIF files due to their compactness and comprehensive nature, and it advises the use of compression to expedite the download process.

To demonstrate the utility of `get_pdb_file`, consider the retrieval of the CIF file for hemoglobin (PDB ID: 4HHB):

```
pdb_file <- get_pdb_file(pdb_id = "4HHB", filetype = "cif")
```

Upon execution, `get_pdb_file` returns a list object of class *pdb* encompassing several components:

atom: A dataframe containing detailed atomic coordinate data for both ATOM and HETATM records.

```
head(pdb_file$atom)
```

```
#>   type eleno elety  alt resid chain resno insert      x      y      z o     b
#> 1 ATOM     1     N <NA>   VAL     A     1   <NA> 19.323 29.727 42.781 1 49.05
#> 2 ATOM     2    CA <NA>   VAL     A     1   <NA> 20.141 30.469 42.414 1 43.14
#> 3 ATOM     3     C <NA>   VAL     A     1   <NA> 21.664 29.857 42.548 1 24.80
#> 4 ATOM     4     O <NA>   VAL     A     1   <NA> 21.985 29.541 43.704 1 37.68
#> 5 ATOM     5    CB <NA>   VAL     A     1   <NA> 19.887 31.918 43.524 1 72.12
#> 6 ATOM     6   CG1 <NA>   VAL     A     1   <NA> 20.656 32.850 42.999 1 61.79
#>   segid elesy charge
#> 1     1     N   <NA>
#> 2     1     C   <NA>
#> 3     1     C   <NA>
#> 4     1     O   <NA>
#> 5     1     C   <NA>
#> 6     1     C   <NA>
```

xyz: A numeric matrix representing the spatial coordinates of the atomic and heteroatomic data.

```
head(pdb_file$xyz[1,])

#> [1] 19.323 29.727 42.781 20.141 30.469 42.414
```

calpha: A logical vector indicating the presence of C-alpha elements within the dataset.

```
head(pdb_file$calpha)

#> [1] FALSE  TRUE FALSE FALSE FALSE FALSE
```

Building on this foundation, secondary structure determination is a critical step in understanding protein function and interaction. Using the PDB file retrieved, we can visualize the secondary structure elements—such as alpha helices, beta sheets, and coils—using the **r3dmol** package.

The following example demonstrates how to load and visualize the secondary structure of a protein from a PDB file, with distinct coloring applied to different structural elements (Figure 7):

```
# Load necessary libraries
library(rPDBapi)
library(r3dmol)

# Retrieve and save a PDB structure
pdb_path <- get_pdb_file("1XYZ", filetype = "pdb", save = TRUE)
pdb_file <- pdb_path$path  # Use the path returned by get_pdb_file

# Load the structure into r3dmol and set visualization styles
viewer <- r3dmol() %>%
  m_add_model(pdb_file, format = "pdb") %>%
  m_set_style(
    sel = m_sel(ss = "helix"),  # Select alpha helices
    style = m_style_cartoon(color = "red")  # Color helices red
  ) %>%
  m_set_style(
    sel = m_sel(ss = "sheet"),  # Select beta sheets
    style = m_style_cartoon(color = "yellow")  # Color sheets yellow
  ) %>%
  m_set_style(
    sel = m_sel(ss = "coil"),  # Select coils/turns
    style = m_style_cartoon(color = "blue")  # Color coils blue
  ) %>%
  m_zoom_to()

# Display the viewer with secondary structure
viewer
```

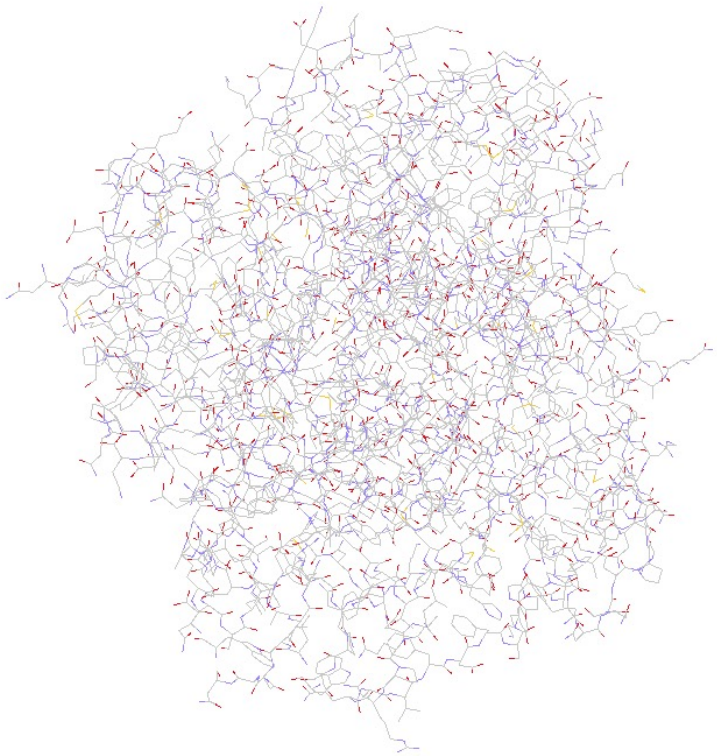


Figure 7: Secondary structure visualization of protein 1XYZ with color-coded elements.

This code provides a clear visualization of the secondary structure of the protein encoded by PDB ID 1XYZ. By applying distinct colors to alpha helices, beta sheets, and coils, the viewer can quickly identify and analyze the structural elements within the protein. The **r3dmol** package facilitates this visualization with a simple yet powerful API, making it an essential tool for structural biology research.

For further examples, please refer to the package documentation.

`get_info(...)` function

The ability to efficiently retrieve information about specific PDB entries is crucial for researchers conducting structural analyses, comparative studies, and drug design. To facilitate this, the `get_info` function has been developed to query the RCSB PDB using its REST API, enabling the retrieval of comprehensive information for any given PDB ID. This function exemplifies the integration of web-based services into bioinformatics workflows, offering streamlined access to a wealth of structural data.

The `get_info` function serves as a gateway to the vast data repository of the PDB, allowing users to look up detailed information about a specific PDB entry by utilizing the REST API. The function is designed to handle JSON data and HTTP requests efficiently, ensuring that users can easily convert and access information associated with both current and legacy PDB entry identifiers.

A typical use case involves querying information for hemoglobin (PDB ID: "4HHB"):

```
hemoglobin_info <- get_info(pdb_id = "4HHB")
```

The query for hemoglobin (PDB ID: "4HHB") through the `get_info` function yields extensive data, reflecting the structure's rich historical and scientific significance.

Key contributors such as Fermi, G., and Perutz, M.F., are acknowledged, highlighting the collaborative efforts behind the structure's elucidation.

```
hemoglobin_info$audit_author

#>           name pdbx_ordinal
#> 1    Fermi, G.            1
#> 2 Perutz, M.F.            2
```

Detailed crystallographic data, including cell dimensions and symmetry space group ("P 1 21 1"), are provided, essential for understanding the structure's geometric context.

```
hemoglobin_info$cell

#> $angle_alpha
#> [1] 90
#>
#> $angle_beta
#> [1] 99.34
#>
#> $angle_gamma
#> [1] 90
#>
#> $length_a
#> [1] 63.15
#>
#> $length_b
#> [1] 83.59
#>
#> $length_c
#> [1] 53.8
#>
#> $zpdb
#> [1] 4

hemoglobin_info$symmetry

#> $int_tables_number
#> [1] 4
#>
#> $space_group_name_hm
#> [1] "P 1 21 1"
```

In addition to retrieving detailed information about specific PDB entries, researchers can also perform more complex analyses, such as multiple sequence alignment, leveraging the get_info function to further investigate structural and experimental characteristics. Below is a step-by-step guide demonstrating how to perform such an analysis:

```
# Load necessary libraries
library(rPDBapi)
library(Biostrings)
library(msa)
library(ggplot2)

# Step 1: Perform a search for kinase-related PDB entries
search_operator <- list(
  type = "terminal",
  service = "full_text",
  parameters = list(value = "kinase")
```

```
)

# Perform the search and limit the number of entries
protein_entities <- perform_search(
  search_operator = search_operator,
  return_type = "ENTRY"
)
protein_entities <- protein_entities[1:10]  # Select the first 10 entries

# Step 2: Fetch sequence data and perform multiple sequence alignment
get_protein_sequence <- function(pdb_id) {
  fasta <- get_fasta_from_rcsb_entry(pdb_id)
  AAString(fasta[[1]])
}

# Retrieve and align protein sequences
sequences <- lapply(protein_entities, get_protein_sequence)
names(sequences) <- protein_entities
alignment <- msa(AAStringSet(sequences), method = "ClustalW")

#> use default substitution matrix

alignment_matrix <- as.matrix(alignment)

# Step 3: Calculate pairwise sequence similarity
pairwise_similarity <- function(seq1, seq2) {
  sum(seq1 == seq2) / length(seq1) * 100
}

n <- length(sequences)
similarity_matrix <- matrix(NA, n, n)
rownames(similarity_matrix) <- names(sequences)
colnames(similarity_matrix) <- names(sequences)
for (i in 1:n) {
  for (j in i:n) {
    similarity_matrix[i, j] <- pairwise_similarity(alignment_matrix[i, ],
                                                   alignment_matrix[j, ])
    similarity_matrix[j, i] <- similarity_matrix[i, j]
  }
}

# Step 4: Fetch and analyze structural and experimental information using get_info
get_structural_info <- function(pdb_id) {
  info <- get_info(pdb_id)

  # Extract relevant structural and experimental data
  resolution <- info$rcsb_entry_info$diffrn_resolution_high$value
  r_free <- info$refine$ls_rfactor_rfree
  r_work <- info$refine$ls_rfactor_rwork
  molecular_weight <- info$rcsb_entry_info$molecular_weight
  citation_year <- info$rcsb_primary_citation$year

  return(list(
    pdb_id = pdb_id,
    resolution = resolution,
    r_free = r_free,
```

```
    r_work = r_work,
    molecular_weight = molecular_weight,
    citation_year = citation_year
  ))
}

structural_info <- lapply(protein_entities, get_structural_info)

# Step 5: Compile the structural information into a data frame
structural_data <- data.frame(
  PDB_ID = sapply(structural_info, function(x)
    if (!is.null(x$pdb_id))
      x$pdb_id
    else
      NA_character_),
  Resolution = sapply(structural_info, function(x)
    if (!is.null(x$resolution))
      x$resolution
    else
      NA_real_),
  R_free = sapply(structural_info, function(x)
    if (!is.null(x$r_free))
      x$r_free
    else
      NA_real_),
  R_work = sapply(structural_info, function(x)
    if (!is.null(x$r_work))
      x$r_work
    else
      NA_real_),
  Molecular_Weight = sapply(structural_info, function(x)
    if (!is.null(x$molecular_weight))
      x$molecular_weight
    else
      NA_real_),
  Citation_Year = sapply(structural_info, function(x)
    if (!is.null(x$citation_year))
      x$citation_year
    else
      NA_integer_)
)

structural_data

#>    PDB_ID Resolution  R_free  R_work Molecular_Weight Citation_Year
#> 1    6GWV       2.80 0.25930 0.23330           519.45          2018
#> 2    2VA1       2.50 0.28500 0.23200           170.10          2007
#> 3    2KDC         NA      NA      NA            39.37          2009
#> 4    3JU6       2.45 0.25400 0.21400           171.77          2010
#> 5    1Z0Z       2.85 0.27500 0.20800           126.81          2005
#> 6    1SUW       2.45 0.25220 0.21100           114.60          2005
#> 7    4IDT       2.40 0.22760 0.19110            79.01          2013
#> 8    3NWY       2.47 0.26859 0.20474           181.30          2011
#> 9    7KAC       1.85 0.23180 0.21670            77.02          2021
#> 10   8FKO       2.10 0.25800 0.21990           224.68          2023
```

This example demonstrates how to integrate the get_info function into a more extensive

workflow that includes searching for specific proteins, aligning their sequences, calculating pairwise sequence similarity, and retrieving associated structural and experimental information. The combination of sequence analysis and detailed structural data retrieval enables comprehensive studies on protein function, evolutionary relationships, and potential applications in drug design.

`get_fasta_from_rcsb_entry(...)` function

A crucial component of RCSB PDB is the FASTA sequence, which represents the primary structure of proteins or nucleic acids in a concise and standardized format. The `get_fasta_from_rcsb_entry` function is designed to streamline the process of fetching FASTA sequences from the PDB for specified entry IDs. This function enhances the accessibility of sequence data, supporting a wide array of bioinformatics analyses and research applications. The function operates by issuing an HTTP request to the PDB's RESTful API, specifically targeting the endpoint associated with FASTA sequence data.

An example use case involves retrieving the FASTA sequence for hemoglobin (PDB ID: "4HHB"), illustrating the function's applicability:

```
fasta_sequence <- get_fasta_from_rcsb_entry(rcsb_id = "4HHB",
                                            chain_id = "A",
                                            verbosity = FALSE)
```

Upon successful execution, the function returns a structured list containing the FASTA sequence data for the specified PDB entry. For "4HHB", the output encapsulates critical details, including entity ID, chains, sequence and FASTA header:

```
# Split the sequence into chunks of 60 characters
split_sequence <- strsplit(fasta_sequence, '(?<=.{60})', perl = TRUE)[[1]]

# Print the split sequence with line breaks
cat(split_sequence, sep = "\n")

#> VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHGK
#> KVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPA
#> VHASLDKFLASVSTVLTSKYR
```

In this scenario, the function queries the RCSB PDB for the FASTA file associated with "4HHB", returning a list of FASTA sequences that represent the various polypeptide chains within the hemoglobin structure.

To further illustrate the utility of the `get_fasta_from_rcsb_entry` function, we conducted a sequence analysis and motif search using hemoglobin sequences retrieved from PDB entries 4HHB and 1A6M.

```
library(rPDBapi)
library(Biostrings)

# Retrieve the FASTA sequence for the first PDB entry (4HHB)
fasta_sequence_1 <- get_fasta_from_rcsb_entry("4HHB", chain_id = "B")
protein_sequence_1 <- fasta_sequence_1

# Convert the first sequence to an AAStringSet object for protein sequences
aa_sequence_1 <- AAStringSet(protein_sequence_1)

# Retrieve the FASTA sequence for a second PDB entry (1A6M)
fasta_sequence_2 <- get_fasta_from_rcsb_entry("1A6M", chain_id = "A")
protein_sequence_2 <- fasta_sequence_2
```

```
# Convert the second sequence to an AAString object for alignment
aa_sequence_2 <- AAString(protein_sequence_2)
```

Next, we performed a pairwise alignment between the two protein sequences using the `pairwiseAlignment` function from the **Biostrings** package. The sequences were aligned using the BLOSUM62 substitution matrix, which is commonly used for protein sequence alignment due to its balanced scoring of amino acid substitutions. This alignment allowed us to examine the similarities and differences between the sequences at the amino acid level:

```
# Perform pairwise alignment between the two protein sequences
alignment <- pairwiseAlignment(
  aa_sequence_1,
  aa_sequence_2,
  substitutionMatrix = "BLOSUM62",
  gapOpening = 10,
  gapExtension = 0.5
)
alignment

#> Global PairwiseAlignmentsSingleSubject (1 of 1)
#> pattern: VHLTPEEKSAVTALWGKVNVDEVGG--EALGRLL...HFGKEFTPPVQAAYQKVVAGVANALAHKY----H
#> subject: V-LSEGEWQLVLHVWAKVEADVAGHGQDILIRLF...RHPGDFGADAQGAMNKALELFRKDIAAKYKELGY
#> score: 86.5
```

Finally, we conducted a motif search on the 4HHB sequence to identify specific amino acid patterns:

```
# Search for motifs (e.g., a specific amino acid pattern) in the first sequence
motif <- "VHLTPEEKS"
match_positions <- vmatchPattern(motif, aa_sequence_1)
match_positions

#> MIndex object of length 1
#> [[1]]
#> IRanges object with 1 range and 0 metadata columns:
#>           start       end     width
#>       <integer> <integer> <integer>
#>   [1]         1         9         9
```

The motif search successfully identified the presence and positions of the specified pattern within the 4HHB sequence.

We further quantified the alignment by calculating the alignment score and the percentage identity between the two sequences:

```
# Extract the alignment score
alignment_score <- score(alignment)
cat("Alignment Score:", alignment_score, "\n")

#> Alignment Score: 86.5

# Calculate the percentage identity
identity <- pid(alignment)
cat("Percentage Identity:", identity, "%\n")

#> Percentage Identity: 24.34211 %
```

Furthermore, phylogenetic analysis is a powerful tool in bioinformatics, enabling researchers to explore evolutionary relationships between proteins, nucleic acids, or organisms based on sequence data. The R package ecosystem, including **ape**, **phangorn**, and **ggtree**, provides comprehensive tools for constructing and visualizing phylogenetic trees.

In this example, we demonstrate how to construct a phylogenetic tree for a set of hemoglobin-related protein sequences retrieved from the RCSB PDB using the query_search and get_fasta_from_rcsb_entry functions. The process involves multiple sequence alignment, distance matrix calculation, tree construction using the Neighbor-Joining method, and tree visualization with enhanced styling.

```
# Load necessary libraries
library(rPDBapi)
library(Biostrings)
library(ape)
library(ggtree)
library(msa)
library(phangorn)

# Step 1: Use query_search to retrieve PDB IDs related to a specific keyword
pdb_ids <- query_search(search_term = "hemoglobin")

# Filter or limit to the first 50 PDB IDs for demonstration purposes
pdb_ids <- pdb_ids[1:50]

# Function to retrieve sequences
get_protein_sequence <- function(pdb_id) {
  fasta <- get_fasta_from_rcsb_entry(pdb_id, chain_id = "A")
  return(AAString(fasta))
}

# Retrieve sequences for all PDB IDs
sequences <- lapply(pdb_ids, get_protein_sequence)

# Combine sequences into a single AAStringSet object
combined_sequences <- AAStringSet(sequences)
names(combined_sequences) <- pdb_ids

# Perform multiple sequence alignment
alignment <- msa(combined_sequences, method = "ClustalW")

#> use default substitution matrix

alignment

#> CLUSTAL 2.1
#>
#> Call:
#>    msa(combined_sequences, method = "ClustalW")
#>
#> MsaAAMultipleAlignment with 50 rows and 145 columns
#>      aln                                                  names
#>  [1] -VLSPADKTNIKSTWDKIGGHAGDY...FTPAVHASLDKFFAAVSTVLTSKYR 3PEL
#>  [2] -VLSPADKTNIKSTWDKIGGHAGDY...FTPAVHASLDKFFAAVSTVLTSKYR 3GOU
#>  [3] -VLSPADKTNIKSTWDKIGGHAGDY...FTPAVHASLDKFFAAVSTVLTSKYR 2QLS
#>  [4] -VLSPADKTNIKSTWDKIGGHAGDY...FTPAVHASLDKFFTAVSTVLTSKYR 1FHJ
#>  [5] -VLSAADKSNVKACWGKIGSHAGEY...FTPAVHASLDKFFSAVSTVLTSKYR 3D4X
```

```
#>  [6] -VLSAADKSNVKACWGKIGSHAGEY...FTPAVHASLDKFFSAVSTVLTSKYR 3GQP
#>  [7] -VLSPADKTNIKASWEKIGSHGGEY...FTPAVHASLDKFLSSVSTVLTSKYR 4YU3
#>  [8] -VLSPADKTNIKASWEKIGSHGGEY...FTPAVHASLDKFLSSVSTVLTSKYR 4YU4
#>  [9] -VLSPADKTNIKTAWEKIGSHGGEY...FTPAVHASLDKFLANVSTVLTSKYR 2RAO
#>  ... ...
#> [43] -VLSGTDKTNVKSIFSKIGGQADDY...LTPEAHASLDKFLCAVGLVLTAKYR 2ZFB
#> [44] -VLSASDKTNVKGVFAKVGGSAEAY...LTPEVHASLDKFMCAVAKELTAKYR 3WR1
#> [45] -MLTEDDKQLIQHVWEKVLEHQEDF...YTPQVQVAYDKFLAAVSAVLAEKYR 1V75
#> [46] -MLTEDDKQLIQHVWEKVLEHQEDF...YTPQVQVAYDKFLAAVSAVLAEKYR 1WMU
#> [47] -MLTAEDKKLIQQAWEKAASHQEEF...YTPEVHAAFDKFLSAVSAVLAEKYR 1HBR
#> [48] XSLSSKDKDTVKALWGKIADKAEEI...FTPEVHISYDKFFSALARALAEKYR 1XQ5
#> [49] XSLSATDKARVKALWDKIEGKSAEL...FTPEVHLSVDKFLACLALALSEKYR 1SPG
#> [50] ---------MPKSFYDAVGG--AKT...LDDEHRRELLDYLEMAAHSLVNSPF 1NGK
#>  Con -VLSAADK?NVKAAW?KVGGHAGEY...FTPAVHASLDKFL??VSTVLTSKYR Consensus
```

The multiple sequence alignment step aligns the sequences to identify conserved regions, which is crucial for constructing a meaningful phylogenetic tree. The aligned sequences are then converted into a format suitable for phylogenetic analysis:

```
# Convert the aligned sequences to a matrix
alignment_matrix <- as.matrix(alignment)

# Convert the alignment matrix to a phyDat object
alignment_phyDat <- phyDat(alignment_matrix, type = "AA")

# Construct a distance matrix using a protein distance measure
dist_matrix <- dist.ml(alignment_phyDat)

# Construct the tree using Neighbor-Joining method
tree <- NJ(dist_matrix)

tree

#>
#> Phylogenetic tree with 50 tips and 48 internal nodes.
#>
#> Tip labels:
#>   3PEL, 3GOU, 2QLS, 1FHJ, 3D4X, 3GQP, ...
#>
#> Unrooted; includes branch length(s).
```

The Neighbor-Joining method (Saitou and Nei, 1987) was used to construct the phylogenetic tree, which is commonly employed due to its efficiency and ability to produce unrooted trees that represent the evolutionary distances between sequences. The resulting tree can then be visualized using the **ggtree** package (Figure 8), which offers extensive customization options:

```
# Visualize the phylogenetic tree using ggtree with enhanced styling
ggtree(tree) +
  geom_tiplab(aes(label = label), size = 2, fontface = "italic", align = TRUE) +
  geom_nodepoint(color = "blue", size = 3, shape = 21, fill = "lightblue") +
  geom_treescale(x = 0, y = -2, width = 0.05) +
  theme_tree2() +
  labs(title = "Phylogenetic Tree of Selected PDB Entries",
       subtitle = "Based on Multiple Sequence Alignment") +
  theme(plot.title = element_text(hjust = 0.5, size = 10, face = "bold"),
```

```
plot.subtitle = element_text(hjust = 0.5, size = 8),
plot.caption = element_text(size = 8, face = "italic"))
```

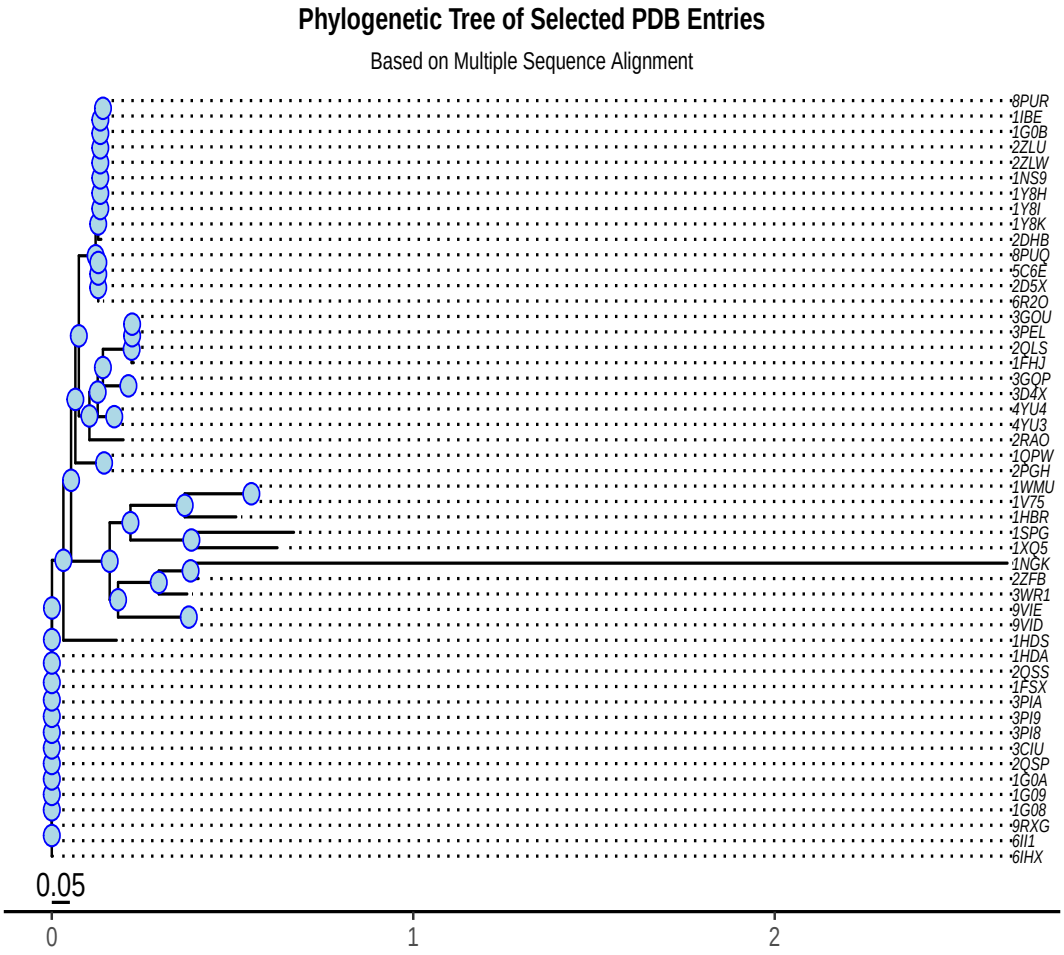


Figure 8: Phylogenetic tree of hemoglobin-related proteins with styled nodes for clarity.

The phylogenetic tree allows for the clear presentation of evolutionary relationships between the selected hemoglobin-related proteins. It highlights how sequences cluster based on similarity, providing insights into the evolutionary history of these proteins. The enhanced styling, including colored nodes and labels, improves the readability and interpretability of the phylogenetic tree, making it a valuable tool for both research and presentation.

2.5 Retrieving Specific Fields and Papers

To enhance the utility of this PDB database for targeted data exploration, we introduce two R functions: find_results and find_papers. These functions allow for precise retrieval of specific data fields from search results, customized to the unique informational needs of researchers.

The `find_results` function executes a search in the PDB using a provided term and retrieves information for a designated field (e.g., citation) from each search result. It leverages the `query_search` and `get_info` functions for searching the PDB and fetching the requested details. This approach enables users to specify the search term and the field of interest, with a default focus on citation information. The versatility of `find_results` is further enhanced by its support for a wide array of fields, including but not limited to audit_author, exptl_crystal_grow, and struct_keywords, allowing for comprehensive data mining customized to the researcher's needs.

Retrieving citation information for structures related to *CRISPR*:

```
crispr_citations <- find_results("CRISPR", field = "citation")
crispr_citations[[1]]

#>   country      id       journal_abbrev journal_id_issn journal_volume
#> 1      US primary Nat.Struct.Mol.Biol.       1545-9993             18
#>   page_first pdbx_database_id_doi pdbx_database_id_pub_med rcsb_authors
#> 1        680    10.1038/NSMB.2043                 21572442 Sashital....
#>   rcsb_is_primary rcsb_journal_abbrev
#> 1               Y Nat Struct Mol Biol
#>                                                                      title
#> 1 An RNA-Induced Conformational Change Required for Crispr RNA Cleavage by the Endoribonuclease Cse
#>   year
#> 1 2011
```

This example underscores the function's efficacy in aggregating citation information, facilitating researchers in keeping abreast of recent structural investigations related to CRISPR, a revolutionary genome editing technology.

The `find_papers` function is specifically designed for bibliographic data retrieval, fetching paper titles from the PDB related to a given search term. This facilitates an efficient overview of scholarly articles pertaining to a specific subject area.

To compile titles of papers related to *CRISPR*:

```
covid_paper_titles <- find_papers("CRISPR", max_results = 5)

# Truncate paper titles to 50 characters
covid_paper_titles <- lapply(covid_paper_titles, function(title) {
  if(nchar(title) > 50) {
    paste0(substr(title, 1, 47), "...")
  } else {
    title
  }
})

# Display the truncated titles
covid_paper_titles

#> $`2Y9H`
#> [1] "An RNA-Induced Conformational Change Required f..."
#>
#> $`4ILM`
#> [1] "Recognition and cleavage of a nonstructured CRI..."
#>
#> $`4XTK`
#> [1] "Crystal structure and nuclease activity of tm17..."
#>
#> $`5FCL`
#> [1] "Structural plasticity and in vivo activity of C..."
#>
#> $`7EI1`
#> [1] "A distinct structure of Cas1-Cas2 complex provi..."
```

Focusing on *CRISPR*, this example facilitates access to pertinent literature, assisting researchers in staying abreast of ongoing studies and findings.

To further demonstrate the power of these functions, let's explore entries related to "ligase," a key enzyme involved in the process of joining two molecules. We will analyze the structural data and related literature to gain insights into the scientific focus surrounding ligase research.

First, we load the necessary R libraries required for performing searches. We will use **tm** for text processing, and **wordcloud** for visualization. Next, we define the search operator, specifying that we want to find PDB entries related to "ligase." The perform_search function will later use this operator to query the PDB database.

```
# Load necessary libraries
library(rPDBapi)
library(tm)
library(wordcloud)

# Define the search operator to find PDB entries related to "ligase"
search_operator <- list(
  type = "terminal",
  service = "full_text",
  parameters = list(value = "ligase")
)
```

We perform the search in the PDB using the previously defined operator. The return_type is set to "ENTRY" to ensure that we retrieve specific PDB entries related to ligase.

```
# Perform the search with the return type set to "ENTRY"
ligase_entries <- perform_search(
  search_operator = search_operator,
  return_type = "ENTRY"
)
```

Since the search might return a large number of results, we select a manageable subset (e.g., the first 100 entries) for detailed analysis.

```
# Select a subset of entries for analysis (e.g., the first 100)
pdb_ids <- ligase_entries[1:100]
```

For each selected PDB ID, we fetch both the structural information and related papers using the find_results and find_papers functions. This helps in linking the structural data with the corresponding scientific literature.

```
# Function to fetch results and related papers for a given PDB ID
fetch_pdb_info <- function(pdb_id) {
  results <- find_results(pdb_id)  # Fetch structural data
  papers <- find_papers(pdb_id)    # Fetch related papers

  return(list(results = results, papers = papers))
}

# Apply the function to all selected PDB IDs
pdb_info <- lapply(pdb_ids, fetch_pdb_info)
```

This section analyzes the fetched PDB information, focusing on the number of related papers and their titles. This helps in understanding the research focus and the amount of attention each structure has received in the scientific community.

```
# Function to analyze PDB info
```

```
analyze_pdb_info <- function(info) {
  # Extract the number of related papers
  num_papers <- length(info$papers[[1]])

  # Extract titles of related papers
  paper_titles <- paste(info$papers[[1]], collapse = "; ")

  return(list(num_papers = num_papers, paper_titles = paper_titles))
}

# Apply the analysis to each PDB ID
pdb_analysis <- lapply(pdb_info, analyze_pdb_info)
```

For easier analysis and visualization, we convert the list of results into a data frame. This data frame will contain the PDB IDs, the number of related papers, and the titles of those papers.

```
# Convert the list into a data frame for easier analysis
pdb_analysis_df <- data.frame(
  PDB_ID = pdb_ids,
  Number_of_Papers = sapply(pdb_analysis, function(x) x$num_papers),
  Paper_Titles = sapply(pdb_analysis, function(x) x$paper_titles)
)

# Truncate with ellipses if the title is longer than 50 characters
pdb_analysis_df_truncated <- pdb_analysis_df
pdb_analysis_df_truncated$Paper_Titles <-
  ifelse(
    nchar(pdb_analysis_df_truncated$Paper_Titles) > 50,
    paste0(substr(
      pdb_analysis_df_truncated$Paper_Titles, 1, 47
    ), "..."),
    pdb_analysis_df_truncated$Paper_Titles
  )

head(pdb_analysis_df_truncated)
```

```
#>   PDB_ID Number_of_Papers                                        Paper_Titles
#> 1   1B04               1 Structure of the adenylation domain of an NAD+-...
#> 2   4EEQ               1 Design, Synthesis and Activity Evaluation of Po...
#> 3   3JSL               1 Structure of the adenylation domain of NAD(+)-d...
#> 4   3RR5               1 ATP-dependent DNA ligase from Thermococcus sp. ...
#> 5   2HIV               1 A Flexible Interface between DNA Ligase and PCN...
#> 6   6JX5               1 Structural insights into a HECT-type E3 ligase ...
```

To identify common themes and keywords in the related papers, we perform a word frequency analysis. This involves creating a text corpus, cleaning the text, and generating a word cloud that visually represents the most frequent terms.

```
# Combine all paper titles into a single string
all_titles <- paste(pdb_analysis_df$Paper_Titles, collapse = " ")

# Create a corpus and clean the text
corpus <- Corpus(VectorSource(all_titles))
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removePunctuation)
```

```
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, removeWords, stopwords("en"))
```

Finally, we create a term-document matrix from the cleaned corpus and generate a word cloud (Figure 9). This visualization highlights the most common words in the paper titles, providing insights into the main research topics associated with the selected ligase-related structures.

```
# Create a term-document matrix
tdm <- TermDocumentMatrix(corpus)
m <- as.matrix(tdm)
word_freqs <- sort(rowSums(m), decreasing = TRUE)
word_freqs_df <- data.frame(word = names(word_freqs), freq = word_freqs)

# Generate a word cloud to visualize common themes
wordcloud(words = word_freqs_df$word, freq = word_freqs_df$freq, min.freq = 2,
        max.words = 100, random.order = FALSE, colors = brewer.pal(8, "Dark2"))
```

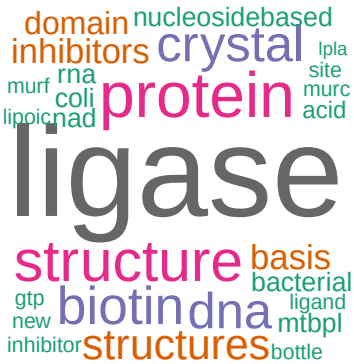


Figure 9: Word cloud showing frequent terms in paper titles related to ligase structures.

2.6 Constructing JSON Queries

Efficient access and retrieval of specific subsets of RCSB PDB data are essential for supporting a broad range of scientific inquiries. The `generate_json_query` function represents a sophisticated tool designed to facilitate this process, enabling researchers to construct customized JSON queries based on specific criteria such as identifiers, data types, and desired properties. This function exemplifies the integration of programmable queries into research workflows, enhancing the accessibility and utility of PDB data.

Consider a scenario where a researcher wishes to retrieve specific information about two PDB entries, identified by "1XYZ" and "2XYZ", focusing on certain cell and experimental properties:

```
ids <- c("1XYZ", "2XYZ")
properties <- list(cell = c("volume", "angle_beta"), exptl = c("method"))
json_query <- generate_json_query(ids, "ENTRY", properties)

# Insert a single break line after "entry_ids: [\"1XYZ\", \"2XYZ\"]"
json_query <- gsub("(\\[\"1XYZ\", \"2XYZ\"\\])", "\\1\n", json_query)

# Display the modified output with one break line
json_query

#> [1] "{entries(entry_ids: [\"1XYZ\", \"2XYZ\"]\n){cell {volume, angle_beta}, exptl {method}}}"
```

This example demonstrates the function's utility in generating a JSON query that targets specific PDB entries and requests detailed information about their cellular dimensions and experimental methods. The generated query can then be used to fetch the relevant data from the PDB, facilitating targeted analyses and research activities.

3 Discussion

The development and integration of the **rPDBapi** package into the bioinformatics and structural biology research toolkit represent a significant advancement in the accessibility and utility of the RCSB PDB for the R community. By providing a comprehensive interface, **rPDBapi** facilitates a streamlined and intuitive approach to data retrieval and analysis, addressing the critical need for direct access to PDB resources within R.

rPDBapi offers an array of powerful functions that simplify the process of accessing and analyzing PDB data. These functions, such as constructing customized JSON queries, retrieving FASTA sequences, and performing nuanced searches, empower researchers to efficiently customize their data retrieval strategies. This customization is crucial for accommodating the diverse needs of projects, whether they involve exploratory analyses in genomics or the identification of potential drug targets. The flexibility in data gathering provided by the package allows researchers to adapt quickly to unique project goals, improving the overall efficiency of research workflows and enabling more sophisticated analyses.

Several key R packages were incorporated into the functionality of **rPDBapi** to enhance data handling and visualization capabilities. Packages such as **dplyr** (Wickham et al., 2023a), **purrr** (Wickham and Henry, 2023), and **magrittr** (Bache and Wickham, 2022) streamline data management and manipulation, providing a user-friendly interface for working with large datasets retrieved from the PDB. Communication with the RCSB PDB API is handled through **httr** (Wickham, 2023), **jsonlite** (Ooms, 2014), and **xml2** (Wickham et al., 2023b), ensuring efficient data access and retrieval. Additionally, the **bio3d** (B.J. et al., 2006) package is utilized for downloading and storing biological molecule structures across various formats, enriching the package's utility in structural biology research.

To support downstream analysis, several key R packages were integrated throughout this manuscript. The **Biostrings** (Pages et al., 2024) and **msa** (Bodenhofer et al., 2015) packages were instrumental in performing sequence alignment and calculating pairwise sequence identity from the retrieved FASTA sequences. These packages enabled the in-depth sequence analysis of proteins retrieved via **rPDBapi**, allowing for comparison and evolutionary insight into the sequences. For text mining and thematic exploration of PDB-related literature, we utilized the **tm** (Feinerer and Hornik, 2024) and **wordcloud** (Fellows, 2018) packages. These were used to extract, clean, and visualize keywords from the titles and abstracts of papers associated with specific PDB entries, generating a word cloud that highlighted the most frequent terms in the dataset.

The **gplots** (Warnes et al., 2024) package played a crucial role in visualizing the pairwise sequence identity data through heatmaps, providing a clear visual representation of the similarity among the aligned protein sequences. Additionally, the **r3dmol** (Su and Johnston, 2021) package was utilized for 3D molecular visualization, which allowed interactive exploration of protein structures retrieved via **rPDBapi**. This integration provided visual insight into structural features, enhancing the analysis of protein conformation and interactions.

For phylogenetic analysis, we incorporated the **ape** (Paradis et al., 2024), **phangorn** (Schliep et al., 2023), and **ggtree** (Yu et al., 2017) packages. These were essential for constructing and visualizing phylogenetic trees, which provided insights into the evolutionary relationships among proteins based on their sequence alignment. This comprehensive analysis pipeline, starting with sequence retrieval from **rPDBapi** and concluding with evolutionary tree construction, demonstrates the package's integration with the broader R ecosystem for complex downstream analyses.

The streamlined data access offered by **rPDBapi**, combined with the aforementioned downstream analysis tools, has the potential to accelerate scientific discovery in bioinformat-

ics and structural biology. Immediate access to a vast repository of structural information enables rapid hypothesis testing, iterative analysis, and exploratory research, which can lead to novel insights into molecular mechanisms, disease pathology, and therapeutic targets.

In addition to its core features, **rPDBapi** promotes collaborative and interdisciplinary research efforts by democratizing access to PDB data within R. The integration of structural biology data into broader research initiatives across computational biology, genomics, and pharmacology is crucial for addressing complex biological questions. By making PDB data more accessible to R users, the package fosters a cross-disciplinary approach that enhances the scope and depth of bioinformatics research.

While **rPDBapi** is a significant advancement for the R community, it is essential to recognize similar tools available in other programming environments. Libraries such as PyPDB for Python (Gilpin, 2016), BioJava (Lafita et al., 2019), and BioPython (Cock et al., 2009) offer robust access to PDB data for their respective communities, enabling similar functionalities in querying and analyzing biological structures.

Looking ahead, the continued development of **rPDBapi** will be instrumental in addressing emerging challenges and opportunities in structural biology research. Future updates could include enhanced integration with other bioinformatics resources, support for novel structural determination methodologies, and tools for visualizing and interactively exploring molecular structures. As the PDB grows in complexity, innovative approaches to data analysis and management will be essential, and **rPDBapi** will remain an important tool for the evolving landscape of bioinformatics research.

4 Conclusion

rPDBapi provides an essential tool for the R community to access and interact with the RCSB Protein Data Bank (PDB), bridging a significant gap in bioinformatics and structural biology research. By offering a streamlined interface for querying, retrieving, and analyzing PDB data, the package simplifies the process of handling complex biological datasets. While rPDBapi largely serves as a wrapper around existing RCSB PDB APIs and other packages like bio3d, its key contribution lies in enhancing accessibility and usability for R users. The package's ability to integrate advanced query functions and support flexible data formats allows researchers to efficiently navigate PDB datasets and customize data retrieval to specific research needs. As a result, **rPDBapi** represents a valuable addition to the R ecosystem, promoting a more efficient workflow for bioinformatics and structural biology studies.

References

S. M. Bache and H. Wickham. *magrittr: A Forward-Pipe Operator for R (Version 2.0.3)*, 2022. URL `https://CRAN.R-project.org/package=magrittr`. [p142]

G. B.J., R. A.P.C., E. K.M., M. J.A., and C. L.S.D. Bio3D: An R package for the comparative analysis of protein structures. *Bioinformatics*, 22:2695–2696, Nov 2006. [p142]

U. Bodenhofer, E. Bonatesta, C. Horejš-Kainrath, and S. Hochreiter. msa: an R package for multiple sequence alignment. *Bioinformatics*, 31(24):3997–3999, 2015. [p142]

S. K. Burley, C. Bhikadiya, C. Bi, S. Bittrich, L. Chen, G. V. Crichlow, J. M. Duarte, S. Dutta, M. Fayazi, Z. Feng, et al. Rcsb protein data bank: Celebrating 50 years of the PDB with new tools for understanding and visualizing biological macromolecules in 3d. *Protein Science*, 31(1):187–208, 2022. [p109]

P. J. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski, et al. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422, 2009. [p143]

I. Feinerer and K. Hornik. *tm: Text Mining Package (Version 0.7-14)*, 2024. URL https://CRAN.R-project.org/package=tm. [p142]

I. Fellows. *wordcloud: Word Clouds (Version 2.6)*, 2018. URL https://CRAN.R-project.org/package=wordcloud. [p142]

W. Gilpin. Pypdb: a Python API for the protein data bank. *Bioinformatics*, 32(1):159–160, 2016. [p109, 143]

S. Korkmaz and B. E. Yamasan. *rPDBapi: A Comprehensive Interface for Accessing the Protein Data Bank (Version 2.1)*, 2024. URL https://CRAN.R-project.org/package=rPDBapi. [p111]

S. Korkmaz, J. M. Duarte, A. Prlić, D. Goksuluk, G. Zararsiz, O. Saracbasi, S. K. Burley, and P. W. Rose. Investigation of protein quaternary structure via stoichiometry and symmetry ınformation. *PloS one*, 13(6):e0197176, 2018. [p109]

A. Lafita, S. Bliven, A. Prlić, D. Guzenko, P. W. Rose, A. Bradley, P. Pavan, D. Myers-Turnbull, Y. Valasatava, M. Heuer, et al. Biojava 5: A community driven open-source bioinformatics library. *PLoS computational biology*, 15(2):e1006791, 2019. [p143]

J. Ooms. The jsonlite package: A practical and consistent mapping between JSON data and R objects. *arXiv:1403.2805 [stat.CO]*, 2014. URL https://arxiv.org/abs/1403.2805. [p142]

H. Pages, P. Aboyoun, R. Gentleman, and S. DebRoy. *Biostrings: Efficient Manipulation of Biological Strings (Version 2.72.1)*, 2024. URL https://bioconductor.org/packages/Biostrings. [p142]

E. Paradis, S. Blomberg, B. Bolker, J. Brown, S. Claramunt, J. Claude, H. S. Cuong, R. Desper, G. Didier, B. Durand, J. Dutheil, R. J. Ewing, O. Gascuel, T. Guillerme, C. Heibl, A. Ives, B. Jones, F. Krah, D. Lawson, V. Lefort, P. Legendre, J. Lemon, G. Louvel, E. Marcon, R. McCloskey, J. Nylander, R. Opgen-Rhein, A.-A. Popescu, M. Royer-Carenzi, K. Schliep, K. Strimmer, and D. de Vienne. *ape: Analyses of Phylogenetics and Evolution (Version 5.8)*, 2024. URL https://CRAN.R-project.org/package=ape. [p142]

N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425, 1987. [p136]

K. Schliep, E. Paradis, L. de Oliveira Martins, A. Potts, I. Bardel-Kahr, T. W. White, C. Stachniss, M. Kendall, K. Halabi, R. Bilderbeek, K. Winchell, L. Revell, M. Gilchrist, J. Beaulieu, B. O'Meara, L. Qu, J. Brown, and S. Claramunt. *phangorn: Phylogenetic Reconstruction and Analysis (Version 2.11.1)*, 2023. URL https://CRAN.R-project.org/package=phangorn. [p142]

W. Su and B. Johnston. *r3dmol: Create Interactive 3D Visualizations of Molecular Data (Version 0.1.2)*, 2021. URL https://CRAN.R-project.org/package=r3dmol. [p142]

G. R. Warnes, B. Bolker, L. Bonebakker, R. Gentleman, W. Huber, A. Liaw, T. Lumley, M. Maechler, A. Magnusson, S. Moeller, M. Schwartz, B. Venables, and T. Galili. *gplots: Various R Programming Tools for Plotting Data (Version 3.1.3.1)*, 2024. URL https://CRAN.R-project.org/package=gplots. [p142]

H. Wickham. *httr: Tools for Working with URLs and HTTP (Version 1.4.7)*, 2023. URL https://CRAN.R-project.org/package=httr. [p142]

H. Wickham and L. Henry. *purrr: Functional Programming Tools (Version 1.0.2)*, 2023. URL https://CRAN.R-project.org/package=purrr. [p142]

H. Wickham, R. François, L. Henry, K. Müller, and D. Vaughan. *dplyr: A Grammar of Data Manipulation (Version 1.1.4)*, 2023a. URL https://CRAN.R-project.org/package=dplyr. [p142]

H. Wickham, J. Hester, and J. Ooms. *xml2: Parse XML (Version 1.3.6)*, 2023b. URL https://CRAN.R-project.org/package=xml2. [p142]

G. Yu, D. K. Smith, H. Zhu, Y. Guan, and T. T.-Y. Lam. ggtree: an R package for visualization and annotation of phylogenetic trees with their covariates and other associated data. *Methods in Ecology and Evolution*, 8(1):28–36, 2017. [p142]

Selcuk Korkmaz
Trakya University
Department Biostatistics
Edirne, Türkiye
ORCiD: 0000-0003-4632-6850
selcukkorkmaz@trakya.edu.tr

Bilge Eren Yamasan
Trakya University
Department Biophysics
Edirne, Türkiye
ORCiD: 0000-0002-6525-2503
berenyamasan@trakya.edu.tr

rqPen: An R Package for Penalized Quantile Regression

by Ben Sherwood, Shaobo Li, and Adam Maidman

Abstract Quantile regression directly models a conditional quantile of interest. The R package rqPen provides penalized quantile regression estimation using the lasso, elastic net, adaptive lasso, SCAD, and MCP penalties. It also provides extensions to group penalties, both for groups of predictors and grouping variable selection across quantiles. This paper presents the different approaches to penalized quantile regression and how to use the provided methods in rqPen.

1 Introduction

The package **rqPen** allows users to model conditional quantiles using a penalized quantile regression approach. It supports a wide range of penalties and includes cross-validation and information criterion approaches for selecting tuning parameters. Koenker and Bassett (1978) proposed quantile regression as a robust alternative to mean regression that directly models a conditional quantile of interest without the need for assumptions about the distribution or moment conditions for the error term. Since Tibshirani (1996) introduced the lasso penalty, investigating the combination of different penalties and loss functions has been an active area of interest. Penalties provided in **rqPen** are lasso, elastic net (Zou and Hastie, 2005), SCAD (Fan and Li, 2001), MCP (Zhang, 2010), adaptive lasso (Zou, 2006), group lasso (Yuan and Lin, 2005), group extensions of adaptive lasso, SCAD, and MCP (Wang et al., 2007; Huang et al., 2012; Breheny and Huang, 2009), and a group lasso penalty where variables are grouped across quantiles (Wang et al., 2024). Extending the theoretical results of penalized estimators to the quantile regression setting has been an active area of research. Examples include deriving the rate of convergence for lasso (Belloni and Chernozhukov, 2011) and group lasso (Kato, 2012) and deriving oracle properties for non-convex penalties such as SCAD and MCP (Wang et al., 2012). Discussed in these papers is how minimizing the penalized objective functions for quantile regression can be framed as linear programming problems or, in the case of group lasso, second-order cone programming problems. The linear programming formulation is particularly familiar to researchers in quantile regression because this is the most common approach for solving quantile regression problems, including in the **quantreg** package (Koenker and D'Orey, 1987, 1994), while a second-order cone programming problem can be solved using convex optimization software, including the R package **Rmosek** (Koenker and Mizera, 2014).

The ability to analyze large data sets is one of the major appeals of penalized regression methods. However, linear programming and second-order cone programming algorithms become computationally burdensome for large data sets. Further complicating matters is that the quantile loss function is non-differentiable, while popular algorithms for penalized objective functions rely on a differentiable loss function, for instance, Friedman et al. (2010) (elastic net), Breheny and Huang (2011) (non-convex penalties), Breheny and Huang (2015) (group non-convex penalties), and Yang and Zou (2015) (group lasso). Yi and Huang (2017) proposed using a Huber-type approximation of the quantile loss and coordinate descent algorithm for solving elastic net penalized quantile regression which is implemented in the R package **hqreg**. Assuming a good initial estimator is provided, Peng and Wang (2015) proposed a coordinate descent algorithm (QICD) for non-convex penalties. The R package **conquer** approximates the quantile loss with a convolution of the quantile loss and Kernel function (He et al., 2023; Tan et al., 2022).

The package **rqPen** provides an implementation of the Huber-type approximation and linear programming algorithms. It allows users to fit quantile regression models with all of the penalty functions discussed in the first paragraph and provides tools for using cross-validation or information criterion to select tuning parameters. In addition, it provides plots

for comparing cross-validation results and coefficient estimates as the sparsity parameter, λ, changes. The package allows for estimating multiple quantiles with a call to a single function and provides plots of how coefficient values change with the different quantiles being modeled.

The packages **quantreg**, **conquer**, **hrqglas**, and **hqreg** are alternatives for penalized quantile regression in R. However, there are some substantial differences between **rqPen** and these packages. The package **rqPen** allows users to fit quantile regression models using the elastic net, SCAD, MCP, adaptive lasso, group lasso, group SCAD, group MCP and group adaptive lasso penalties, along with allowing users to choose between an L_1 or L_2 norm for all group penalties. In addition, users can fit models for multiple quantiles, use cross-validation or information criterion (IC) approaches to choose tuning parameters, and create plots of the change in coefficient estimates for different quantiles and tuning parameters. Also, commonly used functions such as `predict()`, `coef()`, and `plot()` can be used with the '`rq.pen.seq`' and '`rq.pen.seq.cv`' objects created by **rqPen** functions. The package **quantreg** fits unpenalized quantile regression models and provides the SCAD and lasso penalty. For both penalized and unpenalized quantile regression, **conquer** offers computationally efficient estimation for large n or p data sets. Users of **conquer** can choose between lasso, elastic net, group lasso, sparse-group (Simon et al., 2013), group SCAD and group MCP. To the best of our knowledge, **conquer** is the only package that offers the sparse-group lasso penalty for quantile regression. While **hrqglas** provides penalized group lasso, it does not provide any of the other group penalties, nor does it allow for simultaneous estimation of multiple quantiles. Similarly, **hqreg** provides the elastic net penalty using the Huber approximation, but not adaptive lasso, SCAD or MCP. However, **hrqglas** and **hqreg** provide methods for robust mean regression using the Huber loss function, something **rqPen** does not do. Both packages are required for **rqPen** and provide the backbone for the algorithms that use a Huber approximation for the group, **hrqglas**, and non-group, **hqreg**, penalties. The package **rqPen** provides a variety of algorithms and penalty functions that users can choose from. In addition, it provides tools not available in other penalized quantile regression packages, such as plots of how coefficients change with the quantile being modeled or the sparsity parameter, allowing for multiple estimates of quantiles in a single line of code, and functions for information criterion based approaches to tuning parameter selection. Finally, it provides a penalty that guarantees consistent variable selection across quantiles, an option not available in the other packages mentioned.

2 Penalized estimation of quantile regression

Consider observations $\{y_i, \mathbf{x}_i\}_{i=1}^n$, where $\mathbf{x}_i = (1, x_{i1}, \ldots, x_{ip})^\top \in \mathbb{R}^{p+1}$, and the model

$$y_i = \mathbf{x}_i^\top \boldsymbol{\beta}_0^\tau + \epsilon_i, \tag{1}$$

where $P(\epsilon_i < 0 | \mathbf{x}_i) = \tau$. Define $\rho_\tau(u) = u[\tau - I(u < 0)]$, Koenker and Bassett (1978) proposed estimating (1) by minimizing

$$\sum_{i=1}^n \rho_\tau(y_i - \mathbf{x}_i^\top \boldsymbol{\beta}), \tag{2}$$

which is available in the package **quantreg**. Let m_i be the weight for observation i, $\boldsymbol{w}$ be a vector of weights for a predictor or group of predictors, and λ and a are penalty tuning parameters. The package **rqPen** provides functions for estimating $\boldsymbol{\beta}_0^\tau$ by minimizing

$$\frac{1}{n}\sum_{i=1}^n m_i \rho_\tau(y_i - \mathbf{x}_i^\top \boldsymbol{\beta}) + P_{\boldsymbol{w},\lambda,a}(\boldsymbol{\beta}), \tag{3}$$

where $\lambda > 0$ and the plausible values of a depend on the penalty being used. The penalty function $P_{\lambda,a}(\boldsymbol{\beta})$ can take the form of a group or individual penalty.

2.1 Individual Penalty

Let w_j be a weight for a variable j. For an individual penalty, (3) has the form of

$$\frac{1}{n}\sum_{i=1}^{n} m_i \rho_\tau(y_i - \mathbf{x}_i^\top \boldsymbol{\beta}) + \sum_{j=1}^{p} p_{w_j\lambda,a}(|\beta_j|). \tag{4}$$

The package **rqPen** supports four different forms of individual penalties: elastic net, adaptive lasso, SCAD and MCP. Users can also specify ridge and lasso, which are special cases of the elastic net. The SCAD, $p^s_{\lambda,a}()$, and MCP, $p^m_{\lambda,a}()$ penalty functions are

$$\begin{aligned} p^s_{\lambda,a}(|x|) &= \lambda|x| I(0 \le |x| < \lambda) + \frac{a\lambda|x| - (x^2+\lambda^2)/2}{a-1} I(\lambda \le |x| \le a\lambda) + \frac{(a+1)\lambda^2}{2} I(|x| > a\lambda), \\ p^m_{\lambda,a}(|x|) &= \lambda\left(|x| - \frac{x^2}{2a\lambda}\right) I(0 \le |x| < a\lambda) + \frac{a\lambda^2}{2} I(|x| \ge a\lambda), \end{aligned}$$

where $a > 2$ for the SCAD penalty function and $a > 1$ for MCP.

The following defines $P_{\lambda,a}(\boldsymbol{\beta})$ for the four different penalty functions, plus two important special cases.

1. Elastic net: $p_{w_j\lambda,a}(|\beta_j|) = \lambda w_j \left[\alpha|\beta_j| + (1-\alpha)\beta_j^2\right]$, where $a \in [0,1]$.
   (a) Lasso: a special case with $a = 1$.
   (b) Ridge: a special case with $a = 0$.
2. Adaptive Lasso: $p_{w_j\lambda,a}(|\beta_j|) = \lambda w_j |\tilde{\beta}_j|^{-a}|\beta_j|$, where $a > 0$ and $\tilde{\beta}_j$ is the Ridge estimator for the same values of w_j and λ.
3. SCAD: $p_{w_j\lambda,a}(|\beta_j|) = p^s_{w_j\lambda,a}(|\beta_j|)$, where $a > 2$.
4. MCP: $p_{w_j\lambda,a}(|\beta_j|) = p^m_{w_j\lambda,a}(|\beta_j|)$, where $a > 1$.

The weights, w_j, allow for different weights for predictors and must be non-negative. If $w_j = 0$ then that variable will be unpenalized. In **rqPen** these weights are labeled as `penalty.factors` or `group.penalty.factors`. The lasso estimator provides the backbone for the algorithm of the three non-elastic net penalties. As $\rho_\tau(x) + \rho_\tau(-x) = |x|$, the lasso estimator minimizes,

$$\frac{1}{n}\sum_{i=1}^{n} m_i \rho_\tau(y_i - \mathbf{x}_i^\top \boldsymbol{\beta}) + \sum_{j=1}^{p} \rho_\tau(\lambda w_j \beta_j) + \rho_\tau(-\lambda w_j \beta_j). \tag{5}$$

For $i \in \{1,\ldots,n\}$ define $\tilde{y}_i = y_i$, $\tilde{\mathbf{x}}_i = \mathbf{x}_i$, and $\tilde{m}_i = m_i$. Let $\mathbf{e}_j \in \mathbb{R}^{p+1}$ represent a unit vector with a value of one in the jth position and zero in all other entries. For each $j \in \{1,\ldots,p\}$ define $\tilde{\mathbf{x}}_{n+2j-1} = -n\lambda w_j \mathbf{e}_j$ and $\tilde{\mathbf{x}}_{n+2j} = n\lambda w_j \mathbf{e}_j$. In addition, for each $i \in \{n+1,\ldots,n+2p\}$ define $\tilde{y}_i = 0$ and $\tilde{m}_i = 1$. Then minimizing (5) is equivalent to minimizing

$$\frac{1}{n}\sum_{i=1}^{n+2p} \tilde{m}_i \rho_\tau(\tilde{y}_i - \tilde{\mathbf{x}}_i^\top \boldsymbol{\beta}). \tag{6}$$

The objective function (6) has the same form as (2) except for the scaling factor, n^{-1}, and the weights. This approach of creating the augmented $2p$ samples and then using standard quantile regression is implemented in **rqPen** where the problem is solved using the `rq()` function from **quantreg**. Note this approach is different from using `rq(method="lasso",...)` within **quantreg**, which uses a linear programming approach but does not formulate the problem using augmented data.

For large values of n and p the linear programming algorithms become computationally burdensome. To decrease computational complexity, Yi and Huang (2017) proposed approximating the quantile loss function with a Huber-like function and a new coordinate descent algorithm that requires a differentiable loss function. The Huber loss function proposed by Huber (1964) is

$$h_\gamma(t) = \frac{t^2}{2\gamma} I(|t| \leq \gamma) + \left[|t| - \frac{\gamma}{2}\right] I(|t| > \gamma).$$

Note $\rho_\tau(u) = u[\tau - I(u < 0)] = \frac{1}{2}(|u| + (2\tau - 1)u)$ and for sufficiently small γ, $|u| \approx h_\gamma(u)$. The Huber-approximated quantile loss is

$$h_\gamma^\tau(u) = h_\gamma(u) + (2\tau - 1)u, \tag{7}$$

and for small γ, $\rho_\tau(u) \approx \frac{1}{2} h_\gamma^\tau(u)$. The package **hqreg** implements the approach of Yi and Huang (2017) and the function `hqreg()`, with `method="quantile"`, solves the problem of

$$\frac{1}{2n} \sum_{i=1}^{n} h_\gamma^\tau(y_i - \mathbf{x}_i^\top \boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} w_j |\beta_j|. \tag{8}$$

In **rqPen**, solving (8) is done by calling `hqreg::hqreg()` and thus **hqreg** is a required package. The Huber approximation with lasso is the default for `rq.pen()`, one of the main functions in **rqPen**. In addition, optimization of the adaptive lasso, SCAD and MCP problems can be solved using a version of (8), which is available in **rqPen** but not **hqreg**. The adaptive lasso can be solved using the same approach because it is a special case of a lasso problem with different weights for each coefficient. The initial estimators necessary for the weights are determined by a ridge estimator with the same value of λ. The SCAD and MCP functions are approximated by a local linear approximation (LLA) as proposed by Zou and Li (2008). Let $p_{w_j\lambda,a}(|\beta_j|)$ represent a generic penalty function and $p'_{w_j\lambda,a}(|\beta_j|)$ be the derivative with respect to β_j. Let $\bar{\beta}_j$ be the lasso estimator for the same value of λ and weights. The LLA approach uses the following approximation,

$$\frac{1}{n} \sum_{i=1}^{n} m_i \rho_\tau(y_i - \mathbf{x}_i^\top \boldsymbol{\beta}) + \sum_{j=1}^{p} p_{w_j\lambda,a}(|\beta_j|) \approx \frac{1}{n} \sum_{i=1}^{n} m_i \rho_\tau(y_i - \mathbf{x}_i^\top \boldsymbol{\beta}) + \sum_{j=1}^{p} p'_{w_j\lambda,a}(|\bar{\beta}_j|)|\beta_j|. \tag{9}$$

Again, the problem becomes a special case of a lasso estimator with specific weights for each predictor. Thus all the non-group penalties, except for elastic net and ridge, can be solved using linear programming or Huber approximation algorithms.

The elastic net penalty of

$$\frac{1}{n} \sum_{i=1}^{n} m_i \rho_\tau(y_i - \mathbf{x}_i^\top \boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} w_j \left[a|\beta_j| + (1-a)\beta_j^2\right], \tag{10}$$

cannot be framed as a linear programming problem because of the ridge penalty. Thus for $a \neq 1$, the **rqPen** implementation of elastic net only uses the Huber approximation approach provided in **hqreg**. While **hqreg** provides a computational backbone, **rqPen** provides SCAD, MCP, and adaptive lasso penalties that are not provided in `hqreg()`. In addition, `hqreg()` does not include any group penalties or allow for weights for observations, m_i. If weights are specified then the group lasso with singleton groups, implemented in **hrqglas**, is used instead of using `hqreg()` from **hqreg**.

2.2 Group Penalty predictors

When there exists a group structure to the predictors then a group penalty can often account for this structure. For instance, non-binary categorical variables or polynomial transformations of predictors. This section assumes the p predictors are partitioned into G

groups and $\boldsymbol{\beta}_g$ represents the coefficients associated with the gth group of predictors. Group penalized quantile regression estimators minimize

$$\frac{1}{n}\sum_{i=1}^{n} m_i \rho_\tau(y_i - \mathbf{x}_i^\top \boldsymbol{\beta}) + \sum_{g=1}^{G} p_{w_g\lambda,a}(||\boldsymbol{\beta}_g||_q). \tag{11}$$

Users can choose four penalty functions for $p_{w_g,\lambda,a}()$: (1) lasso; (2) Adaptive lasso; (3) SCAD; and (4) MCP. Specifically,

1. lasso: $\sum_{g=1}^{G} p_{w_g\lambda,a}(||\boldsymbol{\beta}_g||_2) = \lambda \sum_{g=1}^{G} w_g ||\boldsymbol{\beta}_g||_2$;
2. Adaptive lasso: $\sum_{g=1}^{G} p_{w_g\lambda,a}(||\boldsymbol{\beta}_g||_q) = \lambda \sum_{g=1}^{G} w_g ||\tilde{\boldsymbol{\beta}}_g||_q^{-a} ||\boldsymbol{\beta}_g||_q$, where $a > 0$ and $\tilde{\beta}_j$ is the Ridge estimator for the same values of λ and w_g;
3. SCAD: $P_a(\boldsymbol{\beta}) = \sum_{g=1}^{G} p^s_{w_g\lambda,a}(||\boldsymbol{\beta}_g||_q)$, where $a > 2$;
4. MCP: $P_a(\boldsymbol{\beta}) = \sum_{g=1}^{G} p^m_{w_g\lambda,a}(||\boldsymbol{\beta}_g||_q)$, where $a > 1$.

The values of q in `rq.group.pen()` is limited to $q \in \{1, 2\}$ and can be changed with `norm=q`. If $q = 2$ then group variable selection will be all-or-nothing, that is all variables within a group will be selected or none of them will be. The choice of $q = 1$ allows for bi-level variable selection (Breheny and Huang, 2009). Users cannot specify `norm=1` and `penalty="gLASSO"`, default penalty of group lasso, because that is equivalent to the individual lasso penalty implemented in `rq.pen()`.

Huang et al. (2012) provide an excellent review of group penalties, including a discussion of the use of $q = 1$. For differentiable loss functions, Breheny and Huang (2009) provide a computational perspective on $q = 1$, while Sherwood et al. (2020) compare oracle model selection properties for $q = 1$ and $q = 2$.

For $q = 1$, the group lasso estimator becomes a special case of the individual lasso estimator, and this is also true for the adaptive lasso penalty. For the SCAD and MCP penalties, LLA will be used to approximate the non-convex penalties. Let $\bar{\boldsymbol{\beta}}_g$ be an initial group lasso estimator and the penalized objective function is approximated by

$$\frac{1}{n}\sum_{i=1}^{n} m_i \rho_\tau(y_i - \mathbf{x}_i^\top \boldsymbol{\beta}) + \sum_{g=1}^{G} p'_{w_g\lambda,a}(||\bar{\boldsymbol{\beta}}_g||_q) ||\boldsymbol{\beta}_g||_q. \tag{12}$$

Thus for the case of $q = 1$ this becomes an individual lasso problem and the algorithms discussed in the previous section apply. In particular, the problem can be framed as a linear programming problem. While for $q = 2$ (12) becomes a special case of the group lasso problem. These group lasso problems are not linear programming problems, but second-order cone programming problems. Second-order cone programming problems can be solved by existing convex optimization software such as **Rmosek** (Koenker and Mizera, 2014). However, there are some barriers to using **Rmosek**, for instance it requires a copy of Mosek installed on the user's computer. In addition, similar to linear programming problems, second-order cone programming algorithms can be computationally burdensome for large values of n or p. For $q = 2$, the Huber approximation described in the previous subsection is used. However, **hqreg** cannot be used to solve this problem because the approach of Yi and Huang (2017) is not for a group penalty. Instead, the algorithm of Yang and Zou (2015) is implemented. See Sherwood and Li (2022) for details regarding the application of this algorithm to the asymmetric Huber regression setting. This approach is available in the package **hrqglas**, which is maintained by two of the authors and provides the computational backbone for the L_2 group penalties.

2.3 Estimation of Multiple Quantiles

For a set of quantiles of interest, $(\tau_1, \ldots, \tau_B)$, Belloni and Chernozhukov (2011) proposed minimizing,

$$\frac{1}{n}\sum_{b=1}^{B}\sum_{i=1}^{n} m_i \rho_{\tau_b}\left(y_i - \mathbf{x}_i^\top \boldsymbol{\beta}^{\tau_b}\right) + \frac{\tilde{\lambda}}{n}\sum_{b=1}^{B}\sum_{j=1}^{p}\sqrt{\tau_b(1-\tau_b)}\hat{\sigma}_j|\beta_j^{\tau_b}|, \tag{13}$$

where $\hat{\sigma}_j = n^{-1}\sum_{i=1}^{n} x_{ij}^2$. We use $\tilde{\lambda}$ in (13) to emphasize that the scaling of the tuning parameter is different than that used in **rqPen**, specifically $\lambda = \frac{\tilde{\lambda}}{n}$. The penalized objective function of (13) provides a quantile specific weight of $\sqrt{\tau_b(1-\tau_b)}$ and a predictor specific weight of $\hat{\sigma}_j$, while λ is assumed to have the same value. In **rqPen**, this approach is generalized to allow for different penalties and user choices of the quantile, predictor, and observation weights. For an individual penalty the estimator minimizes,

$$\frac{1}{n}\sum_{b=1}^{B}\sum_{i=1}^{n} m_i \rho_{\tau_b}\left(y_i - \mathbf{x}_i^\top \boldsymbol{\beta}^{\tau_b}\right) + \sum_{b=1}^{B}\sum_{j=1}^{p} p_{w_j d_b \lambda, a}(|\beta_j^{\tau_b}|). \tag{14}$$

For a group penalty the penalized objective function is

$$\frac{1}{n}\sum_{b=1}^{B}\sum_{i=1}^{n} m_i \rho_{\tau_b}\left(y_i - \mathbf{x}_i^\top \boldsymbol{\beta}^{\tau_b}\right) + \sum_{b=1}^{B}\sum_{g=1}^{G} p_{w_g d_b \lambda, a}(||\boldsymbol{\beta}_g||_q). \tag{15}$$

Note, the $w_j d_b \lambda$ term allows users to specify specific weights for a predictor, w_j, and specific weights for a quantile, d_b. For instance, if a user chooses the lasso penalty with $w_j = n^{-1}\sum_{i=1}^{n} x_{ij}^2$, $d_b = n^{-1}\sqrt{\tau_b(1-\tau_b)}$, and $m_i = 1$ then (14) is equivalent to (13). The optimization of (14) and (15) consist of B different optimization problems. In **rqPen** they are solved separately using the algorithms that have been discussed previously, except the same sequence of λ is used for all B problems. In Belloni and Chernozhukov (2011) they suggest choosing the same value of λ for all the quantiles, albeit with the quantile-specific weights. We later present how users can choose a single value of λ and coefficient estimates come from either (14) or (15). Alternatively, users can choose B different values of λ and then the solutions come from B different optimizations of (4) or (11). The latter approach is equivalent to a data-driven approach for estimating the optimal quantile-specific weights, d_b.

Consistent selection across quantiles

The penalties discussed previously do not ensure consistent selection across the quantiles. That is for two quantiles τ_1 and τ_2, it is possible for a variable's coefficient to be non-zero at τ_1, but zero at τ_2. Our opinion is this is often hard to interpret. Consider the case where B quantiles are considered. Define, $\boldsymbol{\beta}^j = (\beta_j^{\tau_1}, \ldots, \beta_j^{\tau_B})^\top \in \mathbb{R}^B$ as the vector of the jth coefficient for all B quantiles. Define $||\boldsymbol{\beta}^j||_{2,\boldsymbol{w},\boldsymbol{d}} = \sqrt{\sum_{b=1}^{B} d_b w_j (\beta_j^{\tau_b})^2}$, as weighted L_2-norm that allows for predictor, m_j, and quantile, w_k, specific weights. If a user desires consistent variable selection across quantiles they can use the group quantile penalized objective function,

$$\frac{1}{n}\sum_{b=1}^{B}\sum_{i=1}^{n} m_i \rho_{\tau_b}\left(y_i - \mathbf{x}_i^\top \boldsymbol{\beta}^{\tau_b}\right) + \lambda\sum_{j=1}^{p}||\boldsymbol{\beta}^j||_{2,\boldsymbol{w},\boldsymbol{d}}. \tag{16}$$

We refer to this penalty as the group quantile penalty (Li and Sherwood, 2025).

2.4 Tuning parameter selection

There are two tuning parameters, λ and a, that need to be set. The value of a defaults to commonly used values for each penalty: (1) elastic-net ($a = 1$); (2) adaptive lasso ($a = 1$); (3) SCAD ($a = 3.7$); and (4) MCP ($a = 3$). Users can also specify values of a, except for the cases of Ridge ($a = 0$) and lasso ($a = 1$) where the value of a is fixed. Typically the value of a is not as much of a concern as the value of λ, a potential notable exception to this would be the elastic net penalty. Users can specify a sequence for values of λ otherwise, a sequence will be automatically generated. Define $H^{\tau}_{\gamma}(\boldsymbol{\beta}) = \frac{1}{2n}\sum_{i=1}^{n} m_i h^{\tau}_{\gamma}(y_i - \mathbf{x}_i^{\top}\boldsymbol{\beta})$. Define $\tilde{a} = a$ if the elastic net penalty is used and one otherwise. If $a = 0$, that is the ridge penalty is being used, then $\tilde{a} = .001$. The default value for an individual penalty is $\lambda_{\max} = \max_{j,b} 1.05 \left|\frac{\partial}{\partial\beta_j} H^{\tau_b}_{\gamma}(\mathbf{0}_{p+1})\right| (w_j d_b \tilde{a})^{-1}$ and for a group penalty is $\lambda_{\max} = \max_{g,b} 1.05 \left|\left|\frac{\partial}{\partial\boldsymbol{\beta}_g} H^{\tau_b}_{\gamma}(\mathbf{0}_{p+1})\right|\right|_q (w_g d_b)^{-1}$. The 1.05 multiplier is used to ensure that $\lambda_{\max}$ provides a completely sparse solution regardless of whether the Huber approximation is used or not. If any predictors or groups are unpenalized at a given quantile then they are excluded from these calculations.

Both cross-validation and information criterion are provided as tools for selecting the optimal pair of (λ, a). Let $\hat{\boldsymbol{\beta}}^{\tau}_{\lambda,a}$ be the coefficient estimates for quantile τ using tuning parameters λ and a. Define $k^{\tau}_{\lambda,a}$ as the number of non-zero coefficients in $\hat{\beta}^{\tau}_{\lambda,a}$. The quantile information criterion (QIC) is defined as

$$QIC(\tau, \lambda, a) = \log\left[\sum_{i=1}^{n} m_i \rho_{\tau}(y_i - \mathbf{x}_i^{\top}\hat{\boldsymbol{\beta}}^{\tau}_{\lambda,a})\right] + \frac{m k^{\tau}_{\lambda,a}}{2n}, \tag{17}$$

where the value of m depends on the criterion being used. Users can select between AIC ($m = 2$), BIC [$m = \log(n)$], and a version of the large p BIC proposed by Lee et al. (2014) [$m = \log(n)\log(p)$].

Cross-validation is the other approach implemented for choosing the optimal pair of (λ, a). Consider the case of K folds, let n_k be the number of observations in the kth fold, y^k_i, $\mathbf{x}^k_i$, and m^k_i be the ith response, predictor vector, and weights for the kth fold and $\hat{\boldsymbol{\beta}}^{\tau_b}_{-k,\lambda,a}$ be the fit for the bth quantile excluding the kth fold for given values of λ and a. By default, even if Huber approximations are used, cross-validation is done using quantile loss. However, this can be changed by setting `cvFunc` in the later described functions `rq.group.pen.cv()`, `rq.gq.pen.cv()`, or `rq.pen.cv()`. For instance, `cvFunc=abs` will use absolute value loss regardless of the quantile being modeled. For simplicity of presentation, the following assumes quantile loss is used. The average quantile loss at a given fold is

$$C^{\tau}_k(\lambda, a) = \frac{1}{n_k}\sum_{i=1}^{n_k} m^k_i \rho_{\tau}(y^k_i - \mathbf{x}^{k\top}_i \hat{\boldsymbol{\beta}}^{\tau}_{-k,\lambda,a}). \tag{18}$$

The cross-validation functions return two summaries of the cross-validation for selecting (λ, a). The return value `btr` provides a table of the values of a and λ that minimize

$$C^{\tau}(\lambda, a) = \frac{1}{K}\sum_{k=1}^{K} C^{\tau}_k(\lambda, a). \tag{19}$$

In addition it provides the λ value associated with the sparsest solution that is within one standard error of the value that minimizes (19). The standard error for a given pair of (λ, a) is calculated as

$$\sqrt{\frac{1}{K-1}\sum_{k=1}^{K}\left[C^{\tau}_k(\lambda, a) - C^{\tau}(\lambda, a)\right]^2}. \tag{20}$$

The average summary can be replaced by changing the parameter `cvSummary`. For instance, `cvSummary=median` would use the median values of $C^{\tau}_k(\lambda, a)$.

Users can also choose to optimize λ and a, so they are the same values across all quantiles. When evaluating the cross-validation error they can choose to not provide equal weight to each quantile by entering quantile specific weights of z_b. The return value `gtr` provides results for the value of λ and a that minimize

$$C(\lambda, a) = \sum_{b=1}^{B} z_b C^{\tau_b}(\lambda, a). \tag{21}$$

Users can choose between a single pair of (λ, a) that minimizes (21) or B, potentially different, pairs that minimize (19) separately for each quantile except for when minimizing (16). That is a joint optimization problem and thus only joint selection of tuning parameters using (21) is allowed. In addition, results for the one standard error approach, where the standard error for a fixed (λ, a) pair is

$$\sqrt{\frac{1}{K-1}\sum_{k=1}^{K}\sum_{b=1}^{B} z_b \left[C_k^{\tau}(\lambda, a) - C^{\tau}(\lambda, a)\right]^2}, \tag{22}$$

are provided. The one standard error rule finds the sparsest solution, by increasing λ, where the value of (19) or (21) are within one standard error of the optimal value, assuming the optimal a is fixed.

By setting `groupError=FALSE` each observation is weighted equally and (19) and (21) are replaced by $\sum_{k=1}^{K} n_k C_k^{\tau}(\lambda, a)$, and $\sum_{b=1}^{B} z_b \sum_{k=1}^{K} n_k C_k^{\tau}(\lambda, a)$, respectively.

Tuning parameter selection for multiple quantiles using an information criterion

For the case of multiple, B, quantiles being modeled, **rqPen** offers two different approaches for selecting the tuning parameters. One approach is to select B pairs of (λ, a) that minimize the B different versions of (17). The other approach is given weights $(z_1, \ldots, z_b)$, the optimal pair of (λ, a) minimizes,

$$\sum_{b=1}^{B} z_b QIC(\tau_b, \lambda, a). \tag{23}$$

The weights offer flexibility to a user who wants to provide more weight to a specific quantile. The default value for all the weights is one thus giving equal weight to each quantile.

2.5 Additional notes on Huber approximation

For Huber approximation, a value of γ is needed. If γ is too large then the estimator will have a large amount of bias. If γ is too small the algorithms will become unstable. The values of γ are updated for each value of λ. Let $\mathbf{r}^k$ be the vector of residuals corresponding to the estimator using the kth value of λ, λ^k, and $Q_{.1}(|\mathbf{r}^k|)$ be the 10th percentile of the absolute values of these residuals. For an individual penalty we use what is implemented in **hqreg**, which is very similar to what is outlined by Yi and Huang (2017). Specifically,

$$\begin{aligned}\gamma^k &= I(|1-\tau| < .05)\max\left\{.0001, \min\left[\gamma^{k-1}, Q_{.01}(|\mathbf{r}^k|)\right]\right\} \\ &\quad + I(|1-\tau| >= .05)\max\left\{.001, \min\left[\gamma^{k-1}, Q_{.1}(|\mathbf{r}^k|)\right]\right\}.\end{aligned}$$

For the group penalties, as presented in Sherwood and Li (2022),

$$\gamma^k = \min\left\{4, \max\left[.001, Q_{.1}(|\mathbf{r}^k|)\right]\right\}.$$

One reason the linear and second-order cone programming problems are computationally slow for larger values of p is they optimize across all potential predictors. Tibshirani et al. (2012) proposed a strong rule for discarding predictors that can greatly decrease the

computational complexity of penalized optimization. The strong rule assumes a differentiable loss function and thus is only implemented for the Huber approximations. In addition, the Huber approximation approaches use warm starts across a path of potential λ values. It starts with the largest potential value of λ then uses that solution as the initial value for the next largest potential value of λ. This iterative process is continued until all solutions have been found. The linear programming approaches rely on `quantreg::rq()` which does not have an option for initial values. Thus warm starts are not implemented for those approaches. For an individual penalty, the calculations of the strong rule are done in the package **hqreg**, and details can be found in Yi and Huang (2017). A similar approach has been used for group penalties (Sherwood and Li, 2022).

3 Description of functions

Below are the main functions of **rqPen** and an S3 method that is unique to **rqPen**, `bytau.plot()`.

- `rq.pen()` minimizes (14) to provide estimates of a quantile regression model, for potentially multiple quantiles, using an individual penalty of either elastic-net, adaptive lasso, SCAD, or MCP. If not specified, a sequence of λ values is automatically generated. Returns an '`rq.pen.seq`' S3 object that works with `bytau.plot()`, `coef()`, `plot()`, `predict()`, and `print()` methods.
- `rq.gq.pen()` minimizes (16) to provide a quantile regression model for multiple quantiles using the group quantile lasso penalty. It also returns an '`rq.pen.seq`' S3 object and can be used with the same methods listed above.
- `rq.group.pen()` minimizes (15) and is a group penalty version of `rq.pen()`. Users have access to group versions of the lasso, adaptive lasso, SCAD, and MCP penalties. The lasso penalty is restricted to the L2-norm. For other penalties, users can choose between the L1- or L2-norm. It also returns an '`rq.pen.seq`' S3 object and can be used with the same methods listed above.
- `rq.pen.cv()` automates K-folds cross-validation for selecting the tuning parameters λ and a. If λ is left undefined, a sequence of λ values will be automatically generated. For a, the default values described earlier will be used, unless the user specifies a sequence. If multiple quantiles are modeled then it provides the optimal pair of (λ, a) for each quantile and the optimal pair across all quantiles. Returns an '`rq.pen.seq.cv`' S3 object that works with `bytau.plot()`, `coef()`, `plot()`, `predict()`, and `print()` methods.
- `rq.gq.pen.cv()` is the group quantile lasso version of `rq.pen.cv()`. It does everything listed above, but for the group quantile penalty. It also returns an '`rq.pen.seq.cv`' object.
- `rq.group.pen.cv()` is the group version of `rq.pen.cv()`. It does everything listed above, but for group penalties. It also returns an '`rq.pen.seq.cv`' object.
- `qic.select()` takes an '`rq.pen.seq`' or '`rq.pen.seq.cv`' object and provides the optimal values of (λ, a) using an information criterion, as explained in the previous section. It returns a '`qic.select`' S3 object that works with `coef()`, `predict()`, and `print()` methods.
- `bytau.plot()` plots coefficient estimates for each predictor as a function of τ.

The `rq.pen()` function provides estimates of conditional quantiles from minimizing a penalized objective function for a sequence of λ values.

```
rq.pen(x, y, tau = 0.5, lambda = NULL,
        penalty = c("LASSO", "Ridge", "ENet", "aLASSO", "SCAD", "MCP"),
        a = NULL, nlambda = 100, eps = ifelse(nrow(x) < ncol(x), 0.05, 0.01),
        penalty.factor = rep(1, ncol(x)),
        alg = ifelse(sum(dim(x)) < 200, "br", "huber"), scalex = TRUE,
        tau.penalty.factor = rep(1, length(tau)),
```

```
    coef.cutoff = 1e-08, max.iter = 10000, converge.eps = 1e-07,
    lambda.discard = TRUE, weights=NULL, ...)
```

The function `rq.pen()` requires a design matrix `x` and vector of response `y`. The predictors in the design matrix will be centered to have mean zero and standard deviation one when minimizing the penalized objective function unless `scalex` is set to FALSE. While the predictors are scaled, the coefficients are returned on the original scale of `x`. The quantiles modeled are set with `tau`. Users can choose the `penalty` function, with the default being lasso. A pre-specified lambda sequence can be set, `lambda`. If the user does not specify a sequence of λ values then $\lambda_{min} = \epsilon\lambda_{max}$, where ϵ can be set by `eps` and `nlambda` is the number of λ values considered. Though very small values of λ may be discarded if the coefficient estimates are not changing much with smaller values of λ, unless `lambda.discard=FALSE`. For non-ridge or non-lasso penalties, a sequence of values can also be selected for `a`. If `a` is not set then default values depend on the penalty, elastic-net ($a = 0$), adaptive lasso ($a = 1$), SCAD ($a = 3.7$) and MCP ($a = 3$). Penalty factors can be set for predictors, `penalty.factor`, or quantiles, `tau.penalty.factor`. Observation weights are set using `weights`. The linear programming algorithms can provide very small, but non-zero estimates, and these coefficients are set to zero if they are below `coef.cutoff`. The choice of algorithm can be set using `alg`. Two linear programming algorithms are provided, the Barrodale and Roberts algorithm (br) (Barrodale and Roberts, 1974), as described in Koenker and D'Orey (1987) and Koenker and D'Orey (1994), and the Frisch-Newton (fn) approach, described in Portnoy and Koenker (1997). Both approaches rely on implementations in **quantreg**. Setting `rq.pen(alg="huber")` uses the approach of Yi and Huang (2017) implemented in **hqreg**. In addition, they can set the maximum number of iterations, `max.iter`, or convergence criteria, `converge.eps`, though these only apply to the "huber' ' algorithms.

```
coef.rq.pen.seq(object, tau = NULL, a = NULL, lambda = NULL, modelsIndex = NULL,
    lambdaIndex = NULL, ...)
```

Using `coef()`, users can extract coefficients for specific quantiles, `tau`, and tuning parameters, `a` and `lambda`, from class '`rq.pen.seq`' object. Alternatively, they can directly specify the models and lambda values using `modelsIndex` and `lambdaIndex`, respectively. If none of these values are set then a matrix of coefficients for all quantiles and tuning parameters will be returned.

```
coef.rq.pen.seq.cv(object, septau = TRUE, cvmin = TRUE, useDefaults = TRUE,
    tau = NULL, ...)
```

With an '`rq.pen.seq.cv`' object, the `coef()` function returns coefficients based on the cross-validation results. If `cvmin=TRUE` then the tuning parameters associated with the minimum cross-validation error are returned, otherwise the one standard error rule is used. When `septau=TRUE` then the tuning parameters are optimized individually for each quantile. Users can identify the specific quantiles, `tau`, they wish to return. The default is to return results for all quantiles. Setting `useDefaults=FALSE` ignores the cross-validation results and users can use the arguments in `coef.rq.pen.seq()` to select coefficients. For instance, if `useDefaults=FALSE` and no other arguments are provided then all coefficients are returned.

Function `bytau.plot()` has a similar form for the '`rq.pen.seq`' and '`rq.pen.seq.cv`' objects.

```
bytau.plot.rq.pen.seq(x, a = NULL, lambda = NULL, lambdaIndex = NULL, ...)

bytau.plot.rq.pen.seq.cv(x, septau = TRUE, cvmin = TRUE, useDefaults = TRUE, ...)
```

These functions produce a plot of how coefficients change with quantiles. The arguments in `bytau.plot.rq.pen.seq()` and `bytau.plot.rq.pen.seq.cv()` are the same as those

covered for `coef()`, but with one major difference. Only one vector of coefficients can be provided for each quantile. When using an '`rq.pen.seq`' object, users must specify a single pair of (λ, a). When using an '`rq.pen.seq.cv`' object, users can rely on the default tools for selecting coefficients or specify a single pair of λ and a.

Both '`rq.pen.seq`' and '`rq.pen.seq.cv`' objects have `plot()` functions. Using `plot()` with an '`rq.pen.seq.cv`' object provides a plot of how the cross-validation error changes with λ for each quantile.

```
plot.rq.pen.seq.cv(x, septau = TRUE, tau = NULL, logLambda = FALSE, main = NULL, ...)
```

If `septau=TRUE` then a separate plot is created for each quantile, otherwise one plot is created that combines the cross-validation error across all quantiles. Users can limit the plots to a subset of quantiles, `tau`. If `logLambda=TRUE` then the x-axis will be $\log(\lambda)$. The `main` text can be set otherwise the main title depends on the name of the variable and the value of `septau`.

While `plot()` for an '`rq.pen.seq`' object creates a plot of the coefficients as they change with λ.

```
plot.rq.pen.seq(x, vars = NULL, logLambda = TRUE, tau = NULL, a = NULL,
    lambda = NULL, modelsIndex = NULL, lambdaIndex = NULL, main = NULL, ...)
```

The `plot()` function will create a plot for each combination of τ and a. Users can specify specific values of `tau` and `a` if they only want to consider a subset. They can also limit the values of `lambda`. Choice of models and λ values can also be done using `modelsIndex` and `lambdaIndex`. The `main` text can be set. If `logLambda=TRUE` then the λ values are reported on the natural log scale.

4 Applications

This section details how to use individual and group penalties. Different individual penalties and algorithms will be applied to the Barro data set from the **quantreg** package. In addition, the group quantile penalty is used with the Barro data set. While the group penalties for predictors will be used with the Ames housing data (De Cock, 2011) available in **AmesHousing**.

4.1 The Barro data set

The Barro data set, available in **quantreg**, contains GDP growth rates and 13 other potential explanatory variables. See Koenker and Machado (1999) for more details.

Using different algorithms

First, we look at fitting a model with the SCAD penalty for $\tau = .5$ using three different algorithms.

```
library(rqPen)
#quantreg is required for rqPen, but call directly here
#because we need the Barro data set
library(quantreg)
data(barro)
y <- barro$y.net
x <- as.matrix(barro[,-1])
qbr <- rq.pen(x,y,alg="br", penalty="SCAD")
qfn <- rq.pen(x,y,alg="fn", penalty="SCAD")
qhuber <- rq.pen(x,y,alg="huber",penalty="SCAD")
```

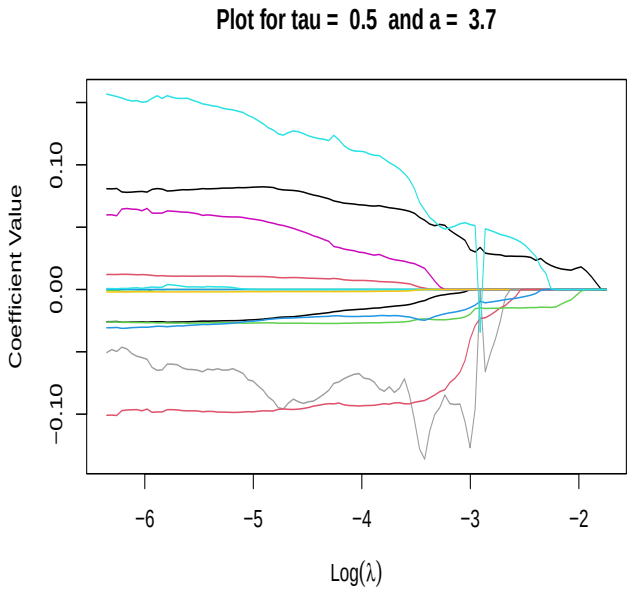
Where "br" and "fn" are the Barrodale and Roberts (Koenker and D'Orey, 1987, 1994) and Frisch-Newton (Portnoy and Koenker, 1997) interior point method for linear programming problems implemented in `quantreg:::rq()`. The following code takes the 25th value in the sequence of λ values and compares the coefficients for the three different approaches creating three different '`rq.pen.seq`' objects.

```
targetLambda <- qbr$lambda[25]
brCoef <- coefficients(qbr,lambda=targetLambda)
fnCoef <- coefficients(qfn, lambda=targetLambda)
huberCoef <- coefficients(qhuber, lambda=targetLambda)
coefDf <- cbind(brCoef,fnCoef,huberCoef)
colnames(coefDf) <- c("br","fn","huber")
coefDf
```

```
#>                    br          fn       huber
#> intercept  0.02018841  0.02018852  0.02131606
#> lgdp2      0.00000000  0.00000000  0.00000000
#> mse2       0.00000000  0.00000000  0.00000000
#> fse2       0.00000000  0.00000000  0.00000000
#> fhe2       0.00000000  0.00000000  0.00000000
#> mhe2       0.00000000  0.00000000  0.00000000
#> lexp2      0.00000000  0.00000000  0.00000000
#> lintr2     0.00000000  0.00000000  0.00000000
#> gedy2     -0.01773640 -0.01773726 -0.06602351
#> Iy2        0.02366121  0.02366100  0.02917322
#> gcony2    -0.02007166 -0.02007206 -0.02269923
#> lblakp2   -0.01433958 -0.01433962 -0.01524323
#> pol2      -0.01112295 -0.01112292 -0.01067910
#> ttrad2     0.04986245  0.04986267  0.04863867
```

The three different algorithms provide coefficient estimates that are different but similar. For larger data sets, with a large number of observations or variables, we recommend using the Huber approximation because this greatly reduces computational time while providing a reasonable solution (Yi and Huang, 2017). The following presents a plot of how the coefficient values from `qhuber` change with λ.

```
plot(qhuber)
```



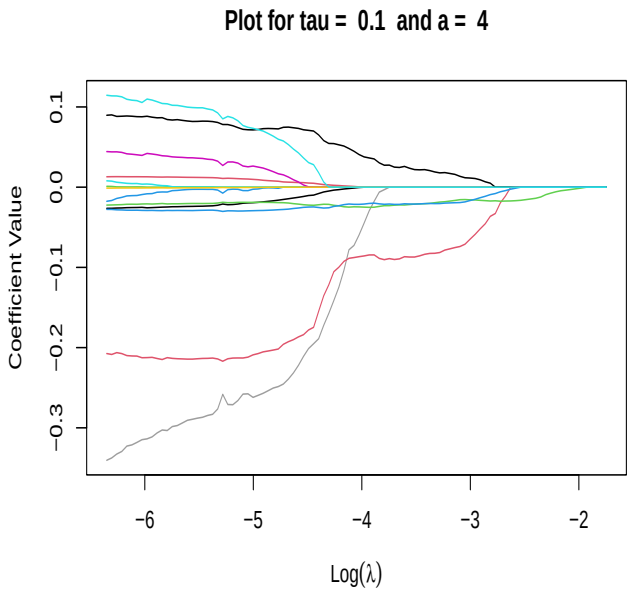
Multiple quantiles

Next, we fit a model for multiple quantiles of $\tau \in \{.1, .5, .9\}$ and will also consider values of $a \in \{3, 4\}$ for the additional tuning parameter for SCAD. A fit with multiple quantiles remains an '`rq.pen.seq`' object.

```
qmult <- rq.pen(x,y,tau=c(.1,.5,.9),a=c(3,4),penalty="SCAD")
```

The following code provides a plot of the coefficients for $\tau = .1$ and $a = 4$.

```
plot(qmult, tau=.1,a=4)
```



Using IC to select tuning parameters

When jointly optimizing the tuning parameter selection the same values of λ and a will be selected for each quantile by minimizing (23). Otherwise, the optimal tuning parameters will be selected by minimizing (17) for each quantile separately. Users can specify `method="AIC"`, `method="BIC"` or `method="PBIC"` for a large p BIC proposed by Lee et al. (2014). When doing joint selection, users can also specify `weights`, z_b in (23). The default is to provide an equal weight, of one, for all quantiles.

The below code selects the tuning parameters of (λ, a) for the `qmult` model using AIC. This code will create two '`qic.select`' objects, `qic_sep` and `qic_joint`.

```
qic_sep <- qic.select(qmult,method="AIC")
qic_joint <- qic.select(qmult, method="AIC", septau=FALSE)
```

Below provides the tuning parameters selected by the two different approaches. The `modelsInfo` attribute provides information on the model selected for each quantile.

```
qic_sep$modelsInfo
```

```
#>      tau modelIndex     a      minQIC lambdaIndex      lambda
#>    <num>      <int> <num>       <num>       <int>       <num>
#> 1:   0.1          1     3 -0.80983835         100 0.001744156
#> 2:   0.5          3     3  0.05987224          71 0.006721156
#> 3:   0.9          5     3 -0.86499611          82 0.004029227
```

```
qic_joint$modelsInfo
```

```
#>          modelIndex a tau      minQIC lambdaIndex      lambda
#> tau0.1a3          1 3 0.1 -0.80983835         100 0.001744156
#> tau0.5a3          3 3 0.5  0.06799888         100 0.001744156
#> tau0.9a3          5 3 0.9 -0.86105610         100 0.001744156
```

The above tables provide the following information for the models selected for each quantile.

- `tau`: The quantile being modeled.
- `a`: The value of a selected.
- `minQIC`: The QIC value for the given quantile and tuning parameters.
- `lambdaIndex`: The position in the sequence for the preceding λ value.
- `lambda`: The value of λ at the minimum cross-validation error.

When defining `qic_joint` we set `septau=FALSE`. Thus only a single pair of (λ, a) is selected to the be the best for all nine quantiles. That is why the values of `lambda` and `a` are the same for all quantiles in `qic_joint$modelsInfo`, but can differ in `qic_sep$modelsInfo`. The coefficients can be extracted using `coef()`.

```
coef(qic_sep)
```

```
#>                 tau=0.1      tau=0.5      tau=0.9
#> intercept  0.0238504916 -0.024276359 -0.045586339
#> lgdp2     -0.0267825117 -0.024713637 -0.034210392
#> mse2       0.0129219159  0.010444821  0.017737529
#> fse2       0.0008487373  0.000000000 -0.004439600
#> fhe2      -0.0186593726  0.000000000  0.000000000
#> mhe2       0.0081453667  0.000000000  0.000000000
#> lexp2      0.0487678850  0.057746988  0.085483398
#> lintr2    -0.0010966058 -0.001761961 -0.002291657
#> gedy2     -0.3527018635 -0.068097230 -0.035234865
#> Iy2        0.0853432506  0.081856907  0.057654273
#> gcony2    -0.2044733845 -0.098175742 -0.084273528
#> lblakp2   -0.0226578393 -0.026987474 -0.033076575
#> pol2      -0.0282535823 -0.025880939  0.000000000
#> ttrad2     0.1155784172  0.140594440  0.225825288
```

```
coef(qic_joint)
```

```
#>                 tau=0.1       tau=0.5      tau=0.9
#> intercept  0.0238504916 -0.0326938848 -0.039997413
#> lgdp2     -0.0267825117 -0.0268792897 -0.034043618
#> mse2       0.0129219159  0.0108844720  0.018840828
#> fse2       0.0008487373  0.0000000000 -0.005800585
#> fhe2      -0.0186593726  0.0003025406  0.000000000
#> mhe2       0.0081453667  0.0052844922  0.002750732
#> lexp2      0.0487678850  0.0639833573  0.083730580
#> lintr2    -0.0010966058 -0.0020594254 -0.002348213
#> gedy2     -0.3527018635 -0.0471775148 -0.047890233
#> Iy2        0.0853432506  0.0797565855  0.059880455
#> gcony2    -0.2044733845 -0.0973417525 -0.088821669
#> lblakp2   -0.0226578393 -0.0264829810 -0.033128704
#> pol2      -0.0282535823 -0.0314157975  0.001284625
#> ttrad2     0.1155784172  0.1654870976  0.227607952
```

Predictions from models

Users can make predictions from either the '`qic.select`' or '`rq.pen.seq`' objects. Prediction from a '`qic.select`' object will return predictions for all quantiles modeled.

```
# creating new data using the mean of all variables
newData <- apply(barro,2,mean)[-1] #removing response
predict(qic_sep,newData)

#>            tau=0.1    tau=0.5    tau=0.9
#> [1,] -0.0002466134 0.01894553 0.03922352
```

The default for `predict()` for an '`rq.pen.seq`' object is to return predictions for all values of λ, a, and τ. For `qmult` this results in 900 predictions for one row of predictors because there are 100 λ values, 3 a values, and 3 quantiles being modeled.

```
allPreds <- predict(qmult,newData)
allPreds[,1:5] #Present the first five predictions

#> tau0.1a3 L1 tau0.1a3 L2 tau0.1a3 L3 tau0.1a3 L4 tau0.1a3 L5
#> -0.01338804 -0.01338804 -0.01338804 -0.01338804 -0.01338804
```

Following the `coef.rq.pen.seq` code presented in the previous section, users can specify predictions from specific values of λ, τ, or a.

```
somePreds <- predict(qmult,newData, tau=c(.1,.5,.9),a=4,lambda=qmult$lambda[25])
somePreds

#>          tau0.1a4   tau0.5a4   tau0.9a4
#> [1,] -0.008353503 0.01773852 0.05166048
```

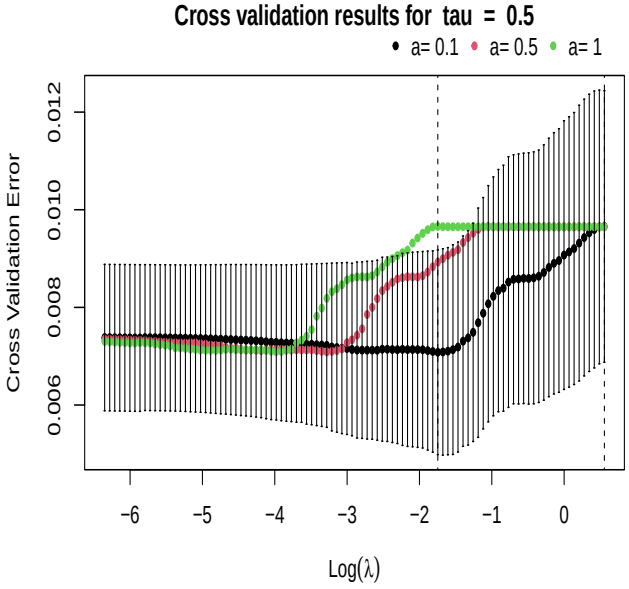
Cross-validation for tuning parameter selection

The function `rq.pen.cv()` creates an '`rq.pen.seq.cv`' object that can be used to select tuning parameters. Below we consider a similar model but use elastic net instead of SCAD. We set `tauWeights` to have $z_b = \tau_b(1-\tau_b)$, which provides more weights to errors at the median and less to the more extreme quantiles.

```
set.seed(1)
tauVals <- c(.1,.5,.9)
qcv <- rq.pen.cv(x,y,tau=tauVals,a=c(.1,.5,1),penalty="ENet",
                 tauWeights = tauVals*(1-tauVals))
```

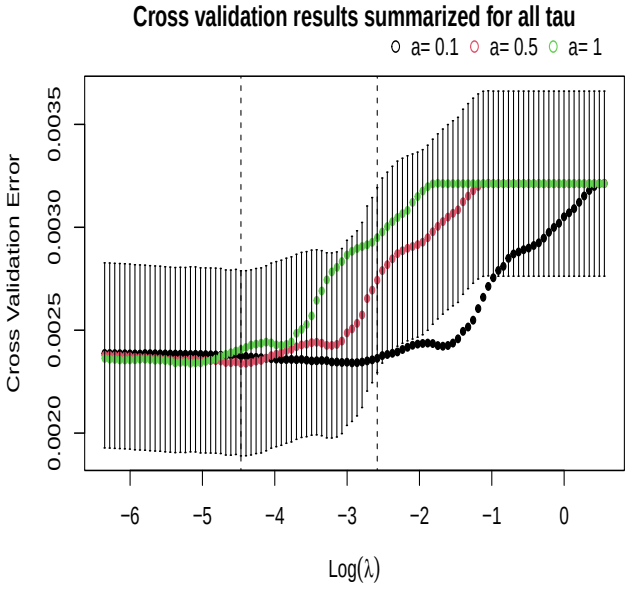
The `plot()` function with an '`rq.pen.seq.cv`' object can create a plot of cross-validation error for a single quantile, (18), or across all quantiles, (21). The following provides a plot for cross-validation error for $\tau = .5$ as a function of λ and a.

```
plot(qcv,tau=.5)
```



The error bars are only provided for the sequence associated with the optimal value of a, $a = .5$ in this example. The dashed lines are only for that optimal value, too. The first dashed line is at the value of λ that minimizes the cross-validation error, while the second dashed line is at the value of λ which is one standard error above the minimum value. Users can also get a similar plot that describes how the cross-validation error across all quantiles changes with λ and a. If a user is only interested in one quantile then the above and below plot would be identical.

```
plot(qcv,septau=FALSE)
```



For an '`rq.pen.seq.cv`' object the `predict()` function relies on the `coef()` function. Below are four predictions using the four combinations of `septau` and `cvmin`.

```
p1 <- predict(qcv,newData)
```

```
p2 <- predict(qcv,newData, septau = FALSE)
p3 <- predict(qcv,newData, cvmin=FALSE)
p4 <- predict(qcv,newData, septau=FALSE,cvmin=FALSE)
pred <- cbind(t(p1),t(p2),t(p3),t(p4))
colnames(pred) <- c("sepMin","jointMin","sep1se","joint1se")
pred

#>                 sepMin     jointMin       sep1se     joint1se
#> tau0.1a0.1 -0.00170812 -0.002409442 -0.006801621 -0.006044798
#> tau0.5a0.1  0.01895257  0.018958411  0.019494565  0.017972817
#> tau0.9a1    0.03892322  0.038575608  0.051660477  0.051154967
```

The `print()` of an '`rq.pen.seq.cv`' object provides information about the cross-validation results.

```
qcv <- rq.pen.cv(x,y,tau=tauVals,a=c(.1,.5,1),penalty="ENet",
                 tauWeights = tauVals*(1-tauVals))
qcv

#>
#> Cross validation tuning parameter optimized for each quantile
#>      tau       minCv      lambda lambdaIndex  lambda1se lambda1seIndex     a
#>    <num>       <num>       <num>       <int>      <num>          <int> <num>
#> 1:   0.1 0.003125878 0.002005356          98 0.09308038             43   0.5
#> 2:   0.5 0.006990980 0.008095657          78 0.23056721             30   0.5
#> 3:   0.9 0.002900349 0.013193903          71 0.08680707             44   0.1
#>            cvse modelsIndex nonzero  nzse
#>           <num>       <int>   <int> <int>
#> 1: 0.0007264311           2      14     5
#> 2: 0.0014415984           5      11     3
#> 3: 0.0006872771           7      13    11
#>
#> Cross validation tuning parameter optimized across all quantiles
#>      tau       minCv     lambda lambdaIndex lambda1se lambda1seIndex     a
#>    <num>       <num>      <num>       <int>     <num>          <int> <num>
#> 1:   0.1 0.002291483 0.01230468          72 0.3268231             25   0.1
#> 2:   0.5 0.002291483 0.01230468          72 0.3268231             25   0.1
#> 3:   0.9 0.002291483 0.01230468          72 0.3268231             25   0.1
#>            cvse modelsIndex nonzero  nzse
#>           <num>       <int>   <int> <int>
#> 1: 0.0003916137           1      14     6
#> 2: 0.0003916137           4      13    11
#> 3: 0.0003916137           7      13     4
```

The printed output returns results optimized at each quantile and grouping across all quantiles.

- `tau`: The quantile being modeled.
- `minCv`: The value of the minimum cross-validation error.
- `lambda`: The value of λ at the minimum cross-validation error.
- `lambdaIndex`: The position in the sequence for the preceding λ value.
- `lambda1se`: The λ value using the one standard error rule.
- `a`: The optimal value of a decided by the minimum cross-validation error, and the same value is used for the one standard error λ value.

- `cvse`: The standard error of the cross-validation error for the values of (λ, a) that provide the smallest cross-validation error.
- `modelsIndex`: The position of the corresponding '`rq.pen.seq`' object in the `$fit$models` list.
- `nonzero`: The number of nonzero coefficients at the minimum value.
- `nzse`: The number of nonzero coefficients at the one standard error rule.

Crossing quantiles

Crossing quantiles occurs when multiple quantiles are being modeled and predictions of a lower quantile are larger than predictions from an upper quantile. Take the following example using the Barro data set that looks at the fitted values for a model of the .1, .2 and .5 quantiles, for the 50th value in the λ sequence.

```
qCross <- rq.pen.cv(x,y,tau=c(.1,.2, .5))
fitCross <- predict(qCross,newx=x)
```

```
#> Warning in checkCrossSep(preds, sort, object$fit$penalty): Crossing quantiles
#> at observations 17, 28, 40, 79, 94, 109, 139, 152 when using seperate
#> optimization for each lambda. Setting septau=FALSE may reduce the number of
#> crossings. In addition, using rq.gq.pen() may reduce the number of crossings.
```

Warnings are provided about potential crossings at observations 17, 28, 40, 79, 94, 109, 139, and 152. Further inspection verifies this is the case as the fitted values at these observations have the .1 quantile larger than the .2 quantile. Examining these predictions verifies that is the case.

```
fitCross[c(17, 28, 40, 79, 94, 109, 139, 152),]
```

```
#>                       tau0.1a1     tau0.2a1     tau0.5a1
#> Zimbabwe75       2.477067e-02  0.024519238  0.031127465
#> United_States75  1.252791e-02  0.009830172  0.014629574
#> Indonesia75      2.545159e-02  0.024898722  0.037985364
#> Gambia85        -3.663382e-03 -0.004988545  0.009901519
#> Zaire85         -3.490489e-02 -0.034921445 -0.019417390
#> United_States85 -5.850236e-05 -0.005162521  0.005242369
#> Yemen85          1.213681e-02  0.006765797  0.016998950
#> Norway85         3.488170e-02  0.034015862  0.046534611
```

There is an option to sort the predictions when there are crosses. The default is not to do this because crossing quantiles can often be evidence of a misspecified model. Even when using the sort option there is a warning provided that the original predictions had crossing values.

```
fitSort <- predict(qCross, newx=x, sort=TRUE)
```

```
#> Warning in checkCrossSep(preds, sort, object$fit$penalty): Quantile predictions
#> sorted at observations 17, 28, 40, 79, 94, 109, 139, 152 when using seperate
#> optimization for each lambda. Setting septau=FALSE may reduce the number of
#> crossings. In addition, using rq.gq.pen() may reduce the number of crossings.
```

```
fitSort[c(17, 28, 40, 79, 94, 109, 139, 152), ]
```

```
#>                     tau0.1a1      tau0.2a1     tau0.5a1
#> Zimbabwe75       0.024519238  2.477067e-02  0.031127465
#> United_States75  0.009830172  1.252791e-02  0.014629574
#> Indonesia75      0.024898722  2.545159e-02  0.037985364
#> Gambia85        -0.004988545 -3.663382e-03  0.009901519
#> Zaire85         -0.034921445 -3.490489e-02 -0.019417390
#> United_States85 -0.005162521 -5.850236e-05  0.005242369
#> Yemen85          0.006765797  1.213681e-02  0.016998950
#> Norway85         0.034015862  3.488170e-02  0.046534611
```

Consistent selection across quantiles

Note the output for `coefficients(qic_sep)` and `coefficients(qic_joint)` both show the sparsity structure changes with the quantile being modeled. We find interpreting such results to be difficult. A penalty that enforces the same sparsity structure across quantiles would avoid this issue. The following code will minimize (16). Coefficients from the 13th entry in the λ sequence demonstrate that this approach provides consistency in variable selection across the quantiles. This property will hold for any value of λ. The function `rq.gq.pen.cv()` can be used for cross-validation of this approach. The function `rq.gq.pen()` returns a '`rq.pen.seq`' object and thus functions such as `predict()`, `plot()`, and `qic.select()` will work as previously described.

```
qCons <- rq.gq.pen(x,y,tau=tauVals)
coefficients(qCons,lambdaIndex=13)
```

```
#>                tau0.1a1      tau0.5a1     tau0.9a1
#> intercept -0.1039717886  0.0565817889  0.212830365
#> lgdp2     -0.0002948174 -0.0003086188 -0.000304075
#> mse2       0.0000000000  0.0000000000  0.000000000
#> fse2       0.0000000000  0.0000000000  0.000000000
#> fhe2       0.0000000000  0.0000000000  0.000000000
#> mhe2       0.0000000000  0.0000000000  0.000000000
#> lexp2     -0.0094731519 -0.0093673950 -0.008568437
#> lintr2     0.0000000000  0.0000000000  0.000000000
#> gedy2      0.0000000000  0.0000000000  0.000000000
#> Iy2        0.0721650464  0.0699737010  0.070700812
#> gcony2    -0.0426233067 -0.0442306488 -0.038750494
#> lblakp2   -0.0290133965 -0.0290175588 -0.028373662
#> pol2      -0.0096515098 -0.0097285059 -0.010002021
#> ttrad2     0.0000000000  0.0000000000  0.000000000
```

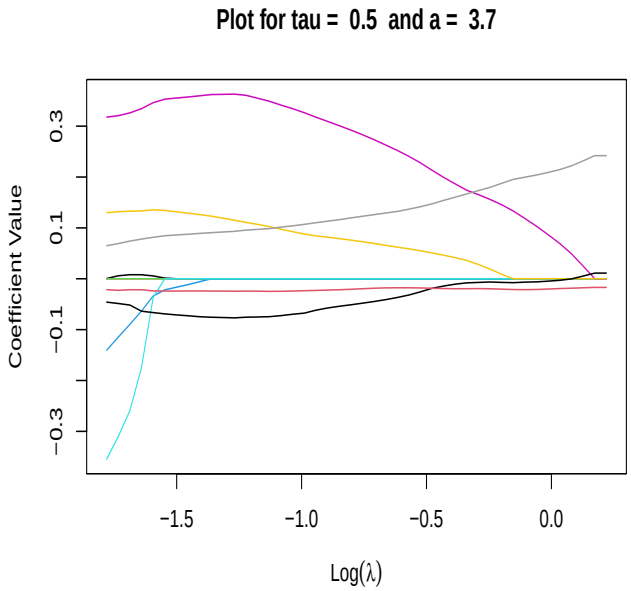
4.2 Group Penalty Example

The Ames Housing data contains many factor variables and is available in **AmesHousing**. For simplicity of presentation, we consider only Overall_Cond, Garage_Type, Full_Bath, Fireplaces, and Lot_Config as potential predictors. Some of these are categorical predictors, and thus a group penalty can account for the group structure. For the group variables some factor levels are merged due to small number of observations in a group. For instance, we combine the groups FR2 and FR3, frontage on two or three sides, in Lot_Config because only 14 houses are in group FR3. The function `rq.group.pen()` creates an '`rq.pen.seq`' object, where the models are fit with a group penalty. In the below example, we fit a model for log sale price using these predictors but do not penalize the Lot_Config variable. For the other groups the square root of the group size is used as the group penalty factor. In addition, quantile-specific penalties of $\tau(1-\tau)$ are used. Two '`rq.pen.seq`' objects are fit, one with $q = 1$, `norm=1`, and the other with $q = 2$, `norm=2`.

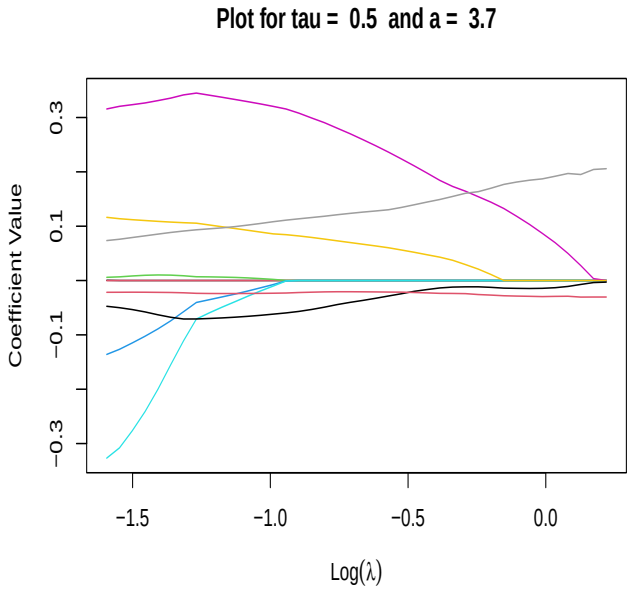
```
library(AmesHousing)
library(forcats)
ames <- make_ames()
ames$Overall_Cond <- fct_collapse(ames$Overall_Cond,
                                  below=c("Very_Poor","Poor","Fair","Below_Average"),
                                  above=c("Above_Average","Good","Very_Good",
                                          "Excellent", "Very_Excellent"))
ames$Garage_Type <- fct_collapse(ames$Garage_Type,
                       other=c("Basment","BuiltIn","CarPort","More_Than_Two_Types"))
ames$Lot_Config  <- fct_collapse(ames$Lot_Config, frontage=c("FR2","FR3"))

x_g <- cbind(model.matrix(~ Overall_Cond+Garage_Type+Full_Bath
                          +Fireplaces+Lot_Config,ames))[,-1]

y_g <- log(ames$Sale_Price)
g <-  c(rep(1,2),rep(2,3),3,4,rep(5,3))
gpf <- sqrt(xtabs(~g))
gpf[5] <- 0
qAmes1 <- rq.group.pen(x_g,y_g,groups=g, group.pen.factor = gpf, tau=tauVals,
                       penalty="gSCAD",norm=1,tau.penalty.factor=tauVals*(1-tauVals))
qAmes2 <- rq.group.pen(x_g,y_g,groups=g, group.pen.factor = gpf, tau=tauVals,
                       penalty="gSCAD",tau.penalty.factor=tauVals*(1-tauVals))
```

The below plots present how the median coefficient estimates change with λ.

```
plot(qAmes1,tau=.5)
```




```
plot(qAmes2,tau=.5)
```



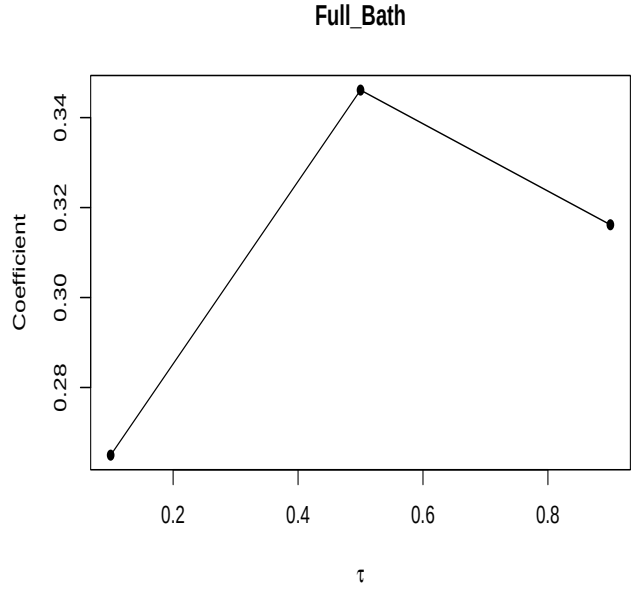
Two important things to note from the above two plots. First, for both models, the unpenalized coefficients for `Lot_Config` are never set to zero. Second, the group coefficients for `qAmes2` go to zero as a group. While the grouped coefficients for `qAmes1` tend to go to zero as a group but do not have to converge to zero at the same time. For instance, for $\tau = .5$ and using the 40th value of λ the garage and overall condition coefficients from `qAmes1` provide a mix of zero and non-zero coefficients. This does not occur for the default setting of $q = 2$, `norm=2`.

```
coefficients(qAmes1,tau=.5, lambdaIndex=40)
```

```
#>                          tau0.5a3.7
#> intercept              11.425638982
#> Overall_CondAverage     0.005564408
#> Overall_Condabove       0.000000000
#> Garage_Typeother        0.000000000
#> Garage_TypeDetchd      -0.033878344
#> Garage_TypeNo_Garage   -0.036836894
#> Full_Bath               0.346115172
#> Fireplaces              0.135504921
#> Lot_ConfigCulDSac       0.081517065
#> Lot_Configfrontage     -0.066503584
#> Lot_ConfigInside       -0.023914057
```

The following plot presents how the coefficients for `Full_Bath` change with τ at the 40th λ value, where `vars=7` because `Full_Bath` is the seventh coefficient.

```
bytau.plot(qAmes1,vars=7,lambdaIndex=40)
```

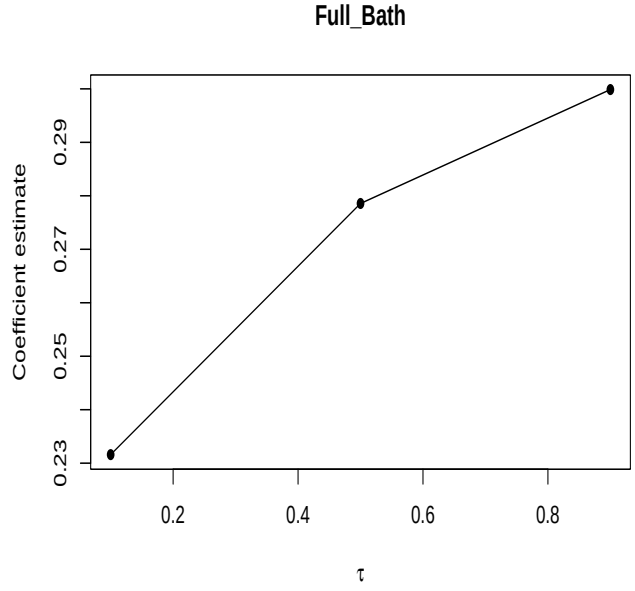


The `bytau.plot()` can also be used with an '`rq.pen.seq.cv`' object. The below code creates an '`rq.pen.seq.cv`' object using the group lasso penalty.

```
qgl <- rq.group.pen.cv(x_g,y_g,g,tau=tauVals)
```

The below code provides the plot for the coefficient of `GarageType_BuiltIn` using the one standard error rule and the λ tuning parameter is selected to be optimal across all quantiles.

```
bytau.plot(qgl,vars=7,septau=FALSE,cvmin=FALSE)
```



Other functions such as `predict()`, `coef()` and `qic.select()` work the same for a group penalty method as they do for an individual penalty, presented in the Barro data set example. The `rq.pen()`, `rq.group.pen()`, and `rq.gq.pen()` functions return '`rq.pen.seq`' objects. While, the `rq.pen.cv()`, `rq.group.pen.cv()` and `rq.gq.pen.cv()` functions return '`rq.pen.seq.cv`' objects.

5 Simulations

Alternative packages for convex penalized quantile regression that were discussed earlier are **quantreg**, **conquer**, **hrqglas**, and **hqreg**. The only one of these four that is not used by **rqPen** is **conquer**. In this section two simulations settings are presented to compare the speed and quality of the solution found by **conquer** and **rqPen** for the lasso and group lasso penalties. The maximum number of iterations for the algorithms are set to 5000 and the tolerance for convergence was set to .001. Otherwise the default options for both packages are used. Simulations were run on a Dell Latitude 7410 laptop with an Intel Core i7 vPRO processor.

5.1 Simulations for lasso penalty

Let $\mathbf{x}_i \sim N(\mathbf{0}, \mathbf{I}_p)$ the responses are generated from a model of,

$$y_i = \mathbf{x}_i^\top \boldsymbol{\beta}^* + \epsilon_i, \tag{24}$$

where $\boldsymbol{\beta}^* = (\tilde{\boldsymbol{\beta}}, \mathbf{0}_{p-10})^\top$, $\epsilon_i \sim N(0,1)$, and the errors are independent. The entries of $\tilde{\boldsymbol{\beta}}$ are generated from a standard normal distribution in each simulation. Data was generated fifty times for each combination of $n \in \{200, 2000, 20000\}$ and $p \in \{30, 100, 300\}$. The methods were fit for 50 λ values equally spaced from .05 to .25. For all methods we model $\tau = .5$.

Three algorithms are considered, `rq.pen()` with `alg="br"` and `alg="huber"`, and `conquer.reg()`. In each simulation the time it took for the algorithms to run and the value of the objective functions at the 50 different λ values were recorded. Using `rq.pen(alg="br")`, which relies on **quantreg**, was dramatically slower than the other approaches. Results, that are consistent with the literature when comparing a penalized approximation approach to commonly used linear programming algorithms (Yi and Huang, 2017; Tan et al., 2022). Thus we focus on the speed differences between the Huber approximation in **rqPen** and the convolution smoothing approach of **conquer**. Figure 1 presents a comparison of the base 10 log time of the two approaches for different values of n and p, which demonstrates that the Huber approximation approach in **rqPen**, which relies on **hqreg**, is consistently faster. To compare the accuracy of the algorithms, we evaluate a ratio of the objective function, using quantile loss, at the solution to the objective function from the BR algorithm. The BR approach does not replace the quantile loss with an approximation and thus this ratio is always greater than or equal to one for both alternative approaches. Figure 2 provides the ratios for different values of n and p. Note, the ratios also would depend on λ, but that information is not included in the figure for ease of presentation. Both approaches are providing minimized objective values very similar to the exact approach, with the Huber approximation approach tending to be a little closer.

5.2 Simulations for group lasso penalty

For group lasso let $\mathbf{x}_{ij} = (x_{ij1}, x_{ij2})^\top$, where $P(x_{ij1} = 1) = P(x_{ij2} = 1) = 1/3$ and these are disjoint events. That is it represents a single observation of a multinomial random variable with 3 categories and equal probability of belonging to any of the three categories. Let $\mathbf{x}_i = (\mathbf{x}_{i1}, \ldots, \mathbf{x}_{ig})^\top$, where the predictors are independent of each other. The data generating mechanism for y is the same as (24). Again, data was generated fifty times for each combination of $n \in \{200, 2000, 20000\}$ and $g \in \{15, 50, 150\}$. The methods were fit for 50 λ equally spaced values from .05 to .001. Again, for all methods we model $\tau = .5$.

The group lasso methods implemented in rqPen, using `rq.group.pen()`, and conquer, using `conquer.reg(penalty="group",...)` are compared in these simulations. Both approaches replace the quantile loss function with a smooth approximation. When the quantile loss is used, the optimization problem is a second order cone programming problem that can be solved using convex optimization software. See Sherwood and Li (2022) for a comparison of the Huber approximation approach and using **Rmosek** for quantile regression with a group lasso penalty. To compare the two methods time and a ratio of the rqPen and conquer

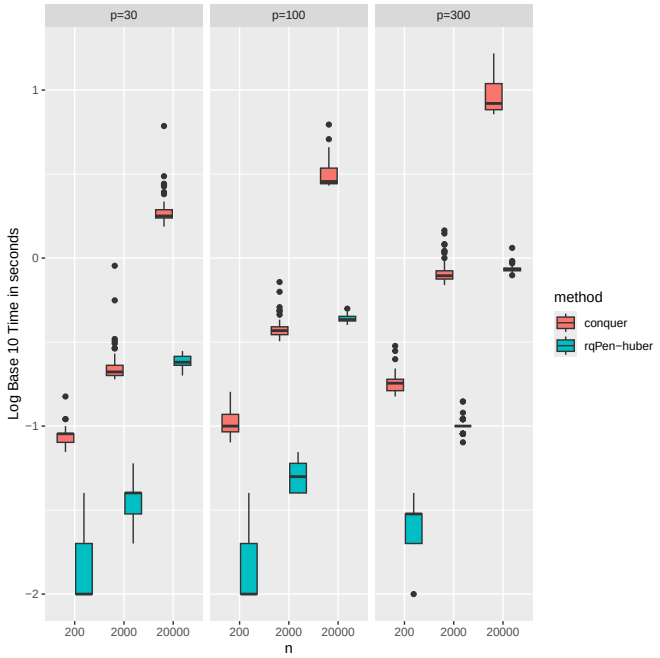


Figure 1: Log time in seconds of lasso quantile regression using conquer and the Huber approximation in rqPen.

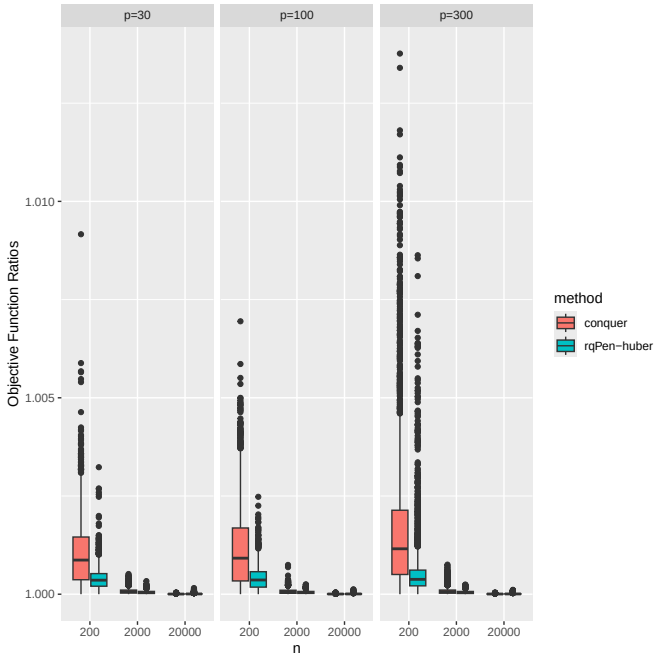


Figure 2: Ratio of quantile regression lasso objective function at the rqPen-huber and conquer solutions, where rqPen-br is the baseline.

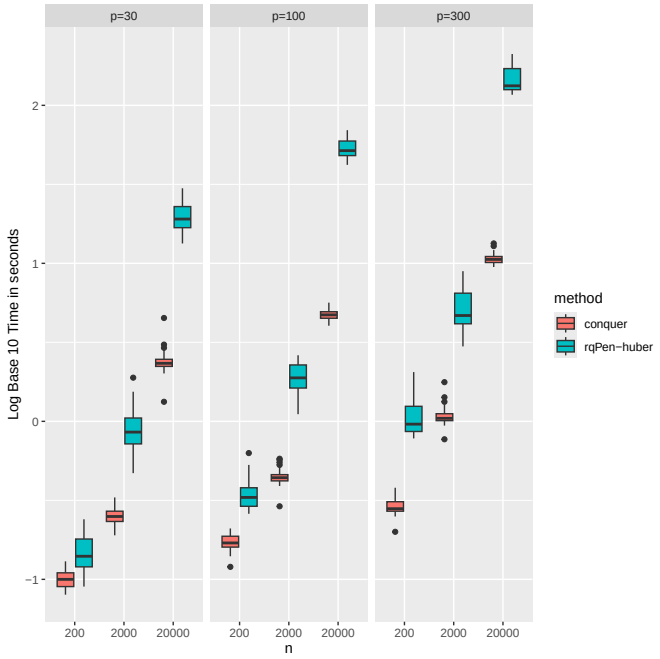


Figure 3: Log time in seconds of group lasso quantile regression using conquer and the Huber approximation in rqPen.

penalized objective functions evaluated at the solutions. Figure 3 and 4 present the time and objective function results, respectively. While, **rqPen** tends to find a solution that provides a smaller value at the objective function, the results from **conquer** are noticeably faster. Note, the implementation of group lasso in **rqPen** relies on **hrqglas**.

6 Summary

The **rqPen** package provides a penalized approach to estimating quantile regression models. The package supports the elastic-net, adaptive lasso, SCAD, and MCP penalty functions. For group predictors there are group penalty versions of the lasso, adaptive lasso, SCAD, and MCP penalties. In addition, users can use a group quantile penalty that ensures consistent variable selection across the quantiles. The package is available on CRAN.

Research in penalized quantile regression, and more generally quantile regression with large data sets, is an active area of research. The package currently does not include some recent innovations in the field. Censored quantile regression is a useful tool for survival analysis and an active area of research. Zheng et al. (2018) present a penalized quantile regression approach for estimating an interval of quantiles for censored data. While, Sze Ming Lee and Xu (2023) provides a unifying approach to estimation and inference for censored quantile regression that possibly could be extended to penalized settings. Another interesting work that provides a framework that could possibly be extended to penalized quantile regression is Volgushev et al. (2019), which proposes a method for estimating quantile regression that splits up initial analysis across multiple computers and then combines the estimates. A challenge here would be dealing with the different sparsity patterns that could arise on the different machines. Inference for penalized methods has also been an active area of research. Recently, Alexandre Belloni and Kato (2019) propose an approach where they do not approximate the quantile loss function and Yan et al. (2023) provide an approach for when the loss function is approximated with a smoothing function. Quantiles are more easily understood when the response is continuous. For users interested in estimating conditional quantiles for count data, one easy solution is to jitter the response (Machado and Silva, 2005). All of these works suggest that there is a variety of existing work that could be developed into useful software and plenty of work to be done in the future.

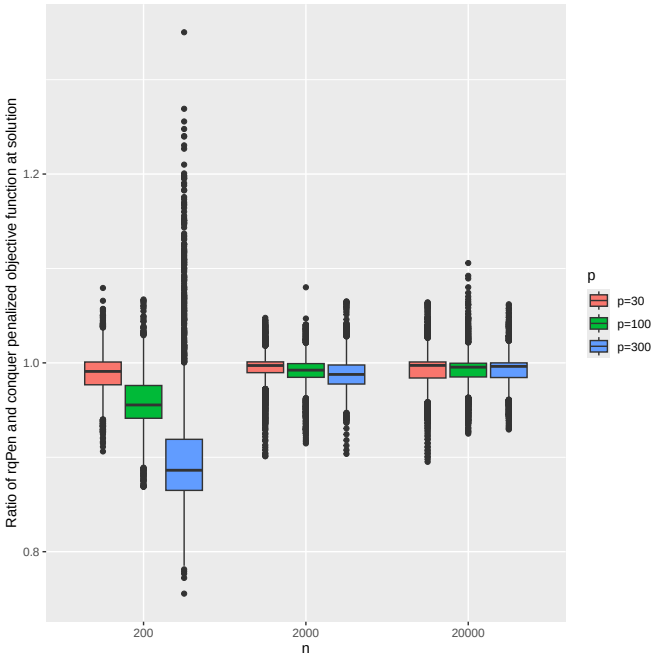


Figure 4: Ratio of quantile regression group lasso objective function at the rqPen and conquer group lasso solutions.

References

V. C. Alexandre Belloni and K. Kato. Valid post-selection inference in high-dimensional approximately sparse quantile regression models. *Journal of the American Statistical Association*, 114(526):749–758, 2019. doi: 10.1080/01621459.2018.1442339. URL https://doi.org/10.1080/01621459.2018.1442339. [p170]

I. Barrodale and F. Roberts. Solution of an overdetermined system of equations in the ℓ_1 norm. *Communications of the ACM*, 17:319–320, 1974. doi: 10.1145/355616.36102. URL https://doi.org/10.1145/355616.36102. [p155]

A. Belloni and V. Chernozhukov. L1-penalized quantile regression in high-dimensional sparse models. *Ann. Statist.*, 39(1):82–130, 2011. doi: 10.1214/10-AOS827. URL https://doi.org/10.1214/10-AOS827. [p146, 151]

P. Breheny and J. Huang. Penalized methods for bi-level variable selection. *Statistics and its Interface*, 2:369–380, 2009. doi: 10.4310/SII.2009.v2.n3.a10. URL https://doi.org/10.4310/SII.2009.v2.n3.a10. [p146, 150]

P. Breheny and J. Huang. Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection. *The Annals of Applied Statistics*, 5(1):232 – 253, 2011. doi: 10.1214/10-AOAS388. URL https://doi.org/10.1214/10-AOAS388. [p146]

P. Breheny and J. Huang. Group descent algorithms for nonconvex penalized linear and logistic regression models with grouped predictors. *Stat. Comput.*, 25:173–187, 2015. doi: 10.1007/s11222-013-9424-2. URL https://doi.org/10.1007/s11222-013-9424-2. [p146]

D. De Cock. Ames, Iowa: Alternative to the boston housing data as an end of semester regression project. *J. Stat. Educ.*, 19(3), 2011. doi: 10.1080/10691898.2011.11889627. URL https://doi.org/10.1080/10691898.2011.11889627. [p156]

J. Fan and R. Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *J. Amer. Statist. Assoc.*, 96(456):1348–1360, 2001. doi: 10.1198/016214501753382273. URL https://doi.org/10.1198/016214501753382273. [p146]

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *J. Stat. Softw.*, 33(1):1–22, 2010. doi: 10.18637/jss.v033.i01. URL https://www.jstatsoft.org/index.php/jss/article/view/v033i01. [p146]

X. He, X. Pan, K. M. Tan, and W.-X. Zhou. Smoothed quantile regression with large-scale inference. *Journal of Econometrics*, 232(2):367–388, 2023. doi: https://doi.org/10.1016/j.jeconom.2021.07.010. URL https://www.sciencedirect.com/science/article/pii/S0304407621001950. [p146]

J. Huang, P. Breheny, and S. Ma. A selective review of group selection in high-dimensional models. *Statistical Science*, 27(4):481–499, 2012. doi: 10.1214/12-STS392. URL https://doi.org/10.1214/12-STS392. [p146, 150]

P. J. Huber. Robust estimation of a location parameter. *Ann. Math. Statist.*, 35(1):73–101, 03 1964. doi: 10.1214/aoms/1177703732. URL https://doi.org/10.1214/aoms/1177703732. [p149]

K. Kato. Group lasso for high dimensional sparse quantile regression models. March 2012. URL https://arxiv.org/pdf/1103.1458. [p146]

R. Koenker and G. Bassett. Regression quantiles. *Econometrica*, 46(1):33–50, 1978. ISSN 00129682, 14680262. doi: 10.2307/1913643. URL http://www.jstor.org/stable/1913643. [p146, 147]

R. Koenker and V. D'Orey. A remark on algorithm as 229: Computing dual regression quantiles and regression rank score. *J. R. Stat. Soc. Ser. C. Appl. Stat.*, 43(2):410–414, 1994. doi: 10.2307/2986030. URL http://www.jstor.org/stable/2986030. [p146, 155, 157]

R. Koenker and J. A. F. Machado. Goodness of fit and related inference processes for quantile regression. *Journal of the American Statistical Association*, 94(448):1296–1310, 1999. doi: 10.2307/2669943. URL http://www.jstor.org/stable/2669943. [p156]

R. Koenker and I. Mizera. Convex optimization in R. *J. Stat. Softw.*, 60(5):1–23, 2014. doi: 10.18637/jss.v060.i05. URL https://www.jstatsoft.org/index.php/jss/article/view/v060i05. [p146, 150]

R. W. Koenker and V. D'Orey. Computing regression quantiles. *J. R. Stat. Soc. Ser. C. Appl. Stat.*, 36(3):383–393, 1987. doi: 10.2307/2347802. URL https://www.jstor.org/stable/2347802. [p146, 155, 157]

E. R. Lee, H. Noh, and B. U. Park. Model selection via bayesian information criterion for quantile regression models. *Journal of the American Statistical Association*, 109(505):216–229, 2014. doi: 10.1080/01621459.2013.836975. URL https://www.jstor.org/stable/24247149. [p152, 158]

S. Li and B. Sherwood. Quantile predictions for equity premium using penalized quantile regression with consistent variable selection across multiple quantiles, 2025. URL https://arxiv.org/abs/2505.16019. [p151]

J. A. F. Machado and J. M. C. S. Silva. Quantiles for counts. *Journal of the American Statistical Association*, 100(472):1226–1237, 2005. ISSN 01621459. doi: 10.1198/016214505000000330. URL http://www.jstor.org/stable/27590667. [p170]

B. Peng and L. Wang. An iterative coordinate descent algorithm for high-dimensional nonconvex penalized quantile regression. *J. Comput. Graph. Statist.*, 24(3):676–694, 2015. doi: 10.1080/10618600.2014.913516. URL https://doi.org/10.1080/10618600.2014.913516. [p146]

S. Portnoy and R. Koenker. The gaussian hare and the laplacian tortoise: computability of squared-error versus absolute-error estimators. *Statist. Sci.*, 12(4):279–300, 11 1997. doi: 10.1214/ss/1030037960. URL https://doi.org/10.1214/ss/1030037960. [p155, 157]

B. Sherwood and S. Li. Quantile regression feature selection and estimation with grouped variables using huber approximation. *Statistics and Computing*, 32(5):75, Sep 2022. doi: 10.1007/s11222-022-10135-w. URL https://doi.org/10.1007/s11222-022-10135-w. [p150, 153, 154, 168]

B. Sherwood, A. J. Molstad, and S. Singha. Asymptotic properties of concave l1-norm group penalties. *Statistics & Probability Letters*, 157:108631, 2020. doi: https://doi.org/10.1016/j.spl.2019.108631. URL https://www.sciencedirect.com/science/article/pii/S0167715219302779. [p150]

N. Simon, J. Friedman, T. Hastie, and R. Tibshirani. A sparse-group lasso. *Journal of Computational and Graphical Statistics*, 22(2):231–245, 2013. doi: 10.1080/10618600.2012.681250. URL https://doi.org/10.1080/10618600.2012.681250. [p147]

T. S. Sze Ming Lee and G. Xu. Efficient estimation for censored quantile regression. *Journal of the American Statistical Association*, 118(544):2762–2775, 2023. doi: 10.1080/01621459.2022.2078331. URL https://doi.org/10.1080/01621459.2022.2078331. [p170]

K. M. Tan, L. Wang, and W.-X. Zhou. High-dimensional quantile regression: Convolution smoothing and concave regularization. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 84(1):205–233, 2022. doi: https://doi.org/10.1111/rssb.12485. URL https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssb.12485. [p146, 168]

R. Tibshirani. Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. Ser. B. Stat. Methodol.*, 58(1):267–288, 1996. doi: 10.1111/j.2517-6161.1996.tb02080.x. URL http://www.jstor.org/stable/2346178. [p146]

R. Tibshirani et al. Strong rules for discarding predictors in lasso-type problems. *J. R. Stat. Soc. Ser. B. Stat. Methodol.*, 74(2):245–266, 2012. doi: 10.1111/j.1467-9868.2011.01004.x. URL https://doi.org/10.1111/j.1467-9868.2011.01004.x. [p153]

S. Volgushev, S.-K. Chao, and G. Cheng. Distributed inference for quantile regression processes. *The Annals of Statistics*, 47(3):1634 – 1662, 2019. doi: 10.1214/18-AOS1730. URL https://doi.org/10.1214/18-AOS1730. [p170]

L. Wang, G. Chen, and H. Li. Group scad regression analysis for microarray time course gene expression data. *Bioinformatics*, 23(12):1486–1494, 2007. doi: 10.1093/bioinformatics/btm125. URL https://doi.org/10.1093/bioinformatics/btm125. [p146]

L. Wang, Y. Wu, and R. Li. Quantile regression of analyzing heterogeneity in ultra-high dimension. *J. Amer. Statist. Assoc.*, 107(497):214–222, 2012. doi: 10.1080/01621459.2012.656014. URL https://doi.org/10.1080/01621459.2012.656014. [p146]

M. Wang, X. Kang, J. Liang, K. Wang, and Y. Wu. Heteroscedasticity identification and variable selection via multiple quantile regression. *Journal of Statistical Computation and Simulation*, 94(2):297–314, 2024. doi: 10.1080/00949655.2023.2243533. URL https://doi.org/10.1080/00949655.2023.2243533. [p146]

Y. Yan, X. Wang, and R. Zhang. Confidence intervals and hypothesis testing for high-dimensional quantile regression: Convolution smoothing and debiasing. *Journal of Machine Learning Research*, 24(245):1–49, 2023. URL http://jmlr.org/papers/v24/22-1217.html. [p170]

Y. Yang and H. Zou. A fast unified algorithm for solving group-lasso penalize learning problems. *Stat. Comput.*, 25(6):1129–1141, Nov 2015. doi: 10.1007/s11222-014-9498-5. URL https://doi.org/10.1007/s11222-014-9498-5. [p146, 150]

C. Yi and J. Huang. Semismooth newton coordinate descent algorithm for elastic-net penalized huber loss regression and quantile regression. *J. Comput. Graph. Statist.*, 26(3): 547–557, 2017. doi: 10.1080/10618600.2016.1256816. URL https://doi.org/10.1080/10618600.2016.1256816. [p146, 149, 150, 153, 154, 155, 157, 168]

M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *J. R. Stat. Soc. Ser. B. Stat. Methodol.*, 68(1):49–67, 2005. doi: 10.1111/j.1467-9868.2005.00532.x. URL https://doi.org/10.1111/j.1467-9868.2005.00532.x. [p146]

C.-H. Zhang. Nearly unbiased variable selection under minimax concave penalty. *Ann. Statist.*, 38(2):894–942, 2010. doi: 10.1214/09-AOS729. URL https://doi.org/10.1214/09-AOS729. [p146]

Q. Zheng, L. Peng, and X. He. High dimensional censored quantile regression. *The Annals of Statistics*, 46(1):308 – 343, 2018. doi: 10.1214/17-AOS1551. URL https://doi.org/10.1214/17-AOS1551. [p170]

H. Zou. The adaptive lasso and its oracle properties. *J. Amer. Statist. Assoc.*, 101(476): 1418–1429, 2006. doi: 10.1198/016214506000000735. URL https://doi.org/10.1198/016214506000000735. [p146]

H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *J. R. Stat. Soc. Ser. B. Stat. Methodol.*, 67:301–320, 2005. doi: 10.1111/j.1467-9868.2005.00503.x. URL https://doi.org/10.1111/j.1467-9868.2005.00503.x. [p146]

H. Zou and R. Li. One-step sparse estimates in nonconcave penalized likelihood models. *Ann. Statist.*, 36(4):1509–1533, 2008. doi: 10.1214/009053607000000802. URL https://doi.org/10.1214/009053607000000802. [p149]

Ben Sherwood
University of Kansas
School of Business
Lawrence, KS
ben.sherwood@ku.edu

Shaobo Li
University of Kansas
School of Business
Lawrence, KS
shaobo.li@ku.edu

Adam Maidman
University of Minnesota
School of Statistics
Minneapolis, MN
abmaidman@gmail.com

bartcs: An R Package for Bayesian Nonparametric Adjustment for Confounding

by Yeonghoon Yoo and Chanmin Kim

Abstract This article presents an overview of the bartcs R package, which employs a Bayesian additive regression trees-based method for selecting confounders. It uses a Dirichlet distribution as a common variable selection probability prior, updating both the exposure and outcome models simultaneously while fitting tree priors for each. This data-driven method determines which variables (i.e., confounders) affect both models by assigning more posterior weight to them. It supports continuous and binary exposure variables, as well as continuous outcome variables, and is written in C++ for improved computational speed. Additionally, it can take advantage of multiple threads for parallel computing if OpenMP is available on the platform.

1 Introduction

In observational studies, drawing causality often relies on the ignorability assumption (Rosenbaum and Rubin, 1983) that all confounders are included in the adjustment procedure. Alternative approaches, such as difference-in-differences (Abadie, 2005), still seek to address confounding without relying on ignorability by adopting a different set of assumptions (e.g., the parallel trends assumption). A confounder or confounding variable is a common cause that simultaneously affects both exposure and outcome (Figure 1 (a)). Two groups with different exposure levels, distinguished by the distribution of the confounding variable, also experience its impact on their respective outcome values. Therefore, to estimate the causal relationship between exposure and outcome, it is crucial to select this common cause in the data and adjust for it. In many recent applications, the number of potential confounders is often enormous, making it difficult to select the optimal set of true confounders among them. In this context, the optimal set is a confounder set with an appropriate level of uncertainty that reduces bias in estimating the final causal effect.

The main distinction between confounder selection and the traditional variable selection method is that variables that meet the ignorability assumption should be chosen. Several criteria need to be met by the selected confounders in order to reduce the bias of estimated causal effects. Among them, "disjunctive cause criterion"(VanderWeele, 2019) requires that the chosen variables be related to exposure and/or outcome. In Figure 1 (a), a confounder set $\boldsymbol{X}$ that satisfies the disjunctive cause criterion consists of variables that either affect exposure A, affect outcome Y, or simultaneously affect both A and Y. A better condition than this is "disjunctive cause criterion without instruments"(VanderWeele, 2019), which removes the variables related to exposure but not directly associated with outcome. An instrument, or instrumental variable, is a variable that influences exposure A but does not affect outcome Y. It is known to amplify bias in causal effect estimation when there is an unmeasured confounder (Myers et al., 2011). In Figure 1 (b), if a certain confounder from $\boldsymbol{X}$ is unmeasured and not adjusted for (i.e., in the presence of an unmeasured confounder), conducting adjustment for instrument Z leads to additional bias, known as "Z-bias" (Ding et al., 2017). Therefore, the best practice is to remove this instrument during the covariate adjustment process. However, manually identifying a set of confounders that meet these criteria among a large number of potential confounders is challenging.

Methods based on data and statistical models for performing such confounder selection have recently been proposed. One such method is the Bayesian adjustment for confounding (BAC) method proposed by Wang et al. (2012); Lefebvre et al. (2014), which connects exposure and outcome models through common variable inclusion indicator variables to identify confounders. Wang et al. (2015) later modified the BAC method to work with generalized linear outcome models. Wilson and Reich (2014) suggested a method based on decision theory with a similar goal, which performs well for a variety of sample sizes. In terms of selecting relevant covariates for use in propensity score, Shortreed and Ertefaie (2017) proposed the outcome-adaptive LASSO method. In addition, Häggström (2018) proposed a method for identifying the causal structure and estimating the causal effect using a probability graphical model.

Despite the advantages of the previously mentioned methods, they each have limitations. To address these shortcomings, Kim et al. (2023) proposed a novel Bayesian non-parametric model that aims to overcome these limitations. They suggested a new method that employs Bayesian additive regression trees (BART; Chipman et al. (2010)) with a shared prior for the selection probabilities, which links the exposure and outcome models. This approach allows for the flexibility and precision of

a Bayesian nonparametric model, while also identifying and integrating covariates that are related to both the exposure and outcome into the final estimator. Caron et al. (2022) similarly applied a sparsity-inducing Dirichlet prior to the selection probabilities in the outcome model of Bayesian Causal Forest, a variant of BART proposed by Hahn et al. (2020). However, while their approach focuses on inducing sparsity among variables used within the trees, our method introduces a common prior shared by both the outcome and exposure models to facilitate confounder identification. This paper introduces bartcs, a new R package developed by Yoo (2024) that implements the Bayesian additive regression trees method for confounder selection proposed by Kim et al. (2023). The package, which is written in C++ and integrated into R via Rcpp for fast computation and easy use, can be downloaded from the Comprehensive R Archive Network (CRAN) at https://cran.r-project.org/package=bartcs. Certain sections of the code referred to the BART package by Sparapani et al. (2021) under the GPL license, with modifications. In particular, the development of efficient code involved referencing the existing BART package algorithm in following aspects: 1) code related to obtaining residuals in the Bayesian backfitting process; 2) code dedicated to efficiently searching for variables eligible for splitting when proposing a splitting variable during the tree alteration process; 3) code for calculating the μ parameter value of leaf nodes; 4) code for obtaining sufficient statistics for all bottom nodes.

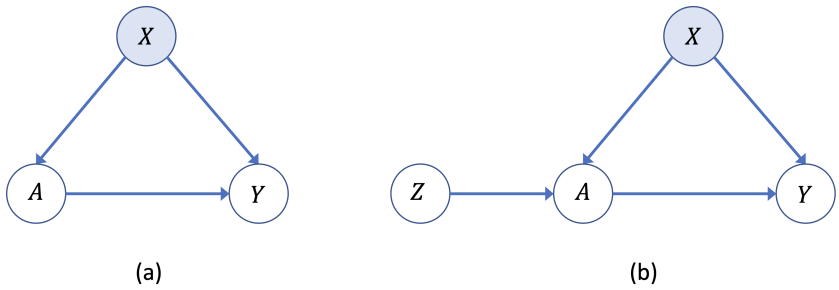


Figure 1: Directed Acyclic Graphs (DAGs): (a) the relationship between exposure A and outcome Y is confounded by covariates X; (b) Adjusting for instrument Z, which affects exposure A but is unrelated to outcome Y, may introduce additional bias if there is an unmeasured covariate in X.

In this paper, we provide an overview of the package, including installation instructions, usage examples, and a demonstration of its performance on simulated data. We also include a comparison with other existing confounder selection methods. Our aim is to provide researchers with a useful tool for identifying relevant confounders in their causal inference studies and to enable them to make more accurate causal inferences.

2 Overview of model

We first express causal estimation within a potential outcome framework (Rubin, 1974). For each unit $i = 1, \cdots, N$, the potential outcome for the i-th unit is defined as $Y_i(a)$, representing the potential value of the outcome Y_i that could be observed under the binary exposure $A_i = a \in \{0, 1\}$. Under the Stable Unit Treatment Value Assumption (SUTVA) (Rubin, 1980), the potential outcomes correspond to the observed outcome as follows: $Y_i = Y_i(A_i)$ for $A_i \in \{0, 1\}$. The target causal estimand is

$$\Delta(1, 0) = E[Y_i(1) - Y_i(0)],$$

which represents the average difference between two potential outcomes under two exposure levels 0 and 1. In the later section, we will also explain the utilization of the proposed model by extending it for cases involving continuous exposure.

However, unlike randomized trials, the exposure assignment is not randomized in observational studies, making it impossible to directly identify either $E[Y_i(1)]$ or $E[Y_i(0)]$ from observed data. With no unmeasured confounders $\boldsymbol{X}_i$, the following strong ignorable treatment assignment assumption (Rosenbaum and Rubin, 1983) holds

$$\{Y_i(1), Y_i(0)\} \perp A_i | \boldsymbol{X}_i,$$

and $0 < Pr(A_i = 1 | \boldsymbol{X}_i = \boldsymbol{x}) < 1$ for all $\boldsymbol{x}; i = 1, \cdots, N$. The first part is also known as the unconfoundedness assumption, and the second part is referred to as the positivity or overlap assumption, which states that each unit has a non-zero probability of being assigned to each treatment condition. This strong ignorable treatment assignment assumption is sufficient to identify the target causal estimand

Package	Lang.	Description
bacr (Wang et al., 2015)	R	Assume (generalized-) linear models (i.e., *parametric* models) for exposure and outcome. Supports binomial, Poisson, Gaussian exposure and outcome.
BayesPen (Wilson et al., 2015)	R	Assume linear models (i.e., *parametric* models) for exposure and outcome. Support continuous outcome.
CovSelHigh (Häggström, 2017)	R	Confounder selection performed via either Markov/Bayesian networks (model-free selection of confounders).
BART (Sparapani et al., 2021)	C++	Incorporate the Dirichlet sparse prior of Linero (2018) for variable selection in the BART outcome model. Support various outcome types (categorical, continuous, binary, survival outcome). *This does not primarily focus on confounder selection, but rather variable selection*, and this variable selection functionality is enabled by setting `sparse=TRUE` in `wbart` (continuous outcome) `pbart/lbart` (binary outcome) `mbart/mbart2` (categorical outcome) `surv.bart` (survival outcome) functions.
bcf † (Hahn et al., 2020)	C++	Specify different BART models for confounding adjustment and heterogeneous effect estimation, and regularizing the treatment effect directly. *This model lacks the ability for both variable selection and confounder selection.* Support continuous outcome.
bartCause† (Hill, 2011)	C++	Fit exposure and outcome models using the BART algorithm, producing estimates of treatment effects. *This model lacks the ability for both variable selection and confounder selection.* Support continuous and binary outcome.
bartcs (Yoo, 2024)	C++	Use BART outcome and exposure models with the common Dirichlet prior for confounder selection. Support binary and continuous exposure, and continuous outcome.

Table 1: Summary of different confounder selection methods.

$\Delta(1,0)$ (Rosenbaum and Rubin, 1983; Ding and Li, 2018; Li et al., 2023). In practice, even if the true treatment assignment mechanism satisfies the above conditions, finite observed data may have only one treatment condition value for certain combinations of $\boldsymbol{X}$. In this case, a non-overlap region occurs for that $\boldsymbol{X}$ combination, and target causal estimates in such cases inevitably rely on extrapolation dependent on the model. When non-overlap is severe, it can amplify bias in the target causal estimate. Therefore, recent research interest lies in whether estimates in such regions are provided with an appropriate level of uncertainty (Papadogeorgou and Li, 2020; Oganisian and Roy, 2020; Li et al., 2023). Any method based on outcome regression cannot provide accurate estimation in the non-overlapping region. Further discussion on this topic is available in Li et al. (2023).

Another notable aspect of this assumption is that it is untestable. Therefore, it is not possible to conduct tests based on the data to determine which confounder X satisfies the above assumption. However, confounders $\boldsymbol{X}$ that meet the criteria presented in the introduction (disjunctive cause criterion or disjunctive cause criterion without an instrument; (VanderWeele, 2019)) can be considered a minimum basis for a "proper" confounder set. With this strong ignorable treatment assignment assumption in place, we can identify the causal effect by the following equation of the observable quantities:

$$\Delta(1,0;\boldsymbol{x}) = E[Y_i|A_i = 1, \boldsymbol{X}_i = \boldsymbol{x}] - E[Y_i|A_i = 0, \boldsymbol{X}_i = \boldsymbol{x}],$$

and finally identify and estimate the target estimand $\Delta(1,0)$ by averaging over confounders $\boldsymbol{X}$. Thus, the two key tasks in estimating causal effects are identifying the confounders among a potentially large set of covariates, and determining the outcome model (i.e., $E[Y_i|A_i = a, \boldsymbol{X}_i = \boldsymbol{x}]$, $a \in \{0,1\}$) with flexibility and precision. The bartcs R package was developed to address these challenges by utilizing Bayesian additive regression trees (BART) models for confounder selection and causal effect

estimation.

2.1 Overview of BART

The BART model (Chipman et al., 2010) is an ensemble of decision trees that can be represented by the following equations:

$$\begin{aligned} y_i &= \mu_0 + f(\boldsymbol{X}_i) + \epsilon_i \\ f(\boldsymbol{X}_i) &= \sum_{t=1}^{T} g(\boldsymbol{X}_i; \mathcal{T}_t, \mathcal{M}_t), \end{aligned}$$

where ϵ_i follows a normal distribution with mean 0 and variance σ^2, and $g(\boldsymbol{X}_i; \mathcal{T}_t, \mathcal{M}_t)$ is a function that maps the tree structure and parameters to the response, for all $i = 1, \cdots, N$. We specify a BART prior on f, comprised primarily of two components: a prior concerning the complexity of each tree, $\mathcal{T}_t$, and a prior concerning its terminal nodes, $\mathcal{M}_t$. For each of T distinct trees, $\mathcal{T}_t$ represents the structure of the t-th tree and $\mathcal{M}_t = \{\mu_{t,1}, \mu_{t,2}, \cdots, \mu_{t,n_t}\}$ represents its mean parameters at the terminal nodes. Each tree has internal nodes that are split based on a "splitting variable" X_j and "splitting value" c (Figure 2). In many papers that use continuous outcome data, the outcome variable y is centered. In this case, μ_0 is set to 0. As mentioned in Sparapani et al. (2016), when the sample size is moderate or larger, centering is not necessarily required due to the flexibility of f. Proceeding with the assumption that the outcome data has also been centered, $\mu_0 = 0$ for the explanation here.

In the Markov Chain Monte Carlo (MCMC) update, Bayesian backfitting(Hastie and Tibshirani, 2000) is utilized within a Metropolis-within-Gibbs sampler. This involves fitting each tree in the ensemble sequentially, using the residual responses: $\mathbf{R}_{-t} := \mathbf{y} - \sum_{j \neq t} g(\boldsymbol{X}; \mathcal{T}_j, \mathcal{M}_j)$ where $\mathbf{R}_{-t}$ denotes unexplained outcome residuals for the t-th tree. In each iteration of the MCMC update, a new tree structure is proposed by randomly selecting one of three possible tree alterations:

> GROW: Choose a terminal node at random, and create two new terminal nodes. This process involves randomly selecting a predictor, X_j, and its associated "splitting value," c, to create the two new terminal nodes.
>
> PRUNE: Pick an internal node at random where both children are terminal nodes (known as a "singly internal node" (Kapelner and Bleich, 2016)) and remove both of its children (thus making it a terminal node).
>
> CHANGE: Select an internal node at random and modify its splitting variable and value according to the priors.

When BART was first introduced by Chipman et al. (2010), four tree alteration steps (GROW, PRUNE, CHANGE, and SWAP) were considered. However, following work by Kapelner and Bleich (2016), who proposed the `bartMachine` package for BART implementation, demonstrated that omitting the SWAP step does not significantly affect the performance in terms of tree updates or parameter mixing. Notably, since GROW and PRUNE are paired as opposite moves, and CHANGE moves are reversible through opposite-direction CHANGE moves, the detailed balance condition continues to hold without the SWAP step. Specifically, when using the grow and change alterations, a new covariate is randomly selected from a set of P available covariates as the splitting variable, according to the assumed prior. The original BART model used a uniform prior of $\{1/P, 1/P, \cdots, 1/P\}$ on the selection probabilities $\boldsymbol{s} = (s_1, s_2, \cdots, s_P)$. However, to promote sparsity, Linero (2018) proposed using a Dirichlet prior $(s_1, s_2, \cdots, s_P) \sim \mathcal{D}(\alpha/P, \cdots, \alpha/P)$. This prior specification, as outlined in Table 1, enables the variable selection functionality of the BART package. Through this, it can be utilized as a Bayesian variable selection method to choose important predictors in regression problems. Kim et al. (2023) have adapted this method for causal inference, proposing a way to select confounders. By specifying a common Dirichlet prior on the selection probabilities of the outcome and exposure models, it allows for the selection of important variables (i.e., confounders) in both models. Additionally, it is worth noting that a tree rotation proposal (Pratola, 2016) has been suggested to enable more radical mixing than the classical 'change' alteration step. This aspect is planned to be incorporated into the future updates of the bartcs package. In the following section, we will explain the specific setting of this method and the steps involved in computing the posterior distributions.

2.2 BART confounder selection

The bartcs package in R is designed for selecting confounding variables, particularly when a large number of potential confounding variables are present, and for estimating the average treatment

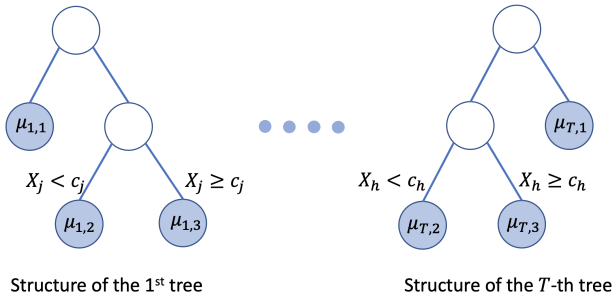


Figure 2: The tree structures consist of T trees, each with nodes represented by circles. Terminal nodes, shown in blue, have μ values. The outcome estimate $\hat{Y}$ of each observation is calculated by adding up the μ values of the terminal nodes where the observation falls within each tree. The method used to split each internal node into two different children nodes is the "splitting rule," which consists of a "splitting variable" (i.e., X_j) and a "splitting value" (i.e., c).

effect (ATE) given the chosen set of confounding variables. To accomplish this, the package uses the Bayesian additive regression trees (BART) model to specify the exposure and outcome models as follows:

$$
\begin{aligned}
P(A_i = 1) &= \Phi\left(\mu_1 + f_1(\boldsymbol{X}_i)\right) && (1)\\
f_1(\boldsymbol{X}_i) &= \sum_{t=1}^{T} g_1(\boldsymbol{X}_i; \mathcal{T}_t, \mathcal{M}_t) \\
\underline{\text{Separate Outcome Models}}: \quad Y_i | A_i = a &= \mu_2^a + f_2^a(\boldsymbol{X}_i) + \epsilon_i^a, \quad \epsilon_i^a \sim N(0, \sigma_a^2) && (2)\\
f_2^a(\boldsymbol{X}_i) &= \sum_{t=1}^{T} g_2^a(\boldsymbol{X}_i; \mathcal{T}_t^a, \mathcal{M}_t^a) \\
\underline{\text{Single Outcome Model}}: \quad Y_i &= \mu_3 + f_3(A_i, \boldsymbol{X}_i) + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma^2), && (3)\\
f_3(A_i, \boldsymbol{X}_i) &= \sum_{t=1}^{T} g_3(A_i, \boldsymbol{X}_i; \mathcal{T}_t', \mathcal{M}_t') \\
f_1 \sim \text{BART}, &\qquad f_2^a \sim \text{BART}, \qquad f_3 \sim \text{BART}
\end{aligned}
$$

for $i = 1, \cdots, N$ in Equations 1 and 3, and for $i \in \mathcal{I}_a$ where $\mathcal{I}_a$ denotes a set of units under each exposure arm $a \in \{0, 1\}$ in Equation 2. In Equation 1, $\Phi(\cdot)$ is the standard normal cumulative distribution function. Note that it is required to replace Equation 1 with $A_i = \mu_1 + f_1(\boldsymbol{X}_i) + \epsilon_i$ where $\epsilon_i \sim N(0, \tau^2)$ when considering a continuous exposure (in Section 5). As mentioned earlier, here we will proceed with the assumption that the outcome data has been centered, setting μ_1, μ_2^a, and μ_3 to 0 for the purpose of elaborating on the methodology. We incorporate a common sparsity-inducing Dirichlet prior $\boldsymbol{s}_{12} = (s_1, s_2, \cdots, s_P) \sim \mathcal{D}(\alpha/P, \cdots, \alpha/P)$ in the exposure model (Equation 1) and the outcome model (Equation 2) resulting in a conjugate update (Figure 3). Alternatively, a common Dirichlet prior is introduced on $\boldsymbol{s}_{13} = (s_0, s_1, s_2, \cdots, s_P)$ in the single outcome model (Equation 3), where s_0 represents the probability of exposure A, and the transformed form of $\boldsymbol{s}_{13}$ is incorporated as the selection probability in the exposure model. We will delve into this aspect in detail later.

If a particular covariate, X_j, is frequently used as a splitting variable in either the model for A or the model for Y, the model will assign more weight to the selection probability s_j through larger numbers of splits on X_j. This means that the selection probabilities will tend to favor covariates that have a relationship with A, Y, or both A and Y. The final confounders chosen for effect estimation in the model for Y will be those that were proposed for splitting through this prior and were accepted during the updating step of the model for Y, which will further prioritize variables that have a relationship with Y. This characteristic satisfies the "disjunctive cause criterion without instruments" in confounder selection.

Remark

The proposed method assumes that the available set of high-dimensional covariates in the dataset includes all true confounders. Therefore, it is not necessary (and indeed may be detrimental) to pre-select covariates, as doing so could inadvertently exclude true confounders. By providing all available P covariates (potential confounders) as input, the proposed method will select confounders in a data-driven manner.

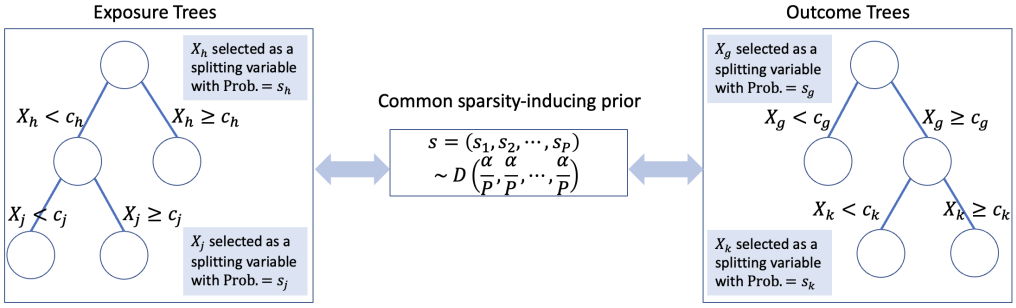


Figure 3: A shared sparsity-inducing prior for the selection probability vector connects the exposure model and outcome model, enabling the selection of the splitting variables in both models. The selection probability vector is updated based on the number of splitting variables used to describe each tree.

However, if there are unmeasured confounders (i.e., if not all true confounders are included in the set of potential confounders), our method cannot entirely avoid biases (e.g., Z-bias, M-bias, etc.). If you suspect that important confounders are unmeasured (thus missing from the potential confounders set), we recommend excluding, in advance, variables that are suspected of being instrumental variables (which affect only the exposure but not the outcome), as well as potential colliders (variables that are believed to have no direct effects on both exposure and outcome), before applying the proposed method.

Separate outcome models

For a binary exposure, we separate the outcome model into two distinct sub-models, in order to align the dimensions of the covariates in both the exposure and outcome models (note that the outcome model includes exposure A as an additional covariate if a single outcome model is specified). For Equations 1 and 2, a sparsity-inducing prior is applied to $\boldsymbol{s}_{12} = (s_1, s_2, \cdots, s_P)$, which is shared among three models: one for exposure and two for outcomes.

We use "Bayesian backfitting"(Hastie and Tibshirani, 2000) to obtain posterior samples for the exposure and outcome models. For the exposure model, this involves a Metropolis-within-Gibbs sampler, where we fit each tree $\mathcal{T}_t$ iteratively using residual responses :

$$R_{i,-t} = Z_i - \sum_{j \neq t} g_1(\boldsymbol{X}_i; \mathcal{T}_j, \mathcal{M}_j)$$

for $i = 1, \cdots, N$ where Z_i is a latent variable for the binary exposure constructed based on Albert and Chib (1993) as follows

$$Z_i \sim \begin{cases} N\left(f_1(\boldsymbol{X}_i), 1\right) I_{(Z_i > 0)} & \text{for } A_i = 1; \\ N\left(f_1(\boldsymbol{X}_i), 1\right) I_{(Z_i \leq 0)} & \text{for } A_i = 0. \end{cases}$$

Note that the variance parameter (σ^2) is assigned a value of 1 as a result of the construction of the latent variable. For each tree $\mathcal{T}_t$ for the exposure model, we propose a new tree structure $\mathcal{T}_t$ from the full conditional $[\mathcal{T}_t | R_{1,-t}, \cdots, R_{n,-t}]$ (i.e., grow, prune or change alterations), and update the parameters within the tree through the full conditional $[\mathcal{M}_t | \mathcal{T}_t, R_{1,-t}, \cdots, R_{n,-t}]$.

To draw samples for $\mathcal{M}_t$, we assume a prior $\mu \sim N(\mu_\mu / T, \sigma_\mu^2)$ on each of the leaf parameters $\mathcal{M}_t = \{\mu_1, \mu_2, \cdots, \mu_{t_b}\}$, where t_b is the number of terminal nodes in tree $\mathcal{T}_t$. The range center of latent variable Z_i's is set as the mean, μ_μ, and σ_μ^2 is empirically determined to satisfy $T\mu_\mu - 2\sqrt{T}\sigma_\mu = Z_{\min}$ and $T\mu_\mu + 2\sqrt{T}\sigma_\mu = Z_{\max}$ where $Z_{\min}$ and $Z_{\max}$ represent the minimum and maximum values of Z_i's (Kapelner and Bleich, 2016).

We generate a sample μ_η from the posterior distribution for the η-th terminal node in tree $\mathcal{T}_t$ by using the following equation:

$$\mu_\eta \sim N\left(\frac{1}{1/\sigma_\mu^2 + n_\eta/\sigma^2}\left(\frac{\mu_\mu/T}{\sigma_\mu^2} + \frac{\sum_{i \in \mathcal{O}_\eta} R_{i,-t}}{\sigma^2}\right), \left(\frac{1}{\sigma_\mu^2} + \frac{n_\eta}{\sigma^2}\right)^{-1}\right),$$

where $\mathcal{O}_\eta$ and n_η correspond to the observation indices and the number of observations, respectively, for the η-th terminal node. In our implementation, we set the μ_μ value to 0, and consequently, the

bartcs package is constructed to shift the Y and Z variables to have a mean value of 0.

For separate outcome models, we also perform a backfitting step to draw samples from $P(\mathcal{T}_1^a, \cdots, \mathcal{T}_T^a, \mathcal{M}_1^a, \cdots, \mathcal{M}_T^a, \sigma_a^2 | \boldsymbol{D})$ for each $A = a \in \{0,1\}$ by computing the residual responses iteratively as follows:

$$H_{i,-t}^a = y_i - \sum_{j \neq t} g_2^a(\boldsymbol{X}_i; \mathcal{T}_j^a, \mathcal{M}_j^a) \quad \text{for } i \in \mathcal{I}_a,$$

where $\mathcal{I}_a$ represents the set of observations corresponding to $A = a \in \{0,1\}$. Afterwards, a process is undertaken to update each tree based on $[\mathcal{T}_t^a | H_{\cdot,-t}^a, \sigma_a^2]$ and the parameters of its corresponding terminal nodes from $[\mathcal{M}_t^a | \mathcal{T}_t^a, H_{\cdot,-t}, \sigma_a^2]$ for each exposure level $a \in \{0,1\}$. This process is analogous to the one described earlier for the exposure model.

For each MCMC iteration, once all the tree structures and corresponding parameters have been updated, we proceed to update the variance parameter (σ_a^2 in each outcome model (2)) using the Gibbs sampler. This is achieved by sampling from the inverse gamma distribution given by:

$$\sigma_a^2 \sim \text{Inv.Gamma}\left(a_\sigma + \frac{|\mathcal{I}_a|}{2}, b_\sigma + \frac{1}{2}\left\{\sum_{i \in \mathcal{I}_a}\left(y_i - \sum_{t=1}^{T} g_2^a(\boldsymbol{X}_i; \mathcal{T}_t^a, \mathcal{M}_t^a)\right)\right\}\right),$$

where $a_\sigma = b_\sigma = 3$.

Next, we update the parameter α in the prior distribution of selection probabilities $\boldsymbol{s}_{12} \sim \mathcal{D}(\alpha/P, \cdots, \alpha/P)$ based on a prior of the form $\alpha/(\alpha + P) \sim \text{Beta}(a_0, b_0)$, where $a_0 = 0.5$ and $b_0 = 1$ (Linero, 2018). The Metropolis-Hastings algorithm is then used to update the parameter. To delve deeper into the α prior in high-dimensional data, refer to Sparapani et al. (2021). Finally, we update $\boldsymbol{s}_{12}$ using a conjugate sampling update as follows: $\boldsymbol{s}_{12} \sim \mathcal{D}(\alpha/P + n_{1,1} + n_{1,21} + n_{1,20}, \cdots, \alpha/P + n_{P,1} + n_{P,21} + n_{P,20})$, where $n_{j,21}$ and $n_{j,20}$ represent the numbers of splits on the confounder X_j in two separate outcome models, and $n_{j,1}$ represents the number of splits on X_j in the exposure model.

The posterior computation process for the approach employing the separate outcome models strategy is outlined in Algorithm 1 through pseudocode.

Single outcome model

Using two separate outcome models for two exposure levels, as outlined in Hill (2011) and Hahn et al. (2020), can result in biased estimates if there is a lack of common support in confounders. While a single outcome model can be a viable alternative, it can be challenging to apply a shared sparsity-inducing prior to $\boldsymbol{s}_{12} = (s_1, s_2, \cdots, s_P)$ due to differences in covariate dimensions between the exposure and outcome models. Let $\boldsymbol{s}_{13} = (s_0, s_1, s_2, \cdots, s_P)$ represent the selection probabilities, with s_0 denoting the probability of exposure A used in the outcome model. To apply this vector to the exposure model, $\boldsymbol{s}_{13}$ is transformed to $\boldsymbol{s}'_{13} = (s_1/(1-s_0), s_2/(1-s_0), \cdots, s_P/(1-s_0))$. Then, updating $\boldsymbol{s}_{13}$ is based on the following equation (likelihood $\times$ prior):

$$Q = (1-s_0)^{-n_{\cdot,1}} s_0^{n_{0,3}+\alpha/P-1} s_1^{n_{1,3}+n_{1,1}+\alpha/P-1} \cdots s_P^{n_{P,3}+n_{P,1}+\alpha/P-1},$$

using the Metropolis-Hastings algorithm, where $n_{j,3}$ represents the number of splits on the confounder X_j in the single outcome model and $n_{\cdot,1} = \sum_{j=1}^{P} n_{j,1}$. The proposal distribution for $\boldsymbol{s}_{13}$ is designed to follow the full conditional in the separate outcome models, $\mathcal{D}(n_{0,3} + c + \alpha/P, n_{1,1} + n_{1,3} + \alpha/P, n_{2,1} + n_{2,3} + \alpha/P, \cdots, n_{P,1} + n_{P,3} + \alpha/P)$, and a positive value c is added to prevent proposals for infrequent exposure. In the bartcs package, the value of c is set to the number of splits on A in the outcome model ($n_{0,3}$). This is to ensure sufficient proposal probability for exposure. Whether the proposed variables are actually accepted is determined through the M-H step, so based on our experience, this setting does not significantly affect the performance of confounder selection.

All posterior computation steps are identical to the separate outcome models method, except for the difference that there is only one outcome model. Therefore, updates for the trees and parameters of the outcome model are based on one $[\mathcal{T}'_t | \boldsymbol{H}_{\cdot,-t}, \sigma^2]$ and one $[\mathcal{M}'_t | \mathcal{T}'_t, \boldsymbol{H}_{\cdot,-t}, \sigma^2]$ for each tree t. Subsequently, sampling for σ^2 is carried out based on the following inverse gamma distribution:

$$\sigma^2 \sim \text{Inv.Gamma}\left(a_\sigma + \frac{N}{2}, b_\sigma + \frac{1}{2}\left\{\sum_{i=1}^{N}\left(Y_i - \sum_{t=1}^{T} g_3(\boldsymbol{X}_i; \mathcal{T}'_t, \mathcal{M}'_t)\right)\right\}\right),$$

where $a_\sigma = b_\sigma = 3$. The posterior computation process for the approach employing the single outcome model strategy is outlined in Algorithm 2 through pseudocode.

Given the M set of posterior samples for BART parameters, the causal effect estimand $\Delta(1,0)$ can be estimated using either the separate models or the single model. For the separate outcome models,

Algorithm 1 Posterior Computation (Separate Outcome Models)

Require: Samples from the previous iteration $(\mathcal{T}_t, \mathcal{T}_t^1, \mathcal{T}_t^0, \mathcal{M}_t, \mathcal{M}_t^1, \mathcal{M}_t^0)$ for $t = 1, \ldots, T$ and $(\sigma_1^2, \sigma_0^2, \boldsymbol{s}_{12})$, and data $(y_i, A_i, \boldsymbol{X}_i)$ for $i = 1, \cdots, N$

for $r = 1, \ldots, M$ iteration **do**

  for $i = 1, \ldots, N$ **do**

$$Z_i \sim \begin{cases} N\left(\sum_{t=1}^T g_1(\boldsymbol{X}_i; \mathcal{T}_t, \mathcal{M}_t), 1\right) I_{(Z_i > 0)} & \text{for } A_i = 1; \\ N\left(\sum_{t=1}^T g_1(\boldsymbol{X}_i; \mathcal{T}_t, \mathcal{M}_t), 1\right) I_{(Z_i \leq 0)} & \text{for } A_i = 0 \end{cases}$$ ▷ latent exposure variable

  end for

  for $j = 1, \ldots, T$ **do**

    for $i = 1, \ldots, N$ **do**

      $R_{i,-j}^{(r)} = Z_i - \sum_{t \neq j} g_1(\boldsymbol{X}_i; \mathcal{T}_t, \mathcal{M}_t)$ ▷ residual of the exposure model

      $H_{i,-j}^{1,(r)} = y_i - \sum_{t \neq j} g_2^1(\boldsymbol{X}_i; \mathcal{T}_t^1, \mathcal{M}_t^1)$ ▷ residual of the outcome model for $i \in \mathcal{I}_1$

      $H_{i,-j}^{0,(r)} = y_i - \sum_{t \neq j} g_2^0(\boldsymbol{X}_i; \mathcal{T}_t^0, \mathcal{M}_t^0)$ ▷ residual of the outcome model for $i \in \mathcal{I}_0$

    end for

    $\mathcal{T}_j^{(r)} \sim [\mathcal{T}_j | R_{1,-j}^{(r)}, \cdots, R_{N,-j}^{(r)}, 1]$ ▷ based on one of the three acceptance ratios

    $\mathcal{T}_j^{1,(r)} \sim [\mathcal{T}_j^1 | \boldsymbol{H}_{\cdot,-j}^{1,(r)}, \sigma_1^2]$ ▷ based on one of the three acceptance ratios

    $\mathcal{T}_j^{0,(r)} \sim [\mathcal{T}_j^0 | \boldsymbol{H}_{\cdot,-j}^{0,(r)}, \sigma_0^2]$ ▷ based on one of the three acceptance ratios

    $\mathcal{M}_j^{(r)} \sim [\mathcal{M}_j | \mathcal{T}_j^{(r)}, R_{1,-j}^{(r)}, \cdots, R_{N,-j}^{(r)}, 1]$

    $\mathcal{M}_j^{1,(r)} \sim [\mathcal{M}_j^1 | \mathcal{T}_j^{1,(r)}, \boldsymbol{H}_{\cdot,-j}^{1,(r)}, \sigma_1^2]$

    $\mathcal{M}_j^{0,(r)} \sim [\mathcal{M}_j^0 | \mathcal{T}_j^{0,(r)}, \boldsymbol{H}_{\cdot,-j}^{0,(r)}, \sigma_0^2]$

    where, for each $a \in \{0, 1\}$, $\boldsymbol{H}_{\cdot,-j}^{a,(r)}$ denotes $\{H_{i,-j}^{a,(r)} | i \in \mathcal{I}_a\}$

  end for

$$\sigma_1^2 \sim \text{Inv.Gamma}\left(a_\sigma + \frac{|\mathcal{I}_1|}{2}, b_\sigma + \frac{1}{2}\left\{\sum_{i \in \mathcal{I}_1}\left(y_i - \sum_{t=1}^T g_2^1(\boldsymbol{X}_i; \mathcal{T}_t^1, \mathcal{M}_t^1)\right)\right\}\right)$$

$$\sigma_0^2 \sim \text{Inv.Gamma}\left(a_\sigma + \frac{|\mathcal{I}_0|}{2}, b_\sigma + \frac{1}{2}\left\{\sum_{i \in \mathcal{I}_0}\left(y_i - \sum_{t=1}^T g_2^0(\boldsymbol{X}_i; \mathcal{T}_t^0, \mathcal{M}_t^0)\right)\right\}\right)$$

  Update $\boldsymbol{s}_{12}^{(r)}$ via the Gibbs algorithm:

$$\boldsymbol{s}_{12}^{(r)} \sim \mathcal{D}\left(n_{1,1} + n_{1,21} + n_{1,20} + \alpha/P, \cdots, n_{P,1} + n_{P,21} + n_{p,20} + \alpha/P\right)$$

end for

Algorithm 2 Posterior Computation (Single Outcome Model)

Require: Samples from the previous iteration $(\mathcal{T}_t, \mathcal{T}_t', \mathcal{M}_t, \mathcal{M}_t')$ for $t = 1, \ldots, T$ and $(\sigma^2, \boldsymbol{s}_{13})$, and data $(y_i, A_i, \boldsymbol{X}_i)$ for $i = 1, \cdots, N$

for $r = 1, \ldots, M$ iteration **do**

  for $i = 1, \ldots, N$ **do**

$$Z_i \sim \begin{cases} N\left(\sum_{t=1}^{T} g_1(\boldsymbol{X}_i; \mathcal{T}_t, \mathcal{M}_t), 1\right) I_{(Z_i > 0)} & \text{for } A_i = 1; \\ N\left(\sum_{t=1}^{T} g_1(\boldsymbol{X}_i; \mathcal{T}_t, \mathcal{M}_t), 1\right) I_{(Z_i \leq 0)} & \text{for } A_i = 0 \end{cases} \quad \triangleright \text{ latent exposure variable}$$

  end for

  for $j = 1, \ldots, T$ **do**

    for $i = 1, \ldots, N$ **do**

      $R_{i,-j}^{(r)} = Z_i - \sum_{t \neq j} g_1(\boldsymbol{X}_i; \mathcal{T}_t, \mathcal{M}_t)$ ▷ residual of the exposure model

      $H_{i,-j}^{(r)} = y_i - \sum_{t \neq j} g_3(\boldsymbol{X}_i; \mathcal{T}_t', \mathcal{M}_t')$ ▷ residual of the outcome model

    end for

    $\mathcal{T}_j^{(r)} \sim [\mathcal{T}_j | R_{1,-j}^{(r)}, \cdots, R_{N,-j}^{(r)}, 1]$ ▷ based on one of the three acceptance ratios

    $\mathcal{T}_j'^{,(r)} \sim [\mathcal{T}_j' | H_{1,-j}^{(r)}, \cdots, H_{N,-j}^{(r)}, \sigma^2]$ ▷ based on one of the three acceptance ratios

    $\mathcal{M}_j^{(r)} \sim [\mathcal{M}_j | \mathcal{T}_j^{(r)}, R_{1,-j}^{(r)}, \cdots, R_{N,-j}^{(r)}, 1]$

    $\mathcal{M}_j'^{,(r)} \sim [\mathcal{M}_j' | \mathcal{T}_j'^{,(r)}, H_{1,-j}^{(r)}, \cdots, H_{N,-j}^{(r)}, \sigma^2]$

  end for

$$(\sigma^2)^{(r)} \sim \text{Inv.Gamma}\left(a_\sigma + \frac{N}{2}, b_\sigma + \frac{1}{2}\left\{\sum_{i=1}^{N}\left(y_i - \sum_{t=1}^{T} g_3(\boldsymbol{X}_i; \mathcal{T}_t'^{,(r)}, \mathcal{M}_t'^{,(r)})\right)\right\}\right)$$

  Update $\boldsymbol{s}_{13}^{(r)}$ based on the M-H algorithm:

$$\text{Proposal:} \quad \boldsymbol{s}_{13}^{(r)} \sim \mathcal{D}\left(n_{0,3} + c + \alpha/P, n_{1,1} + n_{1,3} + \alpha/P, \cdots, n_{P,1} + n_{P,3} + \alpha/P\right)$$

$$\text{Acceptance Ratio:} \quad P_{\text{AR}}(\boldsymbol{s}_{13} \to \boldsymbol{s}_{13}^{(r)}) = \min\left\{1, \left(\frac{1 - \sum_{j=1}^{P} s_j}{1 - \sum_{j=1}^{P} s_j^{(r)}}\right)^{\sum_{j=1}^{J} n_{j,3}}\right\}$$

end for

the estimate is obtained by

$$\hat{\Delta}(1,0) = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{M}\sum_{m=1}^{M}\left\{f_2^{1,(m)}(\boldsymbol{X}_i) - f_2^{0,(m)}(\boldsymbol{X}_i)\right\}\right],$$

where $f_2^{a,(m)}$ is the m-th posterior samples for $A = a \in \{0,1\}$. For the single outcome model, the estimate is obtained by

$$\hat{\Delta}(1,0) = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{M}\sum_{m=1}^{M}\left\{f_3^{(m)}(1,\boldsymbol{X}_i) - f_3^{(m)}(0,\boldsymbol{X}_i)\right\}\right],$$

where $f_3^{(m)}$ is the m-th posterior samples.

3 Simulated example

The bartcs R package makes it easy to implement the confounder selection process described in the previous section. It includes two main functions, `separate_bart()` for the separate outcome models and `single_bart()` for the single outcome model. The package not only offers a summary of the estimated causal effects but also includes visualizations of posterior inclusion probabilities and convergence.

The bartcs package offers multi-threading support through Open Multi-Processing (OpenMP), an API for shared memory parallel programming that manages thread creation, management, and synchronization for efficient data and computation division among different threads. This allows bartcs to specify intensive computations as parallel regions, leading to improved computational efficiency through parallel computing.

The bartcs package is available under the general public license (GPL $\geq$ 3) from the Comprehensive R Archive Network (CRAN) at https://cran.r-project.org/package=bartcs and can be installed and loaded into the current R session as follows:

```
install.packages("bartcs", dependencies=TRUE)
library("bartcs")
```

We will showcase the practical usage of the features in the bartcs package using simulated examples and the Infant Health and Development Program (IHDP) data.

As a simple example of the bartcs package, we use a simulated dataset from Scenario 1 in Kim et al. (2023) to illustrate its features. The data-generating model incorporates both the non-linear propensity score and outcome models, and serves to evaluate the ability to detect 5 true confounding variables out of a huge set of possible covariates, along with the precision of the model's estimation. The dataset consists of 300 observations with 100 potential confounders ($X_1 - X_{100}$), each generated from a normal distribution with mean 0 and variance 1. Of the 100 possible confounders, $X_1 - X_5$ are true confounders. The outcome model includes the five true confounders and two additional predictors, X_6 and X_7 as follows:

$$\begin{aligned} P(A_i = 1) &= \Phi(0.5 + h_1(X_{i,1}) + h_2(X_{i,2}) - 0.5|X_{i,3} - 1| + 1.5X_{i,4}X_{i,5}) \\ Y_i &= f(\boldsymbol{X}_i) + \epsilon_i, \quad \epsilon_i \sim N(0, 0.3^2) \\ f(\boldsymbol{X}_i) &= h_1(X_{i,1}) + 1.5h_2(X_{i,2}) - A_i + 2|X_{i,3} + 1| + 2X_{i,4} + \exp(0.5X_{i,5}) \\ &\quad -0.5A_i|X_{i,6}| - A_i|X_{i,7} + 1| \end{aligned}$$

where $h_1(x) = (-1)^{I(x<0)}$ and $h_2(x) = (-1)^{I(x\geq 0)}$ for $i = 1, \cdots, 300$. The data was generated with the following code:

```
set.seed(42)
N <- 300
P <- 100
cov <- list()
for (i in 1:P) {
     cov[[i]] <- rnorm(N, 0, 1)
   }
X <- do.call(cbind, cov)
h1 <- ifelse(X[, 1] < 0, 1, -1)
h2 <- ifelse(X[, 2] < 0, -1, 1)
```

```
prob <- pnorm(0.5 + h1 + h2 - 0.5 * abs(X[, 3] - 1) + 1.5 * X[, 4] * X[, 5])
Trt <- rbinom(N, 1, prob)
mu1 <- 1 * h1 + 1.5 * h2 - 1 + 2 * abs(X[, 3] + 1) + 2 * X[, 4] + exp(0.5 * X[, 5]) -
     0.5 * 1 * abs(X[, 6]) - 1 * 1 * abs(X[, 7] + 1)
mu0 <- 1 * h1 + 1.5 * h2 - 0 + 2 * abs(X[, 3] + 1) + 2 * X[, 4] + exp(0.5 * X[, 5]) -
     0.5 * 0 * abs(X[, 6]) - 1 * 0 * abs(X[, 7] + 1)
Y1 <- rnorm(N, mu1, 0.3)
Y0 <- rnorm(N, mu0, 0.3)
Y <- Trt * Y1 + (1 - Trt) * Y0
```

For users' convenience, the same data can be generated using the built-in function `synthetic_data( )` from the bartcs package with the following arguments settings: `N` = 300 (sample size), `P` = 100 (number of potential confounders), and `seed` = 42 (seed number).

Examining the standardized mean differences (SMD) of the (potential) confounders generated through the data generating process above, the following observations can be made. The standardized mean differences (SMD) values presented below were computed using the tableone R package (Yoshida and Bartel, 2022), which can be installed from CRAN.

```
library("tableone")
Xdata <- as.data.frame(cbind(Trt,X))
names(Xdata) <- c("Trt", paste0(rep("X", 100),1:100))
Table <- CreateTableOne(vars = paste0(rep("X", 12),1:12), strata = "Trt",
     data = Xdata, test = FALSE)
print(Table, smd = TRUE)
                  Stratified by Trt
                   0            1            SMD
  n                  164          136
  X1 (mean (SD))    0.28 (0.96) -0.39 (0.90)  0.718
  X2 (mean (SD))   -0.25 (0.99)  0.24 (0.92)  0.517
  X3 (mean (SD))   -0.14 (1.02)  0.03 (0.90)  0.178
  X4 (mean (SD))    0.06 (1.08) -0.07 (1.04)  0.118
  X5 (mean (SD))   -0.08 (0.86)  0.01 (1.05)  0.091
  X6 (mean (SD))   -0.03 (1.06)  0.15 (0.98)  0.177
  X7 (mean (SD))   -0.04 (1.03)  0.01 (0.94)  0.050
  X8 (mean (SD))   -0.11 (0.99)  0.20 (1.00)  0.312
  X9 (mean (SD))    0.07 (1.04)  0.05 (1.02)  0.017
  X10 (mean (SD)) -0.04 (1.13) -0.13 (0.96)  0.087
  X11 (mean (SD))   0.05 (1.02) -0.12 (0.98)  0.169
  X12 (mean (SD))   0.13 (1.01) -0.23 (0.99)  0.363
```

When looking at the results for the first 12 X variables, it is noted that for true confounders X_1 and X_2, SMD values greater than 0.1, indicative of inadequate covariate balance between the groups, are observed. Similar lack of covariate balance between the groups is also noticed for X_3 and X_4. However, due to randomness, differences between the groups are observed for some covariates other than the true confounders. In this simulation scenario, with the partial presence of the signal from some covariates other than true confounders, the goal is to assess the performance of the model under consideration.

With a generated data set, we fit the BART confounder selection model (the separate outcome models) using `separate_bart()`.

```
library("bartcs")
separate_fit <- separate_bart(
     Y = Y, trt = Trt, X = X, num_tree = 200, num_chain = 4,
     num_burn_in = 10000, num_thin = 5, num_post_sample = 2000
   )
```

The following are the main arguments used in the `separate_bart()` function call:

- `Y` represents a vector of observed outcome values.
- `trt` denotes a vector of exposure(treatment) values, which is binary. Binary treatment values need to be either 0 or 1. Continuous exposure values can be handled in `single_bart()` function.
- `X` is a data frame of potential confounders.

The following are the remaining settings for the fit: 4 MCMC chains (`num_chain`) with 200 trees (`num_tree`) are used. Each MCMC chain runs 20000 iterations, with 10000 burn-in iterations (`num_burn_in`) and a thinning factor of 5 (`num_thin`). There are other optional arguments available for hyper-parameter settings with the following default values:

- $\alpha = 0.95$ (alpha) and $\beta = 2$ (beta): these govern the probability that a node at depth d is nonterminal as follows
$$\alpha(1+d)^{-\beta}.$$
- $\nu = 3$ (nu) and $q = 0.95$ (q): to set a conjugate prior for the variance σ^2 with $\sigma^2 \sim \nu\lambda/\chi^2_\nu$, we use the following equation to determine the values $P(\sigma < \hat{\sigma}) = q$, where $\hat{\sigma}$ represents the residual standard deviation obtained from a linear regression of Y on X.
- $P_{\text{GROW}} = 0.28$, $P_{\text{PRUNE}} = 0.28$, $P_{\text{CHANGE}} = 0.44$ (step_prob = c(0.28, 0.28, 0.44)): probabilities of three tree alteration steps.
- dir_alpha = 5: this is an initial value for hyperparameter α in the sparsity inducing Dirichlet prior $\mathcal{D}(\alpha/P, \alpha/P, \cdots, \alpha/P)$.

```
separate_fit

`bartcs` fit by `separate_bart()`

           mean       2.5%      97.5%
SATE -2.2851546 -2.6022894 -1.9692134
Y1    0.7195622  0.4663024  0.9833689
Y0    3.0047169  2.8116436  3.1946016
```

The separate_bart() returns a S3 bartcs object. A bartcs object includes the posterior means and 95% credible intervals for the sample average treatment effect ($SATE$), and the potential outcomes $Y(1)$ and $Y(0)$. It is important to note that the true values for the $SATE$, $E[Y(1)]$, and $E[Y(0)]$ are -2.55, 0.64, and 3.19 respectively, and the 95% credible intervals produced by the separate_bart() function include these values.

For a more in-depth understanding of the output, the summary() function can be used. It provides details regarding the treatment values, tree structure, MCMC chain, and outcomes for each of the chains.

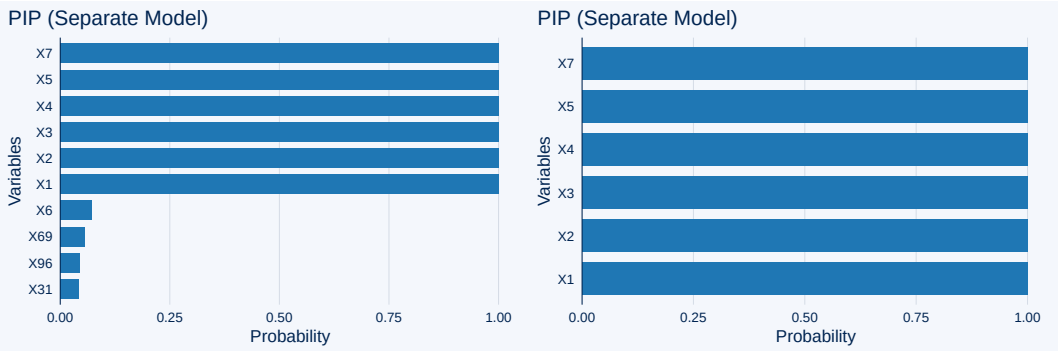
```
summary(separate_fit)

`bartcs` fit by `separate_bart()`

Treatment Value
  Treated group    :      1
  Control group    :      0

Tree Parameters
  Number of Tree   :    200               Value  of alpha    :   0.95
  Prob.  of Grow   :   0.28               Value  of beta     :      2
  Prob.  of Prune  :   0.28               Value  of nu       :      3
  Prob.  of Change :   0.44               Value  of q        :   0.95

Chain Parameters
  Number of Chains :      4               Number of burn-in  :  10000
  Number of Iter   :  20000               Number of thinning :      5
  Number of Sample :   2000

Outcome
 estimand chain       2.5%         1Q       mean     median         3Q      97.5%
     SATE     1 -2.6070044 -2.3892357 -2.2830389 -2.2800555 -2.1765359 -1.9757766
     SATE     2 -2.6013548 -2.4017854 -2.2877997 -2.2877071 -2.1798863 -1.9611401
     SATE     3 -2.5961329 -2.3952700 -2.2794876 -2.2793208 -2.1609143 -1.9644475
     SATE     4 -2.6090523 -2.4001084 -2.2902924 -2.2923171 -2.1812900 -1.9761443
     SATE   agg -2.6022894 -2.3965077 -2.2851546 -2.2842764 -2.1748201 -1.9692134
       Y1     1  0.4705203  0.6322748  0.7174467  0.7174147  0.8027479  0.9668359
       Y1     2  0.4707973  0.6305094  0.7223111  0.7213076  0.8153911  0.9851455
       Y1     3  0.4653391  0.6277828  0.7190511  0.7194586  0.8080547  0.9804701
       Y1     4  0.4614500  0.6273396  0.7194400  0.7175295  0.8087480  0.9920899
       Y1   agg  0.4663024  0.6292846  0.7195622  0.7185828  0.8087121  0.9833689
       Y0     1  2.8082437  2.9361088  3.0004857  2.9998629  3.0664869  3.1897135
       Y0     2  2.8189069  2.9442181  3.0101107  3.0107896  3.0778268  3.2013420
       Y0     3  2.8002284  2.9362280  2.9985387  2.9972708  3.0646989  3.1920314
```

```
Y0     4  2.8210957  2.9427012  3.0097324  3.0133450  3.0772579  3.1960383
Y0   agg  2.8116436  2.9404458  3.0047169  3.0053406  3.0713420  3.1946016
```

For each estimand category, there are five results (rows) that represent the output from each of the 4 MCMC chains and an aggregated output.

For visualization purposes, there are two options available as S3 methods for the bartcs object. The first option is the posterior inclusion probability (PIP) plot. PIP is the probability that a variable is used as a splitting variable, and can be interpreted as the importance of a variable. The `inclusion_plot()` function is a wrapper for the `bar_chart()` function from the ggcharts package (Neitmann et al., 2020), allowing the use of its arguments to customize the plot. The recommended arguments to use are `top_n` and `threshold`.

```
plot(separate_fit, method = "pip", top_n = 10)
plot(separate_fit, method = "pip", threshold = 0.5)
```



(a) Plotting PIP with `top_n` argument **(b)** Plotting PIP with `threshold` argument

Figure 4: Posterior inclusion probability (PIP) plots

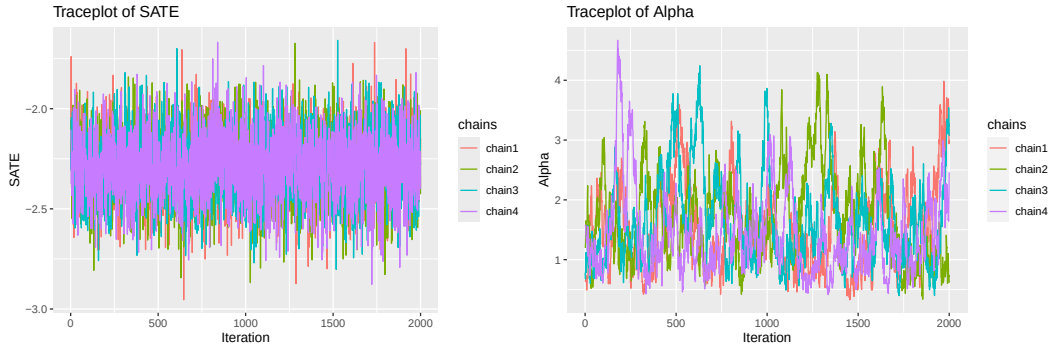
In Figure 4, the argument `top_n` allows us to select variables with the top `top_n` highest PIPs. The argument `threshold` displays variables with PIP greater than `threshold`. From a decision-theoretical perspective (Barbieri and Berger, 2004; Linero, 2018), variables with PIPs larger than 0.5 can be considered chosen confounders. It is worth noting that the five true confounders $X_1 - X_5$ are all correctly selected as true confounders with PIPs of 1, along with one extra predictor X_7 in the outcome model.

The second option for visualization is the traceplot, which is mainly used to check MCMC convergence. The function provides a traceplot of the sample average treatment effect (SATE) for each MCMC chain. Traceplots of other parameters such as `dir_alpha` (the hyperparameter α in the sparsity-inducing Dirichlet prior $\mathcal{D}(\alpha/P, \cdots, \alpha/P)$) and `sigma2_out` (the variance parameter in the outcome model) are also available by using the argument `parameter`.

```
plot(separate_fit, method = 'trace')
plot(separate_fit, method = 'trace', parameter = 'dir_alpha')
```

In Figure 5, the traceplots of the SATE and `dir_alpha` parameters are shown for four different MCMC chains. Concerning the `dir_alpha` parameter (α), the actual value employed as the hyperparameter for the Dirichlet prior is derived by dividing the total number of potential confounders, denoted as P (i.e., α/P). In the setting of simulation data where $P = 100$ is utilized, the hyperparameter to be estimated is notably small, represented as $\alpha/100$. Hence, compared to the variability observed in the traceplot, the variability of the actual α/P (illustrated within the range of $0.5/100$ to $4.5/100$ in Figure 5.(b)) can be interpreted as substantially smaller. Alternatively, extending the chain length for sampling the α parameter could be considered. However, in this dataset, due to the significant presence of confounders (i.e., the counts of splits $n_{j,1}, n_{j,21}, n_{j,20}$ added to α/P during the conjugate update are quite large), confounder selection is minimally influenced by the current α samples.

Although traceplots offer a convenient means of visual inspection, it is recommended to employ the Gelman-Rubin diagnostics provided by the `gelman.diag()` function in the coda package (Plummer et al., 2006) for a comprehensive convergence assessment, as illustrated in the subsequent section. Furthermore, despite the suggestion of a modified $\hat{R}$ (Vehtari et al., 2021) as an alternative to the Gelman-Rubin diagnostic, it is presently unsupported in the coda package. Nonetheless, once the `mcmc.list` object is generated (explained later in this section), it can be directly employed if a function for computing the modified $\hat{R}$ becomes available within the coda package in the future.



(a) Traceplot of *SATE* **(b)** Traceplot of `dir-alpha` parameter

Figure 5: Traceplots for multiple MCMC chains

We evaluated the performance of bartcs in comparison to other models, including those generated by the bacr R package (Wang et al., 2015) that inspired our model development. The bacr package is easily installed via CRAN and loaded into the current R session as follows:

```
install.packages("bacr", dependencies=TRUE)
library("bacr")
```

To fit the model of this package, we used the `bac()` function where the input data needs to be provided in the form of a data frame. To fit the exposure and outcome models in this case, a generalized linear model is used, and it is necessary to specify the family of the model based on the data type (e.g.. `familyX="binomial"` and `familyY="gaussian"`). The MCMC algorithm was run for 10000 iterations after discarding the first 10000 iterations as burn-ins. Additionally, no interaction between the exposure and each confounder was assumed.

```
Z <- as.data.frame(cbind(Y,Trt,X))
fit.bac <- bac(
     data = Z, exposure = "Trt", outcome = "Y",
     confounders = paste("V", 3:(P + 2), sep = ""),
     interactors = NULL, familyX = "binomial", familyY = "gaussian",
     omega = Inf, num_its = 20000, burnM = 10000, burnB = 10000, thin = 5
   )
```

The result can be checked through the `summary()` function as follows:

```
summary(fit.bac)

BAC objects:

Exposure effect estimate:
        posterior mean     95% posterior interval
                  -1.6               (-2.1, -1.3)


Covariates with posterior inclusion probability > 0.5:
    posterior inclusion probability
V3                          1.00000
V4                          1.00000
V5                          1.00000
V6                          1.00000
V7                          1.00000
V99                         0.92100
V14                         0.70305
V54                         0.67480
V90                         0.62345
```

The posterior mean of the SATE was estimated to be -1.6, which was significantly different from the true SATE value of -2.55. Moreover, the 95% credible interval $(-2.1, -1.3)$ did not include the true value. When considering the importance of selected confounders based on the posterior inclusion probability, bacr included all important confounders $X_1 - X_5$ (that is, $V3 - V7$ in the summary), but

also added X_{12}, X_{52}, X_{88}, and X_{97} (that is, $V14$, $V54$, $V90$, $V99$ in the summary) with high PIPs, which were not true confounders. Notably, X_6 and X_7, which are additional predictors of the outcome model, were not included. This result may be attributed to the fact that bacr relies on a parametric model and therefore may struggle to account for the non-linear and complex data structure.

3.1 Connection to coda package

To summarize the results, generic functions such as `summary()` and `plot()` were adapted to work on the bartcs objects. Additionally, `mcmc.list` objects were included as components in the bartcs object to allow for the use of functions from the coda R package (Plummer et al., 2006). The `mcmc_list` component of the bartcs object can produce summary statistics for each of $E[Y(1)], E[Y(0)], SATE$ using the summary function and generate trace plots and posterior densities for parameters using the plot function. Figure 6 displays plot of `mcmc_list` based on coda package.

```
summary(separate_fit$mcmc_list)

Iterations = 10005:20000
Thinning interval = 5
Number of chains = 4
Sample size per chain = 2000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

                Mean        SD  Naive SE Time-series SE
SATE        -2.285155 0.1639173 1.833e-03      2.489e-03
Y1           0.719562 0.1328857 1.486e-03      2.122e-03
Y0           3.004717 0.0977850 1.093e-03      1.790e-03
dir_alpha    1.576421 0.7317213 8.181e-03      6.836e-02
sigma2_out1  0.001731 0.0003359 3.756e-06      5.533e-06
sigma2_out0  0.001310 0.0002334 2.609e-06      3.784e-06

2. Quantiles for each variable:

                2.5%       25%       50%       75%     97.5%
SATE        -2.6022894 -2.396508 -2.284276 -2.174820 -1.969213
Y1           0.4663024  0.629285  0.718583  0.808712  0.983369
Y0           2.8116436  2.940446  3.005341  3.071342  3.194602
dir_alpha    0.5772667  1.024000  1.423480  2.003651  3.390765
sigma2_out1  0.0011718  0.001490  0.001698  0.001935  0.002483
sigma2_out0  0.0009263  0.001146  0.001289  0.001446  0.001845

plot(separate_fit$mcmc_list)
```

The convergence of the MCMC can be assessed by utilizing the convergence diagnostics offered by the coda package. To examine the convergence of six parameters, we can employ the `gelman.diag()` function on the `mcmc.list` object, specifically on `separate_fit$mcmc_list`.

```
library("coda")
gelman.diag(separate_fit$mcmc_list)

Potential scale reduction factors:

            Point est. Upper C.I.
ATE               1.00       1.00
Y1                1.00       1.00
Y0                1.00       1.01
dir_alpha         1.02       1.07
sigma2_out1       1.00       1.00
sigma2_out0       1.00       1.00

Multivariate psrf

1.02
```

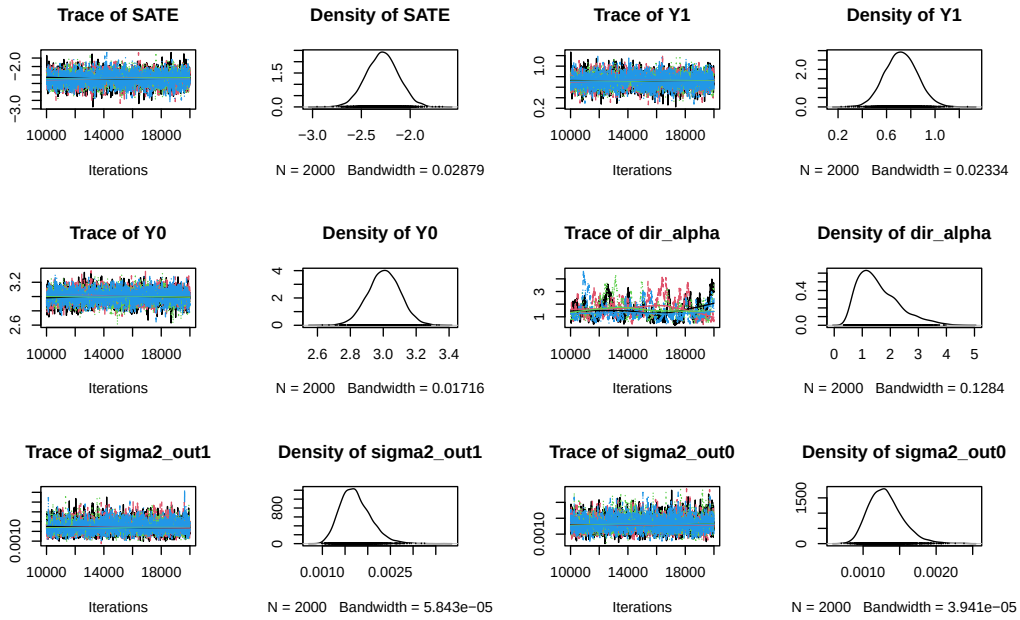


Figure 6: Plot of `mcmc-list` using the **coda** R package

Based on the convergence diagnostics, it can be concluded that there are no issues with the convergence of the MCMC, similar to the visual inspection.

4 Real data example

In the previous section, the `separate_bart()` function was used to demonstrate a separate outcome model strategy. In this section, a single outcome model is tested using the `single_bart()` function, based on the Infant Health and Development Program (IHDP) dataset as an example. This dataset was collected from a longitudinal study that tracked the development of low-birth-weight premature infants. The study participants in the treatment group received intensive care and home visits from trained providers and their cognitive test scores were evaluated at the end of the intervention period. The dataset includes a variety of pretreatment variables, including 6 continuous and 19 binary covariates. The original IHDP data is generated from a randomized experiment setting. However, the IHDP data used by Hill (2011) and Louizos et al. (2017) was manipulated to induce covariate imbalance between treatment groups by removing a subset of the treated group. Specifically, all children with nonwhite mothers were removed from the treated group. We utilize a synthesized variant of the IHDP data as presented in Louizos et al. (2017). This version was created employing the NPCI package (Dorie, 2016) to ascertain the true counterfactual values. As seen in Figure 7, the data generated in this manner significantly violates the overlap assumption for estimating the sample average treatment effect (SATE). This figure depicts the degree of overlap between two groups (Treated vs Control) for selected covariates. Red crosses represent the control group, and blue triangles represent the treated group. In certain intervals of extreme values for each covariate, there are regions where only control group data exists, or very few data points from the treated group are present. For example, in the interval where the X_5 covariate is less than -4, there is no data from the treated group. Non-overlap occurs in these regions. In the case of the binary covariate X_{18}, there is only one data point from the treated group at the value of 0. Therefore, technically speaking, situations like non-overlap can occur in the estimation process. In such a scenario, one of the objectives is to investigate whether a single outcome model can properly estimate the true SATE.

The IHDP data can be loaded by

```
data("ihdp", package = "bartcs")
```

and Table 2 displays the summary statistics of the variables. In the dataset, `y_factual` is the observed outcome Y (i.e., $Y(A)$) and `y_cfactual` is the counterfactual outcome Y (i.e., $Y(1-A)$).

We fit the single outcome model using the `single_bart()` function.

```
single_fit <- single_bart(
  Y               = ihdp$y_factual,
```

	Treatment = 1 (n=139)		Treatment = 0 (n=608)	
Variable	Mean	IQR	Mean	IQR
Y	6.43	(5.83, 7.34)	2.41	(1.45, 3.08)
$X_1^\star$	0.21	(-0.40, 0.95)	-0.05	(-0.75, 0.79)
$X_2^\star$	0.18	(-0.20, 0.60)	-0.04	(-0.60, 0.60)
$X_3^\star$	-0.04	(-0.73, 0.38)	0.01	(-0.73, 0.76)
$X_4^\star$	-0.22	(-0.88, 0.16)	0.05	(-0.88, 0.16)
$X_5^\star$	-0.14	(-0.69, 0.56)	0.03	(-0.50, 0.68)
$X_6^\star$	0.21	(-0.53, 0.96)	-0.05	(-0.86, 0.63)
X_7	0.52	(0.00, 1.00)	0.51	(0.00, 1.00)
X_8	0.09	(0.00, 0.00)	0.1	(0.00, 0.00)
X_9	0.68	(0.00, 1.00)	0.49	(0.00, 1.00)
X_{10}	0.29	(0.00, 1.00)	0.38	(0.00, 1.00)
X_{11}	0.25	(0.00, 0.50)	0.27	(0.00, 1.00)
X_{12}	0.22	(0.00, 0.00)	0.22	(0.00, 0.00)
X_{13}	0.38	(0.00, 1.00)	0.35	(0.00, 1.00)
X_{14}	1.58	(1.00, 2.00)	1.44	(1.00, 2.00)
X_{15}	0.14	(0.00, 0.00)	0.14	(0.00, 0.00)
X_{16}	0.94	(1.00, 1.00)	0.97	(1.00, 1.00)
X_{17}	0.69	(0.00, 1.00)	0.57	(0.00, 1.00)
X_{18}	0.99	(1.00, 1.00)	0.96	(1.00, 1.00)
X_{19}	0.15	(0.00, 0.00)	0.13	(0.00, 0.00)
X_{20}	0.06	(0.00, 0.00)	0.15	(0.00, 0.00)
X_{21}	0.17	(0.00, 0.00)	0.15	(0.00, 0.00)
X_{22}	0.04	(0.00, 0.00)	0.09	(0.00, 0.00)
X_{23}	0.01	(0.00, 0.00)	0.09	(0.00, 0.00)
X_{24}	0.06	(0.00, 0.00)	0.14	(0.00, 0.00)
X_{25}	0.27	(0.00, 1.00)	0.13	(0.00, 0.00)

Table 2: Summary statistics for the IHDP data set. $\star$ denotes a continuous potential confounder.

```
  trt             = ihdp$treatment,
  X               = ihdp[, 6:30],
  num_tree        = 50,
  num_chain       = 4,
  num_post_sample = 2000,
  num_thin        = 5,
  num_burn_in     = 10000
)
single_fit

`bartcs` fit by `single_bart()`

         mean     2.5%    97.5%
SATE 3.964842 3.747028 4.180764
Y1   6.382810 6.188199 6.581852
Y0   2.417969 2.338264 2.496962
```

The function `single_bart()` returns a bartcs object, which displays the posterior means and 95% credible intervals for the sample average treatment effect (SATE), and the potential outcomes $Y(1)$ and $Y(0)$. The `summary()` and `plot()` functions can also be used with this bartcs object generated by `single_bart()`.

```
summary(single_fit)

`bartcs` fit by `single_bart()`

Treatment Value
  Treated group    :      1
  Control group    :      0

Tree Parameters
```

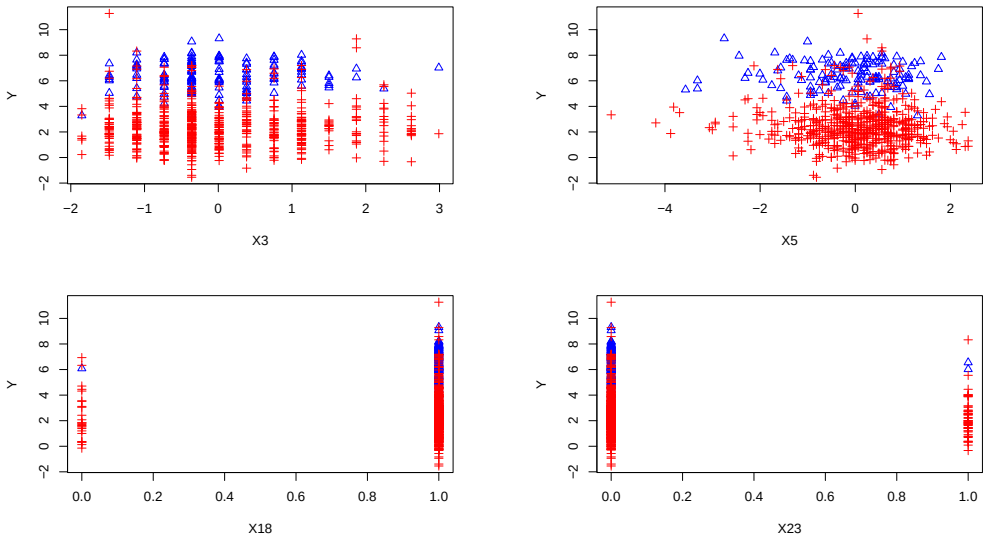


Figure 7: A plot illustrating the degree of overlap between two groups (Treated vs Control) for selected covariates. Red crosses represent the control group, and blue triangles represent the treated group. In certain intervals for each covariate, there are regions where only control group data exists, or very few data points from the treated group are present. Non-overlap occurs in these regions.

```
  Number of Tree   :     50              Value  of alpha    :   0.95
  Prob.  of Grow   :   0.28              Value  of beta     :      2
  Prob.  of Prune  :   0.28              Value  of nu       :      3
  Prob.  of Change :   0.44              Value  of q        :   0.95

Chain Parameters
  Number of Chains :      4              Number of burn-in  :  10000
  Number of Iter   :  20000              Number of thinning :      5
  Number of Sample :   2000

Outcome
 estimand chain     2.5%       1Q     mean   median       3Q    97.5%
     SATE     1 3.758373 3.894465 3.969119 3.968867 4.042380 4.183131
     SATE     2 3.744731 3.886575 3.957434 3.956101 4.026455 4.165961
     SATE     3 3.760480 3.905973 3.980315 3.980162 4.054086 4.206488
     SATE     4 3.730287 3.879606 3.952498 3.953050 4.028315 4.158430
     SATE   agg 3.747028 3.891543 3.964842 3.965384 4.038288 4.180764
      Y1     1 6.196530 6.318675 6.387760 6.387443 6.453303 6.589611
      Y1     2 6.181788 6.310026 6.376027 6.376233 6.439727 6.573960
      Y1     3 6.196317 6.329945 6.396885 6.397153 6.464297 6.601299
      Y1     4 6.169429 6.303404 6.370570 6.371514 6.435679 6.562172
      Y1   agg 6.188199 6.314489 6.382810 6.382215 6.449542 6.581852
      Y0     1 2.339020 2.391137 2.418640 2.418824 2.446414 2.498677
      Y0     2 2.336131 2.392407 2.418593 2.418124 2.446167 2.495229
      Y0     3 2.337997 2.388738 2.416570 2.416414 2.444457 2.495583
      Y0     4 2.340288 2.389536 2.418073 2.418218 2.446018 2.497264
      Y0   agg 2.338264 2.390199 2.417969 2.418042 2.445718 2.496962
```

We also fitted separate outcome models to the `ihdp` data and compared the results from the single outcome model.

```
separate_fit <- separate_bart(
     Y              = ihdp$y_factual,
     trt            = ihdp$treatment,
     X              = ihdp[, 6:30],
     num_tree       = 50,
     num_chain      = 4,
     num_post_sample = 2000,
     num_thin       = 5,
```

```
    num_burn_in     = 10000
  )
separate_fit

`bartcs` fit by `separate_bart()`

         mean     2.5%    97.5%
SATE 3.924013 3.702316 4.148937
Y1   6.342504 6.134043 6.550242
Y0   2.418491 2.340920 2.497081
```

Similar to the separate outcome models strategy, in `single_bart()`, the `plot()` function for the bartcs object can also be employed to check the convergence of the MCMC chain. The traceplots for the ATE is presented in Figure 8 with the following line.

```
plot(single_fit, method = 'trace')
```

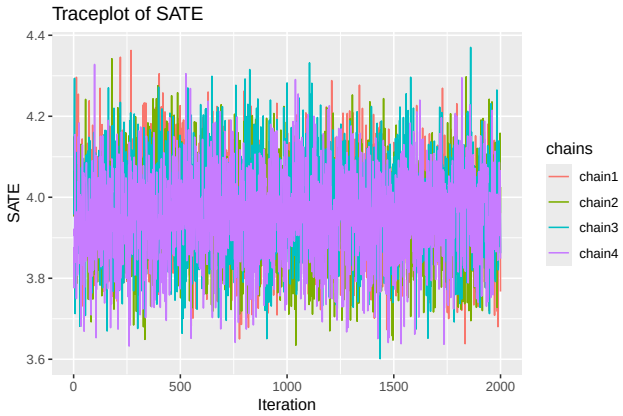


Figure 8: Traceplot of SATE for IHDP dataset

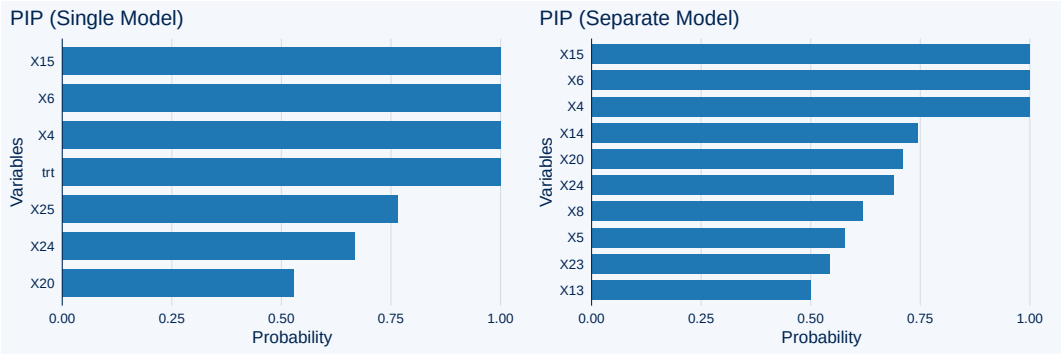
As this is a simulated version of the IHDP data, the true values are known and are 4.02 for the sample average treatment effect (SATE), 6.45 for $E[Y(1)]$, and 2.43 for $E[Y(0)]$. The outputs from the two models accurately reflect these true values within their 95% credible intervals. Additionally, the PIP plots (Figure 9) depict chosen confounders with PIP values larger than 0.5.

The important aspect here is that in the case of the single outcome model, the exposure variable (`trt`) is also incorporated into the selection process. As indicated in Equation 2, because the exposure variable is included as one of the covariates in the outcome model, it is subject to variable selection. This means that in the computation of PIP, it is treated similarly to other confounders, producing the following plot (a) in Figure 9. In Figure 9, plot (a) displays the potential confounders for the single outcome model, which have a posterior inclusion probability of 0.5 or more, while plot (b) illustrates the confounders with a posterior inclusion probability of 0.5 or more when the separate outcome models strategy is used. It is noteworthy that X_4, X_6, and X_{15} were consistently chosen as confounders with posterior inclusion probability 1.

4.1 Computation speed

In Figure 10, the computational speed of two models, the separate and single models, is depicted for two different settings of the number of trees (100 vs. 200) based on the scenario in Section 3. The speed was assessed using 5000 MCMC iterations across various combinations of N and P. We considered three values of N (100, 500, and 1000) and three values of P (circle for $N \times 0.3$, triangle for $N \times 0.5$, and cross for $N \times 1$).

For 100 BART trees, the separate models required 8 to 98 seconds (18 to 190 seconds for 200 BART trees) for computation, while the single model took 7 to 86 seconds (14 to 168 seconds for 200 BART trees), depending on the (N, P) combination. Both models exhibited similar computational speeds overall, considering the MCMC iterations. However, the single model, which fits two BART models (exposure and one outcome model), was found to be more efficient with slightly smaller biases and mean square errors (MSEs) across various scenarios (Kim et al., 2023). Therefore, it is recommended to utilize the single model (`single_bart()` function), especially when N is large, due to its faster computational speed.



(a) PIP plot of single model **(b)** PIP plot of separate model

Figure 9: PIP plot for IHDP dataset

Additionally, depending on the number of trees used, a significant improvement in computation speed can be observed. It is generally suggested to start with 50 trees as a "good starting value," (Kapelner and Bleich, 2016) so using a smaller number of trees is also advised to gain computational advantages in terms of speed. The results in this manuscript were obtained using R 4.3.0 on a Mac Studio with a M1 chip and 128 GB of memory. bartcs 1.2.2 and bacr 1.0.1 were used for the analysis.

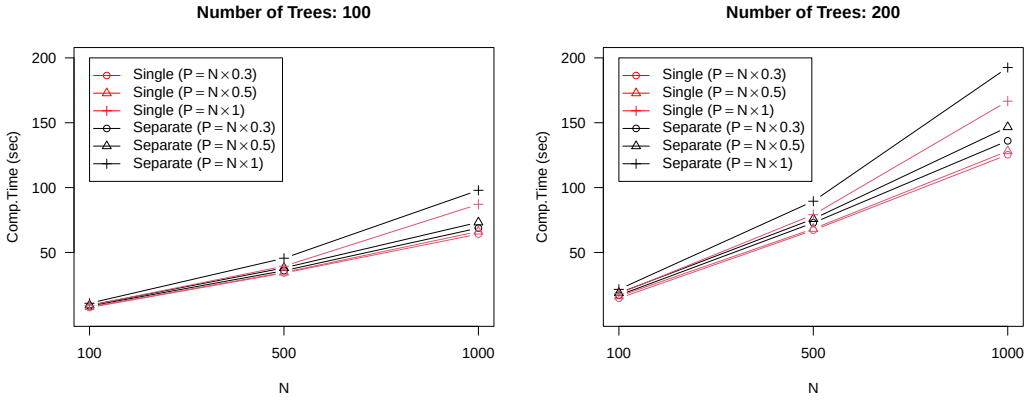


Figure 10: The computation times for both the single outcome model (red) and separate outcome models (black) based on the number of observations (N) under two different numbers of trees. A cross symbol (+) represents the scenario where the number of potential confounders (P) is equal to the number of observations (N), a triangle ($\triangle$) represents the scenario where $P = N \times 0.5$ and a circle ($\bigcirc$) represents the scenario where $P = N \times 0.3$. These results are obtained from 5000 MCMC iterations based on the scenario in Section 3.

5 Continuous exposure example

When it comes to a continuous exposure variable, the formula in Equation 1 is changed as follows:

$$\begin{aligned} A_i &= \mu_1 + f_1(\boldsymbol{X}_i) + \epsilon_i, \quad \epsilon_j \sim N(0, \tau^2). \\ f_1(\boldsymbol{X}_i) &= \sum_{t=1}^{T} g_1(\boldsymbol{X}_i; \mathcal{T}_t, \mathcal{M}_t) \end{aligned} \tag{4}$$

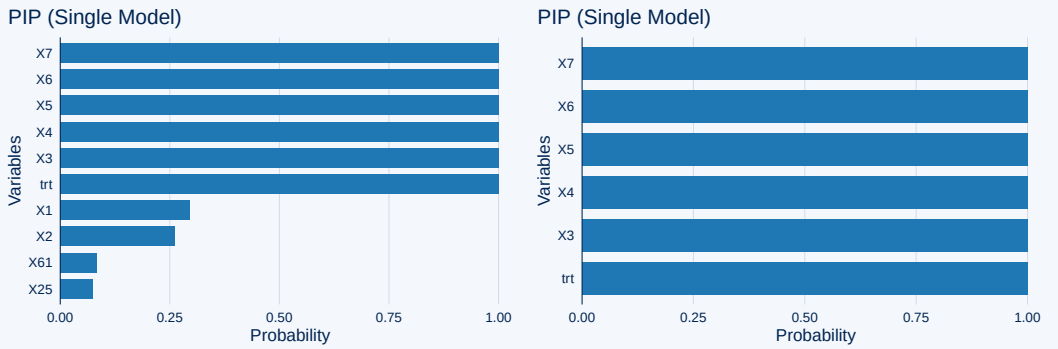
This altered formula is used in conjunction with the single outcome model to perform confounder selection. However, the separate outcome models strategy, which fits two distinct outcome models based on the two exposure levels, is not suitable for the continuous exposure variable. The `single_bart()` function has the versatility to handle both binary and continuous treatments, and automatically identifies the binary treatment when there are only two unique values. To demonstrate this, we generate a data set similar to the previous example.

```
set.seed(42)
N <- 300
P <- 100
cov <- list()
for (i in 1:P) {
     cov[[i]] <- rnorm(N, 0, 1)
   }
X <- do.call(cbind, cov)
h1 <- ifelse(X[, 1] < 0, 1, -1)
h2 <- ifelse(X[, 2] < 0, -1, 1)
mu_trt <- 0.5 + h1 + h2 - 0.5 * abs(X[, 3] - 1) + 0.5 * X[, 4] * X[, 5]
Trt <- rnorm(N, mu_trt, 0.3)
mu_y <- 1 * h1 + 1 * h2 - Trt + 1 * abs(X[, 3] + 1) + 1 * X[, 4] + exp(0.5 * X[, 5]) -
     0.5 * Trt * abs(X[, 6]) - 0.5 * Trt * abs(X[, 7] + 1)
Y <- rnorm(N, mu_y, 0.3)
treatment <- quantile(Trt, 0.75)
control <- quantile(Trt, 0.25)
```

We use the function `single_bart()` to fit the generated data. The first and third quantile values of `Trt` will serve as the basis for comparing two different exposure levels. As arguments in `single_bart()`, we need to provide these two pre-specified exposure levels (a =`trt_treated` and a' =`trt_control`). In the this case, the causal estimand is $\Delta(a, a') = E[(Y(a) - Y(a')]$.

```
single_fit <- single_bart(
    Y = Y, trt = Trt, X = X,
    trt_treated = treatment, trt_control = control,
    num_tree = 200, num_chain = 4,
    num_burn_in = 10000, num_thin = 5, num_post_sample = 2000
  )
single_fit

`bartcs` fit by `single_bart()`

           mean       2.5%     97.5%
SATE -2.8097339 -4.2581469 -1.732448
Y1    0.9982417  0.2753606  1.677726
Y0    3.8079756  3.0967180  4.740133
```

Similar to other bartcs objects, the `summary()` and `plot()` functions can be applied to the continuous exposure scenario. Figure 11 displays a PIP plot, which demonstrates that out of 100 possible confounders, all of the true confounders except X_1, X_2, and two additional predictors were captured effectively, with high PIP values.



(a) Plotting PIP with `top_n` argument **(b)** Plotting PIP with `threshold = 0.5` argument

Figure 11: PIP plot for continuous exposure

6 Heterogeneous effects

The proposed method not only estimates the SATE but also provides posterior samples of $Y_i(1)$ and $Y_i(0)$, enabling inference on individual heterogeneous treatment effects. The return object includes a

component called chains, which is a list containing the results from each MCMC chain. Each element in this list is an mcmc.list object (from the coda package) that holds the posterior samples of $Y_i(1)$ and $Y_i(0)$. The following explains how to extract posterior samples when applying the separate model to the generated data from Section 3.

```
separate_fit <- separate_bart(
  Y = Y, trt = Trt, X = X, num_tree = 200, num_chain = 4,
  num_burn_in = 10000, num_thin = 5, num_post_sample = 2000
)

# Y(1) samples from the 1st MCMC chain
separate_fit$chains[[1]]$Y1_sample
# Y(0) samples from the 1st MCMC chain
separate_fit$chains[[1]]$Y0_sample
```

The posterior means of $Y_i(1)$ and $Y_i(0)$ for $i = 1, \cdots, N$ can be obtained by aggregating the posterior samples across multiple chains, as shown below:

```
Y1_sample <- do.call(cbind, lapply(separate_fit$chains, function(x) x$Y1_sample))
rowMeans(Y1_sample)
Y0_sample <- do.call(cbind, lapply(separate_fit$chains, function(x) x$Y0_sample))
rowMeans(Y0_sample)
```

7 Summary and discussion

In conclusion, the bartcs R package is a powerful tool for causal inference using BART. It allows users to adjust for confounders and estimate treatment effects using a flexible non-parametric method. The package's ability to handle high-dimensional and non-linear confounding, binary exposure, and continuous exposure makes it a versatile tool for a wide range of applications. Additionally, the package's support for parallel computing and visualization of results make it a user-friendly and easy-to-interpret tool. The bartcs package is a valuable resource for researchers in various fields.

In this paper, we assessed the performance of the proposed method in a scenario where all true confounders are included in the potential confounder pool, and additional predictors for the outcome model are also present within the potential confounder pool. In this scenario, the proposed method demonstrated precise average treatment effect (SATE) estimation and accurately identified the true confounders. Moreover, in the study by Kim et al. (2023), the proposed method exhibited accurate confounder selection performance and reliable estimation of SATEs even in scenarios involving instrumental variables in the data generating process. This success is attributed to the satisfaction of the disjunctive cause criterion without instruments by the proposed method, as outlined in the introduction section (VanderWeele, 2019). Additionally, the method demonstrated outstanding results in simulation scenarios with diverse effect sizes and varying numbers of true confounders.

The single outcome model and separate outcome models introduced in this paper both demonstrate excellent performance in confounder selection and average treatment effect estimation. However, in cases where a continuous treatment variable is required, the single outcome model should be applied. Additionally, as indicated in Section 4, the single outcome model has a slightly faster computation speed than the separate outcome models when the sample size is large because it uses one less BART model. On the other hand, the separate outcome models strategy has the advantage of relatively faster convergence of the MCMC chain during the process of updating the selection probability vector of the BART prior using a simple Gibbs update. Therefore, it is necessary to selectively choose between the two models based on the context of the data being applied.

While not currently integrated into the bartcs package, the confounder selection method presented here using BART holds potential for extension to various data types. For count or categorical outcomes, it might be feasible to substitute the proposed outcome model with the log-linear BART model suggested by Murray (2021). Similarly, for survival outcomes, the survival BART model proposed by Sparapani et al. (2016) could serve as the outcome model. Exploring the specific computation algorithms for these extensions could be a fruitful avenue for future research.

One limitation of the proposed method is its lack of consideration for correlation and temporal relationships among potential confounders. Currently, no research has explored the distribution of weights in the selection probability vector when high correlation exists among covariates in the potential confounder pool. An approach worth investigating may involve leveraging a causal Directed Acyclic Graph (DAG) to constrain the selection of certain covariates in the prior setting of the selection probability vector. This too presents a promising direction for future research. Furthermore, our model currently assumes a common hyper-prior for all hyperparameters $\boldsymbol{\alpha}$ associated with the selection

probability vector. This approach restricts our ability to incorporate differing prior weights across potential confounders. Exploring more flexible specifications, such as assigning separate hyper-priors to each α, represents an important avenue for future research.

Another limitation, as mentioned previously in the remark in Section 2, is that biases arising from unmeasured confounders are inherently unavoidable when the set of potential confounders does not include all true confounders. If important confounding variables are suspected to be missing from the available data, one should carefully exclude—prior to using the proposed method—variables suspected to be instruments (which influence the exposure but not the outcome) and potential colliders (which do not directly affect both exposure and outcome).

Computational details

R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/. The results in this manuscript were obtained using R 4.3.0 on a Mac Studio with a M1 chip and 128 GB of memory. bartcs 1.3.0 and bacr 1.0.1 were used for the analysis.

Acknowledgments

This work is supported by the National Research Foundation of Korea (NRF) grants funded by the Korea government (RS-2025-00554477, RS-2024-00407300, NRF-2022R1F1A1062904).

References

A. Abadie. Semiparametric difference-in-differences estimators. *The Review of Economic Studies*, 72(1): 1–19, 2005. [p175]

J. H. Albert and S. Chib. Bayesian analysis of binary and polychotomous response data. *Journal of the American Statistical Association*, 88(422):669–679, 1993. [p180]

M. M. Barbieri and J. O. Berger. Optimal predictive model selection. *The Annals of Statistics*, 32(3): 870–897, 2004. [p187]

A. Caron, G. Baio, and I. Manolopoulou. Shrinkage bayesian causal forests for heterogeneous treatment effects estimation. *Journal of Computational and Graphical Statistics*, 31(4):1202–1214, 2022. [p176]

H. A. Chipman, E. I. George, R. E. McCulloch, et al. BART: Bayesian additive regression trees. *The Annals of Applied Statistics*, 4(1):266–298, 2010. [p175, 178]

P. Ding and F. Li. Causal inference. *Statistical Science*, 33(2):214–237, 2018. [p177]

P. Ding, T. VanderWeele, and J. M. Robins. Instrumental variables as bias amplifiers with general outcome and confounding. *Biometrika*, 104(2):291–302, 2017. [p175]

V. Dorie. *NPCI: Non-parametrics for Causal Inference*, 2016. URL https://github.com/vdorie/npci. [p190]

J. Häggström. *CovSelHigh: Model-Free Covariate Selection in High Dimensions*, 2017. URL https://CRAN.R-project.org/package=CovSelHigh. R package version 1.1.1. [p177]

J. Häggström. Data-driven confounder selection via Markov and Bayesian networks. *Biometrics*, 74(2): 389–398, 2018. [p175]

P. R. Hahn, J. S. Murray, and C. M. Carvalho. Bayesian regression tree models for causal inference: Regularization, confounding, and heterogeneous effects. *Bayesian Analysis*, 15(3):965–1056, 2020. doi: 10.1214/19-BA1195. [p176, 177, 181]

T. Hastie and R. Tibshirani. Bayesian backfitting (with comments and a rejoinder by the authors). *Statistical Science*, 15(3):196–223, 2000. [p178, 180]

J. L. Hill. Bayesian nonparametric modeling for causal inference. *Journal of Computational and Graphical Statistics*, 20(1):217–240, 2011. doi: 10.1198/jcgs.2010.08162. [p177, 181, 190]

A. Kapelner and J. Bleich. bartMachine: Machine learning with Bayesian additive regression trees. *Journal of Statistical Software*, 70(4):1–40, 2016. doi: 10.18637/jss.v070.i04. [p178, 180, 194]

C. Kim, M. Tec, and C. M. Zigler. Bayesian nonparametric adjustment of confounding. *Biometrics*, 79 (4):3252–3265, 2023. doi: 10.1111/biom.13833. [p175, 176, 178, 184, 193, 196, 202]

G. Lefebvre, J. A. Delaney, and R. L. McClelland. Extending the Bayesian adjustment for confounding algorithm to binary treatment covariates to estimate the effect of smoking on carotid intima-media thickness: the multi-ethnic study of atherosclerosis. *Statistics in Medicine*, 33(16):2797–2813, 2014. [p175]

F. Li, P. Ding, and F. Mealli. Bayesian causal inference: a critical review. *Philosophical Transactions of the Royal Society A*, 381(2247):20220153, 2023. [p177]

A. R. Linero. Bayesian regression trees for high-dimensional prediction and variable selection. *Journal of the American Statistical Association*, 113(522):626–636, 2018. [p177, 181, 187]

C. Louizos, U. Shalit, J. M. Mooij, D. Sontag, R. Zemel, and M. Welling. Causal effect inference with deep latent-variable models. *Advances in Neural Information Processing Systems*, 30, 2017. doi: 10.48550/arXiv.1705.08821. URL https://github.com/AMLab-Amsterdam/CEVAE. [p190]

J. S. Murray. Log-linear Bayesian additive regression trees for multinomial logistic and count regression models. *Journal of the American Statistical Association*, 116(534):756–769, 2021. [p196]

J. A. Myers, J. A. Rassen, J. J. Gagne, K. F. Huybrechts, S. Schneeweiss, K. J. Rothman, M. M. Joffe, and R. J. Glynn. Effects of adjusting for instrumental variables on bias and precision of effect estimates. *American Journal of Epidemiology*, 174(11):1213–1222, 2011. [p175]

T. Neitmann, J. Silge, and D. Robinson. *ggcharts: Shorten the Distance from Data Visualization Idea to Actual Plot*, 2020. URL https://CRAN.R-project.org/package=ggcharts. R package version 0.2.1. [p187]

A. Oganisian and J. A. Roy. Invited discussion for Bayesian regression tree models for causal inference: regularization, confounding, and heterogeneous effects. *Bayesian Analysis*, 15(3):998–1006, 2020. [p177]

G. Papadogeorgou and F. Li. Invited discussion for Bayesian regression tree models for causal inference: regularization, confounding, and heterogeneous effects. *Bayesian Analysis*, 15(3):1007–1013, 2020. [p177]

M. Plummer, N. Best, K. Cowles, and K. Vines. Coda: Convergence diagnosis and output analysis for mcmc. *R News*, 6(1):7–11, 2006. URL https://journal.r-project.org/archive/. [p187, 189]

M. T. Pratola. Efficient metropolis–hastings proposal mechanisms for Bayesian regression tree models. *Bayesian Analysis*, 11(3):885–911, 2016. [p178]

P. R. Rosenbaum and D. B. Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1):41–55, Apr. 1983. ISSN 00063444. [p175, 176, 177]

D. B. Rubin. Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of Educational Psychology*, 66(5):688, 1974. [p176]

D. B. Rubin. Discussion of randomization analysis of experimental data: the Fisher randomization test by D. Basu. *Journal of the American Statistical Association*, 75(371):591–593, 1980. [p176]

S. M. Shortreed and A. Ertefaie. Outcome-adaptive lasso: Variable selection for causal inference. *Biometrics*, 73(4):1111–1122, 2017. [p175]

R. Sparapani, C. Spanbauer, and R. McCulloch. Nonparametric machine learning and efficient computation with Bayesian additive regression trees: The BART R package. *Journal of Statistical Software*, 97(1):1–66, 2021. doi: 10.18637/jss.v097.i01. [p176, 177, 181]

R. A. Sparapani, B. R. Logan, R. E. McCulloch, and P. W. Laud. Nonparametric survival analysis using Bayesian additive regression trees (bart). *Statistics in Medicine*, 35(16):2741–2753, 2016. [p178, 196]

T. J. VanderWeele. Principles of confounder selection. *European Journal of Epidemiology*, 34(3):211–219, 2019. [p175, 177, 196]

A. Vehtari, A. Gelman, D. Simpson, B. Carpenter, and P.-C. Bürkner. Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC (with discussion). *Bayesian Analysis*, 16(2):667–718, 2021. [p187]

C. Wang, G. Parmigiani, and F. Dominici. Bayesian effect estimation accounting for adjustment uncertainty. *Biometrics*, 68(3):661–671, 2012. [p175]

C. Wang, F. Dominici, G. Parmigiani, and C. M. Zigler. Accounting for uncertainty in confounder and effect modifier selection when estimating average causal effects in generalized linear models. *Biometrics*, 71(3):654–665, 2015. [p175, 177, 188]

A. Wilson and B. J. Reich. Confounder selection via penalized credible regions. *Biometrics*, 70(4): 852–861, 2014. [p175]

A. Wilson, H. D. Bondell, and B. J. Reich. *BayesPen: Bayesian Penalized Credible Regions*, 2015. URL https://anderwilson.github.io/BayesPen. R package version 1.2. [p177]

Y. Yoo. *bartcs: Bayesian Additive Regression Trees for Confounder Selection*, 2024. URL https://CRAN.R-project.org/package=bartcs. R package version 1.2.2. [p176, 177]

K. Yoshida and A. Bartel. *tableone: Create 'Table 1' to Describe Baseline Characteristics with or without Propensity Score Weights*, 2022. URL https://CRAN.R-project.org/package=tableone. R package version 0.13.2. [p185]

Yeonghoon Yoo
Department of Statistics, SungKyunKwan University
25-2 SungKyunKwan University, Jongno-gu
Seoul 03063, South Korea
yooyh.stat@gmail.com

Chanmin Kim (corresponding author)
Department of Statistics, SungKyunKwan University
25-2 SungKyunKwan University, Jongno-gu
Seoul 03063, South Korea
(ORCiD: 0000-0002-2588-6704)
chanmin.kim@skku.edu

Appendix

The appendix presents results from simulation studies conducted under various scenarios that were not covered in the main text.

Scenario A: 5 true confounders and 2 additional predictors

The results from applying the single model to the example in Section 3 demonstrate that the estimates closely match the true values of SATE (-2.55), $E[Y(1)]$ (0.64), and $E[Y(0)]$ (3.19), consistent with findings from the separate model. Notably, all corresponding 95% credible intervals contain the true parameter values.

```
attach(synthetic_data(N = 300, P = 100, seed = 42))
single_fit <- single_bart(
    Y = Y, trt = Trt, X = X,
    num_tree = 200, num_chain = 4,
    num_burn_in = 10000, num_thin = 5, num_post_sample = 2000
  )
single_fit

`bartcs` fit by `single_bart()`

           mean       2.5%      97.5%
SATE -2.4219319 -2.6327170 -2.2125321
Y1    0.7077994  0.5664953  0.8491844
Y0    3.1297313  3.0100750  3.2520308
```

For a more in-depth understanding of the output, the `summary()` function can be used. It provides details regarding the treatment values, tree structure, MCMC chain, and outcomes for each of the chains.

```
summary(single_fit)

`bartcs` fit by `single_bart()`

Treatment Value
  Treated group    :      1
  Control group    :      0

Tree Parameters
  Number of Tree   :    200               Value  of alpha    :   0.95
  Prob.  of Grow   :   0.28               Value  of beta     :      2
  Prob.  of Prune  :   0.28               Value  of nu       :      3
  Prob.  of Change :   0.44               Value  of q        :   0.95

Chain Parameters
  Number of Chains :      4               Number of burn-in  :  10000
  Number of Iter   :  20000               Number of thinning :      5
  Number of Sample :   2000

Outcome
 estimand chain       2.5%         1Q       mean     median         3Q      97.5%
      SATE    1 -2.6372424 -2.4971912 -2.4278177 -2.4274520 -2.3559219 -2.2195278
      SATE    2 -2.6347996 -2.4962662 -2.4236981 -2.4257645 -2.3521617 -2.2205205
      SATE    3 -2.6229176 -2.4872413 -2.4163175 -2.4188511 -2.3472130 -2.1968565
      SATE    4 -2.6321432 -2.4898965 -2.4198942 -2.4185564 -2.3492909 -2.2160691
      SATE  agg -2.6327170 -2.4929933 -2.4219319 -2.4229442 -2.3510791 -2.2125321
       Y1     1  0.5635365  0.6562010  0.7044118  0.7037438  0.7524049  0.8418673
       Y1     2  0.5678166  0.6558246  0.7050962  0.7054015  0.7548294  0.8466451
       Y1     3  0.5718781  0.6606816  0.7112623  0.7108116  0.7596594  0.8529911
       Y1     4  0.5598710  0.6626540  0.7104274  0.7117518  0.7603410  0.8518614
       Y1   agg  0.5664953  0.6587662  0.7077994  0.7082389  0.7566797  0.8491844
       Y0     1  3.0100293  3.0916675  3.1322295  3.1333000  3.1735625  3.2536797
       Y0     2  3.0101642  3.0857730  3.1287944  3.1295026  3.1688030  3.2518096
```

```
Y0     3  3.0067741  3.0866910  3.1275798  3.1257193  3.1693086  3.2485622
Y0     4  3.0158387  3.0909104  3.1303216  3.1291042  3.1697493  3.2501879
Y0   agg  3.0100750  3.0885669  3.1297313  3.1291107  3.1705643  3.2520308
```

To evaluate the performance of confounder selection, 500 simulated datasets were generated using the data-generating process described in Section 3, with five true confounders and two additional predictors included in the outcome model. Figure A1 presents the average posterior inclusion probabilities across these datasets, comparing the separate model (top panel) with the single model (bottom panel). The results clearly indicate that all five true confounders (highlighted by red circles) are selected with posterior probability equal to one, while irrelevant noise variables are consistently excluded. Additionally, in the single model, the starred point representing the treatment variable is also selected with probability one.

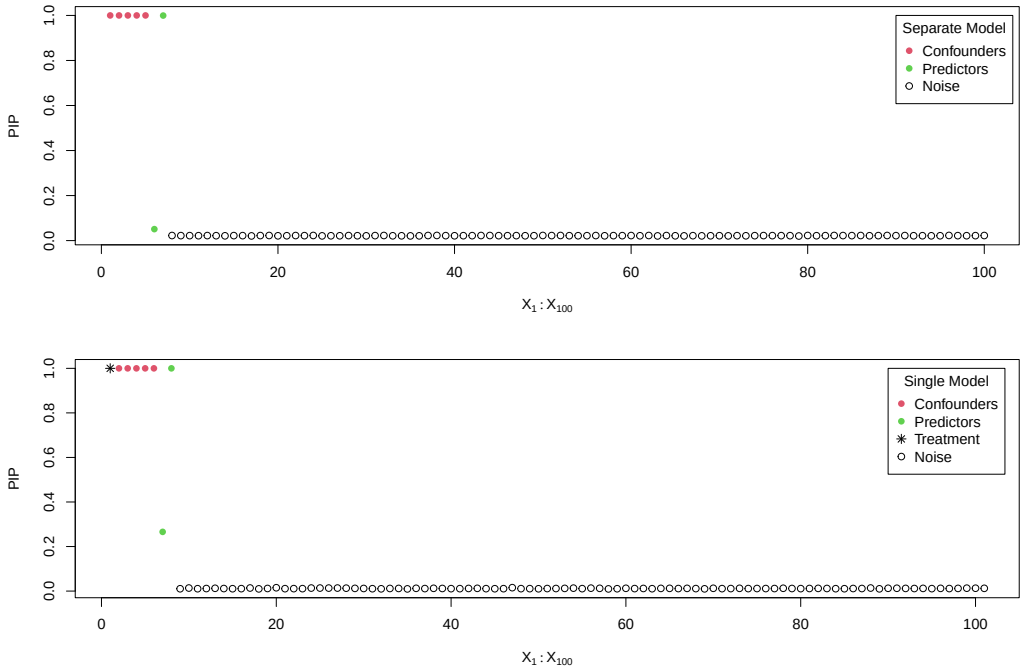


Figure A1: Average PIPs across 500 simulated datasets based on the scenario described in Section 3, comparing the separate and single models. Red circles indicate the true confounders, green circles represent additional predictor variables included in the outcome model, and empty circles denote noise variables. In the single model, the first starred point corresponds to the treatment variable.

Scenario B: 20 true confounders

This scenario is designed to assess the performance of confounder selection when there are 20 true confounders among 100 potential confounders. Data were generated using the following data-generating prcoess:

```
set.seed(42)
N <- 600
P <- 100
cov <- list()
for (i in seq_len(P)) {
  cov[[i]] <- rnorm(N, 0, 1)
}
X <- do.call(cbind, cov)
h1 <- ifelse(X[, 1] < 0, 1, -1)
h2 <- ifelse(X[, 2] < 0, -1, 1)
prob <- pnorm(0.5 + h1 + h2 - 0.5 * abs(X[, 3] - 1) +
                1.5 * X[, 4] * X[, 5] + X[, 6:20] %*% rep(0.5, 15))
Trt <- rbinom(N, 1, prob)
mu1 <- 1 * h1 + 1.5 * h2 - 1 + 2 * abs(X[, 3] + 1) + 2 * X[, 4] + exp(0.5 * X[, 5]) -
```

```
  0.5 * 1 * abs(X[, 6]) - 1 * 1 * abs(X[, 7] + 1) + X[, 6:20] %*% rep(0.5, 15)
mu0 <- 1 * h1 + 1.5 * h2 - 0 + 2 * abs(X[, 3] + 1) + 2 * X[, 4] + exp(0.5 * X[, 5]) -
  0.5 * 0 * abs(X[, 6]) - 1 * 0 * abs(X[, 7] + 1) + X[, 6:20] %*% rep(0.5, 15)
Y1 <- rnorm(N, mu1, 0.3)
Y0 <- rnorm(N, mu0, 0.3)
Y <- Trt * Y1 + (1 - Trt) * Y0
```

Examining the results from the separate model shows that the estimated SATE slightly deviates from its true value of -2.55. This deviation stems primarily from a minor bias in the estimation of $E[Y(0)]$. Such bias arises because, as the number of confounders increases, achieving adequate overlap in confounder distributions between treatment groups becomes more challenging within a limited sample size.

```
separate_fit <- separate_bart(
     Y = Y, trt = Trt, X = X, num_tree = 200, num_chain = 4,
     num_burn_in = 10000, num_thin = 5, num_post_sample = 2000
)

separate_fit
`bartcs` fit by `separate_bart()`

          mean       2.5%     97.5%
SATE -2.207329 -2.4512574 -1.958840
Y1    1.050447  0.8839941  1.224950
Y0    3.257776  3.0882004  3.425323
```

In the case of the single model, the bias is noticeably smaller. This reflects the relative advantage of the single model, as also discussed in Kim et al. (2023).

```
single_fit
`bartcs` fit by `single_bart()`

          mean       2.5%     97.5%
SATE -2.352218 -2.5377421 -2.162899
Y1    1.011433  0.9065068  1.120402
Y0    3.363651  3.2602485  3.465711

single_fit
`bartcs` fit by `single_bart()`

          mean       2.5%     97.5%
SATE -2.352218 -2.5377421 -2.162899
Y1    1.011433  0.9065068  1.120402
Y0    3.363651  3.2602485  3.465711
```

To assess confounder selection performance, 500 simulated datasets were generated based on the data-generating process described above, including 20 true confounders. Figure A2 shows the average posterior inclusion probabilities obtained across these datasets, comparing results from the separate model (top panel) and the single model (bottom panel). The results clearly demonstrate that all 20 true confounders (marked with red circles) are consistently selected with posterior probabilities equal to one, whereas irrelevant noise variables are effectively excluded from the models.

Scenario C: 5 true confounders and 2 instrumental variables

This scenario includes 5 true confounders among 100 potential confounders, and in particular, it includes 2 instrumental variables (i.e., variables that affect only the exposure). The data generating process for this scenario is as follows:

```
set.seed(42)
N <- 300
P <- 100
cov <- list()
for (i in seq_len(P)) {
  cov[[i]] <- rnorm(N, 0, 1)
}
X <- do.call(cbind, cov)
```

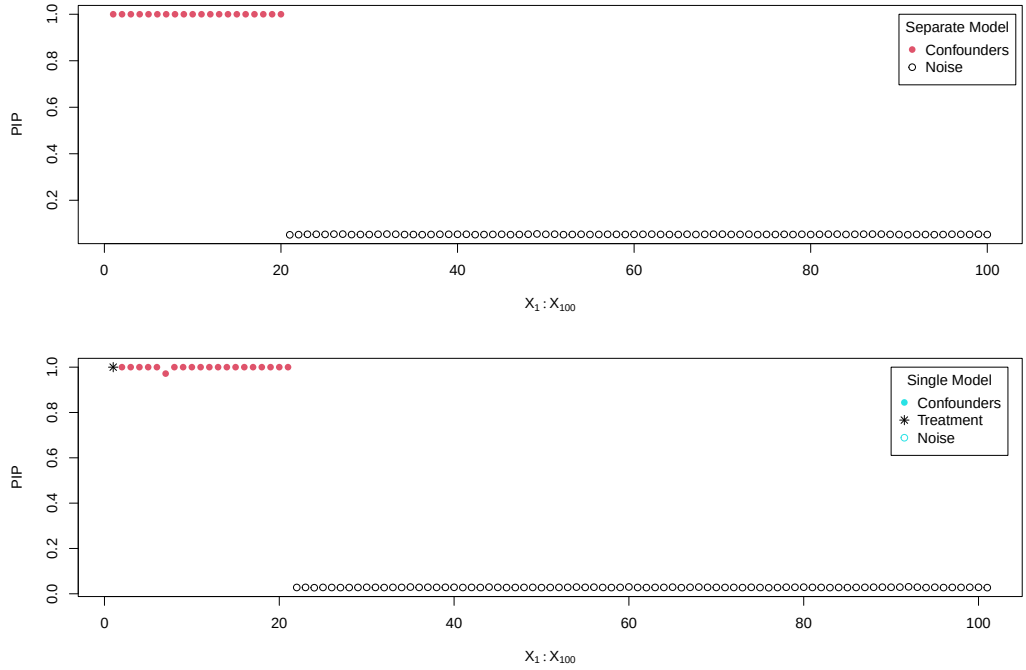


Figure A2: Average PIPs across 500 simulated datasets based on the scenario containing 20 true confounders, comparing the separate and single models. Red circles indicate the true confounders, and empty circles denote noise variables. In the single model, the first starred point represents the treatment variable.

```
h1 <- ifelse(X[, 1] < 0, 1, -1)
h2 <- ifelse(X[, 2] < 0, -1, 1)
prob <- pnorm(0.5 + h1 + h2 - 0.5 * abs(X[, 3] - 1) +
                1.5 * X[, 4] * X[, 5] + 1.5 * X[, 6] - 1 * X[, 7])
Trt <- rbinom(N, 1, prob)
mu1 <- 1 * h1 + 1.5 * h2 - 1 + 2 * abs(X[, 3] + 1) +
  2 * X[, 4] + exp(0.5 * X[, 5])
mu0 <- 1 * h1 + 1.5 * h2 - 0 + 2 * abs(X[, 3] + 1) +
  2 * X[, 4] + exp(0.5 * X[, 5])
Y1 <- rnorm(N, mu1, 0.3)
Y0 <- rnorm(N, mu0, 0.3)
Y <- Trt * Y1 + (1 - Trt) * Y0
```

When the separate model is fitted, the 95% credible intervals successfully capture the true values of SATE, E[Y(1)], and E[Y(0)], which are -0.98, 2.21, and 3.19, respectively. Similarly, the single model also provides accurate estimates of the true values.

```
separate_fit <- separate_bart(
     Y = Y, trt = Trt, X = X, num_tree = 200, num_chain = 4,
     num_burn_in = 10000, num_thin = 5, num_post_sample = 2000
)

separate_fit
`bartcs` fit by `separate_bart()`

          mean      2.5%     97.5%
SATE -0.905955 -1.172369 -0.634617
Y1    2.173088  1.976194  2.375582
Y0    3.079043  2.898203  3.261354

single_fit <- single_bart(
     Y = Y, trt = Trt, X = X, num_tree = 200, num_chain = 4,
     num_burn_in = 10000, num_thin = 5, num_post_sample = 2000
)
```

```
single_fit
`bartcs` fit by `single_bart()`

           mean      2.5%      97.5%
SATE -0.9421112 -1.116511 -0.7714885
Y1    2.1999294  2.089350  2.3124345
Y0    3.1420407  3.040003  3.2451105
```

To assess confounder selection performance, 500 simulated datasets were generated based on the data-generating process described above, including 5 true confounders and 2 instrumental variables. Figure A3 shows the average posterior inclusion probabilities obtained across these datasets, comparing results from the separate model (top panel) and the single model (bottom panel). The results clearly demonstrate that all 5 true confounders (marked with red circles) are consistently selected with posterior probabilities equal to one, whereas irrelevant noise variables are effectively excluded from the models.

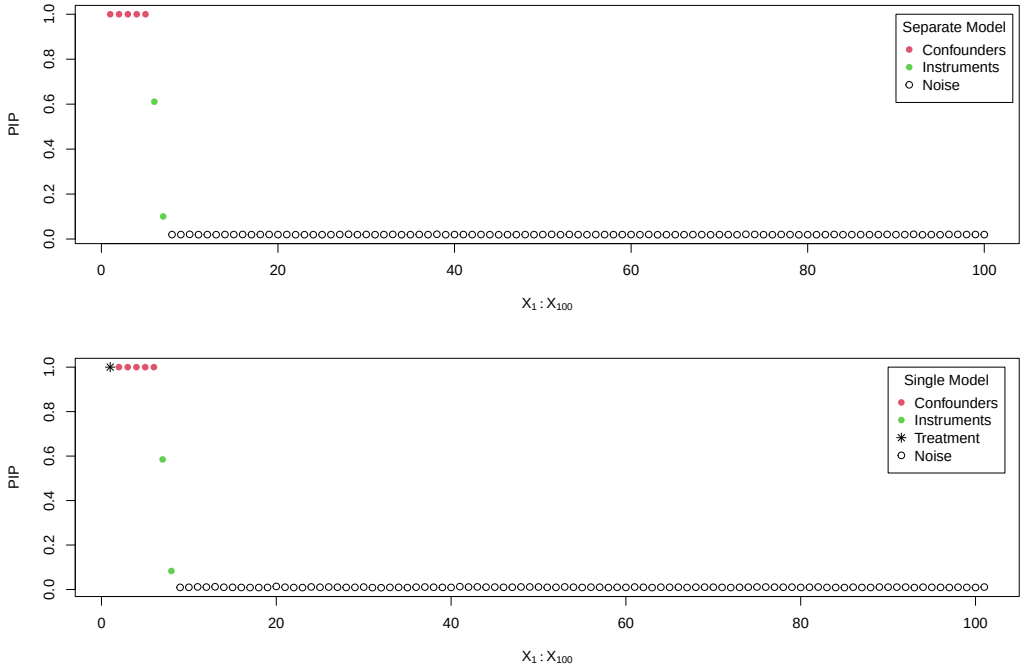


Figure A3: Average PIPs across 500 simulated datasets based on the scenario containing 5 true confounders and 2 instrumental variables, comparing the separate and single models. Red circles indicate the true confounders, green circles represent the instrumental variables, and empty circles denote noise variables. In the single model, the first starred point represents the treatment variable.

MSmix: An R Package for clustering partial rankings via mixtures of Mallows Models with Spearman distance

Marta Crispino[*], *Cristina Mollica*[*], *Lucia Modugno*
[*] *jointly first authors*

Abstract **MSmix** is a recently developed R package implementing maximum likelihood estimation of finite mixtures of Mallows models with Spearman distance for full and partial rankings. The package is designed to implement computationally tractable estimation routines, with the ability to handle arbitrary forms of partial rankings and a large number of items. The frequentist estimation task is accomplished via Expectation-Maximization algorithms, integrating data augmentation strategies to recover the unobserved heterogeneity and the missing ranks. The package also provides functionalities for uncertainty quantification of the parameter estimates, via diverse bootstrap methods and asymptotic confidence intervals. Generic methods for S3 class objects are constructed for more effectively managing the output of the main routines. The usefulness of the package and its computational performance compared with competing software is illustrated via applications to both simulated and original real ranking datasets.

1 Introduction

Ranking data play a pivotal role in numerous research and practical domains, where the focus is on comparing and ordering a set of n items according to personal preferences or other relevant criteria. From market surveys to sports competitions, from academic assessments to online recommendation systems, rankings are ubiquitous in modern society to capture human choice behaviors or, more generally, ordinal comparison processes in various contexts.

Ranking data analysis has attracted significant attention, as evidenced by the extensive literature on the subject (see Critchlow et al., 1991; Marden, 1995, for fundamental reviews) and by the development of a broad spectrum of probabilistic models designed to capture meaningful choice patterns and quantify estimation uncertainty. Traditionally, four main classes of parametric models have been identified, each representing a distinct ranking generative process. The first category, order statistics models (OSs), is originally attributed to Thurstone (1927) and conceptualizes rankings as arising from the ordering of latent item utilities. The second, paired comparison models, is exemplified by the Bradley-Terry model (BT) proposed by Bradley and Terry (1952) and is based on the possibility to decompose a ranking sequence into the corresponding set of pairwise comparisons. The third category, stagewise models, breaks the ranking process into sequential stages and is well represented by the popular Plackett-Luce model (PL) introduced by Luce (1959) and Plackett (1975). Finally, distance-based models, often referred to as *Mallows models* (MMs), trace their origins to the seminal work by Mallows (1957). In this paper, we focus on the last class, which provides an ideal option for applications where a meaningful consensus ranking can be identified in the sample and, consequently, offers a valuable parametric tool for rank aggregation tasks (Marden, 1995). For further insights into probabilistic ranking models and their unique characteristics, that can support the critical choice of suitable parametric families for specific real contexts, the reader is referred to Liu et al. (2019) and Alvo and Yu (2014).

The MM is based on the assumption that a modal consensus ranking of the n items exists in the population, effectively capturing the collective preferences. Under this framework, the likelihood of observing any particular ranking decreases as its distance from the consensus increases. While the distance measure in the MM induces distinct probabilistic models, it only needs to satisfy minimal properties. This simplicity offers researchers remarkable flexibility to choose the distance that best fits their scientific context. For example, Kendall and Cayley distances are well suited for sorting problems, Hamming works well in coding theory, and Spearman distance is particularly suitable for applications involving human preferences or social choice (Diaconis, 1988). Traditionally, the choice of the distance in the MM was mainly driven by computational considerations, specifically the availability of a closed-form expression for the model normalizing constant (or *partition function*). This favored the use of Kendall, Cayley, and Hamming distances, while the Spearman distance has been relatively underexplored due to its perceived intractability, despite its relevance in preference domains. However, Crispino et al. (2023) recently demonstrated that the Spearman distance is a metric that combines both computational feasibility and interpretability. By leveraging the unique properties of the Spearman distance, and by means of a novel approximation of the model partition function,

the authors addressed the critical inferential challenges that historically limited its application. Their approach enabled the development of an efficient strategy to fit the MM with Spearman distance (MMS) for datasets with arbitrary forms of partial rankings. Moreover, they extended the model to finite mixtures, allowing the capture of possible unobserved sample heterogeneity.

Clustering ranking data to detect and characterize groups with similar preferences has long been a major motivation for extending traditional methods beyond their basic forms. This practical need is reflected in the fact that nearly all R packages dedicated to ranking analysis include clustering methods. From a methodological perspective, the clustering problem has been tackled using both model-based strategies and machine learning approaches. For example, the **PLMIX** package (Mollica and Tardella, 2020) fits finite PL mixtures for partial top rankings within the Bayesian framework (Mollica and Tardella, 2017), which notably recovers the maximum likelihood estimation (MLE) described in Gormley and Murphy (2006) as a special case when a noninformative prior is specified. For the same parametric class, the **PlackettLuce** package (Turner et al., 2020) implements MLE procedures for generalizations of the PL model that are capable of handling partial and tied rankings, as well as including covariates to achieve model-based partitioning via PL trees. Concerning OSs, the **StatRank** package addresses the estimation of finite mixture generalizations from both full and partial rankings through the generalized method of moments (Soufiani and Chen, 2015). Additionally, the R package **Rankcluster** (Jacques et al., 2014), offers an innovative model-based clustering approach through mixtures of Insertion Sort Rank data models (Jacques and Biernacki, 2014), which is able to handle partial rankings with arbitrary patterns of incompleteness and, if needed, with multivariate (hierarchical) structures. An additional contribution to rankings with missing data is provided by the **prefmod** package (Hatzinger and Dittrich, 2012), which primarily analyzes preference data in the form of paired comparisons, hence also rankings as a by-product, by using the BT and extensions thereof to accommodate ties, subject- and item-specific covariates, as well as partial observations with different censoring forms and missingness processes. However, the extension proposed in Hatzinger and Dittrich (2012) for clustering heterogeneous data relies on the introduction of non-parametric random-effects models that, in the case of rankings, are implemented only for completely-observed sequences (Hatzinger and Maier, 2023).

Regarding the availability of software which is more related to our proposal, one can first notice that only a few packages of the Comprehensive R Archive Network (CRAN) implement MMs and generalizations thereof. **BayesMallows** (Sørensen et al., 2020) is the unique package adopting the Bayesian perspective to perform inference for the MM and its finite mixture extension. The flexibility of **BayesMallows** stands in the wide range of supported distances (including Spearman) and ranked data formats (complete and partial rankings, as well as pairwise comparisons). Moreover, **BayesMallows** provides estimation uncertainty by building posterior credible sets for the model parameters. Although Bayesian inference of ranking data is effectively addressed, the R packages adopting the frequentist perspective provide users with less flexibility and computational performance. For example, **pmr** (Lee and Yu, 2013) performs MLE of several ranking models, including the MM with Kendall, Footrule, and Spearman distances. However, despite the variety of parametric distributions, **pmr** does not handle partial rankings nor mixtures. Additionally, the estimation routines require the enumeration of all $n!$ permutations for the global search of the consensus ranking MLE and the naïve computation of the partition function, implying that the analysis of ranking datasets with $n \geq 12$ items is unfeasible. The **rankdist** package (Qian and Yu, 2019) fits mixtures of MMs with various basic and weighted metrics (Lee and Yu, 2012), including the Spearman, on a sample of either full or top-k partial rankings. While the implementation for the Kendall distance is highly efficient, it shares similar drawbacks with **pmr**, since the partition function of the MMS is computed by summing over all $n!$ permutations, and the MLE of the consensus ranking is obtained through a time-consuming local search. As a result, the procedures can be highly demanding, especially in mixture model applications. Moreover, the package does not support the analysis of full rankings with $n \geq 12$ items or of top-k rankings with $n \geq 8$ items. Other packages related to the MM, but limited to the Kendall distance, are **RMallow** (Gregory, 2020), which fits the MM and mixtures thereof to both full or partially-observed ranking data, and **ExtMallows** (Li et al., 2018), which supports the MM and the extended MM (Li et al., 2020).

Our review underscores that most of the available packages for frequentist estimation of the MM focus on distances admitting a convenient analytical expression of the model normalizing constant (more often, the Kendall), in the attempt to simplify the estimation task. Moreover, regardless of the chosen metric, these packages face common limitations, particularly in handling large datasets and partial rankings, typically restricted to top-k sequences. These computational constraints impose restrictions on the sample size, the number of items, and the censoring patterns they can feasibly handle. Finally, the current implementations generally lack methods for quantifying MLE uncertainty, particularly for the consensus ranking or when a finite mixture is assumed.

MSmix efficiently enlarges the current suite of methods for model-based clustering of full and partial rankings via mixture-based analysis, by achieving several methodological and computational advances that overcome the practical limitations experienced with the existing packages, namely: 1)

implementation of a recent normalizing constant approximation and of the closed-form MLE of the consensus ranking, to allow inference for the MMS even with a large number of items; 2) analysis of arbitrary forms of incompleteness in the observed sample of partial rankings via data augmentation strategies; 3) availability of routines for measuring estimation uncertainty of all model parameters, through bootstrap and asymptotic confidence intervals (CIs); 4) possible parallel execution of the Expectation-Maximization (EM) algorithms over multiple starting points, to better and more efficiently explore the critical mixed-type parameter space.

The paper is organized as follows. In Section 2, we first provide the methodological background of the MMS specification and its finite mixture extension within the frequentist domain. We then detail the approaches considered for the quantification of inferential uncertainty. Section 3 outlines the package architecture, the main computational aspects, and shows a comparison with existing packages. Section 4 represents the core part of the paper, illustrating the usage of the routines included in **MSmix**, with applications to brand new ranking datasets and simulations. Finally, Section 5 discusses possible directions for future releases of our package.

2 Methodological background

In this work, we consider ranking experiments where the same set of n items is presented to the assessors for comparative evaluation. Respondents can provide either a full ranking, by completely and uniquely attributing the n positions to the items, or a partial ranking, by assigning distinct positions only to a subset of the items and leaving the attribution of the remaining ranks undetermined. The novel R package **MSmix** implements finite mixtures of MMS (MMS-mix) for full and partial rankings with the following key features: i) the unranked items are treated as missing data and are implicitly assumed to occupy the non-assigned positions; ii) missing data may occur at any position within the observed partial ranking, i.e., not necessarily in bottom positions as in the top-k rankings; iii) inferential procedures rely on the implementation of EM algorithms assuming that the missing data generative process is Missing at Random (MAR), see Little (2011) and references therein for a general discussion on ignorable missingness in likelihood-based methods.

2.1 The Mallows model with Spearman distance and its mixture extension

Let $\boldsymbol{r} = (r_1, \ldots, r_n)$ be a full ranking of n items, with the generic entry r_i indicating the rank assigned to item i. A full ranking $\boldsymbol{r}$ is a permutation of the first n integers and belongs to the finite discrete space of permutations, $\mathcal{P}_n$. The MMS assumes that the probability of observing the ranking $\boldsymbol{r}$ is

$$\mathbb{P}(\boldsymbol{r}\,|\boldsymbol{\rho}, \theta) = \frac{e^{-\theta\, d(\boldsymbol{r},\boldsymbol{\rho})}}{Z(\theta)} \qquad\qquad \boldsymbol{r} \in \mathcal{P}_n,$$

where $\boldsymbol{\rho} \in \mathcal{P}_n$ is the *consensus ranking*, $\theta \in \mathbb{R}_0^+$ is the *concentration*, $d(\boldsymbol{r}, \boldsymbol{\rho}) = \sum_{i=1}^n (r_i - \rho_i)^2$ is the Spearman distance, and $Z(\theta) = \sum_{\boldsymbol{r}\in\mathcal{P}_n} e^{-\theta\, d(\boldsymbol{r},\boldsymbol{e})}$, with $\boldsymbol{e} = (1, 2, ..., n)$, is the normalizing constant.

Let $\underline{\boldsymbol{r}} = \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_N\}$ be a random sample of N full rankings drawn from the MMS and N_l be the frequency of the l-th distinct observed ranked sequence $\boldsymbol{r}_l$, such that $\sum_{l=1}^L N_l = N$. As shown in Crispino et al. (2023), the observed-data log-likelihood can be written as follows

$$\ell(\boldsymbol{\rho}, \theta|\underline{\boldsymbol{r}}) = -N\left(\log Z(\theta) + 2\theta\left(c_n - \boldsymbol{\rho}^T\bar{\boldsymbol{r}}\right)\right),$$

where $c_n = n(n+1)(2n+1)/6$, the symbol ^{T} denotes the transposition (row vector), $\bar{\boldsymbol{r}} = (\bar{r}_1, \ldots, \bar{r}_n)$ is the sample mean rank vector whose i-th entry is $\bar{r}_i = \frac{1}{N}\sum_{l=1}^L N_l r_{li}$, and $\boldsymbol{\rho}^T\bar{\boldsymbol{r}} = \sum_{i=1}^n \rho_i \bar{r}_i$ is the scalar product. The MLE of the consensus ranking is given by the ranking arising from ordering the items according to their sample average rank,

$$\hat{\boldsymbol{\rho}} = (\hat{\rho}_1, \ldots, \hat{\rho}_i, \ldots, \hat{\rho}_n) \quad \text{with} \quad \hat{\rho}_i = \text{rank}(\bar{\boldsymbol{r}})_i\,.$$

The MLE $\hat{\theta}$ of the concentration parameter is the value equating the expected Spearman distance under the MMS, $\mathbb{E}_\theta(D)$, to the sample average Spearman distance $\overline{d} = \frac{1}{N}\sum_{l=1}^L N_l d(\boldsymbol{r}_l, \hat{\boldsymbol{\rho}})$. The root of this equation can be found numerically, provided that one can evaluate the expected Spearman distance, given by

$$\mathbb{E}_\theta[D] = \frac{\sum_{\boldsymbol{r}\in\mathcal{P}_n} d(\boldsymbol{r},\boldsymbol{e}) e^{-\theta\, d(\boldsymbol{r},\boldsymbol{e})}}{Z(\theta)} = \frac{\sum_{d\in\mathcal{D}_n} d N_d\, e^{-d\theta}}{\sum_{d\in\mathcal{D}_n} N_d\, e^{-d\theta}},$$

with $\mathcal{D}_n = \left\{d = 2h \,:\, h \in \mathbb{N}_0 \text{ and } 0 \le d \le 2\binom{n+1}{3}\right\}$ and $N_d = |\{\boldsymbol{r} \in \mathcal{P}_n \,:\, d(\boldsymbol{r},\boldsymbol{e}) = d\}|$. The exact

values of the frequencies N_d are available for $n \leq 20$ (sequence A175929 in the Online Encyclopedia of Integer Sequences). In order to tackle inference on rankings of a larger number of items, Crispino et al. (2023) introduced an approximation of the Spearman distance distribution. In **MSmix**, we implement their strategy, so that when the normalizing constant and the expected Spearman distance cannot be computed exactly, inference targets an approximation. Algorithm 1 reported in Appendix A1 illustrates the steps described above.

In order to account for the unobserved sample heterogeneity typical in real ranking data and, more generally, to increase the model flexibility, an MMS-mix can be adopted. Under the MMS-mix, the sampling distribution is assumed to be

$$\mathbb{P}(\boldsymbol{r}|\underline{\boldsymbol{\rho}}, \boldsymbol{\theta}, \boldsymbol{\omega}) = \sum_{g=1}^{G} \omega_g \mathbb{P}(\boldsymbol{r}\,|\boldsymbol{\rho}_g, \theta_g) = \sum_{g=1}^{G} \omega_g \frac{e^{-2\theta_g\left(c_n - \boldsymbol{\rho}_g^T \boldsymbol{r}\right)}}{Z(\theta_g)} \qquad \boldsymbol{r} \in \mathcal{P}_n,$$

with ω_g and $(\boldsymbol{\rho}_g, \theta_g)$ denoting respectively the weight and the pair of MMS parameters of the g-th mixture component. Murphy and Martin (2003) first proposed an EM algorithm to fit such mixture models, but the more efficient version described by Crispino et al. (2023) is implemented in the **MSmix** package (Algorithm 2 in Appendix A1).

2.2 Inference on partial rankings

MSmix implements two schemes to draw inference from partial rankings with arbitrary types of censoring. One is the recent proposal of Crispino et al. (2023), which extends the method originally described by Beckett (1993) to the finite mixture framework. The key idea is to augment each distinct partially observed ranking $\boldsymbol{r}_l$ with the corresponding set $\mathcal{C}(\boldsymbol{r}_l) \subset \mathcal{P}_n$ of compatible full rankings and then maximize the complete-data log-likelihood

$$\ell_c(\underline{\boldsymbol{\rho}}, \boldsymbol{\theta}, \boldsymbol{\omega}, \underline{\boldsymbol{z}}, \underline{\boldsymbol{r}}^* | \underline{\boldsymbol{r}}) = \sum_{m=1}^{M} \sum_{g=1}^{G} N_m z_{mg} \left(\log \omega_g - 2\theta_g \left(c_n - \boldsymbol{\rho}_g^T \boldsymbol{r}_m^* \right) - \log Z(\theta_g) \right), \tag{1}$$

where $\boldsymbol{r}_m^*$ is a generic full ranking belonging to $\mathcal{C}(\boldsymbol{r}_l)$, N_m is its latent frequency, $\sum_{m=1}^{M} N_m = |\cup_{l=1}^{L} \mathcal{C}(\boldsymbol{r}_l)|$, and $\boldsymbol{z}_m = (z_{m1}, \ldots, z_{mG})$ is its latent group membership. The algorithm to maximize (1) is outlined in Algorithm 3 in Appendix A1.

Algorithm 3 requires the computationally intensive construction and iterative computations on the sets $\mathcal{C}(\boldsymbol{r}_l)$ associated to each partial observation. This typically demands a lot of memory, especially in the case of many censored positions (greater than 10, say) and large sample sizes. To address this issue, in **MSmix** we propose the use of a second scheme to draw inference on partial rankings, that uses a Monte Carlo (MC) step in place of the complete augmentation, giving rise to a MCEM-type algorithm (Wei and Tanner, 1990). Let $\kappa > 0$ be a tuning constant and $\mathcal{I}_s \subset \{1, 2, \ldots, n\}$ be the subset of items actually ranked in the observed partial ranking $\boldsymbol{r}_s$.[1] The core idea is to iteratively complete the missing ranks by sampling from the postulated MMS-mix conditionally on the current values of the parameters. Specifically, the MC step is designed as follows:

MC step: for $s = 1, \ldots, N$, simulate

$$\tilde{\boldsymbol{z}}_s \,|\, \hat{\boldsymbol{z}}_s \sim \text{Multinom}\big(1, (\hat{z}_{s1}, \ldots, \hat{z}_{sG})\big) \tag{2}$$

$$\tilde{\boldsymbol{r}}_s \,|\, \underline{\boldsymbol{\rho}}, \boldsymbol{\theta}, \tilde{\boldsymbol{z}}_s \sim \sum_{g=1}^{G} \tilde{z}_{sg} \mathbb{P}\left(\boldsymbol{r}|\boldsymbol{\rho}_g, \kappa\theta_g\right) \tag{3}$$

and complete the partial ranking $\boldsymbol{r}_s$ with the full sequence $\boldsymbol{r}_s^* = (r_{s1}^*, \ldots, r_{sn}^*)$ such that $r_{si}^* = r_{si}$ for $i \in \mathcal{I}_s$ whereas, for $i \notin \mathcal{I}_s$, the positions must be assigned to the items so that their relative ranks match those in $\tilde{\boldsymbol{r}}_s$.

The tuning constant in (3) serves to possibly increase the variability (for $0 < \kappa < 1$) or the concentration (for $\kappa > 1$) of the sampled rankings around the current consensus ranking. The MCEM scheme is detailed in Algorithm 4 in Appendix A1.

[1] Note that for a better account of sampling variability and exploration of the parameter space, the MCEM algorithm works at the level of the single observed units, indexed by $s = 1, \ldots, N$, instead of the aggregated data $(\boldsymbol{r}_l, N_l)_{l=1,\ldots,L}$.

2.3 Uncertainty quantification

To quantify estimation uncertainty, we constructed confidence sets using both asymptotic likelihood theory and bootstrap procedures.

Concerning the former approach, Critchlow (1985) showed that, although the MMS-mix model is not regular due to the presence of the discrete component $\mathcal{P}_n$ in the parameter space, the likelihood asymptotically behaves as if the consensus ranking parameters were known (Marden, 1995). This result justifies the construction of CIs based on the asymptotic likelihood theory for the continuous parameters of the MMS-mix. In particular, we adopt the methodology described in Mclachlan and Peel (2000), which allows us to derive the standard errors from the output of the EM algorithm without an additional computational burden.

Since asymptotic CIs rely on large sample approximations, their validity depends on having a sufficiently large sample size. This is especially crucial in mixture models, where the required sample size must be very large (Mclachlan and Peel, 2000). Therefore, we also employ a non-parametric bootstrap approach (Efron, 1982). Specifically, for $b = 1, \ldots, B$, we draw with replacement a sample $\underline{\boldsymbol{r}}^{(b)} = \{\boldsymbol{r}_1^{(b)}, \ldots, \boldsymbol{r}_N^{(b)}\}$ from the observed data $\underline{\boldsymbol{r}}$, and then compute the MLEs $\hat{\boldsymbol{\rho}}^{(b)}$ and $\hat{\theta}^{(b)}$.[2] Then, to summarize the uncertainty on $\hat{\boldsymbol{\rho}}$, we construct itemwise CIs, providing plausible sets of ranks separately for each item. To guarantee narrower intervals as well as a proper account of possible multimodality, these are obtained as highest probability regions of the n bootstrap first-order marginals, that is the sets of most likely ranks for each item at the given $100(1-\alpha)\%$ level of confidence. We also provide a way to visualize the variability of the bootstrap MLEs through a heatmap of the corresponding first-order marginals, that is, the $n \times n$ matrix whose $(i,j)-$th element is given by $\frac{1}{B}\sum_{b=1}^{B} \mathbb{I}_{[\hat{\rho}_i^{(b)}=j]}$. For the continuous concentration parameter, the bounds of the $100(1-\alpha)\%$ CIs are determined as the quantiles at level $\alpha/2$ and $(1-\alpha/2)$ of the MLE bootstrap sample.

In the presence of multiple mixture components ($G > 1$), the bootstrap CIs of the component-specific parameters are determined using the non-parametric bootstrap method applied on each subsample of rankings allocated to the G clusters (Taushanov and Berchtold, 2019). We considered two approaches to perform this allocation: i) the deterministic *Maximum A Posteriori* (MAP) classification (*separated method*) or ii) a simulated classification at each iteration b from a multinomial distribution with the estimated posterior membership probabilities $\hat{\underline{\boldsymbol{z}}}$ (*soft method*). The key difference between the two methods is that the separated one ignores the uncertainty in cluster assignment, hence, it does not return CIs for the mixture weights and, in general, leads to narrower CIs for the component-specific parameters. In contrast, the soft method accounts for this uncertainty, allowing the construction of intervals for the mixture weights and providing more conservative CIs.

3 Package architecture and implementation

The **MSmix** package is available on the CRAN at https://cran.r-project.org/web/packages/MSmix. The software is mainly written in R language, but several strategies have been designed to effectively address the computational challenges, especially related to the analysis of large samples of partial rankings with a wide set of alternatives. The key approaches adopted to limit execution time and memory load are described below.

- Even though the input ranking dataset is required in non-aggregated form, as detailed in Section 4.1, most of the proposed inferential algorithms first determine the frequency distribution of the observations, and then work at aggregated level. This step reduces data volume and, consequently, the overall computational burden.
- For very large n, the approximate Spearman distance distribution is evaluated over a predefined grid of distance values. This approach prevents the computation of frequencies N_d from becoming numerically intractable or prohibitive, both in terms of computational time and memory allocation.
- The ranking spaces $\mathcal{P}_n$ for $n \leq 11$, needed for the data augmentation of partial rankings in Algorithm 3, are internally stored in the package and available for offline use.
- **MSmix** is one of the few R packages for ranking data which includes the parallelization option of the iterative estimation procedures over multiple initializations. This is crucial to guarantee a good parameter space exploration and convergence achievement at significantly reduced costs in terms of execution time.

[2] For full rankings and a single mixture component, the **MSmix** package also offers the parametric bootstrap method, where each simulated sample $\underline{\boldsymbol{r}}^{(b)}$ is obtained by randomly sampling from the fitted MMS rather than from the observed data.

Table 1: Characteristics of the existing R packages for the MLE of MMS mixtures.

	Full rankings		Top partial		Arbitrary partial	
	$G = 1$	$G > 1$	$G = 1$	$G > 1$	$G = 1$	$G > 1$
pmr	✓	✗	✗	✗	✗	✗
rankdist	✓	✓	✓	✓	✗	✗
MSmix	✓	✓	✓	✓	✓	✓

- The implementation of some critical steps is optimized with a call to functions coded in the C++ language, such as the essential computation of the Spearman distance.

According to their specific task, the objects contained in **MSmix** can be grouped into five main categories, namely

Ranking data functions: objects denoted with the prefix `"data_"` that allow to apply several transformations or summaries to the ranking data.

Model functions: all the routines aimed at performing an MMS-mix analysis.

Ranking datasets: objects of class `"data.frame"` denoted with the prefix `"ranks_"`, which collect the observed rankings in the first n columns and possible covariates. Most of them are original datasets never analyzed earlier in the literature.

Spearman distance functions: a series of routines related to the Spearman distance computation and its distributional properties.

S3 class methods: generic functions for the S3 class objects associated with the main routines.

In Section 4, we extensively describe the usage of the above objects through applications on simulated and real-world data.

3.1 Performance benchmarking

The algorithms developed in **MSmix** result in impressive gains in terms of overall efficiency compared to the few existing R packages for the frequentist analysis of ranking data with the MMS, that is, **pmr** and **rankdist**. Their general characteristics are outlined in Table 1, highlighting the greater flexibility of **MSmix** to handle different forms of partial rankings in a finite mixture framework.

Table 2 reports the execution times for an experiment with full rankings and $G = 1$, representing the only case supported by all the three packages. Specifically, we simulated $N = 100$ full rankings from the MMS with increasing number of items n and then fitted the true model. The comparison shows that **MSmix** outperforms the other packages in all scenarios and its remarkable speed seems almost not to be impacted by n, at least up to $n = 20$. This happens because, for the homogeneous case, **MSmix** exploits the theoretical properties of the Spearman distance and conveniently implements the MLEs as a one-step procedure, without the need to iterate (nor to locally search).

The results of two additional experiments, both supported exclusively by **MSmix** and **rankdist**, are reported in Table 3. The first (left panel) concerns inference of a basic MMS on top partial rankings: we simulated $N = 100$ full rankings of $n = 7$ items from the MMS, and then censor them with decreasing

Table 2: Comparison among **MSmix**, **rankdist** and **pmr** in terms of computational times (seconds) to fit the basic MMS ($G = 1$) on full rankings with increasing number of items. Note: *not run* indicates that we did not perform the fit because of the excessive computing time and the symbol ✗ indicates that the fit is not supported.

	MSmix	rankdist	pmr
$n = 5$	0.004	0.01	0.263
$n = 6$	0.004	0.028	3.955
$n = 7$	0.003	0.276	137.781
$n = 8$	0.004	2.748	*not run*
$n = 9$	0.004	32.1	*not run*
$n = 10$	0.004	538.71	*not run*
$n = 15$	0.004	✗	✗
$n = 20$	0.004	✗	✗
$n = 50$	0.031	✗	✗
$n = 100$	0.485	✗	✗

Table 3: Comparison between **Msmix** and **rankdist** to fit a basic MMS on partial top$-k$ rankings (left) and a MMS-mix with $G = 2$ components on full rankings (right). Computational times (in seconds) averaged over 100 independent replications.

	MSmix	**rankdist**
$k = 5$	0.029	0.301
$k = 4$	0.041	0.321
$k = 3$	0.064	0.386
$k = 2$	0.103	0.543
$k = 1$	0.122	0.673

	MSmix	**rankdist**
$n = 5$	0.049	0.089
$n = 6$	0.035	0.185
$n = 7$	0.023	0.262
$n = 8$	0.024	0.411
$n = 9$	0.018	0.612

number of top-k ranked items. The second (right panel) concerns inference of MMS-mix with full rankings: we simulated $N = 100$ full rankings of increasing length n from the MMS-mix with $G = 2$ components, and then estimated the true model. Again, **MSmix** turns out to be particularly fast and more efficient when compared to the competing package. Moreover, the choice of $n = 7$ is motivated by the fact that **rankdist** only works with a maximum of 7 items in the case partial rankings are considered.

The comparative analysis of this section was performed using R version 4.4.0 on a macOS Monterey 12.7.3 (2.5GHz Intel Core i7 quad-core). For further results and discussion on the computational performance of the **MSmix** package, see Appendix A2.

4 Using the MSmix package

4.1 Data format

The knowledge of the data format adopted in a package is, especially for ranked sequences, crucial before safely conducting any ranking data analysis. The **MSmix** package privileges the ranking data format, which is a natural choice for the MM, and the non-aggregate form, meaning that observations must be provided as an integer $N \times n$ `matrix` or `data.frame` with each row representing individual observed partial rankings. Missing positions must be coded as `NA`s and ties are not allowed.

We start the illustration of the main functionalities of **MSmix** by using a new full ranking dataset contained in the package, called `ranks_antifragility`. This dataset, stemming from a 2021 survey on Italian startups during the COVID-19 outbreak, collects rankings of $n = 7$ crucial antifragility features.[3] Since covariates are also included, the $N = 99$ full rankings can be extracted from the first $n = 7$ columns as follows

```
R> n <- 7
R> ranks_AF <- ranks_antifragility[, 1:n]
R> str(ranks_AF)

'data.frame':        99 obs. of  7 variables:
 $ Absorption        : int  4 1 3 4 2 2 1 2 4 4 ...
 $ Redundancy        : int  2 4 4 2 3 1 4 1 3 3 ...
 $ Small_stressors   : int  1 3 1 7 4 6 5 4 6 6 ...
 $ Non_monotonicity  : int  3 2 2 1 1 3 2 5 1 7 ...
 $ Requisite_variety : int  5 7 7 3 7 7 7 3 7 2 ...
 $ Emergence         : int  6 6 6 6 6 5 6 7 2 1 ...
 $ Uncoupling        : int  7 5 5 5 5 4 3 6 5 5 ...
```

To facilitate the visualization of the outputs, let us shorten the item labels, and then see the appearance of the rankings provided by the very first three startups.

```
R> names(ranks_AF) <- substr(x = names(ranks_AF), start = 1, stop = 3)
R> ranks_AF[1:3, ]
  Abs Red Sma Non Req Eme Unc
1   4   2   1   3   5   6   7
2   1   4   3   2   7   6   5
3   3   4   1   2   7   6   5
```

[3]The antifragility properties reflect a company's ability to not only adapt but also improve its activity and grow in response to stressors, volatility and disorders caused by critical and unexpected events.

The switch to the ordering format (and vice versa) can be easily realized with the `data_conversion` routine, that has the flexibility to support partial sequences with arbitrary patterns of censoring. Here is the transformation into orderings of the above three full rankings.

```
R> data_conversion(data = ranks_AF[1:3, ])
  [,1] [,2] [,3] [,4] [,5] [,6] [,7]
1    3    2    4    1    5    6    7
2    1    4    3    2    7    6    5
3    3    4    1    2    7    6    5
```

4.2 Data description and manipulation

Descriptive statistics and other useful sample summaries can be obtained with the `data_description` routine that, differently from analogous functions supplied by other R packages, can handle partial observations with arbitrary type of censoring. The output is a list of S3 class `"data_descr"`, whose components can be displayed with the `print.data_descr` method. For the entire Antifragility sample, the basic application of these commands is the following

```
R> data_descr_AF <- data_description(rankings = ranks_AF)
R> print(data_descr_AF)

Sample size: 99
N. of items: 7

Frequency distribution of the number of ranked items:
 1  2  3  4  5  6  7
 0  0  0  0  0  0 99

Number of missing positions for each item:
Abs Red Sma Non Req Eme Unc
  0   0   0   0   0   0   0

Mean rank of each item:
 Abs  Red  Sma  Non  Req  Eme  Unc
2.45 3.27 4.02 2.71 5.38 5.01 5.15

Borda ordering:
"Abs" "Non" "Red" "Sma" "Eme" "Unc" "Req"

First-order marginals:
      Abs Red Sma Non Req Eme Unc Sum
Rank1  37  13   6  34   3   3   3  99
Rank2  28  25  10  18   3   9   6  99
Rank3  13  20  22  18  10   7   9  99
Rank4   6  18  28  16  11   9  11  99
Rank5   6  12  14   4  19  20  24  99
Rank6   6   8   9   3  16  39  18  99
Rank7   3   3  10   6  37  12  28  99
Sum    99  99  99  99  99  99  99 693

Pairwise comparison matrix:
    Abs Red Sma Non Req Eme Unc
Abs   0  67  80  52  86  83  82
Red  32   0  63  41  79  79  75
Sma  19  36   0  33  75  68  64
Non  47  58  66   0  86  84  84
Req  13  20  24  13   0  43  47
Eme  16  20  31  15  56   0  59
Unc  17  24  35  15  52  40   0
```

where the two displayed matrices correspond, respectively, to the first-order marginals, with the (j, i)-th entry indicating the number of times that item i is ranked in position j, and the pairwise comparison matrix, with the (i, i')-th entry indicating the number of times that item i is preferred to item i'. The function `data_description` also includes an optional `subset` argument which allows to summarize specific subsamples defined, for example, through a condition on some of the available

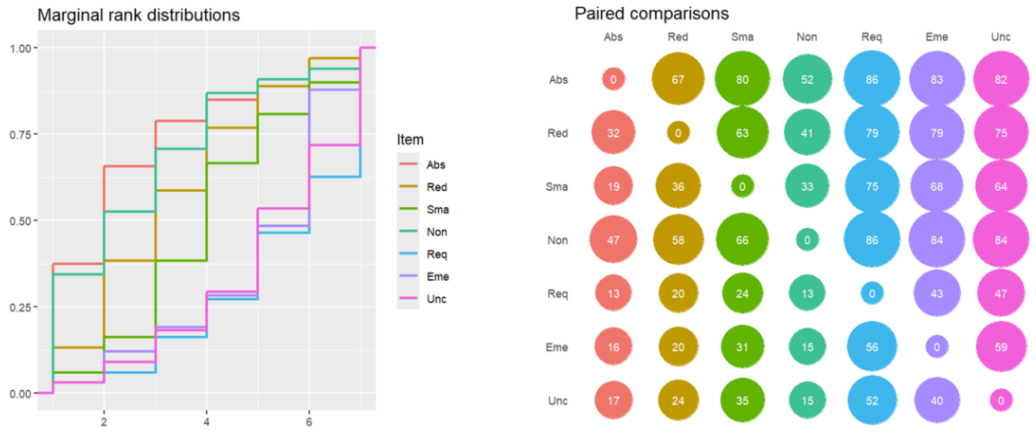


Figure 1: ECDFs of the marginal rank distributions (left) and bubble plot of the pairwise comparison matrix (right) for the Antifragility dataset. These plots correspond, respectively, to the elements named `ecdf` and `pc` of the output list returned by the generic method `plot.data_descr`.

covariates. The idea is to facilitate a preliminary exploration of possible different preference patterns influenced, for example, by some of the observed subjects' characteristics.

Finally, we created a further generic method for the class `"data_descr"` to offer a more attractive and intuitive rendering of the fundamental summaries, that is, the function `plot.data_descr`. This method produces a list of five plots by relying on the fancy graphical tools implemented in the **ggplot2** package (Wickham, 2016), namely: 1) the barplot with the percentages of the number of ranked items in the observed rankings, 2) the pictogram of the mean rank vector, 3) the heatmap of the first-order marginals (either by item or by rank), 4) the Empirical Cumulative Distribution Functions (ECDFs) of the marginal rank distributions and 5) the bubble plot of the pairwise comparison matrix. For the Antifragility dataset, the following code snippet illustrates the creation of the above graphics and how to access, separately, to the ECDFs and the bubble plot displayed in Figure 1.

```
R> p_descr_AF <- plot(data_descr_AF)
R> p_descr_AF$ecdf()
R> p_descr_AF$pc()
```

Concerning ranking data manipulation, **MSmix** provides functions designed to switch from complete to partial sequences, with the routine `data_censoring`, or from partial to complete sequences, with the routines `data_augmentation` and `data_completion`. These functions are particularly useful in simulation scenarios for evaluating the robustness of inferential procedures in recovering the actual data-generating mechanisms under various types and extents of censoring and different data augmentation strategies for handling partial data.

With `data_censoring`, complete rankings can be converted into partial rankings in two distinct ways. One approach obscures only the bottom ranks to produce a top partial ranking (set `topk = TRUE`), while the other obscures ranks at any position (set `topk = FALSE`). In both cases, users can specify how many positions are retained for each sequence through two schemes: (i) a deterministic method, where an integer vector (of length N) is provided to the `nranked` argument specifying the desired number of ranks to be retained in each partial sequence; (ii) a stochastic method, where `nranked = NULL` (default) and a numeric vector is provided to the `probs` argument defining the $(n-1)$ probabilities associated with retaining different number of ranks (from 1 to $n-1$). These probabilities determine the random number of ranks to be retained in each partial sequence after censoring.[4] An example of a deterministic top-k censoring scheme is implemented below to covert the complete Antifragility ranking data into top-3 rankings.

```
R> N <- nrow(ranks_AF)
R> top3_AF <- data_censoring(rankings = ranks_AF, topk = TRUE, nranked = rep(3,N))
R> top3_AF$part_rankings[1:3,]
  Abs Red Sma Non Req Eme Unc
1  NA   2   1   3  NA  NA  NA
2   1  NA   3   2  NA  NA  NA
3   3  NA   1   2  NA  NA  NA
> table(top3_AF$nranked)
```

[4]Recall that a partial sequence with $(n-1)$ observed entries corresponds to a full ranking.

```
 3
99
```

The output of `data_censoring` is a list with a first component, named `part_rankings`, corresponding to the input complete data matrix `rankings` with suitably censored (`NA`) entries, and a second component, named `nranked`, corresponding to the vector with the number of actually visible positions in each partial ranking.

An example of stochastic top-k censoring scheme on the same dataset, that will result in a random number of bottom positions obscured, can be run as follows

```
R> top_AF <- data_censoring(rankings = ranks_AF, topk = TRUE, probs = c(1:(n-2),0))
R> top_AF$part_rankings[1:3,]
  Abs Red Sma Non Req Eme Unc
1   4   2   1   3   5  NA  NA
2   1  NA   3   2  NA  NA  NA
3   3  NA   1   2  NA  NA  NA

R> table(top_AF$nranked)
 1  2  3  4  5
 1 14 16 19 49
```

In this case, the vector `probs` assigns an increasing chance of retaining a higher number of top positions, with the exception of a zero value in the last entry, forcing the non-occurrence of full rankings after censoring. Apart from the different setting for the `topk` argument, applying a censoring scheme to arbitrary positions requires a similar syntax to the top-k. The main difference is that, instead of the censoring process acting only on the bottom part of the rankings, the positions to be censored are determined uniformly at random once the number of ranks to be kept is specified by the user (either deterministically or stochastically).

We conclude this section with an illustration of the counterpart commands of `data_censoring` available in **MSmix**, which act on partial rankings and fill in the missing positions with different criteria. The first, called `data_augmentation`, is the key function for estimating a MMS-mix on partial rankings via Algorithm 3. Here is a toy example with only two partial rankings characterized by different types of censoring.

```
R> ranks_toy <- rbind(c(2, NA, 1, NA, 3), c(NA, 4, NA, 1, NA))
R> ranks_toy
     [,1] [,2] [,3] [,4] [,5]
[1,]    2   NA    1   NA    3
[2,]   NA    4   NA    1   NA

R> data_augmentation(rankings = ranks_toy)
[[1]]
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    4    1    5    3
[2,]    2    5    1    4    3

[[2]]
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    4    3    1    5
[2,]    3    4    2    1    5
[3,]    3    4    5    1    2
[4,]    2    4    5    1    3
[5,]    5    4    2    1    3
[6,]    5    4    3    1    2
```

The output list contains the matrices of all full rankings compatible with each partial sequence.[5] We remark that, despite the name `rankings` of the input (partially ranked) data matrix, the function `data_augmentation` can also be applied to partial observations expressed in ordering format. In general, it supports the data augmentation of sequences containing at most 10 missing entries.

The second function, named `data_completion`, completes each partial ranking with a single compatible full ranking. To complete the rankings in `ranks_toy`, one needs to set the `ref_rho` argument equal to a matrix of the same dimensions as `ranks_toy`, containing the reference full rankings in each row. In the example below, we use the identity permutation and its opposite as the reference sequences for completion.

[5] These correspond to the sets $\mathcal{C}(\boldsymbol{r})$ introduced in Section 2.2.

```
R> data_completion(rankings = ranks_toy, ref_rho = rbind(1:5, 5:1))
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    4    1    5    3
[2,]    5    4    3    1    2
```

The output is the matrix obtained by filling in the missing entries of each partial sequence with the relative positions of the unranked items according to the reference full ranking.[6] The `data_completion` command accommodates any type of censoring, similar to `data_augmentation`, but without the need to enumerate all possible orders of missing positions. Consequently, there is no upper limit on the number of `NA` entries in the partial sequences.

4.3 Sampling

The function devoted to simulating an i.i.d. sample of full rankings from a MMS-mix is `rMSmix`, which relies on the Metropolis-Hastings (MH) procedure implemented in the R package **BayesMallows** (Sørensen et al., 2020). When $n \leq 10$, the routine also offers the possibility to perform exact sampling by setting the logical `mh` argument to `FALSE`.

The `rMSmix` function requires the user to specify: i) the desired number of rankings (`sample_size`), ii) the number of items (`n_items`) and iii) the number of mixture components (`n_clust`). The mixture parameters can be separately passed with the (optional) arguments `rho`, `theta` and `weights`, set to `NULL` by default. If the user does not input the above parameters, the concentrations are sampled uniformly in the interval $(1/n^2, 3/n^{3/2})$,[7] while the simulation of the consensus parameters and the weights can be selected with the logical argument `uniform`. The option `uniform = TRUE` consists in generating the non-specified parameters uniformly in their support. Here is an example where $N = 100$ full rankings of $n = 8$ items are exactly generated from a 3-component MMS-mix, with assigned and equal concentrations $\boldsymbol{\theta} = (.15, .15, .15)$ and the other parameters sampled uniformly at random.

```
R> sam_unif <- rMSmix(sample_size = 100, n_items = 8, n_clust = 3, theta = rep(.15, 3),
+               uniform = TRUE, mh = FALSE)
```

The function `rMSmix` returns a list of five named objects: the $N \times n$ matrix with the simulated complete rankings (`samples`), the model parameters actually used for the simulation (`rho`, `theta` and `weights`) and the simulated group membership labels (`classification`). For the previous example, they can be extracted as follows

```
R> sam_unif$samples[1:3,]
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    6    1    7    5    8    3    2    4
[2,]    2    1    3    7    5    4    8    6
[3,]    6    2    7    5    8    3    1    4

R> sam_unif$rho
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    6    2    1    5    4    3    8    7
[2,]    4    2    5    3    8    7    1    6
[3,]    6    2    8    4    7    3    1    5

R> sam_unif$weights
[1] 0.49165535 0.04123627 0.46710838

R> table(sam_unif$classification)
 1  2  3
35  5 60
```

One can note that, with uniform sampling, cluster separation and balance of the drawings among the mixture components are not guaranteed. In fact, cluster 2 has a very small weight ($\omega_2 \approx 0.04$) corresponding to only 5 observations; moreover, the consensus rankings of clusters 2 and 3 are quite similar, as testified by their low relative Spearman distance obtained by dividing the output of command `spear_dist` included in **MSmix** by the maximum value of the metric.[8]

[6] These sequences correspond to the result of data completion from the MC step described in Section 2.2.

[7] The concentration parameters play a delicate role. In fact, if θ is too close to zero, the MMS turns out to be indistinguishable from the uniform distribution on $\mathcal{P}_n$, while if θ is too large the MMS distribution would tend to a Dirac on the consensus ranking $\boldsymbol{\rho}$. The critical magnitude turns out to be $\theta \sim c/n^2$ with $c > 0$ fixed (Zhong, 2021).

[8] The maximum Spearman distance among two rankings of a given length n is equal to $2\binom{n+1}{3}$.

(a) Samples with uniformly generated parameters.

(b) Samples from separated clusters.

Figure 2: Heatmap of the Spearman distance matrix between all pairs of full rankings for two simulated samples from a 3-component MMS-mix, obtained by setting `uniform = TRUE` (a) and `uniform = FALSE` (b) in the `rMSmix` routine.

```
R> max_spear_dist <- 2*choose(8+1,3)
R> spear_dist(rankings = sam_unif$rho[2,], rho = sam_unif$rho[3,])/max_spear_dist
[1] 0.1904762
```

To ensure separation among the mixture components and non-sparse weights, the user can set the option `uniform = FALSE`. Specifically, the consensus rankings are drawn with a minimum Spearman distance from each other equal to $\frac{2}{G}\binom{n+1}{3}$, and the mixing weights are sampled from a symmetric Dirichlet distribution with (large) shape parameters $\boldsymbol{\alpha} = (2G,\ldots,2G)$ to favour populated and balanced clusters.

```
R> sam_sep <- rMSmix(sample_size = 100, n_items = 8, n_clust = 3, theta = rep(.15, 3),
+                    uniform = FALSE, mh = FALSE)
```

The three clusters are now more balanced and their central rankings have a larger relative distance.

```
R> sam_sep$weights
[1] 0.5214495 0.2594782 0.2190723

R> spear_dist(rankings = sam_sep$rho)/max_spear_dist
     1         2
2 0.6309524
3 0.7023810 0.6666667
```

In Figure 2, we show the separation among clusters in the two examples through the Spearman distance matrix of the simulated samples, which quantifies the dissimilarity between each pair of observations. Specifically, Figures 2a and 2b can be constructed as follows[9]

```
R> plot(spear_dist(rankings = sam_unif$samples), show_labels = FALSE)
R> plot(spear_dist(rankings = sam_sep$samples), show_labels = FALSE)
```

where the argument `show_labels = FALSE` allows to drop the labels of the observations over the axes in the case of large samples. The heatmaps indicate the presence of only two well-separated clusters in the sample obtained with uniformly generated parameters (Figure 2a), while three groups are evident when the simulation is performed by controlling the distance among components (Figure 2b).

In conclusion, `rMSmix` is designed to facilitate the implementation of alternative sampling schemes, that can be fruitful to assess the performance of the inferential procedures and their robustness under a variety of simulation scenarios.

4.4 Application on full rankings

In this section, we show how to perform a mixture model analysis on the Antifragility rankings. To this aim, we use the command `fitMSmix`, the core function of the **MSmix** package, which performs

[9]Notably, the `plot.dist` function of **MSmix** fills in the gap of a generic method for objects of class `"dist"` in R, since it allows to visualize, and hence compare, distance matrices of any metric.

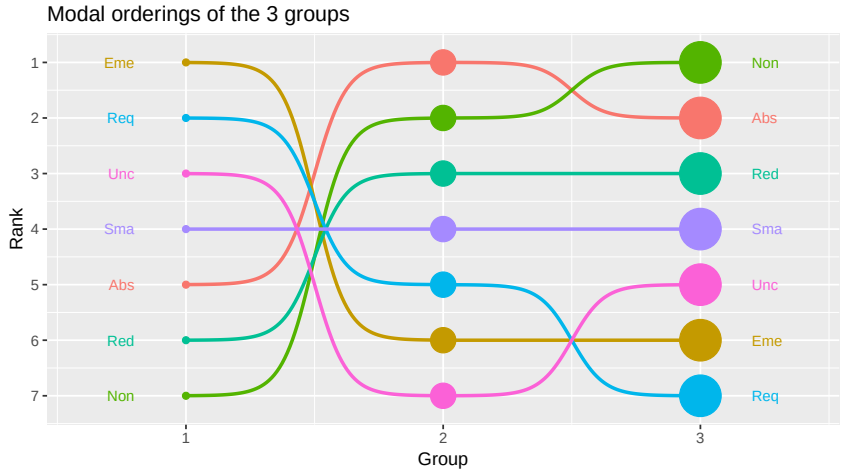


Figure 3: Bump plot depicting the estimated consensus rankings of the $G = 3$ clusters for the `ranks_antifragility` dataset. This plot corresponds to the element named `bump_plot` of the output list returned by the generic method `plot.emMSmix`.

MLE of the MMS-mix on the input `rankings` via EM algorithm with the desired number `n_clust` of components. The number of multiple starting points, needed to address the issue of local maxima, can be set through the argument `n_start`, and the list `init` possibly allows to configure initial values of the parameters for each starting point.

The code below shows how to estimate the MMS-mix with a number of components ranging from 1 to 6 and save the values of the Bayesian information criterion (BIC) in a separate vector for then choosing the optimal number of clusters.

```
R> FIT.try <- list()
R> BIC <- setNames(numeric(6), paste0('G = ', 1:6))
R> for(i in 1:6){
+    FIT.try[[i]] <- fitMSmix(rankings = ranks_AF, n_clust = i, n_start = 50)
+    BIC[i] <- FIT.try[[i]]$mod$bic}
```

The BIC values of the six estimated models are

```
R> print(BIC)
   G = 1    G = 2    G = 3    G = 4    G = 5    G = 6
1494.435 1461.494 1442.749 1444.223 1449.714 1453.101
```

suggesting `G = 3` as the optimal number of groups (lowest BIC). The function `fitMSmix` creates an object of S3 class `"emMSmix"`, which is a list whose main component, named `mod`, describes the best fitted model over the `n_start` initializations. It includes, for example, the MLE of the parameters (`rho`, `theta` and `weights`), the fitting measures (`log_lik` and `bic`), the estimated posterior membership probabilities (`z_hat`) and the related MAP allocation (`map_classification`) as well as the binary indicator of convergence achievement (`conv`).

The MLEs of the best fitted model can be shown also through the generic method `summary.emMSmix`,

```
R> summary(object = FIT.try[[3]])
Call:
fitMSmix(rankings = ranks_AF, n_clust = 3, n_start = 50)

-----------------------------
--- MLE of the parameters ---
-----------------------------

Component-specific consensus rankings:
       Abs Red Sma Non Req Eme Unc
Group1   5   6   4   7   2   1   3
Group2   1   3   4   2   5   6   7
Group3   2   3   4   1   7   6   5

Component-specific consensus orderings:
       Rank1 Rank2 Rank3 Rank4 Rank5 Rank6 Rank7
```

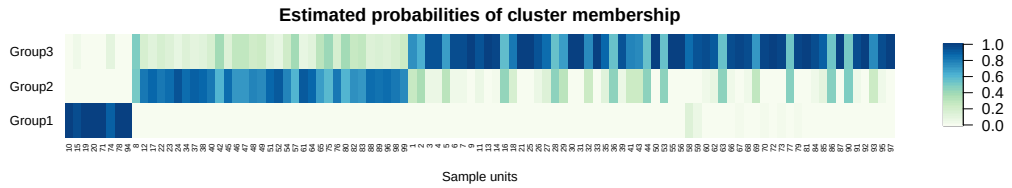


Figure 4: Heatplot of the estimated cluster membership probabilities for each observation of the `ranks_antifragility` dataset. This plot corresponds to the element named `est_clust_prob` of the output list returned by the generic method `plot.emMSmix`.

```
Group1 "Eme" "Req" "Unc" "Sma" "Abs" "Red" "Non"
Group2 "Abs" "Non" "Red" "Sma" "Req" "Eme" "Unc"
Group3 "Non" "Abs" "Red" "Sma" "Unc" "Eme" "Req"

Component-specific precisions:
Group1 Group2 Group3
 0.111  0.241  0.087

Mixture weights:
Group1 Group2 Group3
 0.083  0.343  0.574
```

which also displays the estimated modal orderings in the rows of the second output matrix. The generic function `plot.emMSmix` is also associated to the class `"emMSmix"` and constructs a list of two fancy plots, see commands below.

```
R> p_fit3_AF <- plot(FIT.try[[3]])
R> p_fit3_AF$bump_plot()
R> p_fit3_AF$est_clust_prob()
```

The first one is the bump plot (Figure 3) depicting the consensus ranking of each cluster, with different colors assigned to each item, circle sizes proportional to the estimated weights and lines to better highlight item positions in the modal orderings of the various components. For this example, we note that the size of the second cluster is almost half that of the third cluster, while the first cluster is very small. Moreover, the two larger groups (2 and 3) exhibit very similar modal rankings and quite opposite preferences with respect to the first cluster (items such as "Emergence", "Requisite variety", and "Uncoupling" are ranked at the top in cluster 1, but placed at the bottom in groups 2 and 3).

Figure 4 shows, instead, the individual cluster memberships probabilities, describing the uncertainty with which each observation could be assigned to the mixture components. For example, the units 10, 15, 19, 20, 71, 74, 78 and 94 have high probabilities (close to 1) of belonging to group 1. Instead, some units (e.g., unit 8, 28, 36, and 44) have similar membership probabilities of belonging to clusters 2 or 3, indicating less confidence in their assignment to one of the two groups. On the other hand, when some clusters are close on the ranking space, a certain degree of uncertainty in recovering the true membership is expected.

The package provides also routines for computing the CIs, working with the object of class `"emMSmix"` as first input argument. For example, we can produce asymptotic CIs for the precisions and mixture weights with `confintMSmix`, which is a function specific for full ranking data. With the default confidence level (`conf_level = 0.95`), one obtains

```
R> confintMSmix(object = FIT.try[[3]])
Asymptotic 95%CIs for the precisions:
       lower upper
Group1 0.000 0.226
Group2 0.153 0.329
Group3 0.068 0.106

Asymptotic 95%CIs for the mixture weights:
       lower upper
Group1 0.021 0.144
Group2 0.195 0.491
Group3 0.473 0.676
```

Another possibility relies on bootstrap CI calculation. Let us opt for the soft bootstrap method (the default choice when $G > 1$) which, unlike the separated one (`type = "separated"`), produces CIs

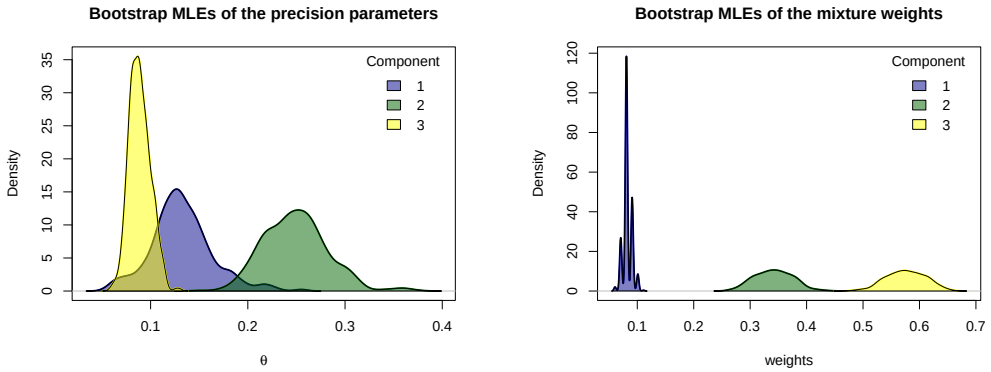


Figure 5: Kernel densities of the soft bootstrap MLEs of the precision parameters (left) and the weights (right) for the `ranks_antifragility` dataset. These plots correspond, respectively, to the elements named `theta_density` and `weights_density` of the output list returned by the generic method `plot.bootMSmix`.

also for weights. We require `n_boot = 500` bootstrap samples and then print the output object of class `"bootMSmix"` through the generic function `print.bootMSmix`.

```
R> CI_bootSoft <- bootstrapMSmix(object = FIT.try[[3]], n_boot = 500, all = TRUE)
R> print(CI_bootSoft)
Bootstrap itemwise 95%CIs for the consensus rankings:

       Abs           Red         Sma           Non     Req         Eme
Group1 "{3,4,5,6,7}" "{4,5,6,7}" "{2,3,4,5,6}" "{6,7}" "{1,2,3,4}" "{1,2}"
Group2 "{1}"         "{2,3}"     "{4}"         "{2,3}" "{5}"       "{6}"
Group3 "{1,2,3}"     "{2,3}"     "{4,5}"       "{1,2}" "{7}"       "{5,6}"
       Unc
Group1 "{2,3,4,5}"
Group2 "{7}"
Group3 "{4,5,6}"

Bootstrap 95%CIs for the precisions:
       lower upper
Group1 0.068 0.212
Group2 0.193 0.314
Group3 0.069 0.112

Bootstrap 95%CIs for the mixture weights:
       lower upper
Group1 0.071 0.101
Group2 0.283 0.404
Group3 0.505 0.636
```

The logical argument `all` indicates whether the MLEs estimates obtained from the bootstrap samples must be returned in the output. When `all = TRUE`, as in this case, the user can visualize the bootstrap sample variability with the generic function `plot.bootMSmix`. It returns a list with the heatmap of the first-order marginals of the bootstrap samples, and the kernel densities for the precisions and weights. For this application, the latter two plots (Figure 5) are obtained as follows.

```
R> p_ci_soft <- plot(CI_bootSoft)
R> p_ci_soft$theta_density()
R> p_ci_soft$weights_density()
```

4.5 Application on partial rankings

In this section, we illustrate how to perform inference on the partial rankings collected in the original `ranks_beers` dataset. These data were gathered through an online survey administered to the participants of the 2018 Pint of Science festival held in Grenoble. A sample of $N = 105$ subjects provided their partial rankings of $n = 20$ beers according to their personal tastes. The partial rankings, characterized

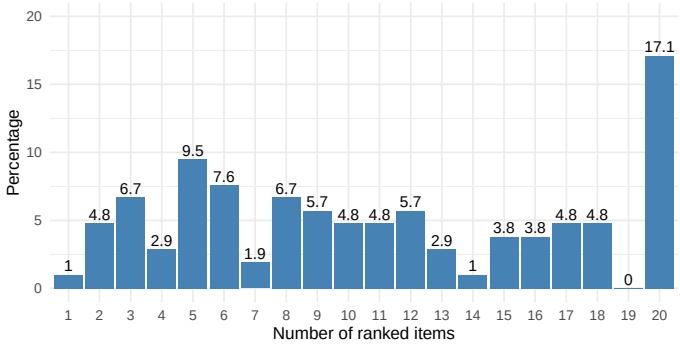


Figure 6: Percentages of the number of ranked items in the `ranks_beers` dataset. This barplot corresponds to the element named `n_ranked_distr` of the output list returned by the generic method `plot.data_descr`.

by different censoring patterns (that is, not exclusively top-k sequences), are recorded in the first 20 columns of the dataset, while column 21 contains a covariate regarding respondents' residency.

The barplot with the percentages of the number of beers actually ranked by the participants is reported in Figure 6. We restrict the analysis to partial rankings with maximum 8 missing positions, to show both the data augmentation schemes (Algorithms 3 and 4) implemented in the package. Note that, since our EM algorithms rely on the MAR assumption, we preliminarily conducted an empirical evaluation to assess whether the realized missingness pattern significantly deviates from this hypothesis. This check is described in Appendix A3.

Thanks to the `subset` argument of `fitMSmix`, we can specify the subsample of observations to be considered directly in the fit command. To speed up the estimation process, we parallelize the multiple starting points by setting `parallel = TRUE`.[10]

```
R> rankings <- ranks_beers[,1:20]
R> subset_beers <- (rowSums(is.na(rankings)) <= 8)
R> library(doParallel)
R> registerDoParallel(cores = detectCores())
R> FIT_aug <- fitMSmix(rankings,n_clust = 1, n_start = 15,
+                      subset = subset_beers, mc_em = FALSE, parallel = TRUE)
R> FIT_mcem <- fitMSmix(rankings, n_clust = 1, n_start = 15,
+                       subset = subset_beers, mc_em = TRUE, parallel = TRUE)
```

The logical `mc_em` argument indicates whether the MCEM scheme (Algorithm 4) must be applied. When `mc_em = FALSE` (default), Algorithm 3 is implemented.[11] We note that, for this application, the results of the two methods are very similar.

```
R> spear_dist(FIT_aug$mod$rho,FIT_mcem$mod$rho)/(2*choose(20+1,3))
[1] 0.001503759

R> c('theta_aug' = FIT_aug$mod$theta, 'theta_mcem' = FIT_mcem$mod$theta)
  theta_aug  theta_mcem
0.008580397 0.008964391
```

One can then evaluate the uncertainty associated to the consensus ranking estimated via the MCEM with the non-parametric bootstrap (default for $G = 1$). Also in this case, we can parallelize over the multiple starting points of the EM algorithm used to fit the bootstrap samples.

```
R> boot_mcem <- bootstrapMSmix(object = FIT_mcem, n_boot = 300, n_start = 15,
                               all = TRUE, parallel = TRUE)

R> print(boot_mcem)
Bootstrap itemwise 95%CIs for the consensus rankings:
```

[10] Note that exact reproducibility of this section may not be possible due to the use of parallelization, which can lead to minor variations in inferential results between runs.

[11] This type of data augmentation is supported for up to 10 missing positions in the partial rankings. However, it is important to note that while this operation may be feasible in principle for some datasets, it can be slow and memory-intensive. For instance, augmenting and storing all rankings compatible with the subset of the beers dataset with a maximum of 10 missing positions requires more than 3GB of storage space.

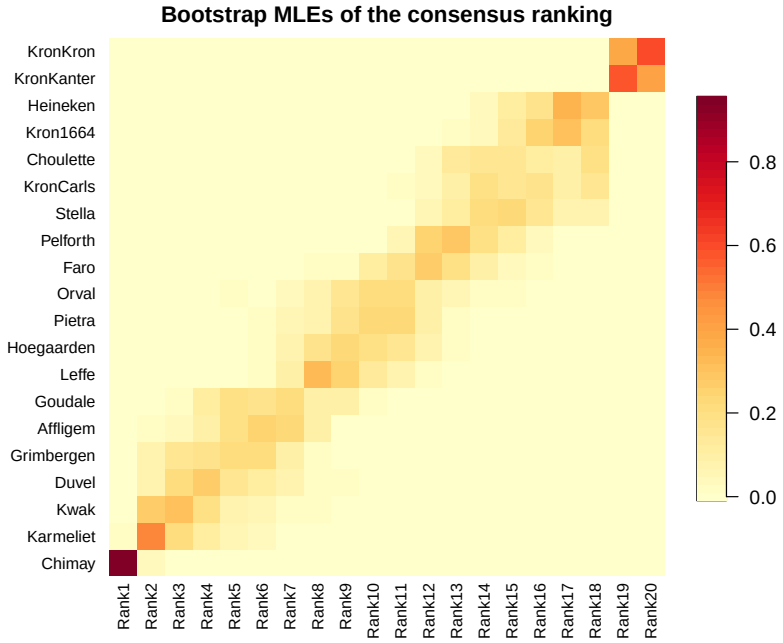


Figure 7: Heatmap of the bootstrap MLE of the consensus ranking for a subsample of the `ranks_beers` dataset. On the y-axis, items are ordered according to the MLE of $\boldsymbol{\rho}$ (top-ranked beer at the bottom). This plot corresponds to the element named `rho_heatmap` of the output list returned by the generic method `plot.bootMSmix`.

```
       Stella                   Kwak          KronKron  Faro
Group1 "{12,13,14,15,16,17,18}" "{2,3,4,5,6}" "{19,20}" "{8,9,10,11,12,13,14,15}"
       Kron1664           Chimay  Pelforth           KronCarls
Group1 "{14,15,16,17,18}" "{1,2}" "{11,12,13,14,15}" "{12,13,14,15,16,17,18}"
       KronKanter Hoegaarden           Grimbergen      Pietra
Group1 "{19,20}"  "{6,7,8,9,10,11,12}" "{2,3,4,5,6,7}" "{6,7,8,9,10,11,12,13}"
       Affligem         Goudale           Leffe             Heineken
Group1 "{3,4,5,6,7,8}" "{4,5,6,7,8,9,10}" "{6,7,8,9,10,11}" "{14,15,16,17,18}"
       Duvel              Choulette               Orval
Group1 "{2,3,4,5,6,7,8}" "{12,13,14,15,16,17,18}" "{5,6,7,8,9,10,11,12,13,15}"
       Karmeliet
Group1 "{1,2,3,4,5,6}"

Bootstrap 95%CIs for the precisions:
       lower upper
Group1 0.007 0.013
R> plot(boot_mcem)$rho_heatmap()
```

The heatmap of the bootstrap output, displayed in Figure 7, helps in understanding the variability and confidence in the rankings of the beers. In fact, the top ranked beer (Chimay) and the two bottom ranked ones (KronKanter and KronKron) are quite reliably ranked in those positions. On the contrary, the ranks of the other beers are more uncertain, with itemwise 95% bootstrap-based CIs for some beers being as wide as 10 positions (out of 20). Note also that some itemwise regions can result in subsets of non-contiguous ranks, as in the case of `Orval` whose CI does not include rank 14. We conclude this section by stressing that the application to the beers dataset represents a non-trivial case of ranking data analysis, since currently there are no other R packages supporting MLE of the MM on partially-ranked sequences with arbitrary missing positions.

4.6 Additional options

The **MSmix** package also supplies some functions to deal with the distribution of the Spearman distance. Although these functions are primarily used internally to fit the model (see the algorithms in Appendix A1), they are made available for external use due to their standalone utility.

The function `spear_dist_distr` returns the (log-)frequency distribution of the Spearman distance under the uniform model. If $n \leq 20$, the function returns the exact distribution by relying on a call to the `get_cardinalities` routine of **BayesMallows**. Here is an example with $n = 5$.

```
R> spear_dist_distr(n_items = 5)
$distances
 [1]  0  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
$logcard
 [1] 0.000000 1.386294 1.098612 1.791759 1.945910 1.791759 1.386294 2.302585
 [9] 1.791759 2.302585 1.791759 2.302585 1.791759 2.302585 1.386294 1.791759
[17] 1.945910 1.791759 1.098612 1.386294 0.000000
```

When $n > 20$, the approximate distribution introduced by Crispino et al. (2023) is returned and, in the case $n \geq 170$, its calculation is restricted over a fixed grid of values of the Spearman distance to limit computational burden.

The functions `partition_fun_spear`, `expected_spear_dist` and `var_spear_dist` provide, respectively, the partition function $Z(\theta)$, the expected value $\mathbb{E}_\theta[D]$ and the variance $\mathbb{V}_\theta[D]$ of the Spearman distance under the MMS. For $n = 5$, one has

```
R> partition_fun_spear(theta = 0.1, n_items = 5)
[1] 3.253889
R> expected_spear_dist(theta = 0.1, n_items = 5)
[1] 2.421115
R> var_spear_dist(theta = 0.1, n_items = 5)
[1] 4.202741
```

For these functions, the computation is exact or approximate according to the same principle described for `spear_dist_distr`.

5 Conclusions

The new **MSmix** package enriches the R software environment with functions to analyze finite mixtures of MMS on full and partial rankings with arbitrary patterns of censoring. Inference is conducted within the ML framework via EM algorithms. Estimation uncertainty is quantified with bootstrap methods and approximate CIs from the asymptotic likelihood theory.

The innovative contributions of **MSmix** span from both methodological and computational advancements to address the lacks and limitations found in most of the existing packages, especially the possibility of realizing a ranking data analysis with many items and missing positions or assessing estimation uncertainty of model parameters. Moreover, the estimation procedures have been generalized and optimized to work effectively across a spectrum of censoring patterns, rather than being limited solely to the top-k scenario. The package also exploits the construction of S3 class objects and related generic methods to offer a unified and original analysis framework. In this regard, a special attention was devoted to the development of effective visualization tools and summaries, that can assist the users in the reporting results and designing conclusions with a more transparent account of the associated uncertainty.

Our inferential procedures rely on EM algorithms assuming a MAR missing data mechanism. Under the MAR assumption, missingness does not depend on the unobserved preferences, allowing the EM algorithm to yield unbiased estimates without explicitly modeling the missing data process (Rubin, 1976; Little and Rubin, 2019). Although MAR is a common simplifying assumption in partial ranking analysis (Beckett, 1993; Jacques and Biernacki, 2014; Piancastelli and Barreto-Souza, 2025), it may not hold in real-world settings. For instance, top-k rankings can arise when respondents omit to rank certain items due to unfamiliarity or low popularity, indicating that missingness depends on unobserved preferences and could, thus, significantly depart from the pure MAR assumption. It is well known that ignoring the missingness process under non-MAR scenarios can bias estimates obtained from standard EM algorithms (Little and Rubin, 2019). However, the effects of MAR violations on the estimation accuracy of EM algorithms, as well as methods to address non-ignorable missingness, remain largely unexplored in the partial ranking literature. In this context, extending the methods provided by **MSmix** to better handle missing data in ranking models would represent a valuable methodological contribution for future research.

The package architecture and its computational achievements can facilitate code extensibility for accomplishing these innovative directions. For example, its flexibility in accommodating diverse data censoring patterns could be of support for exploring the plausibility of the standard MAR assumption or developing extensions of parametric mixture models incorporating non-ignorable missing data

mechanisms. Moreover, the package capability to analyze data characterized by a large number of alternatives could motivate the interest in clustering similar items, as recently proposed for the MM in Piancastelli and Friel (2025), or even in developing methods to solve bi-clustering problems. Finally, to better characterize choice processes, the EM algorithms could be integrated with an additional step for estimating the impact of individual and/or item-specific covariates - a typical but complex task in preference analysis from ranking data (see e.g., Gormley and Murphy, 2008; Zhu et al., 2021). We are currently working in this direction with a proposal to enrich the MMS-mix with a *Mixture of Experts* model (Jacobs et al., 1991; Jordan and Jacobs, 1994), that is, a mixture model in which the weights are functions of the covariates (Crispino et al., 2024). Future releases of **MSmix** will also include functions to deal with different distances among rankings.

6 Acknowledgments

The authors would like to thank the anonymous reviewers for their constructive and insightful comments that greatly improved the manuscript and the package. The authors wish to thank Prof. Luca Tardella and Prof. Enrico Casadio Tarabusi for the insightful discussions on various aspects of the methodology used in this paper. Additionally, the authors wish to thank Prof Maria Vincenza Ciasullo, Dr. Nicola Cucari, Dr. Raffaella Montera, Prof Maria Iannario and Prof. Rosaria Simone for generously sharing their data.
The opinions expressed are those of the authors and do not necessarily reflect the views of the Bank of Italy or the Eurosystem.

1 Appendix

1.1 Estimation algorithms

We here provide the pseudo-code of the estimation algorithms implemented in **MSmix**. In the heterogeneous case ($G > 1$), the latent group membership of the l-th distinct observed ranking $\boldsymbol{r}_l$ is denoted with $\boldsymbol{z}_l = (z_{l1}, \ldots, z_{lG})$, where $z_{lg} = 1$ if the observation belongs to component g and $z_{lg} = 0$ otherwise.

Algorithm 1 MLE of the MMS parameters from full rankings

Input: $\underline{r} = \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_N\}$ full n-rankings.

Preliminary steps:

- For $l = 1, \ldots, L$, compute the frequency N_l of each distinct observed ranking $\boldsymbol{r}_l$.
- Compute either the exact or the approximate frequency distribution of the Spearman distance $\{d, N_d\}_{d \in \mathcal{D}_n}$.

1. Compute the MLE of the consensus ranking $\boldsymbol{\rho}$:
   (a) Compute the sample mean rank vector $\bar{\boldsymbol{r}} = (\bar{r}_1, \ldots, \bar{r}_n)$.
   (b) Compute $\hat{\boldsymbol{\rho}} = \text{rank}(\bar{\boldsymbol{r}})$.
2. Compute the MLE of the concentration parameter θ:
   (a) Compute the sample average distance $\bar{d} = \frac{1}{N} \sum_{l=1}^{L} N_l d(\boldsymbol{r}_l, \hat{\boldsymbol{\rho}}) = 2(c_n - \hat{\boldsymbol{\rho}}^T \bar{\boldsymbol{r}})$.
   (b) Apply `uniroot` to find the solution of the equation $\mathbb{E}_\theta(D) = 2(c_n - \hat{\boldsymbol{\rho}}^T \bar{\boldsymbol{r}})$ in θ.

Output: $\hat{\boldsymbol{\rho}}$ and $\hat{\theta}$.

1.2 Performance of the algorithms

In this section, we further explore and discuss the computational efficiency of the estimation algorithms under a variety of scenarios illustrated in the following:

- *Homogeneous data ($G = 1$).* When the data is homogeneous, the Algorithm 1 performs efficiently even for large sample sizes N (see Table 4). This is because the typical computational challenges associated to the MLE of $\boldsymbol{\rho}$ (the consensus ranking) with other distance specification in the MM are eliminated. In fact, with Spearman distance the problem simplifies to the straightforward application of the Borda rank aggregation method, where items are ranked based on their

Algorithm 2 MLE of the MMS-mix parameters from full rankings

Input: $\underline{r} = \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_N\}$ full n-rankings; G number of clusters; $\underline{\boldsymbol{\rho}}^{(0)}, \boldsymbol{\theta}^{(0)}, \boldsymbol{\omega}^{(0)}$ initial values.

Preliminary steps:

- For $l = 1, \ldots, L$, compute the frequency N_l of each distinct observed ranking $\boldsymbol{r}_l$.
- Compute either the exact or the approximate frequency distribution of the Spearman distance $\{d, N_d\}_{d \in \mathcal{D}_n}$.

Repeat the E- and M-step below until convergence:

E-step: for $l = 1, \ldots, L$ and $g = 1, \ldots, G$, compute $\hat{z}_{lg} = \frac{\hat{\omega}_g \mathbb{P}(\boldsymbol{r}_l | \hat{\boldsymbol{\rho}}_g, \hat{\theta}_g)}{\sum_{g'=1}^{G} \hat{\omega}_{g'} \mathbb{P}(\boldsymbol{r}_l | \hat{\boldsymbol{\rho}}_{g'}, \hat{\theta}_{g'})}$.

M-step: for $g = 1, \ldots, G$ compute

(a) $\hat{\omega}_g = \hat{N}_g / N$ with $\hat{N}_g = \sum_{l=1}^{L} N_l \hat{z}_{lg}$.

(b) The MLE of $\boldsymbol{\rho}_g$ as in step 1 of Algorithm 1, by replacing $\bar{\boldsymbol{r}}$ with $\bar{\boldsymbol{r}}_g = (\bar{r}_{g1}, \ldots, \bar{r}_{gn})$, where $\bar{r}_{gi} = \frac{1}{\hat{N}_g} \sum_{l=1}^{L} N_l \hat{z}_{lg} r_{li}$.

(c) The MLE of θ_g as in step 2 of Algorithm 1, by replacing $\bar{\boldsymbol{r}}$ with $\bar{\boldsymbol{r}}_g$ and $\hat{\boldsymbol{\rho}}$ with $\hat{\boldsymbol{\rho}}_g$.

Output: $\underline{\hat{\boldsymbol{\rho}}} = \{\hat{\boldsymbol{\rho}}_1, \ldots, \hat{\boldsymbol{\rho}}_G\}, \hat{\boldsymbol{\theta}} = \{\hat{\theta}_1, \ldots, \hat{\theta}_G\}, \hat{\boldsymbol{\omega}} = \{\hat{\omega}_1, \ldots, \hat{\omega}_G\}$, and $\underline{\hat{\boldsymbol{z}}} = \{\hat{\boldsymbol{z}}_1, \ldots, \hat{\boldsymbol{z}}_N\}$.

Algorithm 3 MLE of the MMS-mix parameters from partial rankings

Input: $\underline{r} = \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_N\}$ partial n-rankings; G number of clusters; $\underline{\boldsymbol{\rho}}^{(0)}, \boldsymbol{\theta}^{(0)}, \boldsymbol{\omega}^{(0)}$ initial values.

Preliminary steps: for $l = 1, \ldots, L$,

- compute the frequency N_l of each distinct observed ranking $\boldsymbol{r}_l$.
- Compute and store the sets $\mathcal{C}(\boldsymbol{r}_l)$ of full rankings compatible with each distinct $\boldsymbol{r}_l$.

Repeat the E- and M-step below until convergence:

E-step:

(a) For each distinct $\boldsymbol{r}_l$ with $l = 1, \ldots, L$ and for each $\boldsymbol{r}^*_{m'} \in \mathcal{C}(\boldsymbol{r}_l)$, compute

$$\hat{p}_{lm'} = \mathbb{P}(\boldsymbol{r}^*_{m'} | \boldsymbol{r}_l, \underline{\hat{\boldsymbol{\rho}}}, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\omega}}) = \frac{\sum_{g=1}^{G} \hat{\omega}_g e^{-2\hat{\theta}_g \left(c_n - \hat{\boldsymbol{\rho}}_g^T \boldsymbol{r}^*_{m'}\right) - \log Z\left(\hat{\theta}_g\right)}}{\sum_{\boldsymbol{s}^* \in \mathcal{C}(\boldsymbol{r}_l)} \sum_{g=1}^{G} \hat{\omega}_g e^{-2\hat{\theta}_g \left(c_n - \hat{\boldsymbol{\rho}}_g^T \boldsymbol{s}^*\right) - \log Z\left(\hat{\theta}_g\right)}}.$$

(b) For $m = 1, \ldots, M$, compute $\hat{N}_m = \sum_{l: \boldsymbol{r}^*_{m'} \in \mathcal{C}(\boldsymbol{r}_l)} N_l \hat{p}_{lm'}$.

(c) For $m = 1, \ldots, M$, and $g = 1, \ldots, G$, compute $\hat{z}_{mg} = \frac{\hat{\omega}_g \mathbb{P}\left(\boldsymbol{r}^*_m | \hat{\boldsymbol{\rho}}_g, \hat{\theta}_g\right)}{\sum_{g'=1}^{G} \hat{\omega}_{g'} \mathbb{P}\left(\boldsymbol{r}^*_m | \hat{\boldsymbol{\rho}}_{g'}, \hat{\theta}_{g'}\right)}$.

M-step: for $g = 1, \ldots, G$, compute

- $\hat{\omega}_g = \hat{N}_g / N$ with $\hat{N}_g = \sum_{m=1}^{M} \hat{N}_m \hat{z}_{mg}$.
- The MLE of $\boldsymbol{\rho}_g$ as in M-step (b) of Algorithm 2, by replacing $\bar{\boldsymbol{r}}_g$ with $\bar{\boldsymbol{r}}^*_g = (\bar{r}^*_{g1}, \ldots, \bar{r}^*_{gn})$, where $\bar{r}^*_{gi} = \frac{1}{\hat{N}_g} \sum_{m=1}^{M} \hat{N}_m \hat{z}_{mg} r^*_{mi}$.
- The MLE of θ_g as in M-step (c) of Algorithm 2, by substituting $\bar{\boldsymbol{r}}_g$ with $\bar{\boldsymbol{r}}^*_g$.

Output: $\underline{\hat{\boldsymbol{\rho}}} = \{\hat{\boldsymbol{\rho}}_1, \ldots, \hat{\boldsymbol{\rho}}_G\}, \hat{\boldsymbol{\theta}} = \{\hat{\theta}_1, \ldots, \hat{\theta}_G\}, \hat{\boldsymbol{\omega}} = \{\hat{\omega}_1, \ldots, \hat{\omega}_G\}$ and $\underline{\hat{\boldsymbol{z}}} = \{\hat{\boldsymbol{z}}_1, \ldots, \hat{\boldsymbol{z}}_N\}$.

Algorithm 4 MLE of the MMS-mix parameters from partial rankings (MCEM)

Input: $\underline{r} = \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_N\}$ partial n-rankings; G number of clusters; $\underline{\boldsymbol{\rho}}^{(0)}, \boldsymbol{\theta}^{(0)}, \boldsymbol{\omega}^{(0)}$ initial values.

Preliminary step: for $s = 1, \ldots, N$, complete $\boldsymbol{r}_s$ at random, obtaining a full ranking $\boldsymbol{r}^*_s \in \mathcal{C}(\boldsymbol{r}_s)$.

Repeat the E-, M- and MC-step below until convergence:

E-step: for $s = 1, \ldots, N$, compute $\hat{\boldsymbol{z}}_s$ as in E-step of Algorithm 2, by replacing $\boldsymbol{r}_l$ with $\boldsymbol{r}^*_s$.

M-step: same as in Algorithm 2.

MC step: for $s = 1, \ldots, N$, complete $\boldsymbol{r}_s$ with the scheme (2)-(3), obtaining an updated $\boldsymbol{r}^*_s \in \mathcal{C}(\boldsymbol{r}_s)$.

Output: $\underline{\hat{\boldsymbol{\rho}}} = \{\hat{\boldsymbol{\rho}}_1, \ldots, \hat{\boldsymbol{\rho}}_G\}, \hat{\boldsymbol{\theta}} = \{\hat{\theta}_1, \ldots, \hat{\theta}_G\}, \hat{\boldsymbol{\omega}} = \{\hat{\omega}_1, \ldots, \hat{\omega}_G\}$, and $\underline{\hat{\boldsymbol{z}}} = \{\hat{\boldsymbol{z}}_1, \ldots, \hat{\boldsymbol{z}}_N\}$.

sample average rank. This closed-form solution avoids the need of time consuming iterative estimation procedures and multiple initializations for addressing issues of local optima. This is a major advantage over existing R packages, which rely on global or local search methods that quickly become computationally prohibitive as n increases. Concerning the MLE of θ (the dispersion parameter), we recall that this step relates to N through the sample average Spearman distance, whose computation is optimized with the use of an internal C++ routine called from the **BayesMallows** package. Moreover, the analytical approximation of the expected Spearman distance proposed by Crispino et al. (2023), and implemented in **MSmix**, improves the computational efficiency for the MLE of θ, particularly for large n.

N	500	1000	5000	10000	20000	50000	1e+05
time (s)	0.009	0.01	0.08	0.29	1.04	8.3	54.56

Table 4: Computational times (seconds) of Algorithm 1 applied on rankings of $n = 20$ items sampled from a MMS with increasing sample size N.

- *Heterogeneous data ($G > 1$).* as G increases, the computational burden naturally grows since the EM algorithm must iteratively estimate both the cluster-specific parameters and the individual cluster membership probabilities. However, Algorithm 2 maintains efficiency by leveraging the Borda method for estimating the cluster-specific consensus rankings, significantly simplifying a critical step in the clustering process. Nevertheless, since clustering involves iterative updates, our package optimizes the E- and M-steps to ensure computational efficiency, even in large N and G scenarios, as shown in Table 5. Let us also add that, in general, more complex problems - such as when clusters are less distinct - require a great number of initializations to reliably reach the global optimum. The number of starting points is controlled by the argument `n_start` of the `fitMSmix` function and a parallelization of the EM algorithm over multiple initializations is also possible thanks to the `parallel` argument. These options enlarge the applicability and efficiency of MMS-mixtures, and reduce computational time by improving the exploration of the mixed-type parameter space for increasing G.

G	5	10	15	30	60
time (s)	3.1	4.6	15.8	27.1	47.6

Table 5: Computational times (seconds) of Algorithm 2 running with 10 starting points on $N = 5000$ rankings of $n = 20$ items sampled from a MMS-mix with increasing number G of clusters.

- *Big data scenarios.* In the context of ranking data, "big data" typically refers to cases with a large number of items (n) and a small sample size (N), where the primary focus is on rank aggregation. Even in such cases, Algorithm 1 remains efficient (see Table 6). For extremely large n, the main computational challenge is the computation of the partition function which, in our implementation, relies on the approximation of the frequency distribution of the Spearman distance among n-rankings (provided by the `spear_dist_distr` function in the package). When $n > 170$, the function returns an approximation restricted over a fixed grid of values for the Spearman distance to limit both the computational and memory load.

n	200	500	1000	5000	10000
time (s)	2.74	2.58	3.38	3.66	3.9

Table 6: Computational times (seconds) of Algorithm 1 applied on $N = 200$ rankings sampled from a MMS with increasing number n of items.

1.3 Empirical evaluation of the MAR assumption for a subsample of the Beers dataset

In the reduced Beers dataset used in our application, 18 respondents (39%) provided full rankings, while 28 (61%) ranked only a subset of the alternatives.

```
R> n_items <- 20
R> rankings <- ranks_beers[,1:n_items]
R> subset_beers <- (rowSums(is.na(rankings)) <= 8)
R> rankings_subset <- rankings[subset_beers,]
R> is_partial <- rowSums(is.na(rankings_subset))>1
```

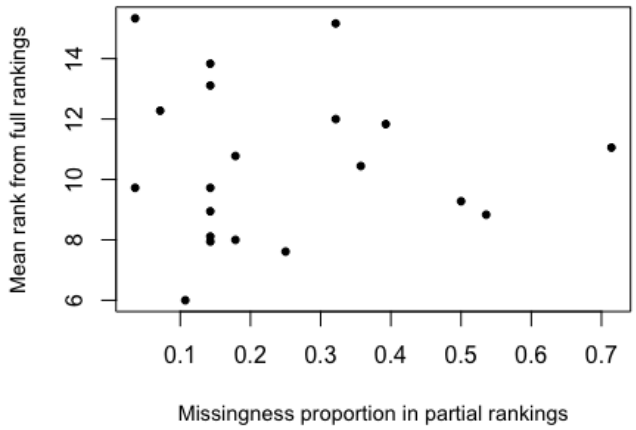


Figure 8: Relationship between the missingness proportion of the items in the partial rankings and their mean ranks from full rankings for the reduced `ranks_beers` dataset.

```
R> cbind(Freq = table(is_partial), '%' = round(100*prop.table(table(is_partial))))
     Freq  %
FALSE   18 39
TRUE    28 61
```

Notably, although the survey design did not require respondents to prioritize their most liked items, the incomplete rankings exhibit a clear top-like censoring pattern, as highlighted by the higher rate of missingness for bottom positions.

```
R> rankings_subset_part <- rankings_subset[is_partial,]
R> orderings_subset_part <- data_conversion(rankings_subset_part)
R> na_perc_by_rank <- round(100*colMeans(is.na(orderings_subset_part)),1)
R> names(na_perc_by_rank) <- paste("Rank", 1:n_items)
R> na_perc_by_rank
 Rank 1  Rank 2  Rank 3  Rank 4  Rank 5  Rank 6  Rank 7  Rank 8  Rank 9 Rank 10
    7.1     7.1     3.6     3.6     3.6     7.1     7.1    14.3     3.6     7.1
Rank 11 Rank 12 Rank 13 Rank 14 Rank 15 Rank 16 Rank 17 Rank 18 Rank 19 Rank 20
   28.6    32.1    35.7    39.3    42.9    50.0    53.6    46.4    46.4    46.4
```

In the case of top-k rankings, missing data could be significantly related to lower preferences, suggesting a Missing Not At Random (MNAR) process. To assess a possible critical deviation from the assumed MAR mechanism, we checked whether items which are more frequently missing in partial sequences systematically receive lower preferences from respondents who provided full rankings. Specifically, we explored the relationship between the average ranks resulting from the complete rankings and the missingness rate of each item in the partial rankings. The code below shows that, thanks to descriptive tools supplied by **MSmix** designed to facilitate a focused analysis of subsamples, the computation of these quantities is straightforward.

```
R> descr_full=data_description(rankings_subset,subset=!is_partial)
R> descr_part=data_description(rankings_subset,subset=is_partial)
R> na_prop_part_by_item <- descr_part$n_ranks_by_item[1,]/nrow(rankings_subset_part)
```

The relationship was first evaluated through a graphical inspection as follows.

```
R> plot(na_prop_part_by_item, descr_full$mean_rank,
     xlab="Missingness proportion in partial rankings",
     ylab="Mean rank from full rankings",
     cex.lab=0.9, pch=20)
```

The scatterplot in the Figure 8 does not reveal a clear association pattern. To formally assess this, we

performed a rank-correlation test[12] as follows

```
R> library(coin)
R> test_df=data.frame(NaProp=na_prop_part_by_item, AvgRank=descr_full$mean_rank)
R> perm_test <- spearman_test(AvgRank ~ NaProp, data = test_df,
                              distribution = approximate(nresample = 10000))
R> perm_test
 Approximative Spearman Correlation Test
data:  AvgRank by NaProp
Z = -0.18296, p-value = 0.8603
alternative hypothesis: true rho is not equal to 0
```

The resulting p-value is well above the conventional 0.05 threshold, indicating no statistically significant evidence against the MAR assumption.

References

M. Alvo and P. L. H. Yu. *Statistical Methods for Ranking Data*. Frontiers in Probability and the Statistical Sciences. Springer, New York, USA, 2014. [p205]

L. A. Beckett. Maximum likelihood estimation in Mallows's model using partially ranked data. In M. A. Fligner and J. S. Verducci, editors, *Probability Models and Statistical Analyses for Ranking Data*, pages 92–107. Springer New York, 1993. [p208, 222]

R. A. Bradley and M. E. Terry. Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. *Biometrika*, 39(3/4):324–345, 1952. [p205]

M. Crispino, C. Mollica, V. Astuti, and L. Tardella. Efficient and accurate inference for mixtures of Mallows models with spearman distance. *Statistics and Computing*, 98(33), 2023. [p205, 207, 208, 222, 225]

M. Crispino, L. Modugno, and L. Mollica. The Mallows model with respondents' covariates for the analysis of preference rankings. In *Proceedings of the SDS 2024 conference*, 2024. ISBN 978-88-5509-645-4. [p223]

D. E. Critchlow. *Metric Methods for Analyzing Partially Ranked Data*. Lecture Notes in Statistics No. 34, Springer New York, 1985. [p209]

D. E. Critchlow, M. A. Fligner, and J. S. Verducci. Probability Models on Rankings. *Journal of Mathematical Psychology*, 35(3):294–318, 1991. [p205]

P. Diaconis. *Group Representations in Probability and Statistics*, volume 11 of *Lecture Notes–Monograph Series*. Institute of Mathematical Statistics, Hayward, USA, 1988. [p205]

B. Efron. *The Jackknife, the Bootstrap and Other Resampling Plans*. SIAM, Philadelphia, USA, 1982. [p209]

I. C. Gormley and T. B. Murphy. Analysis of Irish Third-Level College Applications Data. *Journal of the Royal Statistical Society A*, 169(2):361–379, 2006. [p206]

I. C. Gormley and T. B. Murphy. A Mixture of Experts Model for Rank Data with Applications in Election Studies. *The Annals of Applied Statistics*, 2(4):1452–1477, 2008. [p223]

E. Gregory. *RMallow: Fit Multi-Modal Mallows' Models to Ranking Data*, 2020. URL https://CRAN.R-project.org/package=RMallow. R package version 1.1. [p206]

R. Hatzinger and R. Dittrich. prefmod: An R Package for Modeling Preferences Based on Paired Comparisons, Rankings, or Ratings. *Journal of Statistical Software*, 48(10):1–31, 2012. URL https://www.jstatsoft.org/v48/i10/. [p206]

R. Hatzinger and M. J. Maier. *prefmod: Utilities to Fit Paired Comparison Models for Preferences*, 2023. URL https://CRAN.R-project.org/package=prefmod. R package version 0.8-36. [p206]

R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive Mixtures of Local Experts. *Neural Computation*, 3(1):79–87, 1991. [p223]

[12] We opted for the permutation test based on the Spearman correlation, as it is nonparametric and allows to better captures monotonic relationships in the case of small samples and ties occurring after rank-transformation.

J. Jacques and C. Biernacki. Model-Based Clustering for Multivariate Partial Ranking Data. *Journal of Statistical Planning and Inference*, 149:201–217, 2014. [p206, 222]

J. Jacques, Q. Grimonprez, and C. Biernacki. Rankcluster: An R Package for Clustering Multivariate Partial Rankings. *The R Journal*, 6(1):101–110, 2014. [p206]

M. I. Jordan and R. A. Jacobs. Hierarchical Mixtures of Experts and the em Algorithm. *Neural Computation*, 6(2):181–214, 1994. [p223]

P. H. Lee and P. L. H. Yu. Mixtures of Weighted Distance-Based Models for Ranking Data with Applications in Political Studies. *Computational Statistics & Data Analysis*, 56(8):2486–2500, 2012. [p206]

P. H. Lee and P. L. H. Yu. An R Package for Analyzing and Modeling Ranking Data. *BMC Medical Research Methodology*, 3(1):1–11, 2013. [p206]

H. Li, M. Xu, J. S. Liu, and X. Fan. *ExtMallows: An Extended Mallows Model and Its Hierarchical Version for Ranked Data Aggregation*, 2018. URL https://CRAN.R-project.org/package=ExtMallows. R package version 0.1.0. [p206]

H. Li, M. Xu, J. S. Liu, and X. Fan. An extended Mallows model for ranked data aggregation. *Journal of the American Statistical Association*, 115(530):730–746, 2020. [p206]

R. Little. Calibrated Bayes, for Statistics in General, and Missing Data in Particular. *Statistical Science*, 26(2):162–174, 2011. [p207]

R. J. Little and D. B. Rubin. *Statistical Analysis with Missing Data*. John Wiley & Sons, 2019. [p222]

Q. Liu, M. Crispino, I. Scheel, V. Vitelli, and A. Frigessi. Model-Based Learning from Preference Data. *Annual Review of Statistics and Its Application*, 6(1):329–354, 2019. doi: 10.1146/annurev-statistics-031017-100213. [p205]

R. D. Luce. *Individual Choice Behavior: A Theoretical Analysis*. Wiley, New York, USA, 1959. [p205]

C. L. Mallows. Non-Null Ranking Models. I. *Biometrika*, 44(1/2):114–130, 1957. [p205]

J. I. Marden. *Analyzing and Modeling Rank Data*, volume 64 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, 1995. [p205, 209]

G. Mclachlan and D. Peel. *Finite Mixture Models*. New York: Wiley, 09 2000. [p209]

C. Mollica and L. Tardella. Bayesian Plackett-Luce Mixture Models for Partially Ranked Data. *Psychometrika*, 82(2):442–458, 2017. [p206]

C. Mollica and L. Tardella. PLMIX: An R Package for Modelling and Clustering Partially Ranked Data. *Journal of Statistical Computation and Simulation*, 90(5):925–959, 2020. [p206]

T. B. Murphy and D. Martin. Mixtures of Distance-Based Models for Ranking Data. *Computational Statistics & Data Analysis*, 41(3–4):645–655, 2003. [p208]

L. Piancastelli and W. Barreto-Souza. Time series analysis of rankings: A garch-type approach. arXiv preprint arXiv:2502.05102, 2025. [p222]

L. S. Piancastelli and N. Friel. The clustered Mallows model. *Statistics and Computing*, 35(1):21, 2025. [p223]

R. L. Plackett. The Analysis of Permutations. *Journal of the Royal Statistical Society C*, 24(2):193–202, 1975. [p205]

Z. Qian and P. L. H. Yu. Weighted Distance-Based Models for Ranking Data Using the R Package rankdist. *Journal of Statistical Software*, 90(5):1–31, 2019. [p206]

D. B. Rubin. Inference and Missing Data. *Biometrika*, 63(3):581–592, 1976. [p222]

Ø. Sørensen, M. Crispino, Q. Liu, and V. Vitelli. BayesMallows: An R Package for the Bayesian Mallows Model. *The R Journal*, 12(1):324–342, 2020. [p206, 215]

H. A. Soufiani and W. Chen. *StatRank: Statistical Rank Aggregation: Inference, Evaluation, and Visualization*, 2015. URL https://CRAN.R-project.org/package=StatRank. R package version 0.0.6. [p206]

Z. Taushanov and A. Berchtold. Bootstrap validation of the estimated parameters in mixture models used for clustering. *Journal de la Société Française de Statistique*, 160(1):114–129, 2019. [p209]

L. L. Thurstone. A Law of Comparative Judgment. *Psychological Review*, 34(4):273–286, 1927. doi: 10.1037/h0070288. [p205]

H. L. Turner, J. van Etten, D. Firth, and I. Kosmidis. Modelling Rankings in R: the PlackettLuce package. *Computational Statistics*, 35:1027–1057, 2020. doi: 10.1007/s00180-020-00959-3. URL https://doi.org/10.1007/s00180-020-00959-3. [p206]

G. C. G. Wei and M. A. Tanner. A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *Journal of the American Statistical Association*, 85(411):699–704, 1990. [p208]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p213]

C. Zhong. Mallows permutation models with L^1 and L^2 distances I: Hit and run algorithms and mixing times. arXiv preprint arXiv:2112.13456, 2021. [p215]

W. Zhu, Y. Jiang, J. S. Liu, and K. Deng. Partition–Mallows model and its inference for rank aggregation. *Journal of the American Statistical Association*, 118(541):343–359, 2021. [p223]

Marta Crispino
Department of Economics, Statistics and Research
Bank of Italy, Rome, Italy
marta.crispino@bancaditalia.it

Cristina Mollica
Department of Statistical Sciences
Sapienza University of Rome, Italy
cristina.mollica@uniroma1.it

Lucia Modugno
Department of Economics, Statistics and Research
Bank of Italy, Rome, Italy
lucia.modugno@bancaditalia.it

CvmortalityMult: Cross-Validation for Multi-Population Mortality Models

by David Atance, and Ana Debón

Abstract This article presents the CvmortalityMult *R* package, a novel tool designed for modelling, forecasting and evaluating mortality models for several populations. The package facilitates the fitting and forecasting of multipopulation mortality models, providing accurate projections in an increasingly interconnected world characterized by minimal or no borders between countries. By incorporating different cross-validation (CV) techniques, the package allows for the assessment of the forecasting accuracy of multipopulation mortality models for specific countries or regions within a country. Through an empirical application to Spanish regions, we demonstrate the efficacy and simplicity of the CvmortalityMult *R* package in selecting and evaluating multipopulation mortality models. By providing accessible tools for mortality modelling, forecasting and testing, this package stands out as a valuable resource for advancing the understanding and forecasting of mortality trends across diverse populations. Its contributions extend to enhancing decision-making in critical fields such as life insurance, public health, and pension plan sustainability.

1 Introduction

Currently, the loss of clear and defined borders between states/countries is leading populations worldwide to experience a similar dynamic of mortality. Indeed, mortality improvements or reductions can rapidly spread to other countries, causing correlated mortality dynamics, as observed with the COVID-19 pandemic. Thus, multipopulation mortality models provide a valuable approach for considering mortality membership in a group rather than individually (Li and Lee, 2005). These models enable the joint fitting of multiple populations, regions in the same country, or both sexes simultaneously.

On the basis of the original Lee and Carter (1992) model, many researchers have developed models to fit the mortality of related populations or countries with similar socioeconomic statutes or even both sexes in the same population (Brouhns et al., 2002; Debón et al., 2011; Dowd et al., 2011; Jarner and Kryger, 2011; Li and Hardy, 2011; Russolillo et al., 2011; Villegas and Haberman, 2014; Danesi et al., 2015; Chen and Millossovich, 2018; Bégin et al., 2023). The application of these multipopulation mortality models is relevant in diverse contexts, facilitating the concurrent modelling of multiple populations. This is particularly true within the field of life insurance, where companies operate worldwide and make assumptions about future mortality trends. Therefore, using such models ensures the consistency and reliability of the products across different countries. Notably, in the European Union, where gender-neutral pricing is enforced, it is necessary to model both sexes simultaneously (Ahmadi and Li, 2014). Moreover, multipopulation mortality models can include the correlation structure among populations when projecting future trends (Antonio et al., 2017; Bozikas and Pitselis, 2020).

The rise of "big data" has shifted the focus of many problems, and its methods have become a new field complementary to statistics. Within this domain, "resampling methods" are fundamental tools; these techniques are based on repeatedly drawing samples from a dataset and refitting models to obtain additional information. Advances in computational power have increased researchers' interest in these methods, which were developed in 1990. The two most common types of "resampling methods" are bootstrap and "cross-validation" (CV) methods (James et al., 2013a).

Bootstrap is a fundamental tool in the actuarial field that has had multiple applications throughout the literature. This method has been employed for various purposes, such as prediction errors in claim insurance (England and Verrall, 1999), establishing confidence bounds for discounted reserves (Hoedemakers et al., 2003), and estimating confidence

intervals in mortality through different bootstrap versions (Brouhns et al., 2005; Koissi et al., 2006; Debón et al., 2008b; Liu and Braun, 2010; D'Amato et al., 2012). The CV method divides sample data into k folds, where $k-1$ subsets are typically employed to train the model(s), and the remaining set is used to test the forecasting accuracy (Hastie et al., 2009). The process is iterated k times. For time series data, preserving the chronological order of the sets is necessary. This technique has been employed in other fields, such as finance, biology, and marketing. When applying CV methods to mortality modelling, these techniques must be adapted to time series analysis to use all available data for both testing and training (Tashman, 2000; Bergmeir and Benítez, 2012), which are also known as time series CVs (Hastie et al., 2009). Specifically, in each iteration, the training set must consist of observations that chronologically occurred before the test set observations corresponding to the end of each series (Hyndman and Athanasopoulos, 2021). Furthermore, in our analysis, each observation corresponds to a three-way array involving three categories: age, period and region or country. As a result, these methods have been adapted to assess the forecasting ability of multipopulation mortality models appropriately.

In the context of mortality modelling it should be noted that only recently researchers have applied CV techniques. However, most applications have focused on single-population models, see, for instance, Villegas et al. (2017); Hyndman et al. (2019); Atance et al. (2020b); Kessy et al. (2022); SriDaran et al. (2022); Lindholm and Palmborg (2022), and Barigou et al. (2023). To the best of our knowledge, the existing literature on the use of resampling methods in mortality modelling has focused on single-population mortality models, and among these studies, only one SriDaran et al. (2022) introduced a CV function designed for fitting, forecasting, and testing out-of-sample age-specific probabilities of death, with the selection of the testing period based on the mean squared error (MSE) measure.

This paper introduces the CvmortalityMult *R* package, which allows us to fit and forecast five multipopulation mortality models. Moreover, this package enables the application of CV methods to select the "best" multipopulation mortality model in forecasting among different scenarios. The idea is to determine which model produces the best forecasting outcomes for the period and the selected countries or regions. To achieve this goal, we implement several CV techniques following the terminology established by Tashman (2000) and Bergmeir and Benítez (2012) for evaluating time-series forecasts. Additionally, we adapt the methodology proposed by Atance et al. (2020b), which is primarily designed for single-population mortality models, to evaluate the forecasting ability of multipopulation mortality models over short, medium and long term horizons. The package incorporates multiple CV techniques to facilitate this evaluation.

Moreover, the package includes five variations of the classical Lee and Carter (1992) model to fit and forecast mortality in regions/populations that form part of a group rather than considering them individually. First, Russolillo et al. (2011) proposed adding a new multiplicative effect to represent different countries/regions within a multiplicative mortality model. Second, Debón et al. (2011) integrated the region/country effect as an additive index through an additive model. Third, Carter and Lee (1992) and Li and Lee (2005) first modelled the entire group and then incorporated a specific term for each region/country using the common-factor model. Fourth, Carter and Lee (1992) and Wilmoth and Valkonen (2001) proposed the joint-K model, which includes two specific country/region terms along with a common trend. Finally, Li and Lee (2005) and Hyndman and Ullah (2007) extended the common-factor model by incorporating two additional region/country terms with the augmented common-factor model. These five multipopulation mortality models were chosen because of their promising results compared with those of other mortality models (Debón et al., 2011; Dong et al., 2020), and their frequent usage in multipopulation modelling literature (Villegas et al., 2017). We introduce into the CvmortalityMult *R* package several functions that allow us to fit, forecast and evaluate those five multipopulation mortality models. However, if the functions detect only one population, they can fit the well-known Lee and Carter (1992) model.

The paper is structured as follows. Section 2 focuses on describing multipopulation mortality models. Section 3 discusses the CV methods in multipopulation mortality models.

Section 4 presents the CvmortalityMult *R* package, installation and the main functions. Section 5 presents a case study detailing the use of the package. Finally, Section 6 draws conclusions from the results in the previous section.

2 Models

In this section, we introduce five of the most important multipopulation mortality models, which serve as benchmarks to test the forecasting accuracy using CV methods. Indeed, the CvmortalityMult *R* package requires a set of crude age-specific probabilities of death for age x, period t and, in each region i, $\dot{q}_{x,t,i}$. These crude rates are directly obtained as $\dot{q}_{x,t,i} = d_{x,t,i}/E^0_{x,t,i}$, where $d_{x,t,i}$ represents the number of recorded deaths and $E_{x,t,i}$ denotes the initial population exposed to risk for an age x, period t and, region i. Crude mortality rates, along with other life table indicators, can be obtained using the `LifeTable` function from the **MortalityLaws** *R* package (Pascariu, 2022). This set of crude probabilities is then used to generate smoothed and forecasted estimates, $\hat{q}_{x,t,i}$, of the true but unknown mortality probabilities $q_{x,t,i}$. Therefore, in the context of multipopulation mortality data, "one observation ahead" corresponds to a set of data containing the probabilities of death for all ages and populations considered for the following year.

2.1 Multiplicative mortality model

Russolillo et al. (2011) proposed incorporating a multiplicative index term into the Lee and Carter (1992) model to shift the mortality for each region in the population group. That is:

$$\text{logit}\,(q_{x,t,i}) = \log\left(\frac{q_{x,t,i}}{1-q_{x,t,i}}\right) = a_x + b_x \cdot k_t \cdot I_i + \varepsilon_{x,t,i}; \tag{1}$$

where a_x captures the general age shape of the mortality curve, k_t describes the general trend of the group of populations over time, b_x represents how each age-specific probability of death reacts to changes in the general level of mortality, and I_i represents a multiplicative index associated with mortality for each member of the group of populations considered. Both a_x and b_x are the common age-dependent and time-independent parameters for all the regions considered respectively, whereas k_t is the same for all considered regions and corresponds with the time-dependent parameter, as in the initial version of Lee and Carter (1992), I_i is a different index for each population considered.

Notably, the logit link function is employed to fit the age-specific probabilities of death. The logit transformation ensures that values of $q_{x,t,i}$ between 0 and 1 (Lee, 2000) are obtained. This transformation also maintains the historical ties to the early actuarial work of Perks (1932), as noted by Haberman and Renshaw (2011).

2.2 Additive mortality model

Debón et al. (2011) propose incorporating an additive index term into the Lee-Carter structure to modify the mortality of each region in the multipopulation model. Its expression is as follows:

$$\text{logit}\,(q_{x,t,i}) = \log\left(\frac{q_{x,t,i}}{1-q_{x,t,i}}\right) = a_x + b_x \cdot k_t + I_i + \varepsilon_{x,t,i}. \tag{2}$$

As in the previous model, the same components a_x, b_x and k_t are shared across all the studied regions (populations). This fact is a necessary and sufficient condition to avoid divergence in the forecasting of age-specific probabilities of death of subpopulations; see Debón et al. (2011) and Ahcan et al. (2014). The number of parameters is the same as that in the model of Russolillo et al. (2011), with similar interpretations except for I_i; more details are given in Debón et al. (2011). However, the additive formulation provides a more straightforward structure than the multiplicative formulation does because it incorporates regional effects through an additive index term.

2.3 Common-factor mortality model

Carter and Lee (1992) and Li and Lee (2005) proposed modelling the mortality of different populations through a common long-term component for the whole group combined with an age-dependent specific term for each population i. This model is represented as:

$$\text{logit}\left(q_{x,t,i}\right) = \log\left(\frac{q_{x,t,i}}{1-q_{x,t,i}}\right) = a_{x,i} + B_x \cdot K_t + \varepsilon_{x,t,i}, \tag{3}$$

where $a_{x,i}$ represents the baseline shape of the mortality curve for each ith specific population, while the long-term change over time across the whole mortality group is captured by $B_x \cdot K_t$. These parameters serve the same function as b_x and k_t do in previous mortality models but apply to the whole group of populations.

2.4 Joint-k mortality model

Carter and Lee (1992) and Wilmoth and Valkonen (2001) introduced a model that assumes two specific population age-dependent terms, and a common trend among the group of populations, is given by:

$$\text{logit}\left(q_{x,t,i}\right) = \log\left(\frac{q_{x,t,i}}{1-q_{x,t,i}}\right) = a_{x,i} + b_{x,i} \cdot k_t + \varepsilon_{x,t,i}. \tag{4}$$

where k_t represents a common mortality trend among the different considered populations while a_{x_i} and b_{x_i} are specific for each population i. Indeed, the a_{x_i} parameter retains the same meaning as in the common factor mortality model, and b_{x_i} captures the effect of a time-varying mortality index k_t at age x for each population i.

2.5 Augmented common-factor mortality model

Li and Lee (2005) and Hyndman et al. (2013) introduced two population-specific terms to the common factor model. This model is expressed as:

$$\text{logit}\left(q_{x,t,i}\right) = \log\left(\frac{q_{x,t,i}}{1-q_{x,t,i}}\right) = a_{x,i} + B_x \cdot K_t + b_{x,i} \cdot k_{t,i} + \varepsilon_{x,t,i}. \tag{5}$$

The first three terms correspond to the components of the common-factor mortality model (3), whereas $b_{x_i} \cdot k_{t_i}$ captures deviations in the short to medium-term changes in the age-specific probability of death for each population i to the common trend. Importantly, including the population-specific terms b_{x_i} and k_{t_i} may imply significant divergences in the mortality forecasts across different populations. Our choice of capital letters in Equations (3) and (5) follows the notation introduced by Li and Lee (2005), who use upper-case symbols to denote the common (or "group-wide") Lee–Carter factor and lower-case symbols for the population-specific effects.

All the models discussed are implemented using the software R Core Team (2022) via the **gnm** library (Turner and Firth, 2023). Details on the calibration approach can be found in Debón et al. (2011). The parameters are obtained by maximizing the model's log-likelihood, assuming a quasi-Binomial distribution of deaths in all considered models:

$$L\left(q_{x,t,i}; \hat{q}_{x,t,i}\right) = \sum_x \sum_t \sum_i w_{x,t,i} \left\{q_{x,t,i} \cdot \log\left(\hat{q}_{x,t,i}\right) + \left(1-q_{x,t,i}\right) \cdot \log\left(1-\hat{q}_{x,t,i}\right) + \text{cte}\right\}, \tag{6}$$

where $w_{x,t,i}$ corresponds to weights assigned to each age, period, and population considered, $\hat{q}_{x,t,i}$ is derived by rearranging the terms in Equation (1):

$$\hat{q}_{x,t,i} = \frac{e^{a_x + b_x \cdot k_t \cdot I_i}}{1 + e^{a_x + b_x \cdot k_t \cdot I_i}}, \tag{7}$$

in the case of the multiplicative model. For the other models, the corresponding inverse transformations apply for (2) – (5). We maximize the log-likelihood function because it is an effective estimation method in the actuarial and demography literature for the parameter estimation process (Brouhns et al., 2002; Renshaw and Haberman, 2006; Cairns et al., 2009).

2.6 Forecasting multipopulation mortality models

To project the age-specific probabilities of death, $q_{x,t,i}$, it is essential to forecast the value of the trend parameters k_{t_n}, $k_{t_n,i}$, and K_{t_n} for all the multipopulation mortality models. These models are formulated as follows:

$$\begin{aligned}
\text{Multiplicative} \rightarrow \text{logit}\,(q_{x,t_n+s,i}) &= a_x + b_x \cdot k_{t_n+s} \cdot I_i, \\
\text{Additive} \rightarrow \text{logit}\,(q_{x,t_n+s,i}) &= a_x + b_x \cdot k_{t_n+s} + I_i, \\
\text{Common-factor} \rightarrow \text{logit}\,(q_{x,t_n+s,i}) &= a_{x,i} + B_x \cdot K_{t_n+s}, \\
\text{Joint-k} \rightarrow \text{logit}\,(q_{x,t_n+s,i}) &= a_{x,i} + b_{x,i} \cdot k_{t_n+s}, \\
\text{Augmented common-factor} \rightarrow \text{logit}\,(q_{x,t_n+s,i}) &= a_{x,i} + B_x \cdot K_{t_n+s} + b_{x,i} \cdot k_{t_{(n+s)},i},
\end{aligned} \tag{8}$$

where $q_{x,t_n+s,i}$ corresponds to the forecasted age-specific probability of death for age x, period $t_n + s$ and population i, and k_{t_n+s}, $k_{t_{(n+s)},i}$, K_{t_n+s} are the projections of the trend parameters k_{t_n}, $k_{t_n,i}$, and K_{t_n} considering that t_n is the last in-sample period.

In the CvmortalityMult *R* package, three alternative approaches are considered to project the time series k_t, K_t and k_{t_i}, assuming they follow ARIMA (autoregressive integrated moving average model) independent processes. First, a random walk with drift (ARIMA (0,1,0) with drift) is assumed, which is a common assumption in the actuarial literature (Cairns et al., 2006; Haberman and Renshaw, 2011; Villegas et al., 2018). Second, the CvmortalityMult *R* package allows the user to assume the best ARIMA (p,d,q) model according to the `auto.arima` function in the **forecast** R- package (Hyndman and Khandakar, 2008; Hyndman et al., 2023) for each trend parameter k_t, K_t and/or k_{t_i}, as described by Debón et al. (2008b), Villegas et al. (2018) and Hunt and Blake (2020). The `auto.arima` function determines the best ARIMA (p,d,q) model on the basis of the outcomes according to the corrected Akaike information criterion (AICc). Third, users can specify the (p, d, q) order for each ARIMA model by setting the corresponding parameters. Across the three approaches, the user can decide whether to include different ARIMA configurations by changing the arguments in the `auto.arima` or `Arima` functions as in the **forecast** *R* package.

3 Cross-validation methods

CV is a tool that focuses on assessing the predictive power of models. Identifying the best model benefits insurance companies in actuarial and financial applications, such as pricing and reserving, where forecasting is arguably more relevant than explanation (SriDaran et al., 2022). Thus, this methodology is valuable for identifying which model is the most accurate forecaster for single-population mortality models (Atance et al., 2020b) and can be used for a set of related populations. This section describes the CV methods (Burman, 1989; Bergmeir and Benítez, 2012) applied to evaluate the out-of-sample accuracy of multipopulation models.

The performance of a model varies between in-sample and out-of-sample evaluations (Bartolomei and Sweet, 1989; Pant and Starbuck, 1990). Therefore, partitioning data into training and test sets is fundamental for accurately assessing the forecasting ability of models. Various possibilities exist for evaluating time series forecasts, also referred to as the CV in time series (Hastie et al., 2009). These methods differ on the basis of the forecast horizon and the method of forecasting the out-of-sample validation, also known as "last block evaluation" in individual time series analysis (Tashman, 2000; Bergmeir and Benítez, 2012). Among the different available methods, we have adapted several approaches in CvmortalityMult *R* package to assess the forecasting ability of multipopulation mortality

models from different perspectives. Specifically, we follow the terminology established by Tashman (2000) and Bergmeir and Benítez (2012) to evaluate time series forecasting but adapt it for multipopulation mortality models.

3.1 Fixed-origin evaluation

Fixed-origin evaluation, also known as the out-of-sample test or hold-out method (Lachenbruch and Mickey, 1968; Tashman, 2000), is one of the most commonly used methods for assessing the forecasting accuracy of mortality models (Ahcan et al., 2014; Atance et al., 2020a). In this approach, adapted for time series analysis, the dataset is chronologically divided only once into training and test sets. The model is fitted using the training set, with its final point as the fixed origin for forecasting, as shown Figure 1 for a three-way array that incorporates three dimensions: ages in rows, periods in columns, and regions in the third dimension. This fixed-origin method generates a single forecast to predict all or specific periods in the test set. The forecasting accuracy is then evaluated for different forecast horizons.

This CV approach has also been employed for assessing the forecasting ability of multipopulation mortality models; see, for instance, Danesi et al. (2015), Antonio et al. (2017) and Bozikas and Pitselis (2020).

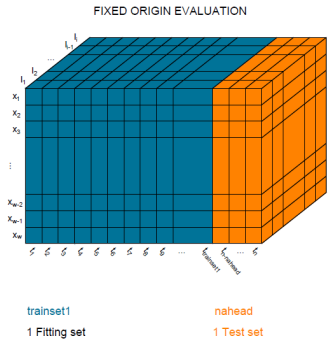


Figure 1: A schematic representation of the three-way array utilizing the fixed-origin cross-validation (CV) technique. The training and test sets are shown in blue and orange, respectively. In our package, the initial training set size is specified by the argument `trainset1`, whereas the forecast horizon is defined by `nahead`. To ensure the proper application of the fixed-origin evaluation, the sum of `trainset1` and `nahead` must equal the total number of provided periods.

3.2 Rolling-origin-recalibration evaluation

In rolling-origin-recalibration (RO-recalibration) evaluation for time series, we initiate the procedure by partitioning the sample into "k" subsets of data, while maintaining the chronological order. The first subset corresponds to the training set, and forecasts are generated with a fixed horizon to assess the model's performance. In each iteration, the model is recalibrated by incorporating all preceding information in the training set (Armstrong and Grohman, 1972; Tashman, 2000; Bergmeir and Benítez, 2012; Hyndman and Athanasopoulos, 2021). The test set periods are sequentially added to the training set to forecast the next set of periods, as shown in Figure 2. Consequently, the beginning of the evaluation shifts forward at each iteration. The model's accuracy is assessed using the average forecasting performance across the k iterations:

$$\text{RO-recalibration}_k = \frac{1}{k}\sum_{i=1}^{k}\text{Goodness of fit measure}_i. \tag{9}$$

In this type of time series CV, the specific variant of rolling-origin (RO) recalibration applied depends on the size of the initial training set and the forecast horizon of each test set. For

instance, k-fold CV (Hastie et al., 2009; James et al., 2013b; Bergmeir et al., 2018) requires the training set size and forecast horizon for each test set to be equal. Notably, to our knowledge, the application of the k-fold CV for analysing the forecasting proficiency of multipopulation mortality models has yet to be documented. We recommend that the initial training set contain more periods than the forecast horizon does to ensure reliable results. If the training and test set sizes differ, the common version RO-recalibration should be applied.

Notably, the "leave-one-out CV" (LOOCV) (Burman, 1989; Shao, 1993) is a special case of RO-recalibration, where the forecast horizon is equal to one, regardless of the initial training set size. Unlike approaches that generate two subsets of comparable size, LOOCV is a distinct approach that involves selecting and forecasting a single observation as the test set, whereas the preceding available observations constitute the training set. To implement this approach, the procedure is repeated (n - `trainset1`) times, where n denotes the total number of observations in the dataset and (`trainset1`) is the size of the initial training set.

Among the different resampling methods, this technique is widely acknowledged for assessing the predictive performance of single mortality models, as evidenced in various studies, such as those in Li and O'Hare (2019), Atance et al. (2020b), Barigou et al. (2023), and Atance and Navarro (2024). Additionally, only one preliminary work Atance and Debón (2022) applied RO-recalibration LOOCV, i.e., the prediction of multipopulation mortality models moving forward by one year for each iteration.

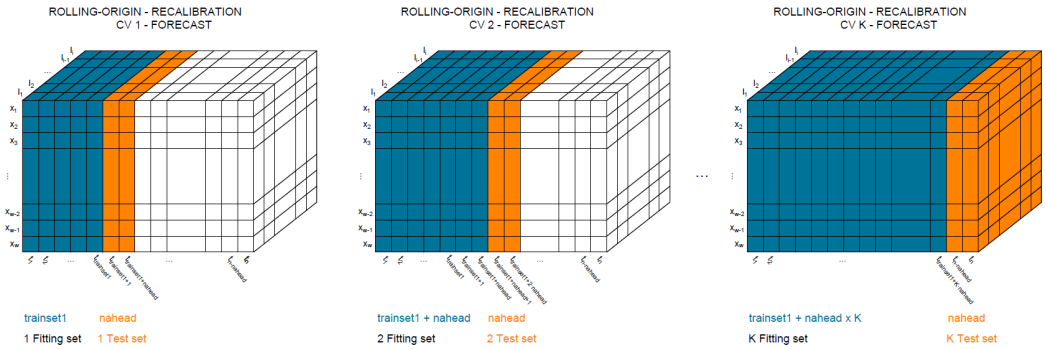


Figure 2: Schematic representation of the cross-validation method for a three-way array using the rolling-origin (RO) recalibration technique. The training, test, and omitted sets are depicted in blue, orange, and white, respectively. The initial training set size is denoted as `trainset1`, whereas `nahead` specifies both the forecast horizon and the size of each test set.

3.3 Rolling-window evaluation

"Rolling-window evaluation" is similar to RO-recalibration but maintains a constant training set size across each forecast iteration (Armstrong and Grohman, 1972; Tashman, 2000). It is also referred to as "time series CV" (TSCV) (Hart, 1994; Bergmeir and Benítez, 2012), "fixed-size rolling-window" (Swanson and White, 1997), or "fixed-size rolling sample" (Callen et al., 1996). Data are partitioned into training and test sets as in previous techniques. In each iteration, the training set incorporates the forecasted periods from the test set while discarding the earliest observations and preserving chronological order, as shown in Figure 3. The model and the forecast origin are also recalibrated at each window/iteration. The forecasting accuracy of the model is assessed using the same procedure as that in RO-recalibration, as expressed in Eq. (9).

Similar to the RO-recalibration technique, there are different variants of rolling-window evaluation. Indeed, the common CV, k-fold CV and LOOCV approaches are also variants. However, in these approaches, the training set size is constant throughout the iterations.

To our knowledge, the CvmortalityMult *R* package provides the first function for analysing the forecasting ability of multipopulation mortality models using CV techniques.

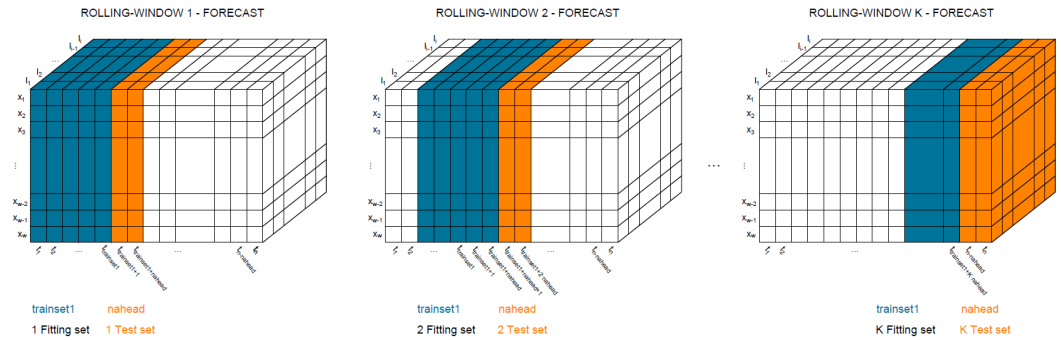


Figure 3: Schematic representation of the cross-validation method for a three-way array using the rolling-window evaluation technique. The training, test, and omitted sets are depicted in blue, orange, and white, respectively. The training set size, denoted as `trainset1`, remains consistent across iterations, starting with an initial training set. The argument `nahead` specifies both the forecast horizon and the size of each test set.

Notably, RO recalibration and rolling-window evaluation have not previously been implemented in any *R* package for assessing the forecasting accuracy of multipopulation mortality models. The CvmortalityMult *R* package addresses this gap by enabling the application of various time series CV methods.

4 The CvmortalityMult R-package

Table 1 introduces the main functions incorporated in the CvmortalityMult *R* package, along with a brief description of every function. Furthermore, these functions have been categorized into four groups: fitting, forecasting, plotting, and CV. In the following sections, we explain with several examples the procedural details and parameter structure essential for using the primary functions of the package.

Initially, the procedure involves the use of five mortality multipopulation functions to calibrate age-specific probabilities of death across various regions (populations). Notably, the CvmortalityMult *R* package allows for the fitting of a single-mortality model when the user provides data for only one population.

During the forecasting stage, the package subsequently facilitates age-specific probabilities of death projections under various $ARIMA(p, d, q)$ specifications, employing the object obtained in the preceding step. During the plotting stage, users have the opportunity to visualize the parameters obtained in the initial fitting stage, forecasts of the age-specific probability of death, and displays specific values across the Spanish regions in a geographical map. This procedure allows for a comprehensive understanding of the behaviour exhibited by each population under consideration.

Finally, during the CV stage, we introduce the function for applying different resampling methods to assess the forecasting accuracy of multipopulation mortality models. This function enables the modification of the fitting and forecasting periods, uses the functions of the preceding steps, and allows for the evaluation of the model forecasting performance using various goodness-of-fit measures. Notably, the structure of the multipopulation mortality data was aligned with a three-way array (age $\times$ time $\times$ region/population). For each probability of death, data are available for different ages, periods and regions/populations. Consequently, the applications of CV techniques need to be adapted to evaluate the projecting ability of these models effectively.

Notably, this paper does not include an example of every function argument. However, the reader is referred to the function documentation and the package vignette for a complete description of the CvmortalityMult *R* package.

Two datasets are included in the package: `SpainRegions` and `SpainNat`. These datasets originate from the Spanish National Institute (Instituto Nacional de Estadística, INE). Life

	Function name	Brief description
Fitting	`fitLCmulti()`	Fitting the multiplicative, additive, common-factor, joint-k or augmented common-factor multipopulation mortality models, and the single version of the Lee-Carter model.
Forecasting	`forecast.fitLCmulti()` or S3 method `forecast()`	Forecasting the multiplicative, additive, common-factor, joint-k or augmented common-factor multipopulation mortality models, and the single version of the Lee-Carter model.
Plotting	`plot.fitLCmulti()` the S3 method `plot()`	Plot the parameters for the multipopulation or single-population mortality models.
	`plot.forLCmulti()` the S3 method `plot()`	Plot the forecasting parameters for the multipopulation or single-population mortality models.
	`SpainMap()`	Plot the regions of Spain with the percentiles of the variable chosen by the users.
CV	`multipopulation_cv()`	CV techniques using the methods described in Section 2.3.
Measures of accuracy	`MeasureAccuracy()`	Measure for testing the accuracy of the single-population or multipopulation mortality models.

Table 1: Summary of the main functions in the CvmortalityMult *R* package.

tables and abridged lifetables were obtained with the methodology proposed by Muriel et al. (2010) on the basis of the work in Elandt-Johnson and Johnson (1980). On the one hand, the `SpainRegions` dataset comprises 10800 observations, encompassing 20 age groups, 30 periods (1990 - 2020), and 18 regions for both males and females in Spain, including national data (the regions in Spain are referred to as autonomous communities). On the other hand, the `SpainNat` dataset contains 600 observations, corresponding to national data for males and females in Spain covering 20 age groups, 30 periods (this dataset was created for the application of a single-population mortality model, and it can be obtained as a subgroup of the `SpainRegions` database). These datasets are structured as a data frame and include the following variables:

- `ccaa`: A vector of the 17 different regions of Spain, including national data. Figure 4 shows the identification of each region on the Spanish map.
- `years`: A vector of the period range, spanning from 1990 - 2020 for both datasets.
- `ages`: A vector of the age groups (children under 1 year, between 1 year and 5 years, and then by groups of 5 years, with the last group being between 90 and 94 years).
- `qx_male`: A vector of the age-specific probabilities of death for the male population.
- `qx_female`: A vector of the age-specific probabilities of death for the female population.
- `lx_male`: A vector of the estimated number of individual males living in each age group during each period in a specific region/population, based on an initial group of $l_0 = 100,000$ individuals aged 0 (Pitacco et al., 2009).
- `lx_female`: A vector of the estimated number of individual females alive in each age group during each period in a specific region/population.
- `series`: The sex included in both datasets "male and female population".

- `label`: A tag indicating the dataset type, either "Spain regions" or "Spain National population".

Figure 4: Administrative structure of Spain (regions are referred to as autonomous communities).

Furthermore, we have included the dataset `regions`, which contains the geographical values of the Spanish regions. The `SpainMap` function facilitates the creation of a map displaying the Spanish regions along with the variables incorporated in this dataset.

5 Application of the CvmortalityMult R- package

5.1 Fitting

Model fitting of the age-specific probabilities of death, $q_{x,t,i}$ (at age x, period t, and i region/population), under a quasi-Binomial distribution of deaths and a logit link is performed using the **gnm** package developed by Turner and Firth (2023).

The proposed multipopulation mortality models present challenges related to parameter identifiability. The parameter solution for the considered multipopulation mortality models $(a_x, a_{x_i}, b_x, B_x, b_{x_i}, k_t, K_t, k_{t_i}, I_i)$ are not unique, as any transformation of these parameters that preserves the model structure is also a solution, highlighting inherent identifiability problems in mortality models (Enchev et al., 2017; Villegas et al., 2018). This identifiability problem is addressed as follows: by setting $k_{t_0} = 0$, $b_0 = 1$ and $I_1 = 1$ for the multiplicative model; by setting $k_{t_0} = 0$, $b_0 = 1$ and $I_1 = 0$ for the additive model (Debón et al., 2011), by setting $B_0 = 1$, and $K_{t_0} = 0$ for the common-factor model (Carter and Lee, 1992), by setting $b_{0,1} = 1$, and $k_{t_0} = 0$ for the joint-K model (Carter and Lee, 1992), and by setting $B_0 = 1$, $b_{0,1} = 1$, $K_{t_0} = 0$, and $k_{t_{0,1}} = 0$ for the augmented-common-factor model (Li and Lee, 2005).

The `fitLCmulti()` facilitates the fitting of multipopulation mortality models, including the multiplicative model (Russolillo et al., 2011), additive model (Debón et al., 2011), common-factor model (CFM) (Carter and Lee, 1992), joint-K model (Carter and Lee, 1992) and augmented common-factor model (ACFM) (Li and Lee, 2005). It also supports fitting a single version of the Lee-Carter model (Lee and Carter, 1992) in the CvmortalityMult *R* package. The synopsis of this function is outlined below:

```
fitLCmulti(model, qxt, periods, ages, nPop, lxt = NULL)
```

The fitting function requires the following information as input:

- The `model` refers to the multipopulation mortality model chosen to fit the mortality rates. The available options include `c('additive','multiplicative','CFM','joint-K','ACFM')`. Users must select one model to fit.
- `qxt` is a vector or matrix containing the crude age-specific probabilities of death for every age, period, and region. The function automatically identifies the data structure (vector or matrix) that users provide.
- `lxt` is a vector or matrix with the estimated number of individual males alive in each age group during each period in a specific region. The function automatically identifies the data structure (vector or matrix) that users provide. If this argument is not included (`NULL`), the function internally estimates this value to obtain the parameters for fitting the multipopulation mortality model.
- The `periods`, and `ages` vectors reflect the period range and age range, respectively, from the dataset.
- `nPop` is a numeric value that indicates the number of populations/regions considered in the dataset.

Importantly, for the effective utilization of the fitting functions, the array or matrix containing the `qxt` and `lxt` (if it is included) should be organized chronologically with the primary or general population placed first. This fact is essential for the ACFM, which needs to fit first the mortality of the whole group. In the dataset labelled `SpainRegions`, the principal population pertains to the mortality data encompassing the entire nation of Spain and is positioned as the first entry in the dataset.

We demonstrate the application of this function by fitting the additive and multiplicative multipopulation mortality models to the `SpainRegions` dataset for both male and female cases. However, the other multipopulation mortality models can be applied only by modifying the `model` input. Indeed, in the explanation of the function `fitLCmulti()` in the CvmortalityMult *R* package, users can find examples of how other multipopulation mortality models (common-factor, joint-k and augmented common-factor) are fitted and forecasted for male Spain regions. Additionally, we generate a vector containing the lower age in each age group considered in the paper.

```
> SpainRegions
Mortality Data
Spain Regions for males and females
Years 1991 : 2020
Abridged Ages 0 : 90
> ages <- c(0, 1, 5, 10, 15, 20, 25, 30, 35, 40,
+          45, 50, 55, 60, 65, 70, 75, 80, 85, 90)
```

In fact, multiplicative and additive multipopulation mortality models can be fitted using the following code:

```
> additive_Spainmales <- fitLCmulti(model= 'additive', qxt = SpainRegions$qx_male,
+        periods = c(1991:2020), ages = c(ages), nPop = 18, lxt = SpainRegions$lx_male)
> additive_Spainfemales <- fitLCmulti(model= 'additive', qxt = SpainRegions$qx_female,
+        periods = c(1991:2020), ages = c(ages), nPop = 18, lxt = SpainRegions$lx_female)
> multi_Spainmales <- fitLCmulti(model= 'multiplicative', qxt = SpainRegions$qx_male,
+        periods = c(1991:2020), ages = c(ages), nPop = 18, lxt = SpainRegions$lx_male)
> multi_Spainfemales <- fitLCmulti(model = 'multiplicative', qxt = SpainRegions$qx_female,
+        periods = c(1991:2020), ages = c(ages), nPop = 18, lxt = SpainRegions$lx_female)
```

The output from the fitting functions is an object of the class `fitLCmulti`, which provides a brief summary of the fitting process, including among other things, the following information:

- `ax`, `bx`, `kt`, and `Ii` are the estimated parameters for the multipopulation mortality models.
- `formula`, and `model` refer to the gnm formula and the fitted multipopulation mortality model, respectively.
- `data.used` includes mortality rates to fit the mortality data.
- `qxt.crude` refers to the crude values of the probabilities of death for every age, period, and region. These values are provided by the user for fitting the selected mortality model.
- `qxt.fitted`, and `logit.qxt.fitted` are the fitted values of the probabilities of death for every age, period, and region using the multipopulation mortality model on a probability or logit scale $\left(\frac{q_{x,t}}{1-q_{x,t}}\right)$.

Once we have adjusted the crude age-specific probabilities of death for different groups of ages, periods, and regions, the `plot.fitLCmulti()` function allows us to show the parameters obtained. Figures 5 and 6 provide the fitted parameters of the additive and multiplicative multipopulation mortality models, respectively, for male populations in Spain. The plots are generated using the following code:

```
> plot(additive_Spainmales)
> plot(multiplicative_Spainmales)
```

Notably, the `plot.fitLCmulti()` function generates different plots depending on the selected model. For example, if the augmented common-factor model is chosen by setting `model = 'ACFM'`, the plot function will display the estimated parameters, a_{x_i}, B_x, K_t, b_{x_i}, and k_{t_i}, for the provided populations.

Our example is the dataset of Spanish regions. We have included the `SpainMap` function in the package. This function facilitates the plotting of the I_i parameters of the regions of Spain in Figure 7. We recommend reviewing the `regions` dataset to identify the order of the regions before using the `SpainMap` function. In the context of multipopulation mortality models, the multiplicative and additive indices for the regions of Spain (with the reminder that the first population is the national dataset and will not be shown) can be obtained with the following code:

```
> SpainMap(multiplicative_Spainmales$Ii[2:18],
+        main = c("Multiplicative for males"),
+        name = c("Ii"))
> SpainMap(regionvalue = additive_Spainmales$Ii[2:18],
+        main = c("Additive for males"),
+        name = c("Ii"), bigred = FALSE)
```

Additionally, the fitting function applies to a single population. Specifically, we design this function to fit cases where mortality data are provided for only one population, and the Lee-Carter mortality model for a single population is fitted. Users can implement this by using the following *R* code:

```
> LC_SpainNatmales <- fitLCmulti(model = 'additive', qxt = SpainNat$qx_male,
+        periods = c(1991:2020), ages = c(ages), nPop = 1, lxt = SpainNat$lx_male)
> LC_SpainNatfemales <- fitLCmulti(model = 'multiplicative', qxt = SpainNat$qx_female,
+        periods = c(1991:2020), ages = c(ages), nPop = 1, lxt = SpainNat$lx_female)
```

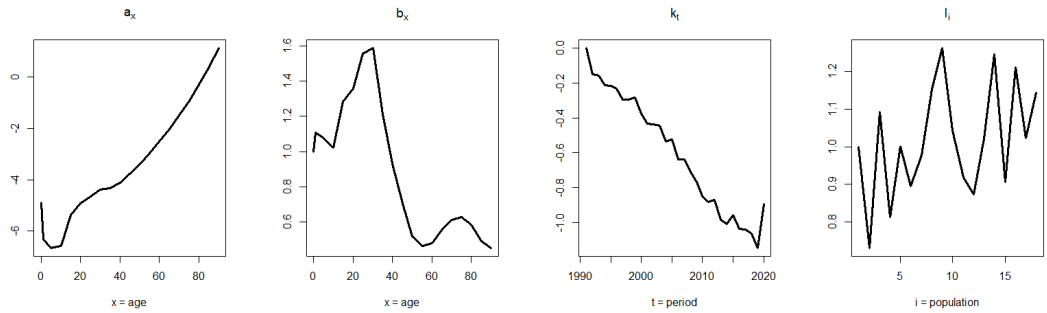


Figure 5: Parameters for the multiplicative mortality model fitted to the male population of Spain for ages 0 – 90 and the period 1991 – 2020.

We use two of the five multipopulation models to demonstrate the operation of the function for one-single population independently of the model provided. However, there is no need to specify this argument in the `fitLCmulti` function, as it inherently fits the Lee-Carter version for a single population.

Similarly, the parameters of the Lee-Carter model for single-population mortality can be plotted using the `plot.fitLCmulti()` function. Therefore, Figure 8 can be obtained using the following code:

```
plot(LC_SpainNatmales)
```

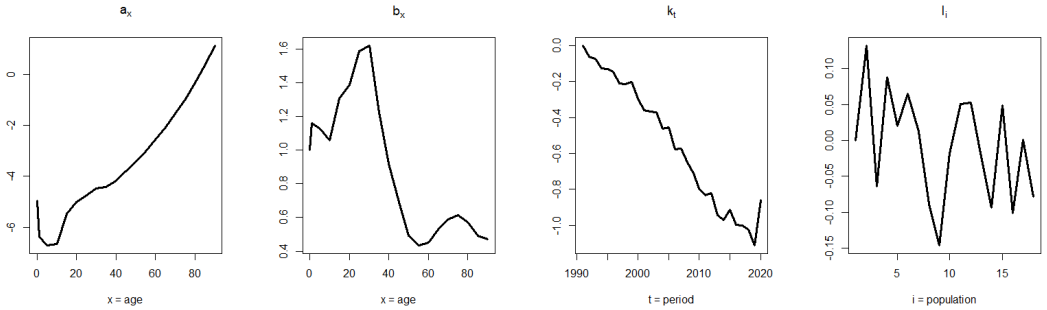


Figure 6: Parameters for the additive mortality model fitted to the male population in Spain regions for ages 0 – 90 and the period 1991 – 2020.

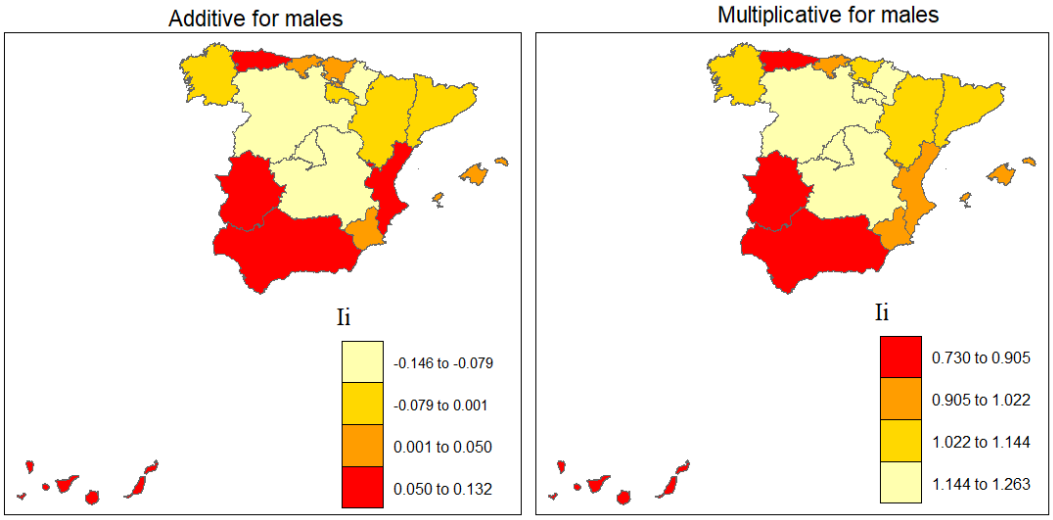


Figure 7: Geographical index, I_i, for multiplicative (left) and additive (right) multipopulation mortality models for the male populations in Spain regions for ages 0 – 90 and the period 1991 – 2020.

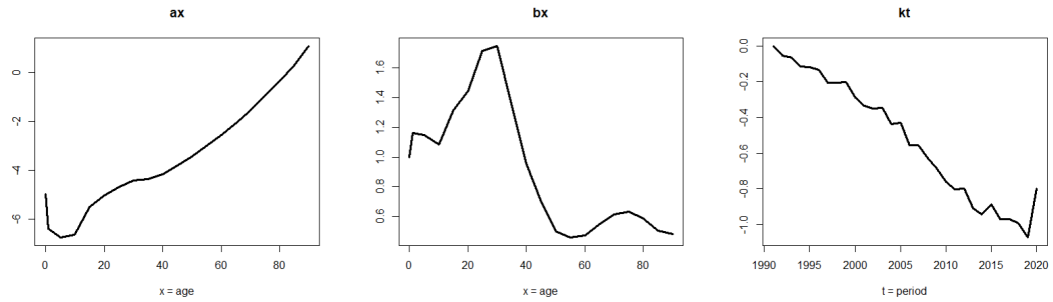


Figure 8: Parameters for the LC single-population mortality model fitted to the Spain male population for ages 0 – 90 and the period 1991 – 2020.

From Figures 5-8, several interesting results emerge:

- All the left panels, a_x, correspond to the average behaviour of age-specific probabilities of death across all studied periods and regions.
- The second panels, b_x, demonstrate how age-specific probabilities of death for each age group (considering all regions) respond to changes in mortality trend, as captured by k_t. Large values of b_x are observed among Spanish males between 20 and 40 years of age; therefore, there is a substantial reduction in the age-specific probabilities of death in this age group (1991 - 2020). This phenomenon is attributed to the impact of AIDS and drugs on Spanish males during the 1980s and 1990s, which led to an initial increase in age-specific probabilities of death and total deaths in these age groups, followed by a significant decline in age-specific probabilities of death due to the introduction of new therapies and medications during the 1990s and 2000s (Felipe et al., 2002; Debón et al., 2008a; Atance et al., 2020a), as can be observed in b_x for the additive and multiplicative models.
- The third panels reveal the impact of the COVID-19 pandemic on the trend parameter k_t for all considered models among males in Spain. Similar trends can be observed for females, although these trends are not shown. However, they can be generated by modifying the fitting object in the plot function. The incorporation of 2020 into the model fitting process induces an upturn in age-specific probabilities of death in the final observed period, disrupting the declining trend observed in the preceding years (1990 - 2019).
- Finally, the right panels, I_i, in the multipopulation approach depict the geographical distribution of the indices corresponding to each region. These panels allow us to discern distinct regional behaviours on the basis of the chosen multipopulation approach. To complement this presentation, we have included Figure 7, which displays the values of I_i for Spanish males in each region using the additive and multiplicative models with the regions with the highest mortality highlighted in red. Notably, the parameter I_i leads to a different interpretation in each model. In the multiplicative model, higher values indicate lower age-specific probabilities of death as the region value is multiplied by the trend parameter. Therefore, with a decreasing trend parameter, as observed in the case of Spain, higher values of I_i correspond to lower age-specific probabilities of death. Conversely, the additive model incorporates the region index to the general trend among the regions ($a_x + b_x \cdot k_t$). Consequently, lower values or the most negative index regions present lower age-specific probabilities of death.

5.2 Forecasting

The CvmortalityMult *R* package enables the projection of future age-specific probabilities of death using the ARIMA (p,d,q) processes. This projection applies to the trend parameters, k_t, K_t, and $k_{t,i}$ in multipopulation mortality models. Two common assumptions for

the trend parameters k_t, K_t, and $k_{t,i}$ in the actuarial and demography literature are often considered: first, a multivariate random walk with drift (ARIMA(0,1,0)) (Cairns et al., 2006, 2009; Haberman and Renshaw, 2011; Villegas et al., 2017), and second, the selection of the best ARIMA (p,d,q) process (Renshaw and Haberman, 2006; Debón et al., 2008b; Villegas et al., 2017; Atance et al., 2020a) for estimating the future values of k_t, K_t, and k_{t_i}. To estimate the future values of the trend parameters k_t, K_t, and k_{t_i}, we employ the `forecast` function from the **forecast** *R* package (Hyndman and Khandakar, 2008), allowing the projection of the future values for various types of ARIMA processes considered in our package. In the CvmortalityMult *R* package, users can choose different ARIMA processes, `ktmethod=c('arima010','arimapdq','arimauser')`. The selection process is applied for single or all trend parameters considered in each multipopulation or single-population mortality model. The ellipsis argument (`...`) provides users with the flexibility to include different ARIMA configurations, changing the arguments in the `auto.arima` or `Arima` functions, depending on the `ktmethod` provided. This functionality mirrors the behavior of the **forecast** *R* package for time series (Hyndman and Khandakar, 2008). Additionally, users must provide `nahead`, indicating the number of periods to forecast the future value of age-specific probabilities of death for each considered region. For example, the code below provides future age-specific probabilities of death for Spanish male and female regions for the next ten years (`nahead = 10`), using different ARIMA options in the package:

```
> fut_additive_Spainmales <- forecast(object = additive_Spainmales,
+        nahead = 10, ktmethod = 'arimapdq')
> fut_multiplicative_Spainmales <- forecast(object = multiplicative_Spainmales,
+        nahead = 10, ktmethod = 'arima010')
> fut_additive_Spainfemales <- forecast(object = additive_Spainfemales,
+        nahead = 10, ktmethod = 'arimapdq')
> fut_multiplicative_Spainfemales <- forecast(object = multiplicative_Spainfemales,
+        nahead = 10, ktmethod = 'arima010')
```

The outputs from these forecast functions are objects of the class `forLCmulti`, which provides a brief summary of the forecasting process, with the following information:

- `ax`, `bx`, `kt`, and `Ii` provide the estimated parameters for the multipopulation mortality models.
- `arimakt` provides the ARIMA (p,d,q) process considered to adjust the time series k_t, K_t, or k_{t_i} and the obtained coefficients.
- `kt.fut` provides the future values of k_t, K_t, or $k_{t,i}$ using the selected ARIMA (p,d,q) configurations for the `nahead` periods.
- `kt.futintervals` provides estimates of the future values of k_t, K_t, or $k_{t,i}$ for the point forecast (`kt.fut`). Additionally, it includes the lower and upper 80% and 95% confidence intervals, utilizing the chosen ARIMA (p,d,q) process for the specified `nahead` periods.
- `formula` and `model` define the `gnm` formula and the forecasted multipopulation mortality model, respectively.
- `qxt.crude` represents the crude values of the probabilities of death for every age, period, and region. These values are provided by the user for fitting the selected mortality models.
- `qxt.future`, and `logit.qxt.future` represent the future values of the probabilities of death for every, age, period and region using the chosen multipopulation mortality model in the probability or logit scale.

Once we have projected the age-specific probabilities of death from different ages, periods, and regions, the `plot.forLCmulti()` function allows us to show the projected values

of trend parameters k_t, K_t, and k_{t_i}. The logit death probabilities for the mean in-sample age and the out-of-sample forecast are shown for all the populations considered. Figures 9 and 10 provide these interesting results for the additive and multiplicative multipopulation mortality models, respectively, for males in Spain. The visualizations are generated with the following code:

```
> plot(fut_additive_Spainmales)
> plot(fut_multiplicative_Spainmales)
```

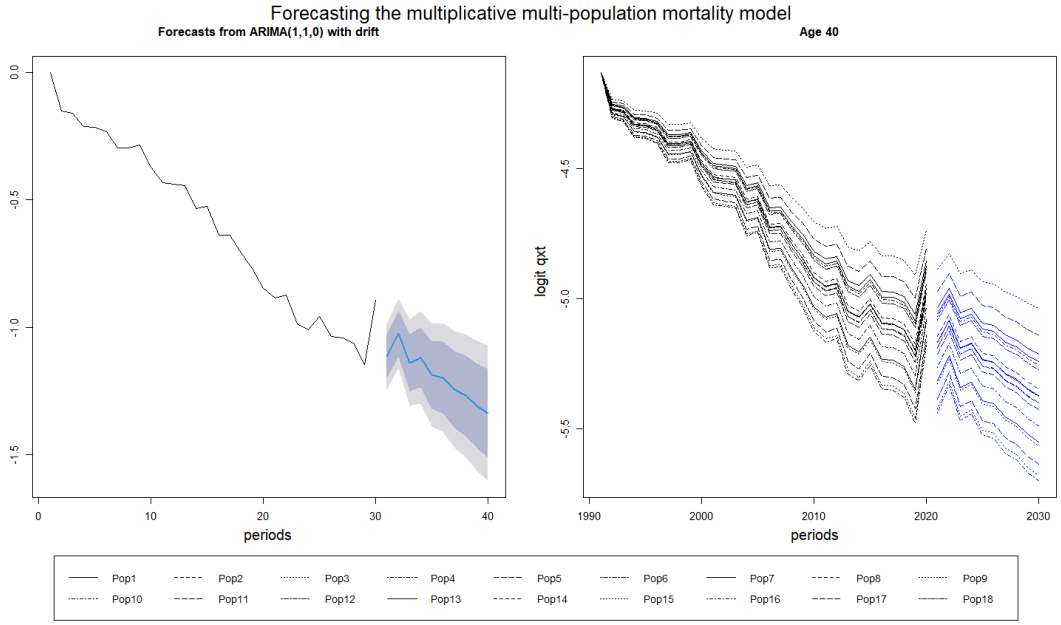


Figure 9: The left panel represents to the in-sample trend parameter k_t and its projected value. The right panel displays the actual and projected logit mortality rates using the multiplicative multipopulation mortality model for the 18 populations considered for age 40 (mean age in the populations considered) in terms of the in-sample period from 1991 – 2020 and the out-of-sample period extending, 10 years ahead.

Similarly, during the forecasting process, users can employ the same function to project future values of age-specific probabilities of death when providing data for a single population. Specifically, the function forecasts age-specific probabilities of death using the Lee-Carter model for a single population, as demonstrated below:

```
> fut_LC_Spainmales <- forecast(object = LC_SpainNatmales,
+         nahead = 10, ktmethod = 'arimapdq')
> fut_LC_Spainfemales <- forecast(object = LC_SpainNatfemales,
+         nahead = 10, ktmethod = 'arima010')
```

Equally, for the single-population mortality model, users can plot two remarkable results. For example, Figure 11 can be obtained using the following code:

```
> plot(fut_LC_Spainmales)
```

5.3 Cross-Validation

In this section, we present the CV function developed in the CvmortalityMult *R* package to evaluate the forecasting ability of multipopulation mortality models with different CV methods. Thus, the CV time series function uses the next synopsis:

```
multipopulation_cv(qxt, model = c('multiplicative', 'additive', 'CFM', 'joint-K', 'ACFM'),
+         periods, ages, nPop, lxt = NULL,
```

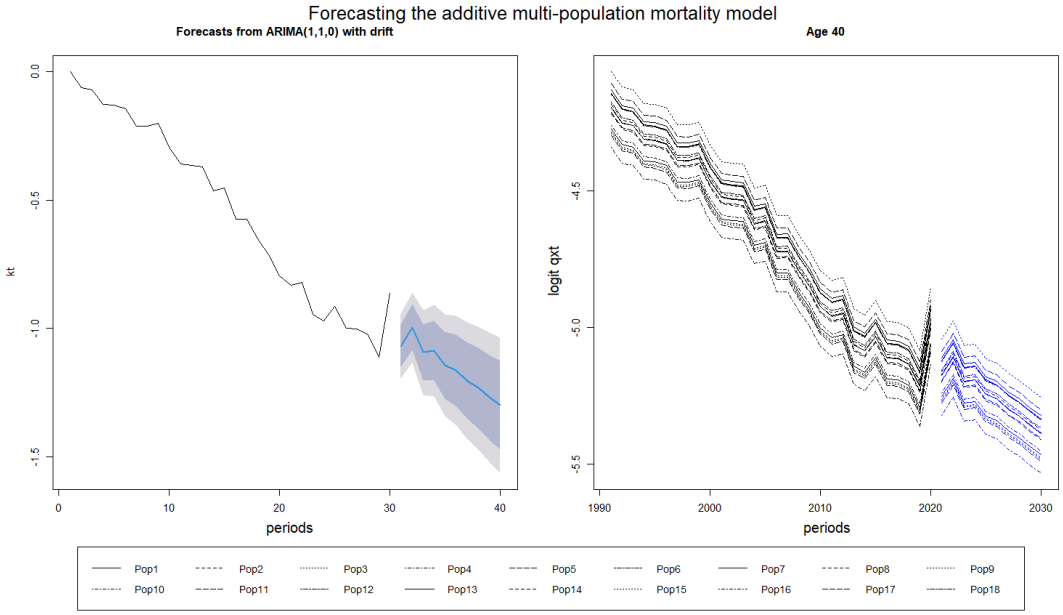


Figure 10: The left panel represents the in-sample trend parameter k_t and its projected value. The right panel displays the actual and projected logit mortality rates using the additive multipopulation mortality model for the 18 populations considered for age 40 (mean age in the populations considered) in terms of the in-sample period from 1991 – 2020 and the out-of-sample period extending, 10 years ahead.

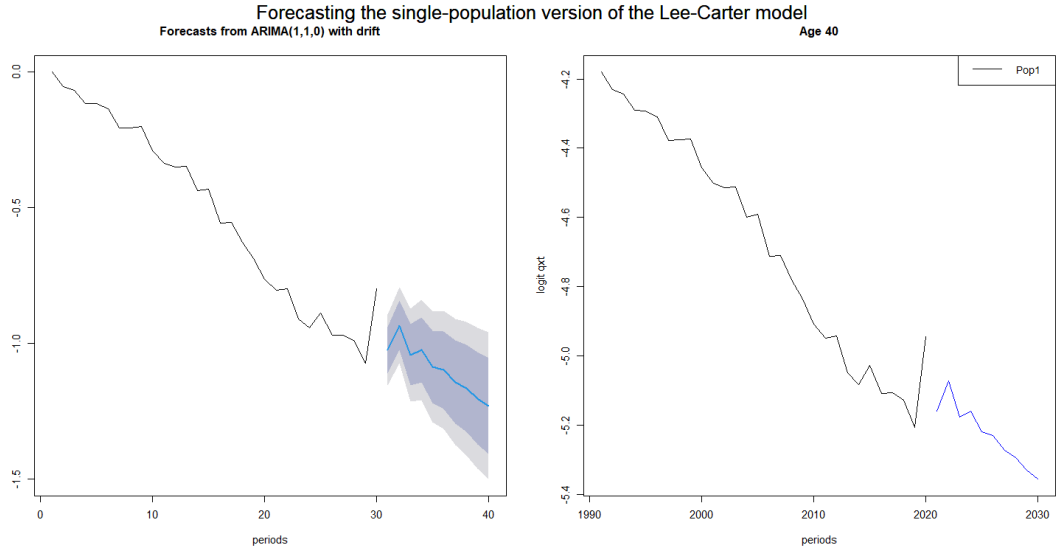


Figure 11: The left panel represents the in-sample trend parameter k_t and its projected value. The right panel displays the actual and projected logit mortality rates using the Lee-Carter model for a single population for the only population considered for age 40 (mean age in the populations considered) in terms of the in-sample period from 1991 – 2020 and the out-of-sample period extending, 10 years ahead.

```
+         nahead, trainset1, fixed_train_origin = TRUE,
+         ktmethod = c('arimapdq', 'arima010', 'arimauser'), order = NULL,
+         measures = c('SSE', 'MSE', 'MAE', 'MAPE', 'All'))
```

This CV function requires the following information as input:

- `qxt`, `lxt`, `periods`, `ages`, `nPop`, `ktmethod` and `order` should match the corresponding values used as in the fitting and forecasting functions for the multipopulation and single mortality models as inputs.

- `model = c('multiplicative','additive','CFM','joint-K','ACFM')` specifies the multipopulation mortality model that users wish to assess for forecasting ability using the specific resampling technique. Users can apply the multiplicative, additive, common-factor, joint-K, and augmented common-factor multipopulation models and the single version of the Lee-Carter model presented in this paper separately.

- `measures = c('SSE','MSE','MAE','MAPE','All')` denotes the adjustment measure that users wish to employ for testing the forecasting ability of the model using the specific resampling technique. If `measures = c('All')`, all the measures will be provided by the function. Additionally, each accuracy measure has a dedicated help function to clarify the underlying equations. Users can access this help function in the CvmortalityMult *R* package using the following code: `?MeasureAccuracy`, where users can select the specific measure of accuracy for testing the age-specific mortality rates (SSE, MSE, MAE or MAPE).

- `trainset1` specifies the number of chronological periods to consider as the initial training set. This value must be greater than 2 to meet the minimum time series size (Hyndman and Khandakar, 2008). Additionally, we recommend that this value be greater than `nahead` to maintain consistency among the forecasts in every iteration (Tashman, 2000).

- `nahead` is the number of periods to project ahead in each iteration and the size of each test set among the selected CV techniques. Moreover, it should be noted that the `multipopulation_cv()` function aims to maintain a uniform length for all the testing sets (iterations). However, the last test set may have fewer periods to align with the total number of periods provided by the user as (`periods`).

- `fixed_train_origin = c(TRUE,FALSE,'add_remove1')` is a logical variable that specifies whether the starting point of the initial training set remains fixed throughout the CV process. This option allows users to maintain a constant starting point where the model will be fitted in every iteration or allow it to shift, thereby determining whether a rolling window evaluation is applied. By default, the package sets `fixed_train_origin = TRUE`, meaning that the first period in the training set remains fixed across all iterations and the model recalibrations of the CV method. However, users can opt to allow the training set starting point to shift by setting `fixed_train_origin = FALSE` or `fixed_train_origin = 'add_remove1'`, thereby implementing a rolling-window evaluation while keeping the training set size constant in each iteration, defined by the user by `nahead` argument. When `fixed_train_origin = FALSE`, in every iteration, the user-defined `nahead` periods are removed from the beginning of the training set, while the next `nahead` periods are added to the training set from the previous test set. Consequently, the number of projected periods (`nahead`) also determines how many periods are added or deleted in every iteration. In contrast, when `fixed_train_origin = 'add_remove1'`, the training set size is also fixed across iterations. However, only one new period is added and one period is removed in each forecast. This process allows users to evaluate the forecasting accuracy of `nahead`-step-ahead projections using more test samples. Users may specify `fixed_train_origin = 'add_remove1'`, and any value of `nahead` greater than or equal to one, provided it is consistent with the number of periods available in the dataset. Notably, when `nahead = 1`, the configuration `fixed_train_origin = 'add_remove1'` yields results equivalent to those obtained using `fixed_train_origin = FALSE`, which means in a LOOCV method keeping the same size of the training set (`trainset1`) across iterations.

The sizes of `nahead` and `trainset1` can be determined by the temporal correlation within the values of the analysed series by using "blocks" of data rather than choosing data randomly (Racine, 2000; Bergmeir and Benítez, 2012).

With this function, the user can apply different CV methods for multipopulation models depending on three main inputs that must be provided: `nahead`, `trainset1`, and `fixed_train_origin`. Indeed, the following CV techniques can be applied:

1. **Fixed-origin evaluation** is implemented by setting the arguments so that nahead + trainset1 = periods while keeping the default value of fixed_train_origin = TRUE.

2. **RO-recalibration evaluation** requires that trainset1 > 2 and that fixed_train_origin = TRUE remain at its default value, regardless of the value assigned to nahead. Specifically, when nahead = 1, leave-one-out CV (LOOCV) is applied. When nahead = trainset1, k-fold CV (CV) is performed. For all other values, a standard time series CV approach is used while keeping the origin of the first training set fixed in all possible options.

3. **Rolling-window evaluation** requires setting fixed_train_origin = FALSE or fixed_train_origin = 'add_remove1', independently of the values assigned to nahead and trainset1. As in the previous CV technique, if nahead = 1, a LOOCV approach with a rolling window of 1 is applied, which remains equivalent whether fixed_train_origin is set to FALSE or 'add_remove1'. When nahead >1 and fixed_train_origin = FALSE, the training set is updated by incorporating and discarding nahead periods in each iteration. Conversely, when fixed_train_origin = 'add_remove1', the training set updates by adding and removing only one observation per iteration while forecasting nahead periods.

We present the results for RO-recalibration using the standard CV approach for male Spanish regions. The main input parameters are set as follows: trainset1 = 8, nahead = 5 and fixed_train_origin = TRUE (default value). This procedure is applied to the five multipopulation mortality models included in the package. To replicate these results, the user can use the following code:

```
> SpainRegions
> ages <- c(0, 1, 5, 10, 15, 20, 25, 30, 35, 40,
+        45, 50, 55, 60, 65, 70, 75, 80, 85, 90)

> cv_SM_multi <- multipopulation_cv(qxt = SpainRegions$qx_male,
+        model = c('multiplicative'), #see options below
+        periods = c(1991:2020), ages = c(ages), nPop = 18, lxt = SpainRegions$lx_male,
+        trainset1 = 8, nahead = 5, ktmethod = c('arimapdq'), measures = c("MSE"))
```

While we executed to female Spanish regions an RO-recalibration was performed using trainset1 = 10, nahead = 1 and fixed_train_origin = TRUE (default value). This configuration corresponds to a LOOCV approach, which fixes the origin in the first training, using the following code:

```
> loocv_SF_multi <- multipopulation_cv(qxt = SpainRegions$qx_female,
+        model = c('multiplicative'), #see options below
+        periods =  c(1991:2020), ages = c(ages), nPop = 18, lxt = SpainRegions$lx_female,
+        trainset1 = 10, nahead = 1, ktmethod = c('arimapdq'), measures = c("MSE"))
```

Available values for model = 'additive','multiplicative','CFM','joint-K', and 'ACFM'. Changing the string assigned to model is sufficient to switch to the corresponding specification. The output from the CV function is an object of the MultiCv class, which provides a brief summary of the CV method employed, including the following information:

- ax, bx, Ii, kt.fitted, kt.future, and kt.arima correspond to the same outputs as those in the adjustment and forecast functions. However, since the adjustment process has been repeated several times (depending on the process), each of these outputs is a list of the iterations executed, denoted as follows: loop-h from period-1 to period-2, where "h" denotes the corresponding iteration.

- meas_ages, meas_periodsfut, meas_pop, and meas_total represent the accuracy measures provided by the resampling technique, each emphasizing different aspects of

the forecasting ability. Indeed, the objective of the CvmortalityMult *R* package is to provide a tool for evaluating the forecasting accuracy of multipopulation models from various ages, namely, across different age groups, future periods, regions considered, or a global measure spanning all ages, future periods, and regions considered. This function allows users the flexibility to choose the specific viewpoint they wish to prioritize in the decision-making process regarding forecasting capabilities.

- `model`, and `CV_method` designate the multipopulation mortality model and the CV-method that users wish to apply for testing the forecasting ability, respectively. Users can apply both, the multiplicative and additive models presented in this paper separately.

Figures 12-13 present the results of the CV techniques for the five multipopulation mortality models applied across ages, periods and regions, respectively. Additionally, we have included Figures 14 and 15 with the MSE measure throughout different regions of Spain only for the multiplicative and additive multipopulation mortality models. The result of the other models are available upon request to the authors and can be found in the reproduction file. These plots can be reproduced using the R script entitled `CvmortalityMult_reproduction.R`. From Figures 13-15, we note the following points:

- The ACFM and joint-K models present lower forecasting results when the MSE measure is used across the ages and future periods considered, using both CV time series techniques.
- The five considered models yield similar results for the age range 0 - 60, whereas for the last section of the mortality curve (60 - 90), the ACFM and joint-K model perform better in terms of the forecasting results.
- Concerning the forecasting periods, the ACFM and joint-K models demonstrate better forecasting results in the medium term, as captured by RO-recalibration CV. However, while the five models exhibit comparable results when evaluating short-term forecasting ability, RO-recalibration LOOCV with the multiplicative model yields the worst result.
- The MSE measures for the different regions of Spain considered for each CV method are shown in Figures 14 and 15. The multipopulation mortality model produces different results depending on the region. Specifically, the multiplicative model yields the best forecasting results for Galicia, Pais Vasco, Cataluña, and Comunidad Valencia. In contrast, depending on the CV model and population considered, the additive model produces superior outcomes for Galicia, Asturias, Pais Vasco, Cataluña, Comunidad Valencia, and Andalucia.

Notably, the CV function allows the computation of global measures of forecasting ability considering all ages, future periods, and regions, as shown in Table 2. The results indicate that the ACFM and joint-K model for standard CV, whereas the CFM and ACFM for LOOCV demonstrate better forecasting ability when all available forecasting information is used. This finding is consistent with the observations mentioned above.

Therefore, the CvmortalityMult *R* package displays the forecasting ability of the multipopulation mortality models in various ways, allowing the user to determine the most suitable model for their specific objectives. We present two alternatives to assess the forecasting ability of the models but there are other additional CV time series techniques that can be implemented with the `multipopulation_cv()` function modifying `nahead`, `trainset1` and `fixed_train_origin`.

6 Summary and discussion

Accurately forecasting age-specific probabilities of death is essential for dealing with life-contingent risk, ensuring solvency within the European (re)insurance industry, and address-

CV method	Multipopulation mortality model				
Common CV males	Multiplicative	Additive	CFM	joint-K	ACFM
MSE	0.000160	0.000142	0.000142	0.000132	0.000124
LOOCV females	Multiplicative	Additive	CFM	joint-K	ACFM
MSE	0.000089	0.000066	0.000061	0.000062	0.000059

Table 2: Summary of the MSE global measure of forecasting ability.

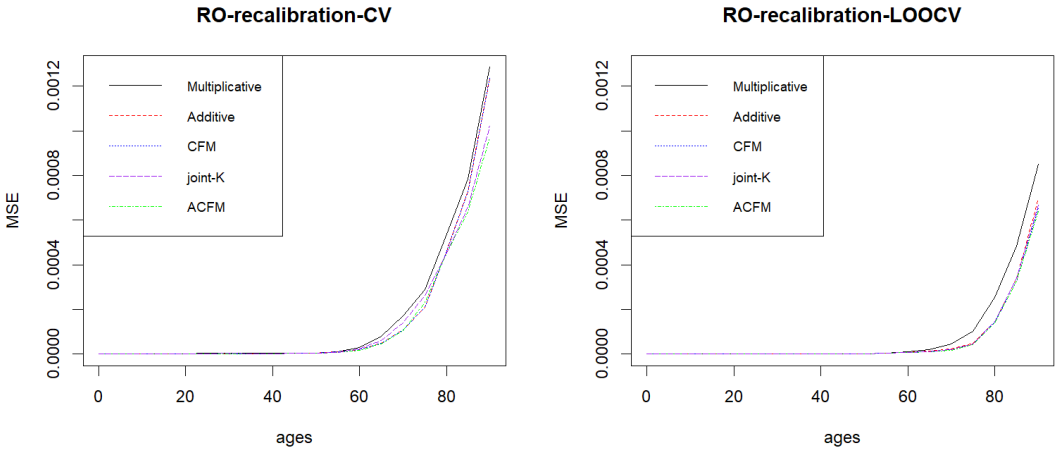


Figure 12: Plot visualizing the MSE over the group of ages considered in regions of Spain for males and females, applying CV time series techniques for five multipopulation mortality models.

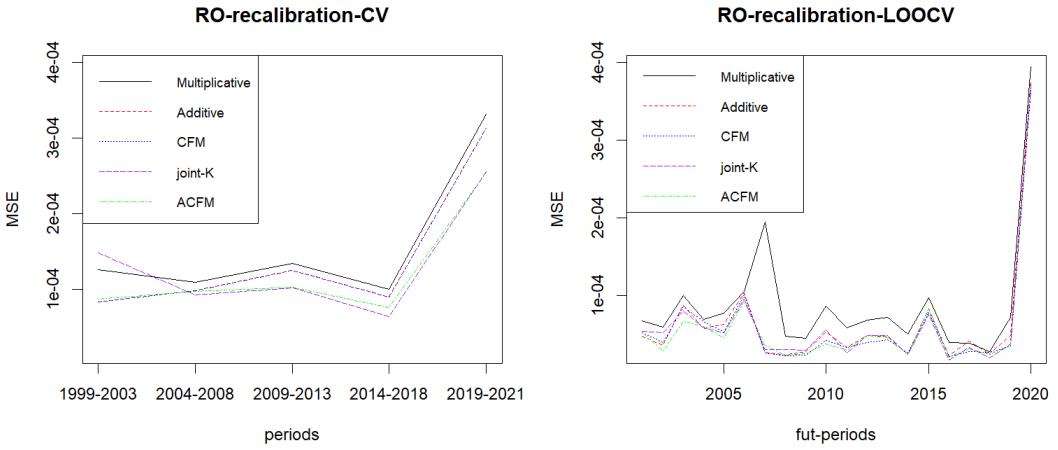


Figure 13: Plot visualizing the MSE over the test sets of future periods considered in regions of Spain for males and females, applying two CV time series techniques for five multipopulation mortality models.

ing the sustainability of public pension system plans, among other purposes. Multipopulation mortality models offer a valuable approach to forecasting age-specific probabilities of death. These models allow the incorporation of data from regions in the same country or group of countries with similar characteristics, transcending borders in a globalized context, where national and international movements occur daily. Moreover, these models are recommended to enrich mortality data with observations from different regions in the same country or group of countries sharing similar characteristics. The CvmortalityMult *R* package facilitates access to five of these multipopulation mortality models, providing an *R* interface to the functions necessary for model fitting and forecasting simply.

Furthermore, comparing various models can be challenging when the most suitable model is selected. Indeed, CV methods offer a valuable tool for evaluating the forecasting

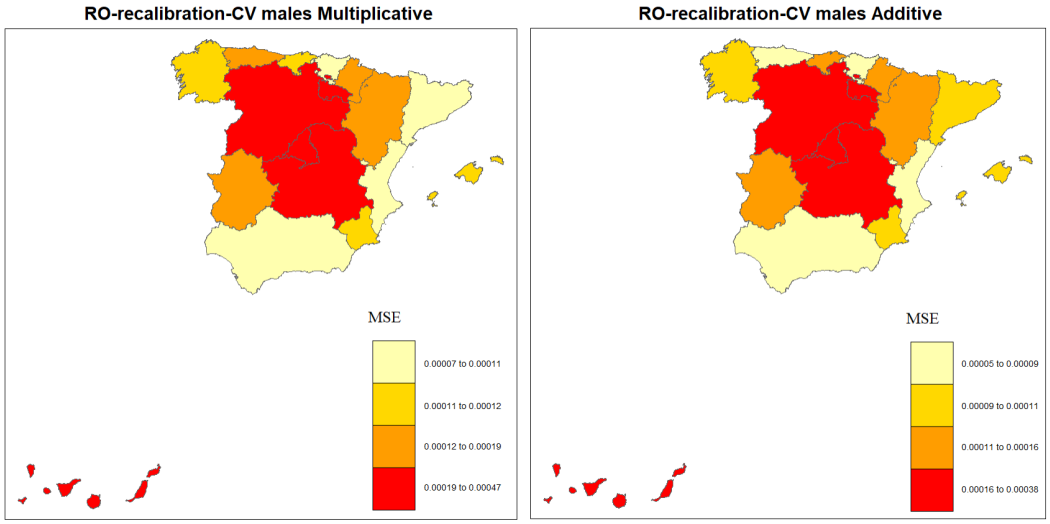


Figure 14: Plot visualizing the MSE in regions of Spain for males, applying RO-recalibration CV for the multiplicative and additive
multipopulation mortality models.

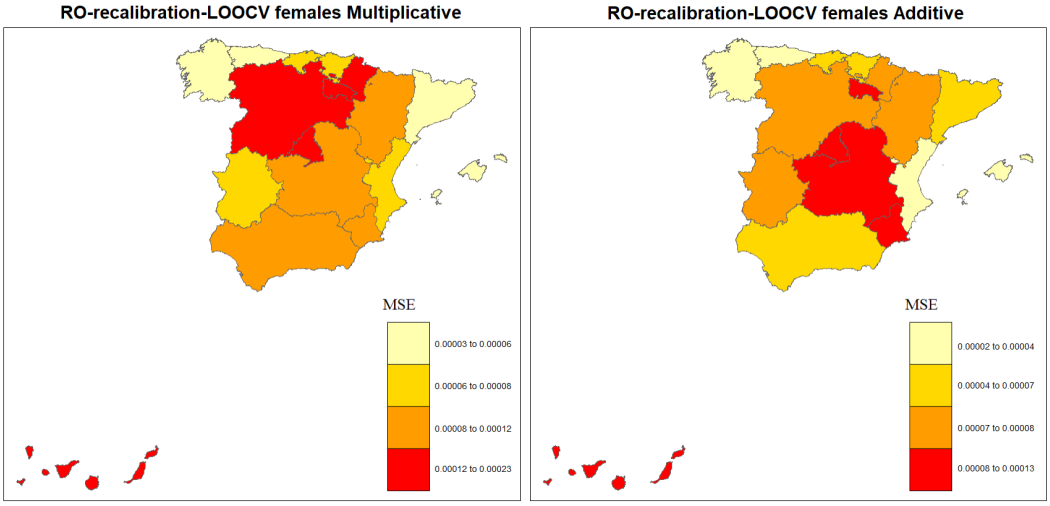


Figure 15: Plot visualizing the MSE in regions of Spain for males, applying RO-recalibration LOOCV for the multiplicative and additive multipopulation mortality models.

ability of models. The CvmortalityMult *R* package allows the application of several CV time series techniques, for assessing the forecasting ability of multiple populations over short, medium and long term horizon. To the best of our knowledge, the CvmortalityMult *R* package is the first to apply these methods to multipopulation mortality models, especially for three-way array data. Users only need to provide multipopulation mortality data, specify the number of periods to be used as the training or testing set, and decide whether it remains fixed at the origin of the first training set. Then, the CvmortalityMult *R* package computes various measures highlighting different forecasting ability aspects. Consequently, users can prioritize selecting the most appropriate multipopulation mortality model on the basis of their specific requirements and perspectives.

7 Acknowledgements

The authors express their gratitude to the anonymous referees for their thorough review and valuable feedback on the manuscript. Appreciation is also extended to Christoph Bergmeir for clarifying doubts related to cross-validation terminology in time series, which

has influenced the final version of the manuscript.
A. Debón's work was partially supported by grants PID2023-152106OB-I00 and CIAICO/2023/272, funded by MCIN/AEI/10.13039/501100011033 and the Generalitat Valenciana, respectively.
D. Atance acknowledges the support of Generalitat Valenciana through projects CIGE/2023/7 (Conselleria de Educación, Universidades y Empleo).

References

A. Ahcan, D. Medved, A. Olivieri, and E. Pitacco. Forecasting mortality for small populations by mixing mortality data. *Insurance: Mathematics and Economics*, 54:12–27, 2014. [p232, 235]

S. S. Ahmadi and J. S.-H. Li. Coherent mortality forecasting with generalized linear models: A modified time–transformation approach. *Insurance: Mathematics and Economics*, 59: 194–221, 2014. [p230]

K. Antonio, S. Devriendt, W. de Boer, R. de Vries, A. De Waegenaere, H.-K. Kan, E. Kromme, W. Ouburg, T. Schulteis, E. Slagter, M. van der Winden, C. van Iersel, and M. Vellekoop. Producing the Dutch and Belgian mortality projections: a stochastic multi–population standard. *European Actuarial Journal*, 7:297–336, 2017. [p230, 235]

J. S. Armstrong and M. C. Grohman. A comparative study of methods for long–range market forecasting. *Management Science*, 19(2):211–221, 1972. [p235, 236]

D. Atance and A. Debón. *Contributions to Risk Analysis: RISK 2022*, chapter Two multi-population mortality models: A comparison of the forecasting accuracy with resampling methods, pages 51–58. Fundación MAPFRE, 2022. [p236]

D. Atance and E. Navarro. A simplified model for measuring longevity risk for life insurance products. *Financial Innovation*, 1:61, 2024. [p236]

D. Atance, A. Balbás, and E. Navarro. Constructing dynamic life tables with a single–factor model. *Decisions in Economics and Finance*, 43(2):787–825, 2020a. [p235, 243, 244]

D. Atance, A. Debón, and E. Navarro. A comparison of forecasting mortality models using resampling methods. *Mathematics*, 8(9):1550, 2020b. [p231, 234, 236]

K. Barigou, P.-O. Goffard, S. Loisel, and Y. Salhi. Bayesian model averaging for mortality forecasting using leave–future–out validation. *International Journal of Forecasting*, 39(2): 674–690, 2023. [p231, 236]

S. M. Bartolomei and A. L. Sweet. A note on a comparison of exponential smoothing methods for forecasting seasonal series. *International Journal of Forecasting*, 5(1):111–116, 1989. [p234]

C. Bergmeir and J. M. Benítez. On the use of cross-validation for time series predictor evaluation. *Information Sciences*, 191:192–213, 2012. [p231, 234, 235, 236, 247]

C. Bergmeir, R. J. Hyndman, and B. Koo. A note on the validity of cross–validation for evaluating autoregressive time series prediction. *Computational Statistics & Data Analysis*, 120:70–83, 2018. [p236]

A. Bozikas and G. Pitselis. Incorporating crossed classification credibility into the Lee–Carter model for multi-population mortality data. *Insurance: Mathematics and Economics*, 93:353–368, 2020. [p230, 235]

N. Brouhns, M. Denuit, and J. K. Vermunt. A Poisson log–bilinear regression approach to the construction of projected lifetables. *Insurance: Mathematics and Economics*, 31(3):373–393, 2002. [p230, 234]

N. Brouhns, M. Denuit, and I. Van Keilegom. Bootstrapping the Poisson log–bilinear model for mortality forecasting. *Scandinavian Actuarial Journal*, 2005(3):212–224, 2005. [p231]

P. Burman. A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods. *Biometrika*, 76(3):503–514, 1989. [p234, 236]

J.-F. Bégin, B. Sanders, and X. Xu. Modeling and forecasting subnational mortality in the presence of aggregated data. *North American Actuarial Journal*, 0(0):1–27, 2023. doi: 10.1080/10920277.2023.2231996. URL https://doi.org/10.1080/10920277.2023.2231996. [p230]

A. Cairns, D. Blake, and K. Dowd. A two–factor model for stochastic mortality with parameter uncertainty: theory and calibration. *Journal of Risk and Insurance*, 73(4):687–718, 2006. [p234, 244]

A. J. Cairns, D. Blake, K. Dowd, G. D. Coughlan, D. Epstein, A. Ong, and I. Balevich. A quantitative comparison of stochastic mortality models using data from England and Wales and the United States. *North American Actuarial Journal*, 13(1):1–35, 2009. [p234, 244]

J. L. Callen, C. C. Kwan, P. C. Yip, and Y. Yuan. Neural network forecasting of quarterly accounting earnings. *International Journal of Forecasting*, 12(4):475–482, 1996. [p236]

L. R. Carter and R. D. Lee. Modeling and forecasting US sex differentials in mortality. *International Journal of forecasting*, 8(3):393–411, 1992. [p231, 233, 239]

R. Y. Chen and P. Millossovich. Sex–specific mortality forecasting for UK countries: a coherent approach. *European Actuarial Journal*, 8(1):69–95, 2018. [p230]

V. D'Amato, S. Haberman, G. Piscopo, and M. Russolillo. Modelling dependent data for longevity projections. *Insurance: Mathematics and Economics*, 51(3):694–701, 2012. [p231]

I. L. Danesi, S. Haberman, and P. Millossovich. Forecasting mortality in subpopulations using Lee–Carter type models: a comparison. *Insurance: Mathematics and Economics*, 62:151–161, 2015. [p230, 235]

A. Debón, F. Montes, J. Mateu, E. Porcu, and M. Bevilacqua. Modelling residuals dependence in dynamic life tables: A geostatistical approach. *Computational Statistics & Data Analysis*, 52(6):3128–3147, 2008a. [p243]

A. Debón, F. Montes, and F. Puig. Modelling and forecasting mortality in Spain. *European Journal of Operational Research*, 189(3):624–637, 2008b. [p231, 234, 244]

A. Debón, F. Montes, and F. Martínez-Ruiz. Statistical methods to compare mortality for a group with non–divergent populations: an application to Spanish regions. *European Actuarial Journal*, 1(2):291–308, 2011. [p230, 231, 232, 233, 239]

Y. Dong, F. Huang, H. Yu, and S. Haberman. Multi–population mortality forecasting using tensor decomposition. *Scandinavian Actuarial Journal*, 2020(8):754–775, 2020. [p231]

K. Dowd, A. J. Cairns, D. Blake, G. D. Coughlan, and M. Khalaf-Allah. A gravity model of mortality rates for two related populations. *North American Actuarial Journal*, 15(2):334–356, 2011. [p230]

R. Elandt-Johnson and N. Johnson. *Survival Models and Data Analysis*. Wiley, New York, 1980. [p238]

V. Enchev, T. Kleinow, and A. J. Cairns. Multi–population mortality models: fitting, forecasting and comparisons. *Scandinavian Actuarial Journal*, 2017(4):319–342, 2017. [p239]

P. England and R. Verrall. Analytic and bootstrap estimates of prediction errors in claims reserving. *Insurance: Mathematics and Economics*, 25(3):281–293, 1999. [p230]

A. Felipe, M. Guillén, and A. Perez-Marin. Recent mortality trends in the Spanish population. *British Actuarial Journal*, 8(4):757–786, 2002. [p243]

S. Haberman and A. Renshaw. A comparative study of parametric mortality projection models. *Insurance: Mathematics and Economics*, 48(1):35–55, 2011. [p232, 234, 244]

J. D. Hart. Automated kernel smoothing of dependent data by using time series cross–validation. *Journal of the Royal Statistical Society Series B: (Methodological)*, 56(3):529–542, 1994. [p236]

T. Hastie, R. Tibshirani, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer Series: New York, NY, USA, 2009. [p231, 234, 236]

T. Hoedemakers, J. Beirlant, M. J. Goovaerts, and J. Dhaene. Confidence bounds for discounted loss reserves. *Insurance: Mathematics and Economics*, 33(2):297–316, 2003. [p230]

A. Hunt and D. Blake. Identifiability in age/period/cohort mortality models. *Annals of Actuarial Science*, 14(2):500–536, 2020. [p234]

R. J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, Melbourne, Australia, 3rd edition edition, 2021. URL http://OTexts.org/fpp3/. [p231, 235]

R. J. Hyndman and Y. Khandakar. Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software*, 26(3):1–22, 2008. doi: 10.18637/jss.v027.i03. [p234, 244, 247]

R. J. Hyndman and M. S. Ullah. Robust forecasting of mortality and fertility rates: a functional data approach. *Computational Statistics & Data Analysis*, 51(10):4942–4956, 2007. [p231]

R. J. Hyndman, H. Booth, and F. Yasmeen. Coherent mortality forecasting: the product–ratio method with functional time series models. *Demography*, 50:261–283, 2013. [p233]

R. J. Hyndman, H. Booth, L. Tickle, and J. Maindonald. *demography: Forecasting mortality, fertility, migration and population data*, 2019. URL https://CRAN.R-project.org/package=demography. R package version 1.22. [p231]

R. J. Hyndman, G. Athanasopoulos, C. Bergmeir, G. Caceres, L. Chhay, M. O'Hara-Wild, F. Petropoulos, S. Razbash, E. Wang, and F. Yasmeen. *forecast: Forecasting functions for time series and linear models*, 2023. URL https://pkg.robjhyndman.com/forecast/. R package version 8.21.1. [p234]

G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning: with applications in R*, volume 1. Springer: New York, NY, USA, 2013a. [p230]

G. James, D. Witten, T. Hastie, R. Tibshirani, et al. *An introduction to statistical learning with application in R*, volume 112. Springer Series: New York, NY USA, 2013b. [p236]

S. F. Jarner and E. M. Kryger. Modelling adult mortality in small populations: the SAINT model. *ASTIN Bulletin*, 41(2):377–418, 2011. [p230]

S. R. Kessy, M. Sherris, A. M. Villegas, and J. Ziveyi. Mortality forecasting using stacked regression ensembles. *Scandinavian Actuarial Journal*, 2022(7):591–626, 2022. [p231]

M.-C. Koissi, A. F. Shapiro, and G. Högnäs. Evaluating and extending the Lee–Carter model for mortality forecasting: Bootstrap confidence interval. *Insurance: Mathematics and Economics*, 38(1):1–20, 2006. [p231]

P. A. Lachenbruch and M. R. Mickey. Estimation of error rates in discriminant analysis. *Technometrics*, 10(1):1–11, 1968. [p235]

R. D. Lee. The Lee–Carter method for forecasting mortality, with various extensions and applications. *North American Actuarial Journal*, 4(1):80–91, 2000. [p232]

R. D. Lee and L. R. Carter. Modeling and forecasting US mortality. *Journal of the American Statistical Association*, 87(419):659–671, 1992. [p230, 231, 232, 239]

H. Li and C. O'Hare. Mortality forecasting: How far back should we look in time? *Risks*, 7 (1):22, 2019. [p236]

J. S.-H. Li and M. R. Hardy. Measuring basis risk in longevity hedges. *North American Actuarial Journal*, 15(2):177–200, 2011. [p230]

N. Li and R. D. Lee. Coherent mortality forecasts for a group of populations: an extension of the Lee-Carter method. *Demography*, 42(3):575–594, 2005. [p230, 231, 233, 239]

M. Lindholm and L. Palmborg. Efficient use of data for LSTM mortality forecasting. *European Actuarial Journal*, 12:749–778, 2022. [p231]

X. Liu and W. J. Braun. Investigating mortality uncertainty using the block bootstrap. *Journal of Probability and Statistics*, 2010:1–15, 2010. [p231]

S. Muriel, M. Cantalapiedra, and F. López. Towards advanced methods for computing life tables. Technical report, Instituto Nacional de Estadística, 2010. URL www.ine.es. [p238]

P. N. Pant and W. H. Starbuck. Innocents in the forest: Forecasting and research methods. *Journal of Management*, 16(2):433–460, 1990. [p234]

M. D. Pascariu. *MortalityLaws: Parametric Mortality Models, Life Tables and HMD*, 2022. URL https://CRAN.R-project.org/package=MortalityLaws. R package version 1.9.4. [p232]

W. Perks. On some experiments in the graduation of mortality statistics. *Journal of the Institute of Actuaries*, 63(1):12–57, 1932. [p232]

E. Pitacco, M. Denuit, S. Haberman, and A. Olivieri. *Modelling longevity dynamics for pensions and annuity business*. Oxford University Press, Oxford, 2009. [p238]

R Core Team. *R: a language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2022. URL https://www.R-project.org/. [p233]

J. Racine. Consistent cross–validatory model–selection for dependent data: hv–block cross–validation. *Journal of Econometrics*, 99(1):39–61, 2000. [p247]

A. E. Renshaw and S. Haberman. A cohort–based extension to the Lee–Carter model for mortality reduction factors. *Insurance: Mathematics and Economics*, 38(3):556–570, 2006. [p234, 244]

M. Russolillo, G. Giordano, and S. Haberman. Extending the Lee–Carter model: a three–way decomposition. *Scandinavian Actuarial Journal*, 2011(2):96–117, 2011. [p230, 231, 232, 239]

J. Shao. Linear model selection by cross–validation. *Journal of the American statistical Association*, 88(422):486–494, 1993. [p236]

D. SriDaran, M. Sherris, A. M. Villegas, and J. Ziveyi. A group regularisation approach for constructing generalised age–period–cohort mortality projection models. *ASTIN Bulletin*, 52(1):247–289, 2022. [p231, 234]

N. R. Swanson and H. White. Forecasting economic time series using flexible versus fixed specification and linear versus nonlinear econometric models. *International Journal of Forecasting*, 13(4):439–461, 1997. [p236]

L. J. Tashman. Out–of–sample tests of forecasting accuracy: an analysis and review. *International Journal of Forecasting*, 16(4):437–450, 2000. [p231, 234, 235, 236, 247]

H. Turner and D. Firth. *gnm: Generalized Nonlinear Models*, 2023. URL https://CRAN.R-project.org/package=gnm. R package version 1.1-5. [p233, 239]

A. M. Villegas and S. Haberman. On the modeling and forecasting of socioeconomic mortality differentials: an application to deprivation and mortality in England. *North American Actuarial Journal*, 18(1):168–193, 2014. [p230]

A. M. Villegas, S. Haberman, V. K. Kaishev, and P. Millossovich. A comparative study of two–population models for the assessment of basis risk in longevity hedges. *ASTIN Bulletin*, 47(3):631–679, 2017. [p231, 244]

A. M. Villegas, V. K. Kaishev, and P. Millossovich. StMoMo: An R package for stochastic mortality modeling. *Journal of Statistical Software*, 84(3):1–38, 2018. doi: 10.18637/jss.v084.i03. [p234, 239]

J. Wilmoth and T. Valkonen. A parametric representation of mortality differentials over age and time. In *Fifth Seminar of the EAPS Working Group on Differentials in Health, Morbidity and Mortality in Europe, Pontignano, Italy*, 2001. [p231, 233]

David Atance
Universidad de Alcalá
Departmento de Economía y Dirección de Empresas
Facultad de Ciencias Económicas, Empresariales y Turismo
Plaza San Diego s/n, 6020, Alcalá de Henares
Spain
ORCiD: 0000-0001-5860-0584
david.atance@uah.es

Ana Debón
Universitat Politècnica de València
Centro de Gestión de la Calidad y del Cambio
Facultad de Administración y Dirección de Empresas
Camino de Vera, s/n, 46022, Valencia
Spain
ORCiD: 0000-0002-5116-289X
andeau@eio.upv.es

Bioconductor Notes, September 2025

by Maria Doyle, Bioconductor Community Manager, and Bioconductor Core Developer Team

1 Introduction

Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data. The project has entered its twenty-first 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.

2 Software

```
#> Loading required package: htmlwidgets

#> 'getOption("repos")' replaces Bioconductor standard repositories, see
#> 'help("repositories", package = "BiocManager")' for details.
#> Replacement repositories:
#>     CRAN: https://cloud.r-project.org
#> 'getOption("repos")' replaces Bioconductor standard repositories, see
#> 'help("repositories", package = "BiocManager")' for details.
#> Replacement repositories:
#>     CRAN: https://cloud.r-project.org
#> 'getOption("repos")' replaces Bioconductor standard repositories, see
#> 'help("repositories", package = "BiocManager")' for details.
#> Replacement repositories:
#>     CRAN: https://cloud.r-project.org
#> 'getOption("repos")' replaces Bioconductor standard repositories, see
#> 'help("repositories", package = "BiocManager")' for details.
#> Replacement repositories:
#>     CRAN: https://cloud.r-project.org
```

Bioconductor 3.21, released in April 2025, is now available. It is compatible with R 4.4 and consists of 2341 software packages, 432 experiment data packages, 928 up-to-date annotation packages, 30 workflows, and 8 books. Books are built regularly from source, ensuring full reproducibility; an example is the community-developed Orchestrating Single-Cell Analysis with Bioconductor.

3 Core Team and Infrastructure Updates

Of note:

A pair of machines with NVIDIA GPUs were added to the system. See devel branch build reports for more information.

An HDF5Array vignette includes new performance analysis data.

NEWS summaries for three contributed packages chosen at random from the 72 new software contributions are:

- **CARDspa**: CARD is a reference-based deconvolution method that estimates cell type composition in spatial transcriptomics based on cell type specific expression information obtained from a reference scRNA-seq data. A key feature of CARD is its ability to accommodate spatial correlation in the cell type composition across tissue locations, enabling accurate and spatially informed cell type deconvolution as

well as refined spatial map construction. CARD relies on an efficient optimization algorithm for constrained maximum likelihood estimation and is scalable to spatial transcriptomics with tens of thousands of spatial locations and tens of thousands of genes.

- **G4SNVHunter** G-quadruplexes (G4s) are unique nucleic acid secondary structures predominantly found in guanine-rich regions and have been shown to be involved in various biological regulatory processes. G4SNVHunter is an R package designed to rapidly identify genomic sequences with G4-forming potential and accurately screen user-provided single nucleotide variants (also applicable to single nucleotide polymorphisms) that may destabilize these structures. This enables users to screen key variants for further experimental study, investigating how these variants may influence biological functions, such as gene regulation, by altering G4 formation.
- **jazzPanda** This package contains the function to find marker genes for image-based spatial transcriptomics data. There are functions to create spatial vectors from the cell and transcript coordinates, which are passed as inputs to find marker genes. Marker genes are detected for every cluster by two approaches. The first approach is by permutation testing, which is implemented in parallel for finding marker genes for one sample study. The other approach is to build a linear model for every gene. This approach can account for multiple samples and background noise.

See the NEWS section in the release announcement for a complete account of changes throughout the ecosystem.

4 Community and Impact

4.1 Outreachy Internships

The December–March 2025 Outreachy program concluded successfully, with interns contributing to Bioconductor and sharing their reflections in a blog post. We also highlighted the rewarding experience of mentoring for Outreachy.

4.2 Community Updates

The Bioconductor community has migrated from Slack to Zulip for discussions; read more about the move here. We also reflected on our two years of Carpentries membership.

In July 2025, Bioconductor blog posts became available through R-bloggers, broadening their reach to the wider R community. One of our first posts became one of the week's most-read, a promising sign of growing visibility. We welcome contributions to the Bioconductor blog; if you have an idea, please get in touch via Zulip or email.

4.3 Publications and Preprints

Vince Carey, a long-time member of the Bioconductor Core Team, has published his reflections on the evolving complexity of the Bioconductor ecosystem in a recent journal article, "Bioconductor: Planning a third decade of comprehensive support for genomic data science". He also shared his thoughts in a blog post, "Ask and you shall receive".

5 Conferences and Workshops

5.1 Recaps

- **Developers' Forum:** A Bioconductor Developers' Forum was held on July 28, 2025. The event sparked discussions that have led to the creation of a new #rust channel on

the Bioconductor Zulip, for community members interested in using Rust with R and Bioconductor.

- **GBCC 2025:** The Galaxy and Bioconductor Community Conference (GBCC 2025) was held in June 2025. A full report of the conference can be found in the July 2025 Galaxy Community Newsletter. Recordings of talks are available in a YouTube playlist.
- **Bioconductor in scverse workshop:** A successful workshop was held in January 2025, focusing on the integration of Bioconductor tools with the scverse ecosystem. Read the recap here.
- **Global Training:** We have had a busy year of training events, with courses in Kenya, a microbiome course in Brazil, and a course in Ethiopia (blog post forthcoming).

5.2 Announcements

- **EuroBioC 2025:** The European Bioconductor conference is taking place from September 17-19, with pre-conference workshops from September 15-16. For more information, visit the conference website.
- **BioCAsia 2025:** BioCAsia 2025 will be held as part of the ABACBS conference in Adelaide on November 27. The focus is on hands-on workshops, and registration is now open. For more information, visit the conference website.

6 Project News

6.1 ggplot2 4.0.0 and Bioconductor

Version 4.0.0 of `ggplot2` was released on September 11, 2025. The new version introduces a significant internal change from the S3 to the S7 object system, and breakages in some Bioconductor packages that customize `ggplot2`'s functionality. We received early notification from ggplot2 developers through Bioconductor Zulip, and outlined the potential impact on developers and users, and provided guidance on how to adapt to this transition in a blog post in July. The `ggplot2` team's proactive communication ahead of their release helped the Bioconductor community prepare, and we appreciate their support.

6.2 Project Collaborations

We highlight several new collaborations:

- **Bioconductor to Galaxy:** This project aims to improve the integration of Bioconductor tools within the Galaxy platform. A group worked on this at the GBCC2025 CoFest, and you can read more about it in this blog post.
- **EDAM Ontology:** We continue our work on the EDAM ontology to improve the FAIRness of our packages. See the latest update here. Our project was selected for the annual BioHackathon Europe in Berlin, November 3-7. In-person places are full but remote participation is possible and free.
- **Physalia Collaboration:** We have partnered with Physalia Courses to offer high-quality training in computational biology. Read about it here.

7 Boards and Working Groups Updates

The Community Advisory Board (CAB) and the Technical Advisory Board (TAB) held their annual call for new members, which closed on August 31, 2025. The selection process is currently underway, and new members will be announced later this year.

In March 2025, we published a blog post introducing Stevie Pederson as the new CAB Co-Chair. The post also highlights the role of the CAB, its working groups, and ways for community members to get involved.

8 Using Bioconductor

Start using Bioconductor by installing the most recent version of R and evaluating the commands

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install()
```

Install additional packages and dependencies, e.g., SingleCellExperiment, with

```
BiocManager::install("SingleCellExperiment")
```

Docker images provides a very effective on-ramp for power users to rapidly obtain access to standardized and scalable computing environments. Key resources include:

- bioconductor.org to install, learn, use, and develop Bioconductor packages.
- A list of available software linking to pages describing each package.
- A question-and-answer style user support site and developer-oriented mailing list.
- A community Zulip workspace (sign up) for extended technical discussion.
- The 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, in addition to video recordings of training courses.
- Our package submission repository for open technical review of new packages.

Upcoming and recently completed events are browsable at our events page.

The Technical and Community Advisory Boards provide guidance to ensure that the project addresses leading-edge biological problems with advanced technical approaches, and adopts practices (such as a project-wide Code of Conduct) that encourages all to participate. We look forward to welcoming you!

We welcome your feedback on these updates and invite you to connect with us through the Bioconductor Zulip workspace or by emailing community@bioconductor.org.

Maria Doyle, Bioconductor Community Manager
University of Limerick

Bioconductor Core Developer Team
Dana-Farber Cancer Institute, Roswell Park Comprehensive Cancer Center, City University of New York, Fred Hutchinson Cancer Research Center, Mass General Brigham

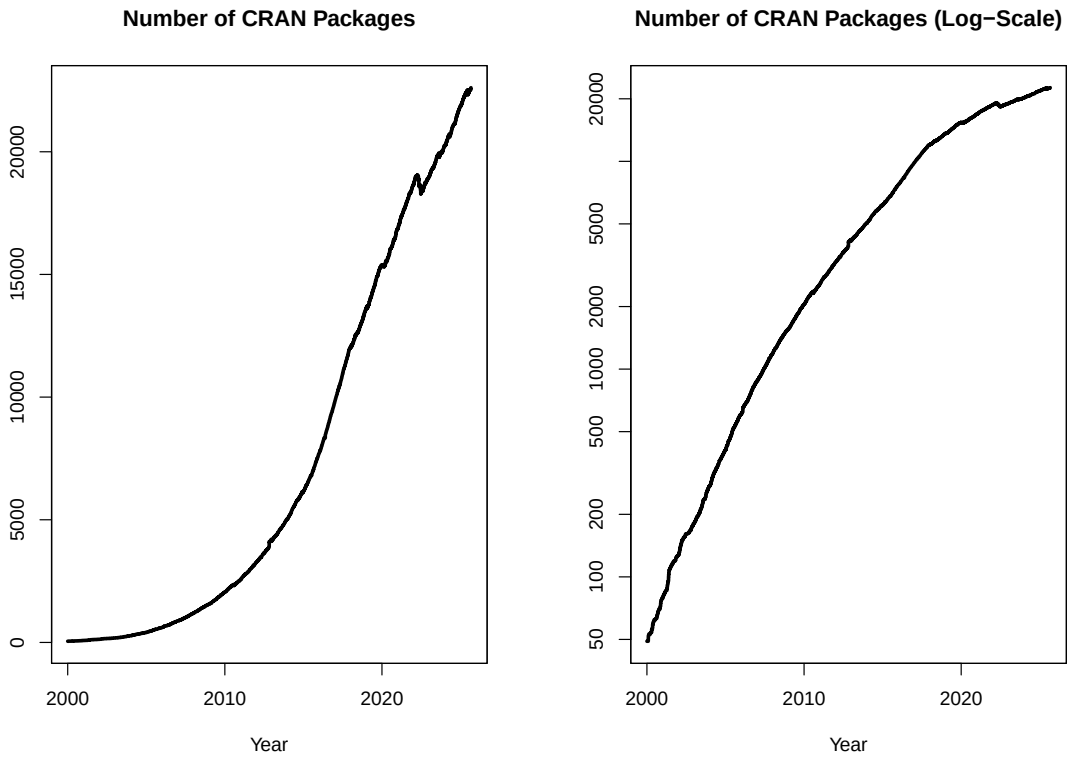
Changes on CRAN

2025-04-01 to 2025-06-30

by Kurt Hornik, Uwe Ligges, and Achim Zeileis

1 CRAN growth

In the past 3 months, 546 new packages were added to the CRAN package repository. 145 packages were unarchived, 570 were archived and 0 had to be removed. The following shows the growth of the number of active packages in the CRAN package repository:



On 2025-06-30, the number of active packages was around 22397.

2 CRAN package submissions

From April 2025 to June 2025 CRAN received 7476 package submissions. For these, 11748 actions took place of which 8890 (76%) were auto processed actions and 2858 (24%) 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	2423	693	1619	132	0	2576	877	311
manual	1166	0	11	5	31	1284	291	69

These include the final decisions for the submissions which were

	archive	publish
auto	2338 (31.8%)	2327 (31.7%)
manual	1152 (15.7%)	1528 (20.8%)

where we only count those as *auto* processed whose publication or rejection happened automatically in all steps.

3 CRAN mirror security

Currently, there are 93 official CRAN mirrors, 77 of which provide both secure downloads via '`https`' *and* use secure mirroring from the CRAN master (via rsync through ssh tunnels). Since the R 3.4.0 release, `chooseCRANmirror()` offers these mirrors in preference to the others which are not fully secured (yet).

4 CRAN Task View Initiative

Currently, there are 48 task views (see https://CRAN.R-project.org/web/views/), with median and mean numbers of CRAN packages covered 107 and 122, respectively. Overall, these task views cover 4891 CRAN packages, which is about 22% of all active CRAN packages.

Kurt Hornik
WU Wirtschaftsuniversität Wien
Austria
ORCiD: 0000-0003-4198-9911
Kurt.Hornik@R-project.org

Uwe Ligges
TU Dortmund
Germany
ORCiD: 0000-0001-5875-6167
Uwe.Ligges@R-project.org

Achim Zeileis
Universität Innsbruck
Austria
ORCiD: 0000-0003-0918-3766
Achim.Zeileis@R-project.org

R Foundation News

by Torsten Hothorn

1 Donations and members

Membership fees and donations received between 2025-03-26 and 2025-09-16.

1.1 Donations

Richard Cousineau (Canada)

1.2 Supporting institutions

Alfred Mueller Analytic Services, München (Germany) Departement Klinische Forschung, Basel (Switzerland) Ef-prime, Inc., Tokyo (Japan) NIFU Nordic Institute for Studies in Innovation, Research and Education, Oslo (Norway) The University of Auckland, Statistics Department, Auckland (New Zealand) University of Iowa, Iowa City (United States)

1.3 Supporting members

Vedo Alagic (Austria) Jose Alaya (Peru) Kristoffer Winther Balling (Denmark) Maurice Baudet von Gersdorff (Germany) Amit Behera (United States) Ashanka Beligaswatte (Australia) Chris Billingham (United Kingdom) Emmanuel Blondel (France) Tom Boulay (United States) Andreas Büttner (Germany) Robert Carnell (United States) Chao Cheng (China) William Chiu (United States) Thomas Collins (United States) Giuseppe Corbelli (Italy) Charles Cowens (United States) Alistair Cullum (United States) Ajit de Silva (United States) Elliott Deal (United States) Dubravko Dolic (Germany) Serban Dragne (United Kingdom) Mitch Eppley (United States) Guenter Faes (Germany) Aurélien Ginolhac (Luxembourg) Susan Gruber (United States) Chris Hanretty (United Kingdom) James Harris (United States) Benedikt Haug (Netherlands) Takehiko Hayashi (Japan) Knut Helge Jensen (Norway) Brian Johnson (United States) Christian Kampichler (Netherlands) Katharina Kesy (Germany) An Khuc (United States) Mohammad Golam Kibria (United States) Sebastian Koehler (Germany) Miha Kosmac (United Kingdom) Sebastian Krantz (Germany) Luca La Rocca (Italy) Teemu Daniel Laajala (Finland) Jindra Lacko (Czechia) Bernardo Lares (Venezuela) Rory Lawless (United States) Thierry Lecerf (Switzerland) Mauro Lepore (United States) Eric Lim (United Kingdom) Michal Majka (Austria) Ivan Marino (Italy) Harvey Minnigh (Puerto Rico) David Monterde (Spain) Markus Näpflin (Switzerland) Maciej Nasinski (Poland) Mark Niemann-Ross (United States) Jens Oehlschlägel (Germany) Jaesung James Park (Korea, Republic of) Matt Parker (United States) Josiah Parry (United States) Bill Pikounis (United States) Kelly Pisane (Netherlands) Dominic Schuhmacher (Germany) Christian Seubert (Austria) Jagat Sheth (United States) Sindri Shtepani (Canada) Murray Sondergard (Canada) Marco Steenbergen (Switzerland) Berthold Stegemann (Germany) Emi Tanaka (Australia) Tim Taylor (United Kingdom) Chris Toney (United States) Nicholas Turner (United States) Philipp Upravitelev (Russian Federation) Mark van der Loo (Netherlands) Frans van Dunné (Costa Rica) Vincent van Hees (Netherlands) Fredrik Wartenberg (Sweden) Yang Hu (New Zealand)

Torsten Hothorn
Universität Zürich
Switzerland
ORCiD: 0000-0001-8301-0471
Torsten.Hothorn@R-project.org